# GORE CLASSIFICATION AND CENSORING IN IMAGES

**WILLIAM LAROCQUE**

Thesis submitted to the University of Ottawa
in partial Fulfillment of the requirements for the degree of
Master of computer science
with concentration in Applied Artificial Intelligence

Ottawa-Carleton Institute for Computer Science
University of Ottawa
Ottawa, Canada

# ABSTRACT

With the large amount of content being posted on the Internet every day, moderators, investigators, and analysts can be exposed to hateful, pornographic, or graphic content as part of their work. Exposure to this kind of content can have a severe impact on the mental health of these individuals. Hence, measures must be taken to lessen their mental health burden. Significant effort has been made to find and censor pornographic content; gore has not been researched to the same extent. Research in this domain has focused on protecting the public from seeing graphic content in images, movies, or online videos. However, these solutions do little to flag this content for employees who need to review such footage as part of their work. In this thesis, we aim to address this problem by creating a full image processing pipeline to find and censor gore in images. This involves creating a dataset, as none are publicly available, training and testing different machine learning solutions to automatically censor gore content.

We propose an Image Processing Pipeline consisting of two models: a classification model which aims to find whether the image contains gore, and a segmentation model to censor the gore in the image. The classification results can be used to reduce accidental exposure to gore, by blurring the image in the search results for example. It can also be used to reduce processing time and storage space by ensuring the segmentation model does not need to generate a censored image for every image submitted to the pipeline. Both models use pretrained Convolutional Neural Network (CNN) architectures and weights as part of their design and are fine-tuned using Machine Learning (ML). We have done so to maximize the performance on the small dataset we gathered for these two tasks. The segmentation dataset contains 737 training images while the classification dataset contains 3830 images.

We explored various variations on the proposed models that are inspired from existing solutions in similar domains, such as pornographic content detection and censoring and medical wound segmentation. These variations include Multiple Instance Learning (MIL), Generative Adversarial Networks (GANs) and Mask R-CNN. The best classification model we trained is a voting ensemble that combines the results of 4 classification models. This model achieved a 91.92% Double F1-Score, 87.30% precision, and 90.66% recall on the testing set. Our highest performing segmentation model achieved a testing Intersection over Union (IoU) value of 56.75%. However, when we employed the proposed Image Processing Pipeline (classification followed by segmentation), we achieved a testing IoU of 69.95%.

# ACKNOWLEDGEMENTS

I am deeply indebted to my two thesis supervisors, Dr. Hussein Al Osman and Dr. Robert Laganière, for the knowledge and experience they shared with me throughout this process. Without their help and support, the ambitious goals that I set out to attain in this thesis will not have been achievable. I also want to thank Dr. Jochen Lang and Dr. Matthew Holden for their evaluations to this thesis.

I cannot begin to express my love, thanks and appreciation to my parents, Lise Guillemette, and Richard LaRocque, and to my brother, Alexandre LaRocque. Their unwavering support especially during the COVID-19 pandemic and the multiple lockdowns has been invaluable. Their willingness to understand my work has not only been emotionally rewarding, but it also became a way to assess and reframe my research progress. This led to improvement of my research methodology, among other things. I am also extremely grateful to my grandparents, Roger and Colette Guillemette as well as Fernand and Véronique LaRocque, for their willingness to see me succeed and their financial support which allowed me to flourish in my post-secondary studies.

I am very grateful to my past and future colleagues in the Government of Canada as they not only provided the research question solved in this thesis, but they also showed me that my desire to help people using Artificial Intelligence was not just a dream, but a fundamental part of their culture and organizational goals.

Finally, I must thank the University of Ottawa, the Vector Institute, and the National Research Council Canada as, without their financial support, my graduate studies and, by extension, this thesis will not have been possible.

# TABLE OF CONTENTS

ABSTRACT ..... ii
ACKNOWLEDGEMENTS ..... iii
TABLE OF CONTENTS ..... iv
LIST OF FIGURES ..... vii
LIST OF TABLES ..... viii
NOMENCLATURE ..... x
GLOSSARY ..... xi
Chapter 1. Introduction ..... 1
1.1 Problem statement ..... 2
1.2 Motivation ..... 2
1.3 Thesis contribution ..... 2
1.4 Thesis structure ..... 3
Chapter 2. Background and Related Work ..... 5
2.1.1 Deep Learning ..... 5
2.1.2 Convolutional Neural Networks ..... 6
2.1.3 Image Classification ..... 11
2.1.4 Image Segmentation ..... 15
2.2 Related Work ..... 20
2.2.1 Harmful Content classification and Censoring ..... 20
2.2.2 Nudity/Pornographic Content classification solutions ..... 20
2.2.3 Violent Content Classification ..... 21
2.2.4 Harmful Content Censoring ..... 23
2.2.5 Medical Wound Segmentation ..... 24
Chapter 3. Methodology ..... 26
3.1 DL Model Training and Testing ..... 26
3.1.1 Order of the Experimental Phases ..... 27
3.2 Image Processing Pipeline ..... 28
Chapter 4. Building a Dataset ..... 29
4.1 Annotation Methodology ..... 29
4.1.1 Segmentation Annotation ..... 29
4.1.2 Classification Annotation ..... 30
4.1.3 Mask R-CNN Annotation ..... 30
4.2 Gathering images ..... 31
4.2.1 Initial Dataset (Segmentation) ..... 31

4.2.2 Reddit Scraping for Classification ..... 34
4.3 Summary of Chapter 4 ..... 35
Chapter 5. Gore Classification ..... 36
5.1 Training Methodology ..... 36
5.2 Experimental Results ..... 37
5.2.1 Phase 1 – Evaluation metric experimentation ..... 38
5.2.2 Phase 2 – Data Augmentation Ablation ..... 39
5.2.3 Phase 3 – Loss function experimentation ..... 39
5.2.4 Phase 4 – CNN encoder experimentation ..... 41
5.2.5 Phase 5 – Uncertainty Estimation ..... 44
5.2.6 Phase 6 – Ensembles ..... 48
5.2.7 Phase 7 – MIL for Gore Classification ..... 55
5.2.8 Phase 8 – Further Uncertainty Experimentation ..... 56
5.2.9 Comparison with Google Cloud's SafeSearch ..... 60
5.2.10 Analysis of Voting Ensembles Results ..... 61
5.2.11 Summary of Chapter 5 ..... 62
Chapter 6. Gore Segmentation ..... 65
6.1 Training Methodology ..... 65
6.2 Experimental Results ..... 67
6.2.1 Phase 1 – Loss function experimentation ..... 67
6.2.2 Phase 2 – CNN encoder experimentation for U-Net ..... 67
6.2.3 Phase 3 – Segmentation model architecture experimentation ..... 68
6.2.4 Phase 4 – CNN encoder experimentation for FPN24 ..... 69
6.2.5 Phase 5 – Experimentation with GANs ..... 70
6.2.6 Phase 6 – Mask R-CNN ..... 73
6.2.7 Phase 7 – Optimal Thresholding ..... 74
6.2.8 Phase 8 - Ablation study ..... 75
6.2.9 Summary of Chapter 6 ..... 77
Chapter 7. Image Processing Pipeline Testing, Deployment and Use ..... 79
7.1 Pipeline Testing ..... 79
7.1.1 Gore Classification ..... 80
7.1.2 Gore Segmentation ..... 81
7.2 Analysis of the Pipeline's Segmentation results ..... 82
7.3 Summary of Chapter 7 ..... 83
Chapter 8. Conclusion ..... 84
8.1 Limitations ..... 85
8.2 Future Work ..... 86

Bibliography ..... 87
Appendix A – Proof of equivalence of the F-Score ..... 103
Appendix B – Pipeline Testing Results ..... 104
B1 - Classification ..... 104
B2 – Segmentation ..... 108
Appendix C - BentoML Services Creation ..... 113
Services Overview ..... 113
Gore Classification Service ..... 114
Gore Segmentation Service ..... 114
BentoML Service Use in Client Applications ..... 115

# LIST OF FIGURES

Figure 1 - Gore Classification Model Architecture. The image is first passes through a CNN encoder then a global pooling average layer and a fully connected network (head of the network). ..... 37
Figure 2 - Post Training Precision Recall Curves. This curve is used for optimal thresholding after training. We use the lowest threshold that achieves either best F1-score or 90% recall for testing. This ensures the model achieve at least a validation recall of a 90%. ..... 37
Figure 3 - Distribution of testing uncertainty for VGG16 model trained with WRL. The uncertainty was computed using the infer-dropout method of Mi et al. [61]. The uncertainty for correct predictions is for the most part lower than the uncertainty of wrong predictions. ..... 47
Figure 4 - Distribution of testing uncertainty for VGG16 model trained with BCE loss. The uncertainty was computed using the infer-dropout method of Mi et al. [61]. The uncertainty for correct predictions is for the most part lower than uncertainty of wrong predictions. This model is more certain than WRL model when making negative predictions. ..... 48
Figure 5 - Cascading Ensemble based on model uncertainty. A model is uncertain if the decision threshold is within a standard deviation of the mean prediction (Equation (5.3)). ..... 49
Figure 6 - Distribution of testing uncertainty for Cascading VGG16 model. The uncertainty was computed using the infer-dropout method of Mi et al. [61]. The first quartile for FPs predictions is lower than in Figure 3 and Figure 4. ..... 49
Figure 7 – Diagram of a Voting Ensemble with learned weighted voting layer. ..... 51
Figure 8 – Testing uncertainty distribution for the 'Voting_WRL' ensemble without no thresholding prior to voting. The uncertainty was computed using the infer-dropout method of Mi et al. [61]. The uncertainty of wrong predictions is much more consistent than models presented in Figures Figure 3, Figure 4 and Figure 6. ..... 52
Figure 9 - Diagram of a Stacking ensemble for Image classification ..... 54
Figure 10 - Segmentation Data Augmentation (Upper bound). This image shows the result of the 5 alteration types using the upper bound values of the random range used when making the augmentation during training. Image used in this figure [204] is in the public domain. ..... 66
Figure 11 - Segmentation Data Augmentation (Lower bound). This image shows the result of the 5 alteration types using the lower bound values of the random range used when making the augmentation during training. Image used in this figure [204] is in the public domain. ..... 66
Figure 12 - Gore Censoring Image Processing Services Workflow ..... 79
Figure 13 - Flowchart of BentoML Service use. It shows how the services are deployed only when a request for its used has been made. ..... 113

# LIST OF TABLES

Table 1 - Image sources for the Initial dataset ..... 32
Table 2 - Image sources of the Reddit additional classification data ..... 35
Table 3 - Comparison of the testing performance of Evaluation Metrics for Gore Classification. Highlighted cells represent the best results on each metric. ..... 39
Table 4 - Effect of Data Augmentation for Gore Classification on the testing results. All models use the MobileNetV2 encoder trained with BCE Loss with early stopping based on validation Double F1-Score. Highlighted cells represent the best results on each metric with recall and Double F1-Score being the most important. 39
Table 5 - Comparison of the testing performance of Loss Functions for Gore Classification. All models use MobileNetV2 encode, are trained with Data Augmentation and early stopping on validation Double F1-Score. Highlighted cells represent the best results on each metric with recall and Double F1-Score being the most important ..... 41
Table 6 - Testing Results of Multiple Models trained with BCE and WRL. All models use MobileNetV2 encode, are trained with Data Augmentation and early stopping on validation Double F1-Score. Highlighted cells represent the best results on each metric with recall and Double F1-Score being the most important. ..... 41
Table 7 - Comparison of the testing performance of CNN Encoders for gore classification. All models are trained with Data Augmentation, WRL and early stopping on validation Double F1-Score. Highlighted cells represent the best results on each metric with recall and Double F1-Score being the most important. ..... 43
Table 8 - Comparison of the testing performance of CNN Encoders at threshold value of 0.5. All models are trained with Data Augmentation, WRL and early stopping on validation Double F1-Score. Highlighted cells represent the best results on each metric with recall and Double F1-Score being the most important. ..... 43
Table 9 - Uncertainty Estimation for CNN Encoders using Inter-Transformation algorithm on the testing set. All models are trained with Data Augmentation, WRL and early stopping on validation Double F1-Score. Highlighted cells represent the best results on each metric with recall and Double F1-Score being the most important. 45
Table 10 - Testing Results (with uncertainty) of Multiple Models trained on BCE loss and WRL with MobileNetV2. The random uncertainty allowed us to test for statistical significance of the performance resulting from training on the two loss functions. Highlighted cells represent the best results on each metric with recall and Double F1-Score being the most important. ..... 46
Table 11 - Uncertainty Estimation of VGG16 models with BCE loss and WRL on the testing set. In this case, we tried all the uncertainty estimation methods presented in Mi et al. [61]. ..... 46
Table 12 - Comparison of testing results of the Cascading model and the models used to make it. Highlighted cells represent the best results on each metric with recall and Double F1-Score being the most important. The cascading ensemble was able to achieve a very slight improvement in terms of Double F1-Score compared to the standalone VGG16 WRL model. The cascading model did not achieve lower uncertainty as hypothesized. ..... 50
Table 13 - Comparison of testing results of different Voting Ensembles for Gore Classification. Highlighted cells in gold represent best performance at the threshold of 0.5. Green coloured cells represent the best performance at the threshold which achieved the highest validation F1-Score. The recall and Double F1-Score being the most important ..... 53
Table 14 - Stacking Ensembles Gore Classification Testing Results. Highlighted cells represent the best results on each metric with recall and Double F1-Score being the most important. ..... 55
Table 15 - MIL Experimental Results for Gore Classification. Highlighted cells in gold represent best performance at the threshold of 0.5. Green coloured cells represent the best performance at the threshold which achieved the highest validation F1-Score. The recall and Double F1-Score being the most important. ..... 56
Table 16 - Testing results of voting models when using the standard deviation during inference. Rows with similar results are coloured using the same colour. Optimal thresholding is able to achieve better precision at higher recall values than when using the standard deviation at a fixed threshold (0.5). ..... 57
Table 17 - Performance analysis of Voting ensembles with different number of uncertainty samples. For a sample size of one, no random dropout is used to ensure no randomness in the results. Highlighted cells show the best results for the evaluation metrics for a given threshold and model. ..... 59

Table 18 - Google SafeSearch's Gore Classification model comparison on the testing set. Highlighted cells represent the best results on each metric with recall and Double F1-Score being the most important. ..... 60
Table 19 - Comparison of the testing performance of Loss Functions for Gore Segmentation on U-Net Architecture. Highlighted cells show the best performance on each metric. ..... 67
Table 20 - Comparison of the testing performance of CNN Encoders for Gore Segmentation on U-Net Architecture. Highlighted cells show the best performance on each metric. When two cells are highlighted for the same metric, it is because the difference is too small to tell the difference between both model ($< 5 * 10 - 4$)..... 68
Table 21 - Comparison of the testing performance of segmentation Model Architectures for Gore Segmentation. Highlighted cells show the best performance on each metric. When two cells are highlighted for the same metric, it is because the difference is too small to tell the difference between both model ($< 5 * 10 - 4$)..... 69
Table 22 - Comparison of the testing performance of CNN Encoders for Gore Segmentation on FPN24 Architecture. Highlighted cells show the best performance on each metric. When two cells are highlighted for the same metric, it is because the difference is too small to tell the difference between both model ($< 5 * 10 - 4$)..... 70
Table 23 – Testing results of Segmentation Pix2Pix GAN Experimentation's models using EfficientNetB7 FPN24 Generator with ImageNet Weights. Highlighted cells show the best performance on each metric. ..... 72
Table 24 - Testing results of Segmentation Pix2Pix GAN Experimentation's models using pretrained EfficientNetB7 FPN24 Generator. Highlighted cells in gold show the best performance on each metric. The IoU values highlighted in blue are when the GAN fine-tuned model improved its performance due to the fine-tuning..... 72
Table 25 – Testing results of Mask R-CNN Gore Segmentation Models. Highlighted cells show the best performance on each metric. When two cells are highlighted for the same metric, it is because the difference is too small to tell the difference between both model ($< 5 * 10 - 4$). The asterisk in this table denotes a model which the training was restarted after the initial 600 epochs were over. ..... 74
Table 26 – Effect of validation Optimal Thresholding for Gore Segmentation on the testing results. Highlighted cells show the best performance on each metric. ..... 75
Table 27 – Testing results of models of our Gore Segmentation Ablation Study of our Training setup. Highlighted cells show the best performance on each metric. ..... 76
Table 28 - Testing results of Fine-Tuning Augmentation-less models with Pix2Pix GAN. Highlighted cells show the best performance on each metric. ..... 76
Table 29 - Gore Segmentation Models standalone testing performance before the Image Processing Pipeline. Highlighted cells show the best performance on each metric. ..... 78
Table 30 - Pipeline Classification Testing Results at 0.5 classification threshold. Highlighted cells in gold show the best performance on each metric. The cells in blue are the best performance of the 4 models of Phase 4... 81
Table 31 - Pipeline Segmentation Testing Results at 0.5 classification threshold. Highlighted cells in gold show the best performance on each metric. ..... 82

# NOMENCLATURE

| | |
|---|---|
| AI | Artificial Intelligence |
| API | Application Programming Interface |
| BCE | Binary Cross-entropy |
| CNN | Convolutional Neural Network |
| COCO | Common Objects in Context |
| DNN | Deep Neural Network |
| DL | Deep Learning |
| $F_\beta$ | F-Score with parameter $\beta$ |
| FCNN | Fully Connected Neural Network |
| FL | Focal Loss |
| FN | False Negative |
| FP | False Positive |
| FPN | Feature Pyramid Network |
| GAN | Generative Adversarial Network |
| GPU | Graphics Processing Unit |
| IoU | Intersection over Union |
| IQR | Interquantile range |
| MIL | Multiple Instance Learning |
| ML | Machine Learning |
| MSE | Mean Square Error |
| NSFW | Not Safe For Work |
| PTSD | Post-Traumatic Stress Disorder |
| ReLU | Rectified linear unit |
| SE | Squeeze-and-Excitation |
| SGD | Stochastic Gradient Descent |
| TN | True Negative |
| TP | True Positive |
| VSD | Violent Scenes Dataset |
| WRL | Weighted Recall Loss |
| 2D | Two dimensional |
| 3D | Three dimensional |
| $\beta$ | Weighting parameter for F-Score |
| $\eta$ | Learning rate |

# GLOSSARY

| ACRONYM/TERM | DEFINITION |
|---|---|
| Backbone | Refers to the model performing feature extraction of the given input to be used by the rest of a larger architecture. The backbone is usually pretrained on another dataset as most of the coarse features it learned can be used even for new applications. |
| classification | Process that labels a given input into one or a set of classes. |
| Convolution | Used as a synonym of cross-correlation. |
| Cross-Correlation | “Process of moving a filter mask (kernel) over the image and computing the sum of products at each location.” [1] |
| Data Augmentation | “Techniques used to increase the amount of data by adding slightly modified copies of already existing data or newly created synthetic data from existing data.” [2] |
| Downsampling | Process by which the size of the input vector or matrix is reduced by a given factor or to a given size. |
| Dropout | A layer in Neural Networks which replaces input values at random location with zeros. |
| Early Stopping | Process used to stop a model’s training when the results on a holdout dataset (the validation set) stops improving. |
| Encoder | See “Backbone” |
| entropy | “The average level of "information", "surprise", or "uncertainty" inherent in the [random] variable's possible outcomes.” [3] |
| Ensemble | “A set of individually trained classifiers (such as neural networks or decision trees) whose predictions are combined when classifying novel instances.” [4] |
| Evaluation Metric | “Used to measure the quality of the statistical or machine learning model.” [5] |
| False Negative | A positive example misclassified as a negative. |
| False Positive | A negative example misclassified as a positive. |
| F-Score | A metric used to evaluate and weight both precision and recall. |
| Fully Connected Neural Network | Classical Deep Learning model architecture where the inputs of each neuron of a layer contains all output of the neurons of the previous layer. |
| Gore | “Gruesomeness depicted in vivid detail.” [6] |
| Image Processing Pipeline | |
| Image segmentation | “Process of partitioning a digital image into multiple segments (sets of pixels, also known as image objects).” [7] |
| Learning rate | Rate at which the model learns from the training samples at each iteration. |
| Loss Function | A function that defines the error (or loss) of the model given an input and the expected results for that input. A loss function is used to optimize the model’s results during the Machine Learning training phase. |
| Metric | See “Evaluation Metric” |
| Patience | The number of training epochs without evaluation metric increase before the training is stopped when early stopping is used during training. |
| Pooling Function | Function that “progressively reduces the spatial size of the representation to reduce the number of parameters and computation of the network.” [8] |
| Precision | The fraction of relevant instances among the retrieved instances. [9] |
| Recall | The fraction of the total amount of relevant instances that were retrieved. [9] |

| | |
|---|---|
| Transfer Learning | A research problem in machine learning (ML) that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem. [10] |
| True Negative | A negative example correctly classified as a negative. |
| True Positive | A positive example correctly classified as a positive. |
| Uncertainty | Confidence of a model in its prediction [11]. Usually calculated by introducing random variations somewhere in the model and seeing its effect on the predictions. |
| Upsampling | Process by which the size of the input vector or matrix is increased by a given factor or to a given size. |
| Wound | An injury to the body (as from violence, accident, or surgery) that typically involves laceration or breaking of a membrane (such as the skin) and usually damage to underlying tissues [12] |

# Chapter 1. Introduction

Recently, the automated detection of harmful content on the Internet has become a prominent research topic. These endeavours have focused on the development of computer algorithms to recognize hateful speech, pornographic, child exploitation, violent, and graphic content. Conventionally, users and content moderators are the ones that will flag and remove such content. With the massive amount of data generated every minute on the internet [13, 14], manual detection cannot keep up with the magnitude of the posted content. Moreover, it has been proven that content moderators working for Facebook and Google suffer lasting effects, such as post-traumatic stress disorder (PTSD), resulting from exposure to graphic content [15, 16, 17]. As such, Artificial Intelligence (AI) methods are currently being researched to scale the detection effort to keep up with the amount of content created while reducing the amount of content moderators must view as part of their job [15, 18]. It is reasonable to expect that national security analysts and criminal investigators suffer similar repercussions to content moderators as they are also exposed to similar content as part of their work.

Violent and graphic content detection efforts have mainly focused on flagging harmful content in videos and movies. The Violent Scenes Dataset (VSD) [19] provides a dataset for the purpose of finding content that is not suitable for children. It also provides a testing set that can be used to benchmark solutions created to solve address this issue [19]. However, the definition of gore and graphic content differs from one application to another depending on who views the content. This is because content moderators, analyst and investigators may become somewhat desensitized to gore and graphic content due to repeated exposure while the public, especially children, are not. Thus, the data and methods used in children protection research may not transfer to the detection of gore in images for professionals, as the notion of gore differs. Even the learned features of the models may not be transferable between both tasks.

The automated detection of pornographic content and medical imagery has been studied extensively [20, 21, 22, 23]. Pornographic data is widely available on the internet and, as penalties arising from the distribution of child pornography are very severe, companies have expended important efforts to detect, remove and report it. For medical imagery, a multitude of techniques have been used. Researchers have conventionally made assumptions to simplify the identification process, such as supposing that the target regions (i.e., wounds) only appear near or within a region of skin-like colour [24, 25, 26]. Recent studies have shown that Convolutional Neural Network (CNN) models can be trained to effectively classify pornographic content and wounds in a medical setting. Similarly, segmentation models, Object Detection models and Generative Adversarial Networks (GAN) have been used to censor pornographic content in images or segment medical images. These solutions will be studied and adapted, if possible, for gore classification and censoring as part of this thesis.

### 1.1 Problem statement

We hypothesize that using deep learning models, we can automatically censor egregious depictions of gore in images. If such models can be successful, they can then be included as part of a gore censoring system that will be particularly useful for professionals who may be exposed to gory images as part of their work. The system will seek to reduce the exposure to such content as much as possible. To do so, it will first need to determine which images have a high probability of containing gore to reduce accidental exposure. Second, it will need to assess where the gore is within the image to censor it while removing as little information as possible from the overall image, allowing observers to glean other potentially useful information. The first task can be fulfilled by an image classification model. The second could be achieved using either Object Detection or Image segmentation models.

The term gore has many varied definitions. Thus, it is important that we clarify what we mean by gore in this work. The Merriam-Webster dictionary defines gore as "gruesomeness depicted in vivid detail." [6] It also defines gruesomeness as "inspiring horror or repulsion" [27]. We adopt this definition for our work as research has shown that horror and disgust lead to traumatic memories and in some cases PTSD [28]. Wounds and burns can be defined as gore in our work if viewing triggers a disgust reaction. Also, large quantities of bodily fluids or entrails are also considered gore according to this definition.

### 1.2 Motivation

Throughout my Master's degree studies, I, the author of this thesis, worked for the Government of Canada. This included a work term with an Applied Research team that was developing Image Processing tools for National Defence analysts. The images that analysts may be exposed to came from a multitude of sources and could contain violent, pornographic or gore content that should ideally be hidden from them. PTSD has affected analysts in the past and part of the mandate of the new system was to help mitigate this problem [15, 16, 17]. Their system used Yahoo's Not Safe For Work (NSFW) classifier to blur images containing pornographic content. While the blurring could also be used to censor gore, analysts and the applied research team were interested in a smarter censoring solution which will hide the gore while allowing the rest of the image to remain visible for analysis. However, there was no such solution for gore. Since the team was small, they did not have enough resources to create a dataset and develop a solution to this problem. As it piqued my and my supervisors interest, and seemed like a realistic problem to solve, we offered to tackle it as part of this thesis. Hence, we worked for that organization on this project for a full work term in the summer of 2020. We then continued our work on this project in the following semesters.

### 1.3 Thesis contribution

Our first contribution is that we systematically trained a CNN to solve the gore classification task. The resulting networks can consistently identify the most egregious cases of gore in images. Our second contribution is a systematically trained CNN solving a novel segmentation task, gore segmentation. The segmentation model produces a feature map that can be used to censor the gore in an image. Our third

contribution is the development of a gore censoring pipeline that processes images in two stages: classification followed by segmentation. Our results show that the pipeline achieves a major performance improvement compared to the standalone segmentation model. This pipeline is, to our knowledge, a novel approach to this kind of problem.

### 1.4 Thesis structure

Chapter 2 provides a summary of papers in relevant domains such as gore segmentation. This includes Pornographic content classification and censoring, Harmful and Violent content detection, and medical Wound segmentation solutions. Moreover, since all the existing solutions, in addition to the proposed one, use ML, Neural Networks, and CNNs for image classification and segmentation, this chapter also discusses relevant background topics.

Chapter 3 covers the general methodology we used to train and test the model presented in this thesis. We will also go over the reasoning behind our choice for the experiments described in Chapter 5 and 6 as well as the order in which we performed them. This chapter will also present the reasoning that led us to create the image processing pipeline and how it related to the rest of the experiment in this thesis.

Chapter 4 describes the process we used to create a standardized gore segmentation dataset. We developed a specific methodology to annotate the exact location and shape of the gore in the image. As such, this annotation process can easily be generalized for other classification or Detection problems. This chapter also discusses the difficulties that were encountered when searching, creating, and sharing such a dataset.

Chapter 5 presents the proposed gore classification. Image classification is a task that consists of identifying if an image contains content of a given class or not. This task is easier to solve than segmentation or detection as it only has one output, the probability that the image containing gore, instead of a probability that each pixel or bounding box contain gore in segmentation and detection respectively. Using this as a first step of processing will allow us to only run more resource-intensive tasks, such as censoring, only if an image is likely to contain gore. As with all binary classification tasks, one must carefully consider the balance between precision and recall. If the recall is too high when compared to the precision, many images that do not contain gore will be misclassified as having it (False Positive). Conversely, a system that is too precise means images containing gore will be misclassified (False Negative). Since we want to reduce exposure to gore, we want to maximize the recall value while still having a relatively high precision value. This subject will be covered in detail in this chapter.

Chapter 6 covers gore segmentation. The output of this task is an image where each pixel captures the probability that the corresponding pixel in the input image contains gore. This information can then be used to censor the original image. Since this is a novel application of this approach, a wide variety of solutions were tested to determine if state of the art results in other applications transfer to this application as well. As it was difficult to annotate a large amount of data for this problem, Transfer Learning is used to train the networks. This consists of training models that were pretrained for another segmentation task. This

significantly reduces the probability of overfitting the training data while leveraging the low-level features learned from the original problem. Since our classification and segmentation training was done in parallel, the model does not share features.

Chapter 7 explains how the trained models were deployed. This chapter explores how both models can be combined into an Image Processing Pipeline. The performance of this Pipeline is also be evaluated on both tasks.

Chapter 8 summarizes the thesis, describes its limitations, and suggests future work on this topic.

# Chapter 2. Background and Related Work

This chapter summarizes the literature and discusses results from the domains of pornographic, violence and harmful content detection and censoring. It also covers literature from ML, Image classification, and Image segmentation as they are the basis of the problem and solution presented in this thesis.

### 2.1.1 Deep Learning

According to LeCun et al. [29], Deep Learning (DL) is a subset representative learning where the methods stack multiple levels of representation combined with non-linear functions. The use of non-linear functions allows a wider set of representations to be explored. This is useful when applications require the representations to be invariant to some changes in the input. These representation layers are usually a weighted sum of the input layer or a subset of the input. As layers are added, the level of abstraction that can be learned becomes higher. LeCun et al. [29] showed that for images the initial levels could only detect edges. As levels are added, the detected features will become motifs and can in the end become objects like eyes, mouths, wheels and so on. Importantly, these representation layers are determined using a learning procedure. This eliminates the need for feature engineering, a laborious and complex task [29].

One of the most popular learning procedures is Stochastic Gradient Descent (SGD) [29]. It consists of selecting a random set of input vectors and averaging the error between the expected output and the model's output for each vector [29]. It is then possible to backpropagate the error to each of the learned weights by calculating partial derivatives [29, 30, 31, 32]. In other words, it is feasible to calculate the proportion of the error caused by each individual weight. And since each representation layer is a sum, it is differentiable. One of the breakthroughs of backpropagation comes from using a differentiable non-linear activation layer between each representation layer [29, 30, 31, 32]. The weight vector $w$ can then be updated for the next epochs using Equation (2.1). It is common practice to update $w$ by subtracting the error gradient multiplied by a learning rate $\eta$ [29, 30, 31, 32].

$$w_t = w_{t-1} - \eta \frac{\partial E}{\partial w}, where\ 0 < \eta < 1 \tag{2.1}$$

SGD can be optimized using various methods. Weight decay optimization aims to regularize the weights, so they do not become too large. It is usually done by adding the L2 norm of the weights multiplied by a parameter to the weight update operation [33]. Momentum based optimization aims to reduce the oscillating nature of SGD [34]. It does so by keeping track of the previous gradient in a momentum variable which uses the scale of previous updates to adjust the current one [34]. There is a weight decay variable which gives more importance to recent updates in the momentum calculation [34]. The Adam optimizer has been recently proposed by Kingma and Ba [35] and has gained wide adoption. It aims to combine the strengths of AdaGrad and RMSProp. It uses the first and second moment variables each with their own decay rates. The learning rate is renamed as the step size $\alpha$ and is only used in the weight update operation. The learning rate for each of the moment updates is linked to their weight decay rates instead.

The choice of non-linear activation function is non-trivial [29]. The logistic or sigmoid function is easy to interpret as a probability, as it maps a real number into a number in the [0,1] range [29]. A variant of this function is the softmax activation which is useful when there are multiple outputs or classes [36]. However, stacking multiple layers with this activation can lead to the vanishing gradient problem as the logit derivative for extreme values is exceedingly small. According to LeCun et al. [29] and, Nair and Hilton [37], the Rectified Linear Unit (ReLU) function returns the positive part of the argument. Its formula is $ReLU(z) = \max(0, z)$ [29] [37]. It as been shown by Glorot et al. [38] that deep networks using a rectifier function learn better when compared to the logit or other similar activations. As such, it is common practice in classification to only use a logit or softmax activation for the final layer so that the output represents a probability while all other activations use ReLU or a similar function [39] [36].

### 2.1.2 Convolutional Neural Networks

In Fully Connected Neural Networks (FCNN), weights are directly tied to all positions in the input vector or matrix of each layer. They are all processed as their own independent dimension. Thus, these networks are unable to detect patterns independently from their positions such as eyes in images or a word in audio file. One possible way to extract high-level features to feed into an FCNN will be to use filters to extract the wanted information. Convolution is then performed on the input with the filter to perform the extraction. Just like it is possible for an FCNN to learn new representation spaces on its own, CNNs learn the filters automatically using backpropagation [31]. CNN has rapidly become a widely adopted DL method for image-based tasks after Krizhevsky et al. [40], C. Szegedy et al. [41] and other models achieved state-of-the-art results.

#### *2.1.2.1 Standard CNNs architectures*

This subsection explores several CNN architectures discussed in the literature. It covers their contributions to solving various research problems, especially in Computer Vision.

##### *2.1.2.1.1 VGG*

Simonyan and Zisserman [42] increased the depth of CNN architectures by only using 3x3 and 1x1 convolutions as well as 2x2 max pooling layers. Using this method, the authors were able to achieve state-of-the-art results on the ILSVRC classification task with a previously unattainable network depths of 16 and 19 weight layers [42]. These networks are abbreviated as VGG16 and VGG19 and are included in the Keras [43] and PyTorch [44] libraries.

Interestingly, the two best performing architectures proposed in Simonyan and Zisserman [42] only used 3x3 convolutional layers. Since all VGG models use the same fully connected head architecture with 3 layers (2 with 4096 nodes and 1 with 1000), this means that the VGG16 models have 13 and VGG19 models have 16 convolutional layers [42]. These layers are combined in 5 separate blocks that end with a max pooling layer. The layers in each block have the same number of filters/channels. The first block has two layers with

64 filters while the second also has two layers but with 128 filters. For VGG16, the third block has 3 layers with 256 filters. The fourth and fifth blocks have 3 layers with 512 filters each. VGG19 has an extra convolutional layer in the third, fourth and fifth blocks.

*2.1.2.1.2 ResNet*

He et al. [45] proposed the residual block and the ResNet Architecture to allow the training of deeper CNNs. He et al. [45] showed the usefulness of this architecture in Image classification and Detection. Since then, this architecture has seen success in Image Denoising [46], Visual Tracking [47] and segmentation [48] [49] in various domains. He et al. [45] mention that the problem of vanishing or exploding gradients that plagued numerous networks was mostly solved using normalized layer initialization and intermediate normalization layers such as Batch Normalization. However, He et al. [45] state that there still seems to be a degradation problem linked to adding too many layers to a model. This degradation leads to a higher training and test error for deeper models.

To address this degradation, He et al. [45] introduced the residual learning blocks for Computer Vision which can be stacked to form a network. Each block consists of two paths. The first path consists of two convolutional layers with a ReLU activation between the two for ResNet-34. ResNet-50/101/152 instead consist of 3 convolutional layers per block where the third is a bottleneck layer which uses a 1x1 kernel to increase the number of channels. The second path is a shortcut connection. If the first path output size is different than the input, the shortcut connection projects the input to that new size. Otherwise, no change is made to the input. The outputs of both paths are then combined using a summation and passed through a final ReLU activation layer. ResNet-50, which is the 3-layer block equivalent to ResNet-34, achieves a lower Top-1 and Top-5 error rates on the ImageNet validation set than previous solutions [45]. This illustrates the usefulness of bottleneck layers.

The residual architecture of He et al. [45] allows models to have more layers with less or equivalent number of parameters than methods that were studied at the time. This is especially useful as it allows the models to gain from having an increased depth without risking overfitting the model to the training data. This behaviour is exhibited by networks with too many learned weights that are trained for too long on the same set of data [50]. Overfitted models do not generalize well to data that is dissimilar from the training set [50].

*2.1.2.1.3 DenseNet*

Based on the success of VGG and ResNet at increasing the number of layers in CNNs, Huang et al. [51] developed a method to increase the number of layers while reducing the vanishing gradient effect. This phenomenon occurs when the gradient passes through so many layers during backpropagation that the weight of the initial layers barely change even if the error was maximal on a given sample. The solution presented in Huang et al. [51] is called the Dense Convolutional Network (DenseNet). The solution is similar to ResNet [45] as it involves combining the outputs of previous layers. However, in DenseNet [51], the output of a convolutional layer is combined to the outputs of the subsequent layer in the same dense block [51].

Moreover, the layer's outputs are concatenated [51] instead of being summed as it is done in ResNet [45]. Three of the DenseNet models of Huang et al. [51] (DenseNet-121, DenseNet-169 and DenseNet-201) are included in TensorFlow's Keras implementation [43] and PyTorch [52]. This approach achieved state-of-the-art results at the time on a multitude of classification benchmark tasks (CIFAR10, CIFAR100, ImageNet and Street View House Numbers (SVHN) datasets). Moreover, the DenseNet models which were able to achieve these results had significantly less parameters than previous state-of-the-art models including Residual Networks based on ResNet [51].

*2.1.2.1.4 MobileNetV2*

The goal of the MobileNetV2 model architecture [53] is to reduce the computational cost of running models. This is especially useful when running models on mobile devices such as cellphones and tablets with a limited computational capacity. Sandler et al. [53] reduced the number of operations and the complexity of the model by using depthwise separable convolutions, linear bottlenecks and what they call inverted residual blocks. Sandler et al. [53] then showed that this architecture performs well on known datasets while also being highly memory and computationally efficient.

*2.1.2.1.5 ResNext*

The ResNext architecture [54] is based on ResNet [45]. Instead of blocks having only two paths (residual and identity paths) like He et al. [45], ResNext blocks each have a cardinality parameter $c$ which specifies the number of independent residual paths. The reasoning behind this approach is that multipaths networks like Inception models [41] showed very good results for Image-based tasks. Xie et al. [54] showed that their approach performed better than ResNet [45] and Inception [41] networks of the same depth.

*2.1.2.1.6 NasNet*

The previously mentioned CNN architectures were designed manually and showed great results on a multitude of tasks. NasNet [55] builds on the idea of repeatable network structures such as Dense and Residual blocks [45, 51], but adds the idea of learning the best structure for a given task. This learning is done using the reinforcement learning framework called Neural Architecture Search (NAS) [56] which is used to optimize Neural Network architecture configurations. The NAS controller creates two repeated structured called cells to create the CNN architecture. One is called a "Normal cell" and does not change the size of its output, while the second is called "Reduction cell" and starts with a chosen operation with a stride of 2 to reduce the output dimensions by half [55]. The process of making the cell structure is recursive where the controller selects two of the previous hidden states, an operation to be performed for each state, and a means to combine the two hidden states to create a new one [55]. The possible operations include the identity layer, convolutional layer, dilated convolutional layer, depthwise-separable convolutional layer, and average and max pooling layers of varying sizes. Summation or concatenation are the two ways of combining the hidden states.

The search for the best cell structure is done by training the convolutional child network with the given structure on the given task and updating the NAS controller with the validation accuracy. This means that this process does not scale well on large datasets. Zoph et al. [55] recognized that and proposed to learn the cells' structure on a smaller similar task. The structure is then applied to the larger dataset by changing the number of normal cell repeats and the number of filters in the convolutional layer placed before the first reduction cell. Zoph et al. [55] showed that this process works by first designing the cells using the CIFAR-10 Image classification tasks, and then using these to create a larger network for the ImageNet tasks. The authors generated a family of networks (NASNet-A) with different hyperparameters to use various image input sizes for comparison [55]. Then, they showed that networks in this family were able to achieve similar or better Top 1 and Top 5 Accuracy than state-of-the-art networks using the same or similar input sizes. Moreover, the NASNet networks were able to achieve these results with much less hyperparameters. They also showed that their architecture could achieve better performance than other published architectures like MobileNet in a 'constrained computational setting'. The latter network is available in TensorFlow as NasNetMobile [43].

#### *2.1.2.2 Learned improvements to the CNN Architecture*

The above CNN improvements were performed by making changes within the architecture of the networks themselves through the creation of convolutional blocks or bottlenecks. The improvements described in this section are performed by adding to or scaling the pre-existing model's architecture.

##### *2.1.2.2.1 Squeeze-and-Excitation Networks and Blocks*

The goal of Squeeze-and-Excitation (SE) networks of Hu et al. [57] is to allow CNNs to learn more complex or abstract features than is normally possible with the traditional convolutional architecture. It does so by performing what Hu et al. [57] calls 'feature recalibration'. It is a way to combine the different channels/features from a convolutional layer in a non-linear way. This 'recalibration' of the channels is done in the SE blocks that are added after the convolutional layer or residual block, depending on the architecture. This means that any CNN can be converted to an SE network. Hu et al. [57] experimentally showed that the SE version of CNNs perform significantly better than the original network with only a slight increase in the number of operations needed to compute the network output.

##### *2.1.2.2.2 Compound Model Scaling (EfficientNet)*

CNNs can be scaled up or down. Scaling up the network usually leads to a gain of accuracy but requires more resources while scaling down does the opposite [58]. Scaling is done by changing 3 aspects of a given CNN: its resolution, width, and depth. The resolution is the input size of the image. The width refers to the number of filters in each convolutional layer. The depth is the number of such layers in the network. Scaling it means adding or removing convolutional layers from the model until the network contains $d$ times the initial number of layers. The width is the number of channels in each convolutional layer. Scaling it means adding channels in each convolutional layer. Increasing it is done by increasing the network's input size.

To show that their scaling method works, Tan and Le [58] created a mobile-sized network called EfficientNet [59], which they scaled incrementally and tested on ImageNet. The initial network, called 'EfficientNet-B0', performs slightly better than ResNet50 and DenseNet168, but with a fraction of the parameters or Floating-Point Operations Per Second (FLOPS) [58]. Then, they incrementally scale up the network obtaining seven different networks ('EfficientNet-B1' to 'EfficientNet-B7'). The EfficientNet-B7 network obtained ImageNet results close to one of the state-of-the-art networks at that time (GPipe [60]), while reducing the number of parameters by approximately 8.4 times and reducing the number of FLOPS 16 times. This is a significant improvement in terms of improving model efficiency.

#### *2.1.2.3 Uncertainty Estimation*

Neural Networks usually provide their results without any measure of certainty. In the last few years, efforts have been made to estimate the uncertainty of DL models. Gal and Ghahramani [11] proposed that adding Dropout or multiplicative Gaussian noise between each layer is "equivalent to a deep Gaussian process". As such, Gal and Ghahramani [11] added that this method can be used to estimate the model's uncertainty which is Gaussian in nature. To obtain the estimate of the uncertainty for a given image in production, Monte Carlo estimation was used by performing the forward pass multiple times with the dropout layers active with known probabilities (set during training).

Mi et al. [61] proposed methods to perform training-free Uncertainty Estimation. They compared these methods to the Monte-Carlo dropout estimation (MC-dropout). The first proposed method, called 'Infer-Transformation', consisted of creating a certain number of transformed versions of the image and performing a forward pass for each [61]. Using these results as well as the output of the untransformed image, the uncertainty can be computed. This method is referred to as a 'black box' method as it can be used even if only the model results are known. The other two proposed approaches are called 'gray box' methods as they require access to intermediate layers in the model [61]. Both of these methods introduced a random element at a given layer to create a variation in the model output. This variation can then be used to estimate uncertainty after a few forward passes. The layer at which the random element is introduced is found through experimentation [61]. The random elements introduced are either dropout (for the 'Infer-dropout' method) or Gaussian noise (for the 'Infer-noise' method) [61]. Mi et al. [61] performed experiments on two Image Regression tasks, Single Image Super Resolution and Monocular Depth Estimation, to show the effectiveness of their proposed methods. The experiments showed encouraging results in which these methods obtained results comparable to MC-dropout. However, reviewers for this paper raised concerns with the experiments as they pointed that it is not a 'systematic study of training free estimation' [62].

In an Appendix, Mi et al. [61] proposed that entropy can be used to estimate uncertainty for 'classification' tasks. They defined the latter as any task for which the output is a distribution or a probability. As such, both classification and segmentation tasks could use this method. However, reviewers [62] were skeptical of this conclusion stating that more experimentation was needed to prove that entropy is equivalent

to MC-dropout. Mi et al. [61] further explained that entropy can overfit and be mis-calibrated but is very easy to use which makes it a good choice for training-free classification Uncertainty Estimation [62].

### 2.1.3 Image Classification

The task of classification consists of assigning a given input in a category or set of categories. The subclass of classification tasks where the input given is an image is called Image classification. An example of such tasks is the recognition of handwritten digits [31] or zip codes [30]. These two problems were solved in the 1980s as they were relatively simple using black and white images. The methodology used in LeCun, Y. et al. [30, 31], including convolution and backpropagation, are still used today. Large and difficult classification datasets like ImageNet [63] became benchmark set used to compare state-of-the art advances in Computer Vision. ImageNet contains almost 15 million images with objects belonging to 21,841 categories [63]. ImageNet led to the creation and recognition of influential CNNs like AlexNet [40], VGG [64], GoogLeNet [41] and ResNet [45]. The use of networks pre-trained on ImageNet is still a common practice as it is included in DL frameworks like TensorFlow [65], Keras [66] and PyTorch [67].

#### *2.1.3.1 Evaluation Metrics*

This section presents evaluation metrics that are widely used to evaluate an Image classification models. These include the Mean Square Error, precision, recall and F-Score metrics.

##### *2.1.3.1.1 Mean Square Error*

In Statistical Regression, Mean Square Error (MSE) is used as an estimator and predictor [68, 69]. It is used in Neural Networks as either an error metric or a loss function [70]. As the name suggests, MSE consists of the mean of the squared difference between the expected result and the actual result (model's prediction). Because the errors are squared, an MSE value of 0 means the predictor perfectly predicts the data. This also means that a smaller MSE value is better. MSE can be decomposed as the sum of the model variance and bias [71, 72]. The model bias refers to the error coming from the choice of the model family. Thus, a model with a low MSE reflects a good fit of the model (low variance), but also a good choice of model distribution $Q$ to predict the unknown distribution $P$ (low bias).

##### *2.1.3.1.2 Precision and Recall*

In a Binary classification setting, there are four possibilities for the predicted label when compared to the expected results [73, 74, 75]. Two of them are correct predictions. If the predicted and expected value are both 1/True, the prediction is referred as a True Positive (TP). If both are 0/False, the prediction is referred as a True Negative (TN). Two of them are incorrect where the model predicts the opposite of the expected value. If the expected label is 1/True and the predicted label is 0/False, this is a False Negative (FN). While the other case is a False Positive (FP). There is usually a trade-off between minimizing FN and FP [73, 74]. As such, it is necessary to have measures or metrics to see the influence of both of these error types on the predictions.

Precision (Equation (5.3)) is the proportion of correctly predicted positive values (TP) over the total number of positive predictions [73, 74, 75]. Recall (Equation (5.3)) is the proportion of correctly predicted positive values (TP) over the total number of positive expected examples [73, 74, 75]. These two metrics show how both type of errors affect model predictions. This contrasts with accuracy (Equation (2.4)) which is simply the proportion of correctly classified examples over the total number of samples. It is possible to have a model that has a high accuracy but a relatively low recall or precision [74]. Depending on the application in which the model is used, this might mean that the model is inadequate.

$$Precision = \frac{TP}{TP + FP} \tag{2.2}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.3}$$

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{2.4}$$

While it is possible to obtain a model with perfect precision and recall, this will mean the model will have been trained on all the possible input data or a subset that represents all possible data. Models are usually trained on a small subset of data that can be noisy. As such, there is usually a trade-off to be made between the precision and the recall [73]. This trade-off can be visualized using a precision-recall curve [73]. To obtain this curve, it is necessary to obtain all the model's predictions on a subset of data. Usually, data that has not been seen by the model previously is used (either a validation or testing set). A multitude of thresholds are then used to determine when a prediction is designated as a positive value. The precision and recall can then be calculated for each of these thresholds which are plotted to obtain the precision-recall curve. The most common shape of this curve is what Buckland and Gey [73] refer to as the "Ordinary Parabolic Recall" which has one optimal balance point between both metrics that leads to maximal accuracy. As previously stated, this optimal point might not be the best for all applications. The curve stills allow to visualize the optimal trade-off of precision and recall for any given application.

Sometimes it can be useful to see how the model performs when trying to predict the negative class. Precision and recall can be used to evaluate this as well. In this context, TP are the TN of the positive class and vice-versa. The FP and FN are also switched. The precision and recall formula for the negative class can be written as presented in Equations (2.5) and (5.3).

$$Negative\ Precision = \frac{TN}{TN + FN} \tag{2.5}$$

$$Negative\ Recall = \frac{TN}{TN + FP} \tag{2.6}$$

#### *2.1.3.1.3 F-Score and Double F-Score*

The F1-Score is a metric defined as the harmonic mean of the precision and recall [76, 77]. The F-Score or F$\beta$-Score is a generalization of F1 which adds a real positive parameter $\beta$ that balances the weight of the precision and recall within the metric [78]. However, F1 and its generalization are usually given in two different forms which can be seen in Equations (2.7) and (2.9). When calculating the metrics using a set of samples, Equation (5.3) can be referred to as the Micro F-Score and Equation (2.7) as the Macro F-Score [79]. Appendix A – Proof of equivalence of the F-Score contains a proof different than the one provided in [77] of the equivalence of the harmonic mean form (Equation (5.3)) and the F-Score (Equation (2.7)). It also shows the equivalence of a simplified form of the F-Score (Equation (2.8)) which is reparametrized in terms of TP, TN, FP, and FN [80]. From the latter equation, if $\beta > 1$, more weight is given to the FNs which increases the effect of the recall on the overall score. On the other hand, a value of $\beta$ between 0 and 1 increases the effect of the precision on the score [77].

$$F_\beta := \frac{(1+\beta^2) * precision * recall}{\beta^2 * precision + recall} \tag{2.7}$$

$$= \frac{(1+\beta^2) * tp}{(1+\beta^2) * tp + \beta^2 * fn + fp} \tag{2.8}$$

$$= \frac{1+\beta^2}{\beta^2 * recall^{-1} + precision^{-1}} \tag{2.9}$$

In multi-class settings, the F-Score can be computed for each class [81]. It is then possible to average these to obtain a single F-Score describing the performance of the overall model on all classes. In a binary setting, there are two classes: the positive and the negative class. In this context, we refer to the average F-Score of both classes as the Double F-Score. It is possible to weigh the F-Score of each class differently in a weighted average F-Score, or weighted F1-Score [81]. Hence, Double F1-Score is a weighted binary F1-Score with the weight set to 0.5 for both classes.

### *2.1.3.2 Loss functions*

This section presents the loss functions that will be used later in this thesis for Image classification. These include Binary Cross-entropy and Macro Soft F1-Loss functions.

#### *2.1.3.2.1 Binary Cross-entropy Loss*

The notion of entropy and Cross-entropy (CE) comes from Information Theory [82]. The former is a measure of the number of bits required to transmit a random event in a distribution [82] while the latter is a measure of the average number of bits from a source distribution $P$ when using another model (or distribution) $Q$ to represent it [82]. Equation (2.10) represents entropy, while Equation (5.3) represents Cross-entropy. Gibbs' inequality [83] means that Cross-entropy will always be larger or equal to entropy. Cross-entropy can therefore be used to learn the unknown distribution $P$ using a model distribution $Q$. Shore and

Johnson [84] and Csiszar [85] proved that there is a unique minimal solution to CE minimization and that this minimization has inherent properties that makes it very useful.

$$H(X) \coloneqq -\sum_{x \in X} P(x) \log_2\big(P(x)\big) \tag{2.10}$$

$$D(P, Q) \coloneqq \sum_{x \in X} P(x) \log_2 \left(\frac{P(x)}{Q(x)}\right) \tag{2.11}$$

In the context of Deep Neural Networks (DNN), the true distribution $P$ is unknown. However, we do know the points of the distribution $P(x)$ for the inputs $x$ of the training set. As such, it is possible to calculate the Cross-entropy at those points and use it as a loss for the network. This is called Cross-entropy Loss. When used in a context where there are only two classes to predict, the loss formula can be simplified to provide the Binary Cross-entropy (BCE) Loss function $L_{BCE} = \frac{1}{N}\sum_{n=1}^{N} H(p_i, q_i) = -\frac{1}{N}\sum_{n=1}^{N}(p_i \log_2(q_i) + (1 - p_i)\log_2(1 - q_i))$ where $q_i$ is the model prediction for input $x_i$ [86]. Zhang and Sabuncu [87] showed that Categorical CE Loss (CCE) which includes BCE is very successful at training DNN with noisy data for classification. Zhang and Sabuncu [87] also showed it performed better than Mean Absolute Error Loss in that context. Kline and Berardi [88] showed that CCE gives significantly better results than Mean Squared Error for classification. As such, BCE and CCE have become the standard loss function for classification tasks.

#### *2.1.3.2.2 Macro Soft F-Loss*

The F-Score could be an interesting function to use as a loss function as it will allow the model to naturally converge to a balance of precision and recall. The F-Score function is usually parametrized in terms of precision and recall which makes it not differentiable. The elementary elements of the latter (TP, FP, FN, TN) are also not differentiable as their value can only be 0 or 1. This is because the predicted and expected label values can only have these two values. To obtain the predicted label value, it needs to be thresholded [89, 90, 91]. If the model's output value is higher or equal to the threshold, the label value is set to 1. Otherwise, the predicted label is 0. We can obtain a smooth version of the predicted label by not applying the threshold [89]. We can then change the formulas of TP, FP, FN and TN to "continuous sum of likelihood values by using probabilities" [89]. The resulting formulas for these are the following continuous functions where $x$ is the continuous output from the network and $q$ is the expected label, $TP_{soft} = xq,\ FP_{soft} = x(1 - q),\ FN_{soft} = (1 - x)q,\ FN_{soft} = (1 - x)(1 - q)$ [89, 90, 91, 92]. We can then use these in Form 3 of the F-Score to obtain the Macro Soft F-Loss (Equation (5.3)). Maiza [89] also presents a Double Macro F-Loss function obtained in a similar manner as presented in Equation (5.3). This loss function is used so that the model can also be successful in predicting the negative label 0. This ensures that the model does not learn to always predict the positive label [89].

$$Macro\ Soft\ F_\beta(x,q) = \frac{(1+\beta^2)*tp}{(1+\beta^2)*tp+\beta^2*fn+fp} = \frac{(1+\beta^2)*x*q}{(1+\beta^2)*x*q+\beta^2*(1-x)*q+x*(1-\ q)} \tag{2.12}$$

$$\begin{aligned} Double\ Macro\ Soft\ F_\beta(x,q) &= 0.5*Macro\ Soft\ F_\beta(x,q) \\ &+ 0.5*Macro\ Soft\ F_\beta\big((1-\ x),(1-\ q)\big) \\ &= 0.5*\frac{(1+\beta^2)*x*q}{(1+\beta^2)*x*q+\beta^2*(1-x)*q+x*(1-\ q)} + 0.5 \\ &*\frac{(1+\beta^2)*(1-\ x)*(1-\ q)}{(1+\beta^2)*(1-\ x)*(1-\ q)+\beta^2*x*(1-\ q)+(1-x)*q} \end{aligned} \tag{2.13}$$

Promising results in multi-label classification and other applications were also shown [89, 90, 91]. However, it is usually not compared to other loss functions such as BCE. Dembczynski et al. [92] shows that F-Measure optimization performed better than Hamming Loss in a multi-label classification, in a regression setting. This means it is unclear how good it is compared to other loss functions when used in DL models.

#### *2.1.3.3 Multiple Instance Learning*

Sun et al. [93] used a different method for augmenting data for image classification annotation. They used the labels for a bag of augmented instances of the original image instead of reusing the label for each instance. This process, called Multiple Instance Learning (MIL), allows the use of augmentations such as rotation and image cropping which can result in objects being removed from the image. However, if the bag contains all of the original images, then the bag annotations are still consistent even if certain regions do not contain the desired objects. The authors showed that this approach could be used for Object Detection and obtained state-of-the-art results on the CIFAR10, CIFAR100 and ILSVRC2015 classification datasets. A similar approach was used in Jin et al. [94] where the instances in the bag are weighted in the final predictions.

### 2.1.4 Image Segmentation

Image segmentation is a task that aims to find homogeneous regions in an image [95]. By adding depth, the regions can be combined to represent complex 'objects' such as pedestrians or cars. Another way to view segmentation is to assign each pixel of the image to a class [95, 96]. That class can represent an object or the background depending on the task.

While the methods presented in Skarbek and Koschan [95] are based on set filters and algorithms, advances in DNNs in the last 20 years has made it possible to perform more complex tasks such as segmenting multiple objects in real time [97] and organ segmentation [96]. This section covers the evaluation metrics, loss functions and model architectures that have been used on this task.

#### *2.1.4.1 Evaluation Metrics*

MSE can be used as a metric for Image segmentation. In Binary segmentation or on a class-by-class basis, precision, recall and F-Score can be used as well [98]. Intersection over Union (IoU) tends to be the metric of choice for Semantic segmentation of objects [98, 99, 100] while the Dice coefficient is more popular in medical image analysis [98]. Some papers use both metrics for comparison [101].

##### *2.1.4.1.1 Intersection over Union Score*

IoU, also known as the Jaccard index and the Jaccard similarity coefficient, measures how much two sets overlap [102, 98]. It is widely used as a metric for Object Detection [103], but it can also be used for Image segmentation [104, 99]. In both of these applications, the prediction and the expected value can be viewed as a set of pixels. IoU can be calculated using the standard formulation $IoU(P,Q) = \frac{|P \cap Q|}{|P \cup Q|} = \frac{|P \cap Q|}{|P| + |Q| - |P \cap Q|}$ where $P$ is the prediction and $Q$ is the expected value [102]. For segmentation, the intersection can be approximated by the element-wise multiplication $|P \cap Q| \cong |P \odot Q|$ [104, 99] if the values of $P$ and $Q$ are contained between 0 and 1. As the value of IoU increases, the amount of pixels in the intersection ($P \cap Q$) increases or the amount of pixels in the complement of $P$ and $Q$ ($P \backslash Q$) decreases which means the predictions are getting closer to the expected values. This highlights why IoU is a useful metric for Object Detection and Image segmentation [103, 104, 99]. Pietz [105] notes that IoU can also be written as a function of TP, FN, and FP. These are obtained by comparing the segmentation model's results pixel by pixel to the expected results for a given class. As a result, we obtain the following formula, $IoU = \frac{TP}{TP+FP+FN}$ [105].

##### *2.1.4.1.2 F1-Score (Dice Coefficient)*

Interestingly, the F1-Score can also be used as a semantic segmentation evaluation metric [106, 98]. In this context, it is a Boolean formulation of the Dice coefficient [98]. Willem [107] explains that this metric and IoU are always positively correlated because of their formulation using TP, FP and FN. This paper [107] also states that these two metrics might sometimes disagree on which two classifiers behave the best, due to their slightly different formulations.

#### *2.1.4.2 Loss Functions*

Similar to the evaluation metrics, loss functions used in Image classification can also be applied to Image segmentation. However, there is a notion of locality that is relevant for segmentation but not classification. Therefore, loss functions can use this in their calculation. This leads to new loss functions for Image segmentation. This section presents Dice Loss, Focal Loss and IoU Loss.

##### *2.1.4.2.1 Dice Loss*

The Dice Score Coefficient is an overlap measure used to compare predictions to the ground truth [108, 109, 110]. It was initially intended for use with 3D segmentation [110]. It can be used to define the Dice Loss

functions by subtracting the coefficient from 1. This gives the following formula $L_{DL} = 1 - \frac{2|P \cap Q|}{|P| + |Q|} = 1 - \sum_{n \epsilon N} \frac{2p_n q_n + \varepsilon}{p_n + q_n + \varepsilon}$ for predictions $p_n \in P$, ground thruth $q_n \in Q$ at pixel positions $n \in N$ with positive scalar $\varepsilon$ [108, 109, 110]. Nieradzik [109] says $\varepsilon$ should be set to 1. Dice Loss can be generalized by adding a parameter $\beta$ that weights the FPs and FNs. This loss function and corresponding index are known as the Tversky loss/index [109, 111] (see Equation (5.3)).

$$L_{TL} = 1 - \sum_{n \epsilon N} \frac{2p_n q_n + \varepsilon}{2p_n q_n + \beta(1 - q_n)p_n + (1 - \beta)q_n(1 - p_n) + \varepsilon} \tag{2.14}$$

#### *2.1.4.2.2 Focal Loss*

The intuition behind Focal Loss (FL) is that class imbalance means that the networks using CE loss learn more on easy examples than on the harder, insightful ones [112, 109]. To adjust for this, FL introduces a focusing parameter $\gamma \geq 0$ to create a modulaing factor $(1 - p)^{\gamma}$ so that the network learns more from harder examples [112]. Using $\gamma = 0$, FL is equivalent to BCE [109]. Lin et al. [112] presented FL for Object Detection. They obtained state-of-the-art results using FL loss [112]. The loss can be adapted for segmentation [109] which results in Equation (5.3).

$$L_{FL} = -\sum_{n \epsilon N} (\alpha(1 - p_n)^{\gamma} q_n \log(p_n) + (1 - \alpha)p_n{}^{\gamma}(1 - q_n)\log(1 - p_n)) \tag{2.15}$$

#### *2.1.4.2.3 Intersection over Union (IoU) Loss*

The properties that make IoU a good metric could also make it a good loss function [104, 99]. As a loss function, we calculate the Jaccard distance [102] which comes with the usual properties of the distance and by extension a loss function. Beers et al. [104] and Rahman and Wang [99] both show that the approximation of IoU (Equation (5.3)) can be differentiated and as such used as a loss function for Neural Networks and CNNs. Beers et al. [104] also prove that this loss function performs better than BCE in their segmentation application. Rahman and Wang [99] prove that IoU loss is better than accuracy-based loss.

$$L_{IoU} = 1 - IoU(P, Q) = \frac{|P \cup Q| - |P \cap Q|}{|P \cup Q|} = \frac{|P| + |Q| - 2|P \odot Q|}{|P| + |Q| - |P \odot Q|} \tag{2.16}$$

### *2.1.4.3 Standard Model Architectures*

#### *2.1.4.3.1 U-Net*

Ronneberger, Fischer and Brox [113] presented a new segmentation network called U-Net, an encoder-decoder CNN network. In this network, the encoder downsamples the image resolution four times. Each decoder block upsamples the lower resolution output of the previous level using a transposed convolution. It then concatenates the upsampled features to the last encoder block output of the same level. In [113], the

network only uses convolution with no padding. This means that the upsampled and the downsampled features at a given level are of different sizes. To deal with this, Ronneberger, Fischer and Brox [113] crop the downsampled image to the right size. This means that there is a loss of information in the downsampled features. However, these features have been used to obtain the upsampled convolution. The output of the network is also smaller than the input image. The output is mirrored to obtain the same image size. This works for applications in the biomedical field such as cell segmentation [113]. In uses outside of this domain, it does not make sense to do so. Implementations of U-Net [114] usually use padding in convolutions which eliminates this problem.

Ronneberger et al. [113] also used Data Augmentation to train the network using only a small training dataset. The augmentation technique needs to consider the nature of the task. For cell segmentation of microscopical images, the augmentations included image shifts, rotations, random deformations, and gray value variations [113]. Drop-out layers are also used at the end of the encoder. These layers randomly reduce some outputs to zero [115]. This is only done during the training phase. Dropout reduces the network overreliance on some features which makes the overall network more robust and less likely to overfit [115]. U-Net [113] was able to achieve state-of-the-art performance with less training data than other methods.

*2.1.4.3.2 Linknet*

Like U-Net [113], Linknet [116] is an encoder-decoder network. One of the major differences with U-Net is the use of 'Full Convolutional' layers with kernel size of 3 by 3 (3x3) for upsampling [117]. These layers use a method called 'shift-and-stitch' instead of using interpolation while upsampling. This consists of input shifting and output interlacing as presented in Sermanet et al. [118]. Their method requires greater computational resources as multiple branches with a different shift are created using 3x3 pooling with a non-overlapping stride $s$. These branches are then convoluted individually and interlaced together. Long et al. [117] contend that the same results can be obtained by performing 'filter rarefaction' using two layers. While the upsampling in 'shift-and-stitch' comes from the interlacing which is at the end of the block [118], the upsampling layer comes first in the purely convolutional block [117].

The Linknet architecture is very similar to U-Net [113], but has been shown to achieve state-of-the-art performance in Semantic segmentation tasks such as the ImageNet Large Scale Visual Recognition Challenge, which is more difficult than Cell segmentation. Linknet [116] uses a pre-trained ResNet-18 as the encoder, but other implementations use other encoders [114, 119, 120]. Chaurasia and Culurciello [116] assert that using a smaller encoder is beneficial for an application that must generate outputs in real time. The decoder works in blocks like the encoder; each of these blocks has 3 layers, a 1x1 convolutional layer which reduces the initial number of channels by 4, a full convolutional layer with 3x3 kernel which upsamples the feature map and a final 1x1 convolution to get the given block output number of channels. Between each layer, batch normalization and ReLU activation is used. Between each decoder block, the output of the previous decoder block and one of the encoder blocks with the same size are either added or concatenated, depending on the implementation.

#### *2.1.4.3.3 FPN*

Feature Pyramid Networks (FPN) [121] were originally designed for Object Detection in images. These networks make predictions using feature maps at different scales. This is useful to detect objects that are at different scale within the image. FPNs [121] create a feature pyramid (FP) by using the encoder features and previous levels of the pyramid. As such, the feature pyramid is computed similarly to the decoders of U-Net [113] and Linknet [116] by combining features from multiple points in the encoder. The FP differs from these decoders since each level has the same number of channels. Like [113, 116], Lin et al. [121] showed that the lateral connections between the encoder and the feature pyramid have a beneficial effect on the results. Lin et al. [121] also showed that using the various levels of the pyramid to make the region proposals for Object Detection improved the results using RPN, Fast R-CNN and Faster R-CNN.

While Lin et al. [121] focused on Object Detection, they also proposed an approach to extend the architecture to perform segmentation. Kirillov et al. [122] showed that the FPN features can be used to make segmentation directly. It does so by performing convolutions on each of the pyramid levels in parallel. Smaller levels are then upsampled so they are all of the same size. These can then be concatenated and used to obtain the final segmentation feature map. This is the approach used for the FPN segmentation model in Yakubovskiy [114].

### *2.1.4.4 Mask R-CNN*

Mask R-CNN [123] is an Object Detection and instance segmentation model. This means it detects instances of objects within a given image and segments the pixel representing this object from the rest of the image. Its architecture is an extension of the Object Detection model Faster R-CNN [124]. Mask R-CNN adds a third output branch that gives a mask of the pixels representing the object within the input image. To do this, the authors of [123] needed to align the output of the model to the input with a layer called RoIAlign. Its name comes from the fact that it aligns the Region of Interests proposals (RoI) with the input image using bilinear interpolation before detection is performed.

The mask branch of Mask R-CNN outputs a mask per class which makes it a segmentation model [123]. The instance segmentation is obtained by combining the bounding boxes and classification scores of the detection branches with the results. He et al. [123] achieved better results using this decoupling of the branches instead of segmenting each instance individually.

Mask R-CNN achieved state-of-the-art instance segmentation results [123], especially on the Cityscapes dataset. It also achieved significant result improvements on the Common Objects in Context (COCO) dataset for Instance segmentation/Object Detection and Keypoint detection.

## 2.2 Related Work

### 2.2.1 Harmful Content classification and Censoring

The proliferation of social media and content sharing platforms in the last decade have put a lot of pressure on moderators to identify and remove harmful content. One possible way to address this problem is using AI. Large companies such as Facebook and Google created datasets to recognize hate speech or toxic comments placed online in either comments [125] or memes (images with overlayed text) [126]. Other datasets have been made for Pornographic Content classification [127] or Violent Scene Detection [128, 129]. For the latter task, detection means finding the frames that contain violence, as opposed to identifying the specific violent content within the frame. In that sense, it can be viewed as the multimodal classification of frames in the video with sound.

The rest of this section covers various solutions for Pornographic Content classification and Violent Scene Detection, as some of the concepts used may transfer to gore classification and segmentation tasks.

### 2.2.2 Nudity/Pornographic Content classification solutions

There has been significant effort over the years to find or censor images and videos that contain adult content. There are many useful applications for such systems such as safeguarding children from seeing adult content [20, 130] or identifying child pornography [22, 131]. Numerous solutions rely on the fact that the pornographic content depicts human bodies by detecting pixels of skin-like color [24, 25]. One approach is to use the YCrCb color space [21, 132] to identify skin colors in the pixels. Other solutions used a Bayesian based classifier [133, 134, 24] or a Support Vector Machine [135] to identify skin pixels. With the skin pixels identified, it is then possible to create a mask to extract the skin regions of the image.

Interestingly, in many solutions, DL is not used. Instead, a more traditional ML approach using feature extraction is used [21, 136]. Shih et al. [21] used two MPEG-7 visual descriptors, scalable color descriptor (SCD) and edge histogram descriptor (EHD), that can then be used for image retrieval from a labelled database. Zhang et al. [136] used the same two descriptors with the addition of "the average skin probability, ratio of skin area to the pornographic regions, complexity of the texture and intensity histogram descriptor" before creating a Bag of Words (BoW) representation of the image's Regions of Interest (RoI). Zaidan et al. [133] extracted colour and shape features before performing a Bayesian classification on the image. Zuo et al. [132] extracted similar features that were then used by a random forest model for classification. The feature extraction of Dewantono and Supriana [134] used computer vision descriptors called "Speeded Up Robust Features" (SURF), opponent-SURF and "Binary Robust Invariant Scalable Keypoints" (BRISK). The child pornography detection system presented in Sae-Bae et al. [137] used skin-based features to classify the image as explicit prior to child face detection.

Xu et al. [138] observed that DL was not used widely in pornographic material recognition. In [138], the authors created an ensemble of CNN classifiers that achieved better performance than prominent existing

methods for pornography detection. However, the paper only compares the computing time to AlexNet, another CNN. Nian et al. [139] proposed a pre-trained CNN network on ImageNet that is then fine-tuned with a fully connected network to obtain a state-of-the-art pornography detection model. Sun et al. [140] trained a multiple-instance CNN model using as input a bag of images containing the original image and augmented copies of it. This works since all of the images in the bag have the same label. Jin et al. [94] proposed a variation of MIL where examples are given weights based on their importance to improve training. A comparative study by Gangwar et al. [22] found that DL models were not only able to achieve the highest accuracy for pornographic content detection, but they were also much faster in terms of performance especially when GPUs were used. This currently makes them the best choice for pornographic and child sexual abuse images detection systems.

#### *2.2.2.1 Videos*

For pornographic content detection in videos, there are different forms of information such as frames (which is equivalent to images), audio, and the motion between frames. Thus, detection systems usually perform feature extraction on each of these individually and then combine the extracted features [141, 142]. A final model then performs the classification/detection of the harmful content in the video. The classification can be for the whole video or for a specific sequence within the video, depending on the system being implemented.

When looking at the video frames, certain systems look at every frame while others only analyse frames at a given interval, or when the scene changes drastically (keyframe) [141, 142, 22, 143]. Caetano et al. [143] used image descriptors such as Scale Invariant Feature Transform (SIFT), SURF, and a mid-level representation called BossaNova to classify keyframes in the video. Other systems use skin detection, BoW, and other image-based approaches to extract visual data from the frames for analysis [141, 142, 22, 144]. Caetano et al. [143] used an SVM to perform the classification of the video clips. Perez et al. [142] also suggested performing the classification using an SVM. The most successful proposed approach was to only combine the features later in the overall architecture [142]. Jansohn et al. [141] proposed a late fusion (post-classification) approach to combined visual and motion information for pornographic video detection. Jansohn et al. [141] differs from Perez et al. [142] because it uses a weighted average for the final voting. The classification of the features was done using SVM and decision tree classifiers.

### 2.2.3 Violent Content Classification

There is an effort to automate the classification of film, television, and other media using AI [145, 19, 129, 146, 147]. Current systems cannot scale up and therefore automated approaches cannot easily be applied to all, or even most of the content produced and uploaded to the internet every day. These systems focus on ensuring children only view content deemed appropriate for their age [148, 149]. In this context, inappropriate content includes violence, gore, nudity, drugs, and coarse language, among others. For instance, the Violent Scene Dataset (VSD) [19] has been created to automate some aspects of this classification. This

dataset's annotations [19, 147, 146] highlight segments of a movie which contain different kinds of violent content. The goal is to train a model which can extract or identify problematic sequences so the movie or video can be rated faster or even automatically.

Surveys of the solutions to the VSD MediaEval tasks [147, 146] found that multimodal solutions were the most common and performed the best. The results of the 2015 version of the VSD task [147] showed that SVMs trained with visual, audio, and DL features achieved better performance. Other solutions used different subsets of traditional visual and audio features and DL and high-level conceptual features [147]. Neural Networks, Clustering, Unsupervised learning were among the techniques used for the detection of violence. Depending on the task formulation and dataset, different paradigms achieved better results. This is because the definition of violence for a given dataset varied depending on the year [147]. For some years, the definition was more objective while for others it was more subjective. In 2015, the definition was "movies that are too violent for an 8-year-old child" [128]. Another factor of note is the variety of content the model is shown during training and testing. For instance, the 2015 version of the MediaEval task tested the generalisation of the model to domains it had not trained on which led to reduced performance compared to other years. Thus, we can say such 'in the wild' formulation is a harder task to solve.

Recent effort has been made to reduce the amount of feature extraction performed to classify movies. Just like for pornographic content classification, DL is used. Gruosso et al. [145] trained a classification CNN based on Inception v3 to classify movies in one of 3 classes 'G', 'PG-13' and 'R'. Gruosso et al. [145] found that single frame analysis had difficulty differentiating 'PG-13' and 'R' but worked well for 'G' and 'R'. Khan et al. [150] uses a MobileNet CNN to perform the classification. The CNN is only provided the frame that is deemed the most informative or salient from every shot of each scene. Their proposed method achieved great performance on multiple datasets including VSD, Violence in Movies [151] and Hockey Fight [152] which showed that the approach generalizes well. Better generalisation performance was achieved in Song et al. [153] by creating a 3D CNN. The third dimension of the convolution is time which means consecutive frames are stacked in the input. They proposed two versions of their 3D CNN, one which takes 16 consecutive frames and another which takes 32 frames. These networks were able to achieve state-of-the-art results on Hockey Fight [152], movies [151] and crowd violence datasets [152].

The DL approach presented previously only used video frames as inputs. There are also solutions which attempt to apply DL to audio features. Gu et al. [154] perform a fusion of deep visual and audio features which was then trained in a multi-task setting with a semantic correspondence task. To extract appearance and motion features, Gu et al. [154] used pseudo-3D convolutional blocks and LSTM layers. Audio feature extraction using a "VGGish" network which is a modified VGG convolutional network for audio. Their approach achieved good results on multiple datasets including VSD 2015. Gu et al. [154] also showed that using semantic correspondence leads to better results than just using the violence loss alone. Mu et al. [155] also extracted audio and video features using CNNs. They then showed that a learned fusion of these features

performed better than the audio or video features alone on VSD 2015. Mu et al. [155] also showed that a late fusion of the audio and visual features performed better on the same task.

### 2.2.4 Harmful Content Censoring

There are multiple approaches for censoring harmful content in images or videos. One simple approach is to blur or not show the image or video if it has been identified as harmful [156, 144]. To do so, a classification model is trained. de Freitas et al. [156] trained a classifier on both the video frames and audio. The image/video could then be censored. They then used a Gaussian Blur filter for the video frames and removed the audio segments corresponding to the harmful content.

Another approach to this problem is to find the harmful content within an image or frame and then censor it. For image and video frames, Object Detection models can be used to perform this task. These ML models give the coordinates of a bounding box that contains the object. The censoring can then be done by blurring, removing, or replacing the content of the box from the image. This is the method used in Mallmann et al. [157] and Joseph et al. [158]. Mallmann et al. [157] used the Faster R-CNN architecture to achieve near real-time censoring of private parts in pornographic videos. Joseph et al. [158] used the Mask R-CNN architecture which also gives a segmentation mask of the object within the bounding box. As a result, the censored image then suffers a lesser loss of information compared to using a bounding box. Segmentation models can also give the pixel locations of an object in an image so it can also be used for censoring. This approach was used in Dolgushev and Brown [159] to censor cigarettes.

Adversarial training can also be used for censoring images. More et al. [20] proposed a GAN that can be used to censor nude images by adding underwear on the exposed breasts and genitals. This network consists of a generator and a discriminator network. The discriminator tries to differentiate between real and generated images while the generator learns when its generated image is classified by the discriminator. The Image generation and segmentation are both image translation tasks since the output is also an image. For instance, in More et al. [20], the U-Net architecture is used to generate images. In this paper, there are two image domains, X for nude women and Y for women wearing bikinis or underwear. The paper has two GANs, one to take images from X to Y and another to go from Y to X. The discriminator attempts to differentiate the real images from the generated ones. While the generative network was able to censor the sensitive regions, it also affected the background and other aspects of the images presented in the paper which may not be ideal in other applications. Simoes et al. [160] iterated on More et al. [20] by adding an Attention CNN which outputs a mask that finds explicit content which was then given to the generative networks. During training, the image combined with the attention mask was provided to the first generator which censored it. This paper's only discriminator was used to train the censoring generator. The discriminator was trained using the same approach as in More et al. [20] by feeding it the generated images and real images of women in bikinis. The inverse attention mask was then computed and combined with the censored image into the second generator which attempted to reconstruct the original image. The L2 Loss was used to train the reconstructing generator. The authors of [160] performed a survey to compare their method of censoring

images [20]. The survey participants preferred the attention-based results. However, 49% of the responses said that all presented methods were not good enough.

### 2.2.5 Medical Wound Segmentation

The problems solved using computer vision in the medical field vary from the problems in other domains. For instance, wound segmentation sometimes involves differentiating the different tissues and elements of a wound, not just the wound itself from the rest of the body [26, 23]. Other problems like wound area measurement use a segmentation image as an input. Moreover, the images usually depict the wound as the primary object within the image. This simplifies the problem in some regards and allows a wider range of solutions to achieve great results. Another factor to note is that certain solutions specialize in a specific type of wound such as pressure ulcers [26, 161, 162], foot ulcers [163], skin ulcers [164] and chronic wounds [23], further simplifying the problem.

A survey of DL in medical image analysis [165] states that CNNs have been used for segmentation of 2D and 3D images in the medical field. The U-Net CNN architecture [113] has specifically been created for medical imagery but is now used for images in other domains. Patch-based training has also been used successfully for organ and substructure segmentation. For lesion segmentation, U-Net and other segmentation methods have been used. Object Detection methods have also shown success on these problems as most pixels represent non-lesion/non-wound class.

Image processing methods for segmenting wounds vary. For instance, Veredas et al. [161] used K-Means clustering of the pixels in different color spaces to segment. They refined their approach using additional algorithms such as Thresholding and Region Fusion of different tissue types. Garcia-Zapirain et al. [26] and Ortiz et al. [162] initially segmented the wounds using synthetic frequencies based on energy density to create a calibrated grayscale representation. They used toroidal geometry and applied a threshold to realize a binary segmentation. This method is calibrated for pressure ulcers and may not generalize well to other types of wounds. Instead, Fauzi et al. [23] segmented using a custom colour space based on probabilistic map of red, yellow, black, and white (RYKW) pixels. They then found the contours of the wound and segmented it using Region Growing and Optimal Thresholding. The authors mention that this method works with more complex background. Song and Sacan [166] used 4 different segmentation algorithms (K-means Clustering, Edge Detection, Thresholding, and Region Growing) to create a polygon representation of the wound before extracting features to feed in their wound identification neural network.

In recent years, CNN models have been used to perform wound segmentation. The main advantage is that these models do not require initial seed points or feature extraction like Thresholding, Region Growing or Region Fusion. Also, CNNs usually need less post-processing to obtain the segmented image when compared to the previously mentioned algorithms. However, a CNN is only as good as the training dataset [167]. In Wang et al. [163], an encoder-decoder architecture and pre-trained encoders to perform segmentation. The method proposed in Wang et al. [163] was only tested against other CNNs, such as Mask

R-CNN, U-Net, VGG16 and SegNet, and not against other segmentation algorithms of the medical field. While segmentation is not the primary subject of Chino et al. [164], the authors used a U-Net-like model to segment before they perform Pixel Density Estimation to measure wounds. Their method was able to achieve state-of-the-art results in wound area estimation for skin ulcers. Liu et al. [168] proposed a new CNN framework for the purpose of wound segmentation called WoundSeg. Most convolutions occur in the encoder which start with a 512x512 image (3 channels) and ends with a 16x16 representation. The decoder then upsamples to 32x32 and performs a convolution. The final 32x32 representation is then concatenated with the one from the decoder using a skip connection before upsampling by a factor of 16 and convolved to obtain the segmentation map. The encoder can be any pre-trained CNN model; the authors tested MobileNet and VGG16, although some layers may have had to be removed. The segmentation results of WoundSeg are then processed with morphological operations to fill holes and remove noise. A skin-detection model is also used to reduce the amount of background that is included within the wound segmentation. This outperformed the other reported methods [168]. However, the only metric that all models could be compared with was accuracy which is not ideal for segmentation. On its own, the results of this framework [168] are promising with a mean IoU of ~ 85%.

Li et al. [167] combined both CNNs and image processing segmentation to obtain a better segmentation. Their system first performs background removal by identifying skin-like pixels. They used a custom encoder-decoder architecture like WoundSeg without the skip connection. This segmentation is then refined by performing semantic and wound based corrections. This method performed better than the tested pure clustering or CNNs methods. Li et al. [167] mention that their method is much more complex and cannot perform end-to-end training.

# Chapter 3. Methodology

The classification of gore in images is a simpler problem compared to segmentation as the former merely requires the detection of gore in the image while the latter necessitates the identification of gore at the pixel level. Since classification would not permit us to censor gore while maintaining the rest of the information in an image, we elect to combine both solutions. Hence, we propose an Image Processing Pipeline composed of a cascade of classification and segmentation models. The pipeline is described in Chapter 7 while the development of the classification and segmentation DL models is detailed in Chapter 5 and 6 respectively. Yet, before we develop our DL models, we need to build a dataset. We explain our methodology for collecting the dataset in Chapter 4.

In this chapter, we will explore the methodology we employed to build and test the classification and segmentation DL models. We will relate this methodology to our related work exploration of Chapter 2. We will further discuss the rationale for the development of the Image Processing Pipeline.

## 3.1 DL Model Training and Testing

Since the gore classification and gore segmentation problems are unexplored in the literature, we did not have a benchmark model to compare to. Furthermore, we could not base our choice of evaluation metric, loss function, model architecture, and encoder on other solutions in the same domain. Our exploration of harmful and pornographic content classification and wound segmentation provides possible choices for our experiments. However, these problems have different objectives and datasets. This means that we cannot assume that the best performing models for these related tasks will be appropriate for the gore censoring task. For instance, gore is not guaranteed to only occur near pixels of skin-like colours, an assumption used for pornographic censoring and wound segmentation. Based on the literature of related applications, we solely considered DL models instead of traditional AI approaches. As we summarized in Section 2.2, recent experiments have shown that DL models consistently outperform traditional ones.

Developing the DL models requires us to make decisions regarding the evaluation metric, loss function, model architecture, and encoder. To ensure we make the right design decisions, we setup our experiments in phases. In each phase, we look at one aspect of the model or experimental setup. To isolate the aspect for which we need to make a design decision, we set all other aspects while varying the one under consideration. For instance, to select an evaluation metric for gore classification in Section 5.2.1, we use a model with MobileNetV2 CNN encoder and BCE loss function. The encoder and loss function are widely used in the literature and industry for classification tasks [169, 170, 171, 172, 173, 82]. Then, we test several evaluation metrics to assess the most appropriate one. This process allows us to understand the impact of each aspect of the models we are developing. At each experimental phase, for each candidate solution, we perform hyperparameter tuning on the validation set. We then test the best performing model on the testing set. We use the testing set results to select the best performing model that we adopt for the subsequent experimental phases.

### 3.1.1 Order of the Experimental Phases

The order in which the experimental phases are performed is non-trivial as it could affect the performance of the end model(s). We employ the following order for the experimental phases: evaluation metric, loss function, model architecture, CNN pre-trained encoders, and improvements from related literature.

To avoid overfitting the models, we use early stopping during hyperparameter tuning. We monitor a metric while evaluating on the validation set. If the model's performance according to this metric does not improve for a certain number of epochs (usually set to 300 for classification and 100 for segmentation) or begins to deteriorate, then we assume the model is overfitting and training is stopped. We then test the model that performed the best in terms of the evaluation metric. For segmentation, IoU is the most widely used metric for both testing and validation [98, 99, 174, 175]. As such, it is the only metric that we consider. For classification, there are many evaluation metrics that are used in literature such as MSE, best loss, precision, recall, F-Score, and Double F-Score. Thus, we need to choose the metric that leads to the best testing results. The metric we prioritize is the recall as it correlates with the amount of gore detected. We also monitor the testing Double F1-Score of the models to ensure that the focus on the recall does not result in a model that always classifies an image as containing gore. We describe the classification evaluation metric experiment in section 5.2.1.

In the literature, the order in which design decisions are made about the loss function or the CNN encoders/architectures changes depending on the work's context. This thesis addresses an unresearched problem. As such, we do not know which loss function, CNN encoders or even architecture lead to the best results. We resolve the design decision pertaining to the loss function first. The loss function provides the same optimization landscape for all the models we train. Thus, if a loss function provides a good landscape for one model to learn, we postulate that it may be appropriate for others as well. The loss functions we tested for both tasks are discussed in sections 2.1.3.2 and 2.1.4.2 respectively.

The ability of a CNN encoder to perform well on a given task is impacted by the choice of the overall architecture. For classification, we use the conventional approach of inputting the CNN representation to a fully connected network which in turn provides the results. For the segmentation task, there are a wide variety of architectures. Wound segmentation models typically use U-Net or U-Net like architecture [165, 113, 165, 164, 176], while censoring networks use object detection models [158, 159, 157]. However, since our segmentation task differs from these related applications, we broaden our architecture consideration to include widely used image segmentation architectures from other domains, namely Linknet and FPN which are included in Yakubovskiy's segmentation package [114].

We test a variety of pre-trained CNN Encoders using the previously chosen loss function and model architecture. We use pre-trained encoders since the dataset we build in Chapter 4 is small. Similar to the choice of architecture, we consider widely used encoders that have shown an ability to perform well on a

variety of tasks. These include VGG, ResNet, DenseNet, MobileNet, MobileNet, NasNet and EfficientNet from tf.keras.applications module [43]. Moreover, from Yakubovskiy's segmentation package [114], we consider the pre-trained encoder model family SE-ResNet (Squeeze-and-Excitation ResNet) and ResNext for the segmentation task.

Once we complete the above-discussed experimental phases, we obtain a performance benchmark on our dataset. This allows us to further investigate approaches to improve our results. Based on the classification literature, we test various ensemble and MIL approaches. The latter is due to the success of Sun et al. [93] on object detection and Jin et al. [94] on pornographic image recognition tasks. For segmentation, we test GAN approaches, based on their recent success in pornographic content censoring by Simoes et al. [160] and More et al. [20]. We also train a Mask R-CNN model as it was proposed by Dolgushev and Brown [159] for a censoring task.

### 3.2 Image Processing Pipeline

During classification training, we wish to maximize the recall of the end model to ensure the least amount of gore is passed to the end users. By considering the Double F1-Score, we also ensure the number of false positives the model produces are reduced. For segmentation, the metrics used for testing the model operate on the pixel level instead of the whole image. We do not know how much of the images censored by the segmentation models are true or false positives. Moreover, the segmentation models are only trained with our initial dataset. This dataset is smaller than the classification one due to the time-consuming annotation process related to this task. This is explained further in Chapter 4. As such, we expect that the gore segmentation models may lead to more false positives than the classification models. This could reduce the confidence of end-users in the models' performance in the real world as most images fed into the model are non-gore.

Hence, we propose an Image Processing Pipeline composed of a classification and segmentation components to solve these problems. We feed the images to the classification model. If the classifier predicts that an image contains gore, it is forwarded to the segmentation to identify the gore pixels so that they can be censored. Since we focus on maximizing testing recall for the classification model, most egregious cases of gore images should be passed to the segmentation model. Thus, this pipeline will likely reduce the number of needlessly censored images (false positives) when compared to the gore segmentation model alone while still censoring the egregious cases of gore. There are also other benefits to this image processing pipeline. The Image Processing Pipeline could also have implementation benefits as well. This will be further explored in Chapter 7.

# Chapter 4. Building a Dataset

One of the main challenges of working with gore is that there is no freely available dataset. This means one must be created, requiring extensive searching, downloading, and annotating of images for both classification and segmentation tasks. When building a dataset that contains graphic imagery, one must be mindful of their mental health during the process. In this thesis, we did so in two ways, by only looking at images where the gore occurs in a medical setting or images in which the gore is simulated. The former was less triggering for us as the injured party is getting medical care by professionals. The same applied to simulated gore, as no matter how disgusting or horrifying an image was, we always knew it was not real. The second way we reduced mental health risk was by playing a video game right after searching for or annotating gory images. This is because research has shown that doing such an activity after a possibly traumatic event significantly reduces the odds of creating intrusive memories [177, 178].

## 4.1 Annotation Methodology

In this section, the methodology adopted to annotate the images that make up this research's dataset is detailed. The dataset was annotated for gore segmentation to start. This is because annotations for simpler tasks like detection or classification can be generated from the segmentation annotation or feature map. For instance, for classification, an image is classified as gore if there is a region in the segmentation annotation that contains it. For detection, one can use bounding box algorithms to generate bounding boxes over the gore region in the feature map. This is possible as the feature map is a binary image. For each region we annotated, we gave it boolean properties explaining the region's content. The Gore and Blood properties if the region contains gore or blood. The Wound and Burn are used to tell if the region is a wound or a burn respectively.

### 4.1.1 Segmentation Annotation

For segmentation, the annotation is performed with the "VGG Image Annotator" [179]. Using the polygon region tool, the gore in images could be highlighted. This tool also allows the addition of region attribute which allowed us to define whether a region contains blood, a wound, a burn, or gore. An image is annotated as gore if we found - or expect that some people might find - the region to be uncomfortable to look at. This notion is subjective and as such we expect there are some annotated samples which could be contentious, especially since the author was the sole annotator of this dataset. However, the notion of gore is not subjective when it comes to the most egregious cases.

Annotating the images for segmentation can be quite time consuming especially if an image contains multiple small gore regions. An example of this will be the results of an explosion in a movie still or a victim of gunshot wounds. In these cases, to reduce annotation time and exposure to the graphic content, gore regions that are close together are joined into super regions. As a result, these regions contain some non-gore

content. Hence, even a model which can perfectly segment gore will not produce perfect evaluation metrics on these images. While not ideal, we felt this compromise was necessary to reduce the strain on the annotator.

When annotating images, we felt we were becoming more and more tolerant to gore which may have led to errors. This could have been mitigated if gore had a universal definition like wound. However, as previously said, different people have dissimilar levels of tolerance to such content. In our case, the target audience are analysts and adults which are moderately used to such content. However, most common annotation for gore are intended to reduce children's exposure to it. Another possible way to mitigate subjectivity will have been to use multiple annotators. Because of the nature of the content and due to monetary constraints, we felt that this was not possible for this thesis.

The annotation file can be downloaded as a JSON file which allows us to generate the expected result feature map. To obtain the gore feature map, we created a Python script which upon loading the image and annotation file generates the feature map. To do so, we create an empty grayscale image with the same size as the original image. Using the Python Image Library's (PIL) ImageDraw class [180], the gore polygonal regions can then be drawn. The points within these regions are set to 255. A 9x9 mean filter is then used to blur the edge of each region, because there is a level of uncertainty around this area. The map can then be saved as a PNG image. When loaded in Tensorflow, the map value is normalized to the [0.0, 1.0] float range instead of the [0, 255] integer range. This is standard practice in ML as it usually leads to stability and performance improvement [181].

### 4.1.2 Classification Annotation

For classification, the annotation is also performed using the "VGG Image Annotator" [179]**.** The classification labels are added as file attributes. As with region attributes, this allows the author to specify if an image contains blood, a wound, a burn, or gore. An image is annotated as gore if we found or expected that some people might find the region to be uncomfortable to look at. Annotating for classification is much faster as the region does not have to be defined using the polygon region tool.

### 4.1.3 Mask R-CNN Annotation

Because the segmentation and classification annotations were already completed, the final step was to use the Mask R-CNN model to generate bounding boxes around each gore region. This is because Mask R-CNN is based on the object detection model Faster R-CNN with an added branch for segmentation [123]. Thus, we need to provide the bounding box as well as the segmentation annotation for this model. This was done using the Shapely library in Python to generate the contour of each region. We were then able to create a COCO-like version of the segmentation dataset which can then be used to fine tune a Mask R-CNN model using Matterport's implementation in TensorFlow [182]. Note that COCO refers to the Common Objects in Context dataset for Instance segmentation/Object Detection and Keypoint detection. Its dataset format is what is used by the Matterport's implementation during use.

### 4.2 Gathering images

Images that contain real gore are ideal as training data as they are varied and provide the type of gore we want to censor. However, searching, and annotating images exposes the researchers to the same risks we want to alleviate for other people. As such, one must be careful when searching for such content. What we found less incongruous are images taken in a medical setting; but these are hard to find. Moreover, the gore in the medical setting is almost always the focus of the image which is less than ideal if we want the end model to find gore in varied contexts. As an alternative source of images, frames from movies and television shows provided an acceptable source of gore images. The frames have simulated gore that can be very realistic and can be blended in the background. Some types of gore are not simulated in movies and others are overrepresented. Therefore, we decided to create a dataset that combine images from both sources.

Since high-quality images are hard to find, almost all images used in this dataset are under copyright. While the use of these images for research falls under fair dealing [183], the publishing of these images may not. As a result, examples from the dataset will only be described in text.

#### 4.2.1 Initial Dataset (Segmentation)

The initial dataset we created was made up of 922 images from varied sources. They are split into three subsets: a training set containing 737 images, a validation set containing 74 images and testing set containing 111 images. Table 1 shows the image sources and the number of images from that source in the initial dataset. Most of the images in the dataset are from the Medetec dataset which are medical images of wounds. Images from movies and television shows are the next biggest sources. Lastly, images we searched for manually were added. These included searches for gore, gunshot wounds and stabbing wounds.

Initially the training set contained only real gore images while the validation and testing sets contained both types of images (real and simulated). However, the model had a lot of difficulty identifying gore in images where it was not the focus of the image. Simulated gore images were therefore added to the training set, improving generalisation. The new training, testing and validation set are now stratified in terms of the image content annotation (Gore, Blood, Wound and Burn) instead of its source.

*Table 1 - Image sources for the Initial dataset*

| Image source | Number of images | |
|---|---|---|
| | Initial/segmentation | classification |
| **Medetec** | **593** | **593** |
| - abdominal-wounds | 12 | 12 |
| - burns | 20 | 20 |
| - epidermolysis-bullosa | 5 | 5 |
| - extravasation-wound-images | 18 | 18 |
| - foot-ulcers | 48 | 48 |
| - haemangioma | 6 | 6 |
| - leg-ulcer-images | 134 | 134 |
| - malignant-wound-images | 9 | 9 |
| - meningitis | 25 | 25 |
| - miscellaneous | 56 | 56 |
| - orthopaedic-wounds | 49 | 49 |
| - pilonidal-sinus | 3 | 3 |
| - pressure-ulcer-images-a | 100 | 100 |
| - pressure-ulcer-images-b | 74 | 74 |
| - toes | 34 | 34 |
| **Movies** | **207** | **316** |
| - Fight Club | 0 | **8** |
| - I Spit on your grave | 6 | 6 |
| - Jigsaw | 21 | 21 |
| - Kill Bill | 0 | **24** |
| - Kill Bill 2 | 0 | **2** |
| - Last House on the Left | 3 | 3 |
| - Logan | 31 | **32** |
| - Pulp Fiction | 0 | **6** |
| - Saving Private Ryan | 0 | **68** |
| - Saw | 11 | 11 |
| - Saw 2 | 19 | 19 |
| - Saw 3 | 32 | 32 |
| - Saw 4 | 21 | 21 |
| - Saw 5 | 9 | 9 |
| - Saw 6 | 15 | 15 |
| - Saw 7 | 20 | 20 |
| - Zero Zero Zero (TV) | 19 | 19 |
| **Other sources** | **122** | **122** |
| - Wounds (Google Images) | 54 | 54 |
| - Non-wounds (Tattoo, Piercing, Scars, Smile) | 68 | 68 |
| **Total** | **922** | **1031** |

#### *4.2.1.1 Real gore*

As previously stated, real gore images are primarily taken in a medical setting. The main source of these images is the Medetec [184] dataset which shows a multitude of wounds at different stages of healing. While this dataset contains a large variety of wounds, the images primarily show the wound and thus simplify the work of the model. We also felt certain type of graphic wounds were missing such as stabbing wounds and gunshot wounds. As such, images from Google Images were also used. On the search engine, we specifically searched for images that were described as containing gore or for the type of wounds we wanted. All the collected images are subject to copyright which means that our dataset cannot be openly distributed.

While there are some wound segmentation papers which use Medetec [184] in their model, we consider the task of gore segmentation to be different enough from these studies. For instance, we cannot assume that the gore is always on a human body. Examples will be bloody teeth or organs outside of the body. Another difference is that we considered some wounds not to be gore. Wounds that were healed were not annotated as gore as we did not - or did not expect the target audience to - have an adverse reaction to viewing these depictions.

#### *4.2.1.2 Simulated gore from fiction (Movies and Television)*

Fiction movies are a gold mine for gore image collection. This is especially true for R-rated action and horror movies. However, it is still difficult to select the movies to go through to extract frames. VSD [19] provides an annotation that specifies problematic sequence categories for a set of movies. One of these categories is gore. However, VSD annotations is for content that is not suitable for children. On the other hand, the goal of this thesis is to reduce exposure of gore for adults that may be exposed to it as part of their profession. This means that all the frames taken from these sequences must be reannotated even for classification data. VSD only has a few movies that have the gore annotation. These movies are “Fight Club”, “Kill Bill (Volume 1 and 2)”, “Pulp Fiction” and “Saving Private Ryan”. It was therefore necessary to get frames from other movies to get a reasonably sized dataset. We decided to do this using movies and television shows that are recognized as being gory. As such, we made searches online to see which fictional works were recognized as being gory. The additions include “I Spit on Your Grave”, “Jigsaw”, “Last House on the Left”, “Logan”, “The Saw Collection Unrated (Saw 1 to 7)” and “Zero Zero Zero”.

When we looked at the breakdown of the images in the initial dataset, we found that because of an error in the processing of the image annotation, certain sources were excluded as it can be seen in Table 1. Thus, the initial dataset and the segmentation set did not contain images from “Fight Club”, “Kill Bill (Volume 1 and 2)”, “Pulp Fiction” and “Saving Private Ryan”. This was corrected when combining the initial dataset and the Reddit images for the larger classification dataset.

Another difficulty is that we do not want to overexpose the model to certain type of wounds like gunshot wounds or knife wounds which are very common in fiction. We also do not want to overexpose the model to repeating aspects such as certain actors and scenery, because this could mean that the model will learn

features specific to these aspects instead of learning features related to the gore itself [185]. To avoid this issue as much as possible, we only collected one or two frames per point-of-view in each scene. This meant we had to scrub the scenes to find the most meaningful frames to extract in each sequence.

#### *4.2.1.3 Non-gore images*

High-quality images that do not contain gore are as important as images that contain it. Using them, the model will be able to learn features to differentiate non-gore content like a smile or a knife from a graphic wound. These images also allow the classification model to learn that the background of gore images is not part of the gore.

Ideally, these images are framed and shot in very similar ways to the images that contain gore. For example, we found that images showing new tattoos and healed scars are very similar to images from medical settings. Accordingly, we searched for these types of images on Google Images. We also extracted frames without gore from the same movies as the ones where gore frames were extracted. This is to ensure the model is also exposed to the cinematographic style of the movie without gore in the frame. This will hopefully translate to the model being able to censor gore in movie frames.

### 4.2.2 Reddit Scraping for Classification

Initially, the same set of images was used to train the classification and segmentation models. However, testing the model using images not included in the training set showed that the classification models classified a lot of non-gore elements as gore with a high probability. This was especially true for food. Saucy food, pastries, and cooked meat where very likely to be classified as gore. On the other hand, certain types of wounds that were underrepresented in the training set were not classified as gore. After some experimentation, we determined that the easiest solution will be to get more images.

The new set of images were added using the “Not Safe For Work (NSFW) Image Scraper”. This scraper works by downloading the top image posts from Reddit communities (subreddits). The subreddits selected for gory content were “Medicalgore” and “InjuriesAndWounds”. The communities for non-gore images were “scars”, “AcneScars”, “tattoos”, “food”, “architecture”, “Mirrorsforsale”. The first non-gore subreddits were chosen since the images posted there are similar to medical imagery. The “architecture” subreddit was chosen because the model trained previously seemed to confuse windows and doors as cavities, and gore by extension. “Mirrorsforsale” was chosen because the context of taking a picture of a mirror can lead to the image only containing disembodied limbs. However, it is in a non-gory context. The intent was that adding all these subreddits will lead the model to learn real gore features instead of circumstantial ones such as the image framing.

Most of the images that were added for classification are non-gore images as it can be seen in Table 2. This is because the classification model is always given the whole image when training. As such, we felt that we needed to rebalance the number of gore and non-gore images since the initial dataset mostly contains

gore, but if the model were ever deployed, it will mostly see non-gore images. This rebalancing makes the gore a minority class. However, we estimate that the amount of gore images in this new classification set is around 33% of the total number of images. This leads us to believe that the classification models will learn to predict that certain images contain gore instead of always predicting non-gore which can be the case when the minority class is exceptionally underrepresented.

Annotating the downloaded images for segmentation like the initial set will have been too time consuming as it was much larger. Thus, these images were only annotated for classification. Similarly, the annotation of each image contains the gore, Blood, Wound and Burn Boolean annotations allowing us to use the same stratified sampling as the one used for the initial dataset. This led to a new gore classification dataset which includes 3830 training, 391 validation and 586 testing images.

*Table 2 - Image sources of the Reddit additional classification data*

| Image sources | Number of images |
|---|---|
| - no_violence_2 (food, architecture, Mirrorsforsale) | 1073 |
| - gore (InjuriesAndWounds, Medicalgore) | 1066 |
| - tattoos_scars (scars, tattoos, AcneScars) | 1639 |
| **Total image annotated** | **3778** |
| **Total image in new classification set (csv read issue)** | **3776** |

### 4.3 Summary of Chapter 4

In this chapter, we covered how we sourced and annotated the datasets used in the rest of this thesis. The images were taken from a variety of sources such as medical imagery, movies, social media and image search websites. The collection contains real gore, simulated gore, and non-gore images. The medical images mainly came from the Medetec [184] dataset. These images contained mostly real gore although some images were classified as non-gore since they showed healed or bandaged wounds. The initial set of images from movies were sourced from films whose annotation are included in VSD [19]. Movies were a useful source of simulated gore and non-gore images.

The initial dataset was annotated for the gore segmentation tasks. These annotations can be repurposed for the gore classification task. However, this annotation was tedious and time consuming. The resulting dataset only contained a total of 922 images. 737 images of which were used for training, 111 for testing and 74 for validation. We found that this small dataset was too small and led to poor gore classification results. To remedy this, we added images that contain food, architecture, tattoos, scars among other elements. This is because these kinds of images caused confusion in early gore classification models. The new gore classification dataset contained a total of 4807 images. Of these, 3830 were used for model training, 391 for model validation and 586 for model testing.

# Chapter 5. Gore Classification

This Chapter covers how the gore classification solutions were designed, trained, and tested. The training methodology will be explored first. Then the experimental results will be discussed.

The training experiments were performed in phases where each aspect of the training setup was evaluated independently (i.e., the evaluation metrics, data augmentation, loss function, CNN encoder, uncertainty estimation, ensembles learning, and MIL). Finally, the best resulting models are compared to the closest publicly available analog to gore classification: Google Cloud's SafeSearch.

## 5.1 Training Methodology

A conventional Image classification model architecture is used for this thesis (see Figure 1). A CNN, which acts as an encoder, receives a 224 by 224 image and outputs a Tensor. Global pooling average is then applied on this matrix to remove the sparsity in each channel. The average result is a vector that has as many dimensions as channels in the convolutional output. This vector is then passed to a FCNN with a single numerical output. This output provides a numerical prediction of whether the image contains gore (i.e., harmful.gore) or not.

When training the model, Data Augmentation is used on the training set to learn as much as possible from it. The training set is loaded using TensorFlow's Keras ImageDataGenerator [186] which dynamically performs the data augmentation. The generator is set to allow rotation of up to 90 degrees as well as flipping the image horizontally. Random brightness range variations are also made on the images.

During training, the threshold used to calculate binary evaluation metrics, such as precision, recall and F-Score, is 0.5. This is because this threshold is the most intuitive. It is also the value at which Shannon's entropy is maximal [3]. Thus, this threshold was preferred. However, the threshold can be changed and the formula for entropy adapted. The latter is performed by remapping the ranges of $[0, threshold)$ and $[threshold, 1.0]$ to $[0, 0.5)$ and $[0.5, 1.0]$ before calculating the entropy. After a model has been trained, the precision recall curve is plotted using Scikit-learn and the validation set as seen in Figure 2. To do this, the package iterates over the threshold value and plots the precision and recall values for each. Three key points are extracted. The first of these points is the point at which the maximal F1-Score is obtained. This is the balance point between precision and recall. As we want to reduce exposure to gore as much as possible, the other two points extracted are the points where 90% and 95% recall are reached. Ideally, the highest F1-score point is in between 90 and 95% recall, as seen in the right plot of Figure 2. This means that a high recall is obtained with high precision. However, the obtained plot usually looks like the left plot of Figure 2. In this case, the highest F1-Score is obtained before reaching 90% recall. This means that to obtain a lower number of FNs, a significant number of FPs will occur which leads to reduce model precision at this threshold value. One thing to note is that, in the results, there is usually only one threshold per model in each table. The

threshold value is the one that gives the best performance overall. This is usually the best balance of testing Recall, F1-Score and Double F1-Score.

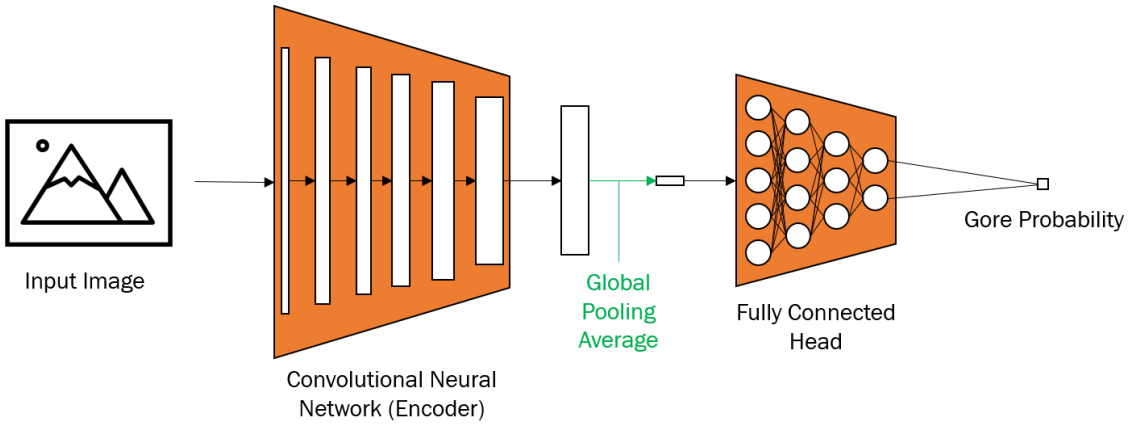


*Figure 1 - Gore Classification Model Architecture. The image is first passes through a CNN encoder then a global pooling average layer and a fully connected network (head of the network).*

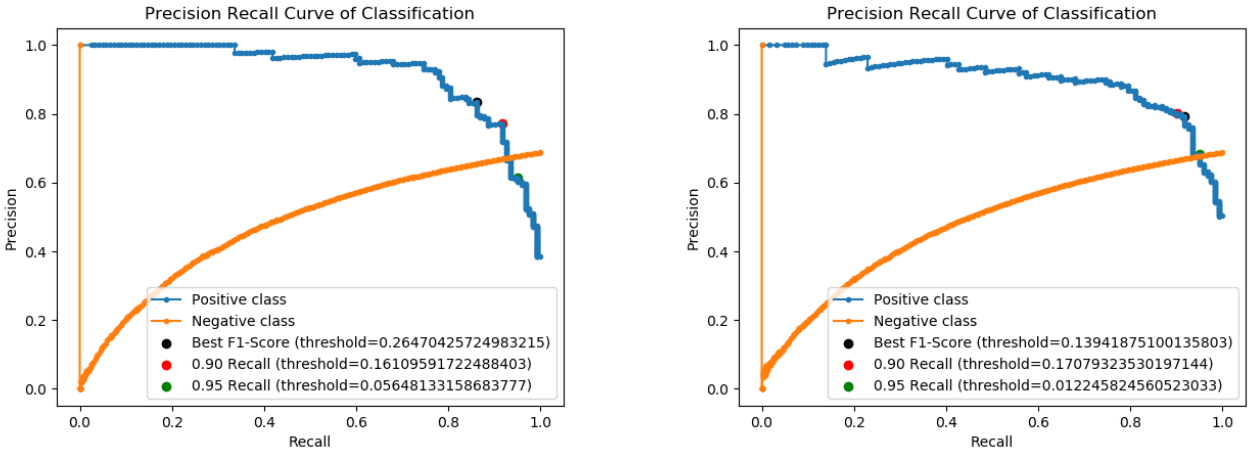


*Figure 2 - Post Training Precision Recall Curves. This curve is used for optimal thresholding after training. We use the lowest threshold that achieves either best F1-score or 90% recall for testing. This ensures the model achieve at least a validation recall of a 90%.*

### 5.2 Experimental Results

Since this application's subject has not been well researched, it is unclear what is the best way to train the model and evaluate the performance of CNN encoders. Experimentations are required to find the best experimental setup to train the model presented in Figure 1. The experiments will modify one aspect of the training setup - or model architecture - at a time to assess the effect of the change on the model's performance.

The first phase of the experiments will attempt to find the best evaluation metric to track for early stopping. In the next phase, the use of Data Augmentation will be revisited to assess its effectiveness. The third phase will analyse the effects of different classification loss functions. In the fourth phase, different pretrained CNN encoders will be tested to find the best one for this task. In the fifth phase, we will experiment

with uncertain estimation to further assess the different models' ability to generalize to new data. In this phase, we will also revisit the chosen loss function and CNN encoder from previous phases using these new metrics. In Phase 6, we will combine the best performing models of previous phases in different ensembles in an attempt to further improve the testing results. The three ensemble types that will be trained and tested in this phase are: cascading, voting, and stacking ensembles. The relatively new paradigm of MIL will be tested in the seventh phase to see if it can improve performance for gore classification. In phase 8, we will conduct experiments to assess whether calculating the model's uncertainty is worth the increased computational cost by yielding a meaningful performance gain. We will compare the best gore classification models to the closest publicly available solution, Google Cloud's SafeSearch. Finally, we will analyse the results of the best gore classification model to get insights into the features they use to make their predictions and what leads to FPs and FNs.

### 5.2.1 Phase 1 – Evaluation metric experimentation

The goal of this first phase of experimentation of gore classification is to find what metric performs the best on this task. Early stopping is used during training to reduce the odds of overfitting the models on the training set. The evaluation metric value on the validation set is monitored after each training epoch. If the best value has not improved for a given number of epochs (the patience hyperparameter [187]) the training is stopped. If the best value improves, the model is saved as the current best model. The different evaluation metrics that will be tested are the best loss, F1-score, Double F1-Score, and Weighted Double F1-Score. The latter is a generalisation of Double F-Score where the weight given to the F-Score of the negative class can be changed. If that weight is set to 0.5, we obtain regular Double F-Score. If that weight is set to 0, the F-Score is obtained. During this phase, the model settings are set separately from the change in evaluation metric. The CNN encoder used is MobileNetV2. The loss function is BCE with a learning rate of $1 * 10^{-5}$. The patience is set for 300 epochs and each model can be trained for up to 1000 epochs.

Table 3 shows the testing results of this experimentation phase. The best loss model has the threshold closest to 0.5. However, its best F1-Score is the lowest of the evaluation metrics. The situation is similar for the model that used F1-Score, although the results are slightly better. This model achieved the highest recall and negative class precision value but has the lowest number of FNs. It also has the lowest precision score (i.e., the most FPs). The model using Weighted Double F1-Score achieved similar results but at a much lower threshold value. This model struggles with FNs and probably will not generalise well as a small variation of signal has a significant effect on its predictions. The Double F1-Score model achieved the best testing F1-Score, Double F1-Score and Precision values in this phase. This performance is interesting since the precision was obtained with a high recall value (0.8956). From this, we gather that this model has learned good features to differentiate both classes which results in low amount of both FPs and FNs. This leads us to believe that this metric is the best of the ones tested and the one that we will use for the rest of the gore classification experimentation.

*Table 3 - Comparison of the testing performance of Evaluation Metrics for Gore Classification. Highlighted cells represent the best results on each metric.*

| Evaluation Metric | Threshold | F1-Score | Double F1-Score | Precision | Recall | Negative Precision | Negative Recall | Entropy |
|---|---|---|---|---|---|---|---|---|
| Best Loss | 0.2647 | 0.8110 | 0.8627 | 0.8087 | 0.8132 | 0.9156 | 0.9134 | 0.3626 |
| Best F1-Score | 0.1630 | 0.8249 | 0.8641 | 0.7319 | 0.9451 | 0.9715 | 0.8441 | 0.4531 |
| Double F1-Score | 0.1394 | 0.8512 | 0.8895 | 0.8109 | 0.8956 | 0.9506 | 0.9059 | 0.1879 |
| Weighted Average F1-Score (0.75 for positive, 0.25 for negative | 0.0695 | 0.8465 | 0.8829 | 0.7703 | 0.9396 | 0.9698 | 0.8738 | 0.2149 |

### 5.2.2 Phase 2 – Data Augmentation Ablation

In this phase, we performed an ablation study for our use of Data Augmentations during training. To do this, two identical models are trained, one with and one without Data Augmentation on the training set. Both models use a MobileNetV2 encoder trained with BCE Loss and a learning rate of $1 * 10^{-5}$. The Data Augmentation setup is explained in Section 5.1. Table 4 shows that the models trained with Data Augmentation performed better on the F1-Score and Double F1-Score metrics. Moreover, the testing loss of the model was much lower when Data Augmentation was used, which means that these models generalise better to new data. The model without augmentation did have better recall and negative precision values but had worse precision and negative recall. This model also had the best entropy value which could indicate less model uncertainty, but this value is still high considering the output is within the range $[0, 1]$. We conclude from Table 4 that Data Augmentation improves the ability of the model to generalize to new data without significantly affecting the other results. Accordingly, it will continue to be used. Moreover, from this point forward, horizontal, and vertical shifts will also be used when training the models.

*Table 4 - Effect of Data Augmentation for Gore Classification on the testing results. All models use the MobileNetV2 encoder trained with BCE Loss with early stopping based on validation Double F1-Score. Highlighted cells represent the best results on each metric with recall and Double F1-Score being the most important.*

| Model | Model Loss | Threshold | F1-Score | Double F1-Score | Precision | Recall | Negative Precision | Negative Recall | Entropy |
|---|---|---|---|---|---|---|---|---|---|
| Without Data Augmentation | 0.4024 | 0.1729 | 0.8506 | 0.8873 | 0.7887 | 0.9231 | 0.9625 | 0.8886 | 0.1469 |
| With Data Augmentation (Without shifts) | 0.3057 | 0.1394 | 0.8512 | 0.8895 | 0.8109 | 0.8956 | 0.9506 | 0.9059 | 0.1879 |
| With Data Augmentation (With shifts) | 0.2630 | 0.1847 | 0.8556 | 0.8931 | 0.8191 | 0.8956 | 0.9509 | 0.9109 | 0.2147 |

### 5.2.3 Phase 3 – Loss function experimentation

The goal of the experiments in this phase is to find the best loss function to use when training a gore classification model. As in the previous phases, a model with MobileNetV2 encoder is trained using the

augmented training set while monitoring the validation set's Double F1-Score. The loss functions that will be evaluated in this phase are BCE, Macro Soft-F1 [89], Double Soft-F1 [89], and the new Weighted Recall Loss (WRL).

WRL is our variation of Macro Double F-Loss in order to simplify Double F-Loss while keeping the information coming from both classes. This is because Double F1-Loss is a weighted sum of a weighted sum and becomes a complex web of interactions between 4 terms: TP, TN, FP and FN. However, this loss function is used to maximise the number correct predictions (TP and TN) while minimizing the number of incorrect ones (FP and FN). This can also be achieved by performing the weighted sum of either the precision or recall of both the positive and negative class as seen in Equations (5.1) and (5.3). These sums can then be transformed into a loss function by calculating 1 minus the sum.

$$\begin{aligned} Weighted\ Recall\ Sum\ (w_0) &= (1 - w_0)Recall_1 + w_0 Recall_0 \\ &= (1 - w_0)\frac{TP}{TP + FN} + w_0 \frac{TN}{TN + FP}. \end{aligned} \tag{5.1}$$

$$\begin{aligned} Weighted\ Precision\ Sum\ (w_0) &= (1 - w_0)Precision_1 + w_0 Precision_0 \\ &= (1 - w_0)\frac{TP}{TP + FP} + w_0 \frac{TN}{TN + FN}. \end{aligned} \tag{5.2}$$

In Table 5, we notice that all the trained models achieved their best testing results around the 80% precision point on the testing set. However, the Macro Double Soft-F1 Loss model only achieved 81.87% testing recall which makes it the worst overall loss function of the ones tested. The other loss functions were able to achieve around ~90% testing recall with similar precision. The Macro Soft-F1 Loss model achieved this at a very low threshold value which could make it susceptible to noise. Both the BCE and WRL models achieved this performance at much higher threshold. The thresholds for WRL are closer to 0.5 than BCE which will make it a slightly better candidate. The entropy is also much lower for WRL. However, it is unclear if this means that these models have less uncertainty or if the entropy overfitted as it was mentioned by OpenReview.net [62]. We will refrain from making any conclusions based on entropy until more experiments are made to compare to other methods, such as the one proposed in Mi et al. [61]. Using only the results of Table 5, it is not possible to conclude whether WRL is a better loss function than BCE or vice-versa. To do so, we will fit multiple models for each loss function and compare the mean and variance of the results at the threshold of 0.5. These testing results are shown in Table 6.

*Table 5 - Comparison of the testing performance of Loss Functions for Gore Classification. All models use MobileNetV2 encode, are trained with Data Augmentation and early stopping on validation Double F1-Score. Highlighted cells represent the best results on each metric with recall and Double F1-Score being the most important.*

| Loss Function | Learning Rate | Threshold | F1-Score | Double F1-Score | Precision | Recall | Negative Precision | Negative Recall | Entropy |
|---|---|---|---|---|---|---|---|---|---|
| Binary Cross-entropy | $1 * 10^{-5}$ | 0.1394 (0.3606) | 0.8512 | 0.8895 | 0.8109 | 0.8956 | 0.9506 | 0.9059 | 0.1879 |
| Macro Soft-F1 Loss | $1 * 10^{-5}$ | 0.0326 (0.4674) | 0.8497 | 0.8880 | 0.8039 | 0.9011 | 0.9529 | 0.9010 | 0.1283 |
| Macro Double Soft-F1 Loss | $1 * 10^{-5}$ | 0.2702 (0.2298) | 0.8120 | 0.8631 | 0.8054 | 0.8187 | 0.9177 | 0.9109 | 0.0987 |
| Weighted Recall Loss (0.30 negative class weight) | $1 * 10^{-5}$ | 0.3556 (0.1444) | 0.8594 | 0.8954 | 0.8168 | 0.9066 | 0.9557 | 0.9084 | 0.0530 |

*Table 6 - Testing Results of Multiple Models trained with BCE and WRL. All models use MobileNetV2 encode, are trained with Data Augmentation and early stopping on validation Double F1-Score. Highlighted cells represent the best results on each metric with recall and Double F1-Score being the most important. An highlighted p-value means a significant test (with at least 95% confidence).*

| Loss Function | Learning Rate | Threshold | F1-Score | Double F1-Score | Precision | Recall | Negative Precision | Negative Recall | Entropy |
|---|---|---|---|---|---|---|---|---|---|
| Binary Cross-entropy | $1 * 10^{-5}$ | 0.5 | 0.8278 | 0.8695 | 0.8529 | 0.8329 | 0.9243 | 0.9438 | 0.1360 |
| | | | 0.7909 | 0.8355 | 0.8432 | 0.7860 | 0.9100 | 0.9289 | 0.2006 |
| | | | 0.8136 | 0.8458 | 0.8714 | 0.8049 | 0.9094 | 0.9390 | 0.1860 |
| | | | 0.7885 | 0.8440 | 0.8187 | 0.7977 | 0.9099 | 0.9340 | 0.1727 |
| | | | 0.8165 | 0.8472 | 0.8633 | 0.8221 | 0.9173 | 0.9358 | 0.2089 |
| | Mean | | 0.8075 | 0.8484 | 0.8499 | 0.8087 | 0.9142 | 0.9363 | 0.1808 |
| | Standard Deviation | | 0.0153 | 0.0113 | 0.0183 | 0.0168 | 0.0059 | 0.0050 | 0.0256 |
| Weighted Recall Loss (0.30 negative class weight) | $1 * 10^{-5}$ | 0.5 | 0.8448 | 0.8786 | 0.8326 | 0.8989 | 0.9512 | 0.9104 | 0.0513 |
| | | | 0.8440 | 0.8807 | 0.8300 | 0.9052 | 0.9467 | 0.9011 | 0.0521 |
| | | | 0.8449 | 0.8813 | 0.8380 | 0.8938 | 0.9475 | 0.9198 | 0.0582 |
| | | | 0.8396 | 0.8779 | 0.8192 | 0.9112 | 0.9580 | 0.9037 | 0.0665 |
| | | | 0.8181 | 0.8684 | 0.8155 | 0.8672 | 0.9386 | 0.9119 | 0.0576 |
| | Mean | | 0.8383 | 0.8774 | 0.8271 | 0.8953 | 0.9484 | 0.9094 | 0.0571 |
| | Standard Deviation | | 0.0103 | 0.0047 | 0.0084 | 0.0152 | 0.0063 | 0.0066 | 0.0054 |
| Independent T-test p-values | | | 0.01227 | 0.00438 | 0.06652 | 6.462e-05 | 4.725e-05 | 0.00025 | 0.00046 |

From Table 6, we can see that the models trained using BCE loss and WRL behave differently at the threshold value of 0.5. For all tracked metrics, except precision, the two models behave differently in a significant way which is verified using the Student's t-test [188]. The WRL models have higher testing recall, F1-Score and Double F1-Score, and lower entropy on average. On the other hand, BCE models are only slightly - and not significantly - more precise at the same threshold value. For the gore classification tasks, a lower number of FNs is the priority over the FPs and thus higher recall is better. Because of this, WRL will be used from now on.

### 5.2.4 Phase 4 – CNN encoder experimentation

In the previous phases, the pretrained CNN encoder model used was always MobileNetV2 as its small number of parameters in relation to its depth allowed for fast training while giving it good odds of learning

insightful features. However, there could be encoders that perform better. The goal of this phase is to find if such models exists and, if possible, find the characteristics that seem to increase the performance over the other models.

In this phase, the CNN encoders tested will be the ones included in version 2.3.0 of TensorFlow Core [43]. This version of TensorFlow added the EfficientNet models as part of the tf.keras.applications module [43]. All these models will be instantiated with the ImageNet weights provided in the package. They will be trained using the data augmented training set and WRL for a maximum of 1000 epochs with early stopping after 300 epochs with no change in the best validation Double F1-Score, like previous phases. The results of this experiment can be found in Table 7 and Table 8. The latter contains the results of select models at the preferred threshold value of 0.5 instead of the value that gave the best validation F1-Score. Of all the models tested in this phase, only 6 were able to achieve an above 90% testing recall at their threshold with the highest F1-Score as seen in Table 5. The EfficientNetB1 model did so with a precision of only 73.25% which means this model is particularly prone to FPs. DenseNet and VGG models generally performed well. In fact, two of the DenseNet and VGG models achieved ~80% precision and ~90% recall on the testing set. However, the high entropy value (0.2592) for DenseNet201 and VGG19 could mean that these models have some difficulty classifying some gore and non-gore images. The same can be said for NASNetMobile. The MobileNetV2 achieved the best entropy value. Unfortunately, this value is a bit of an outlier, and we must not draw any hasty conclusions until more testing of model uncertainty is performed. The VGG16 and DenseNet121 achieved better results on the Double F1-Score testing metric than MobileNetV2 and as such are better candidates until more is known about uncertainty for this task.

The VGG16 model produced the best testing Double F1-Score (0.9024) and did so with a recall of 90.11% and a precision of 83.67%. The DenseNet121 model is only slightly worse for the Double F1-Score, achieving 90.16%. It did so with an 81.95% testing precision and a 92.31% testing recall. This recall value is the best of the models tested by a margin of 2%. VGG16 achieved the second-best entropy value at 0.1259 while DenseNet121 got 0.1871. This could be because VGG16 achieved its highest validation F1-Score at a higher threshold (0.4173) than DenseNet (0.1331). If the threshold is set at 0.5, the entropy of DenseNet121 reduces to 0.1272 as seen in Table 8. However, the testing recall of the model drops to 85.40% which means that the model will produce a higher number of FNs. With the testing performance of MobileNetV2, VGG16 and DenseNet121 models being so close, the questions related to these models' uncertainty needs to be answered before the best one can be selected. The next phase will attempt to estimate that uncertainty to further compare these models.

*Table 7 - Comparison of the testing performance of CNN Encoders for gore classification. All models are trained with Data Augmentation, WRL and early stopping on validation Double F1-Score. Highlighted cells represent the best results on each metric with recall and Double F1-Score being the most important.*

| CNN Encoder | Number of parameters | Threshold | F1-Score | Double F1-Score | Precision | Recall | Negative Precision | Negative Recall | Entropy |
|---|---|---|---|---|---|---|---|---|---|
| MobileNetV2 | 2,921,539 | 0.3556 | 0.8594 | 0.8954 | 0.8168 | 0.9066 | 0.9557 | 0.9084 | 0.0530 |
| VGG16 | 14,981,953 | 0.4173 | 0.8677 | 0.9024 | 0.8367 | 0.9011 | 0.9538 | 0.9208 | 0.1259 |
| VGG19 | 20,291,649 | 0.2624 | 0.8579 | 0.8958 | 0.8377 | 0.8791 | 0.9443 | 0.9233 | 0.3115 |
| ResNet50V2 | 24,624,641 | 0.1295 | 0.8303 | 0.8740 | 0.7910 | 0.8736 | 0.9403 | 0.8960 | 0.1541 |
| ResNet101V2 | 43,686,401 | 0.3319 | 0.8392 | 0.8830 | 0.8324 | 0.8462 | 0.9302 | 0.9233 | 0.1768 |
| ResNet152V2 | 59,391,489 | 0.1203 | 0.8564 | 0.8952 | 0.8449 | 0.8681 | 0.9398 | 0.9282 | 0.1396 |
| NASNetMobile | 4,817,685 | 0.1822 | 0.8462 | 0.8847 | 0.7933 | 0.9066 | 0.9550 | 0.8936 | 0.2711 |
| MobileNet | 3,760,321 | 0.3879 | 0.8347 | 0.8812 | 0.8514 | 0.8187 | 0.9197 | 0.9356 | 0.1719 |
| DenseNet121 | 7,568,961 | 0.1331 | 0.8682 | 0.9016 | 0.8195 | 0.9231 | 0.9633 | 0.9084 | 0.1677 |
| DenseNet169 | 13,504,577 | 0.5449 | 0.8362 | 0.8826 | 0.8605 | 0.8132 | 0.9179 | 0.9406 | 0.1871 |
| DenseNet201 | 19,315,777 | 0.2164 | 0.8462 | 0.8847 | 0.7933 | 0.9066 | 0.9550 | 0.8936 | 0.2592 |
| EfficientNetB0 | 4,713,124 | 0.4238 | 0.8242 | 0.8725 | 0.8242 | 0.8242 | 0.9208 | 0.9208 | 0.2824 |
| EfficientNetB1 | 7,238,792 | 0.2037 | 0.8146 | 0.8574 | 0.7325 | 0.9176 | 0.9581 | 0.8490 | 0.3445 |
| EfficientNetB2 | 8,498,170 | 0.2208 | 0.8285 | 0.8733 | 0.7970 | 0.8626 | 0.9357 | 0.9010 | 0.2979 |

*Table 8 - Comparison of the testing performance of CNN Encoders at threshold value of 0.5. All models are trained with Data Augmentation, WRL and early stopping on validation Double F1-Score. Highlighted cells represent the best results on each metric with recall and Double F1-Score being the most important.*

| CNN Encoder | Number of parameters | Threshold | F1-Score | Double F1-Score | Precision | Recall | Negative Precision | Negative Recall | Entropy |
|---|---|---|---|---|---|---|---|---|---|
| MobileNetV2 | 2,921,539 | 0.5 | 0.8448 | 0.8786 | 0.8326 | 0.8989 | 0.9512 | 0.9104 | 0.0513 |
| VGG16 | 14,981,953 | | 0.8500 | 0.8803 | 0.8500 | 0.8896 | 0.9455 | 0.9255 | 0.1237 |
| VGG19 | 20,291,649 | | 0.8351 | 0.8532 | 0.8711 | 0.8379 | 0.9203 | 0.9460 | 0.2457 |
| NASNetMobile | 4,817,685 | | 0.7410 | 0.8177 | 0.7667 | 0.7658 | 0.9087 | 0.9106 | 0.2195 |
| DenseNet121 | 7,568,961 | | 0.8468 | 0.8815 | 0.8811 | 0.8540 | 0.9352 | 0.9422 | 0.1272 |
| DenseNet201 | 19,315,777 | | 0.8310 | 0.8455 | 0.8568 | 0.8435 | 0.9343 | 0.9397 | 0.2119 |
| EfficientNetB1 | 7,238,792 | | 0.8109 | 0.8161 | 0.8484 | 0.8209 | 0.9139 | 0.9180 | 0.2867 |

### 5.2.5 Phase 5 – Uncertainty Estimation

Since the models were already trained, uncertainty estimation must be done using a training-free method. The methods discussed in Mi et al. [61] will be adopted as they can be used with such models. The simplest algorithm to implement is Infer-Transformation. The uncertainty is calculated as a variance, where the mean is the result obtained using the unaltered image, and the variance calculated using the results of images where a given transformation was performed. We calculated this variance on the testing set using two different set of transformations. The first consists of 90 degrees rotation and flips giving 7 transformed images. The second set adds random Gaussian noise to these 7 images. The Gaussian noise is centered at 0 with a standard deviation of 0.02. The standard deviation was determined by reducing the value until the variance was under 10% for the MobileNet and WRL models. The results of the best CNN encoder models can be found in Table 9. The choice of the best loss function needs to be revisited as entropy was a factor in the analysis for previous phases. As we explain in the next section, according to the results in Table 9, entropy may be a poor estimator of uncertainty.

#### *5.2.5.1 Best CNN Encoder*

The Inter-Transformation testing results of Table 9 create some doubt in the use of entropy to estimate uncertainty as presented in Mi et al. [61]. The MobileNetV2 entropy of 0.0530 is closer to the variance of Inter-Transformation than the standard deviation. However, for all the other models, the entropy is closer to the latter. Moreover, the model that had the lowest testing uncertainty, calculated using Inter-Transformation, is VGG19. This is surprising as this model had one of the highest entropy values in previous experiments. We also observe the wide discrepancies between entropy and uncertainty for the NasNetMobile, DenseNet201, and EfficientNetB1 models. Moving forward, we will not use entropy to describe a model's uncertainty. We will instead use other methods such as Infer-Transformation, Infer-Noise or Infer-dropout.

*Table 9 - Uncertainty Estimation for CNN Encoders using Inter-Transformation algorithm on the testing set. All models are trained with Data Augmentation, WRL and early stopping on validation Double F1-Score. Highlighted cells represent the best results on each metric with recall and Double F1-Score being the most important.*

| CNN Encoder | Threshold | F1-Score | Double F1-Score | Precision | Recall | entropy | Uncertainty Estimation (Rotation and Flip) | | Uncertainty Estimation (Rotation, Flip and Gaussian Noise [σ=0.02]) | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | Variance | Standard deviation | Variance | Standard deviation |
| MobileNetV2 | 0.3556 | 0.8594 | 0.8954 | 0.8168 | 0.9066 | 0.0530 | 0.0231 | 0.1518 | 0.0554 | 0.2353 |
| VGG16 | 0.4173 | 0.8677 | 0.9024 | 0.8367 | 0.9011 | 0.1259 | 0.0199 | 0.1410 | 0.0226 | 0.1505 |
| VGG19 | 0.2624 | 0.8579 | 0.8958 | 0.8377 | 0.8791 | 0.3115 | 0.0127 | 0.1129 | 0.0149 | 0.1222 |
| NASNetMobile | 0.1822 | 0.8462 | 0.8847 | 0.7933 | 0.9066 | 0.2711 | 0.0306 | 0.1750 | 0.0441 | 0.2099 |
| DenseNet121 | 0.1331 | 0.8682 | 0.9016 | 0.8195 | 0.9231 | 0.1677 | 0.0253 | 0.1591 | 0.0357 | 0.1889 |
| DenseNet201 | 0.2164 | 0.8462 | 0.8847 | 0.7933 | 0.9066 | 0.2592 | 0.0290 | 0.1704 | 0.0350 | 0.1870 |
| EfficientNetB1 | 0.2037 | 0.8146 | 0.8574 | 0.7325 | 0.9176 | 0.3445 | 0.0291 | 0.1706 | 0.0364 | 0.1909 |

The VGG19 model has the lowest testing uncertainty values as can be seen in Table 9. However, its lower testing recall when compared to other models makes it a lesser candidate for gore classification. Its smaller variant, VGG16, has slightly worse uncertainty values while still keeping VGG19's resistance to noise. This combined with its higher recall at a similar precision value makes it a very good option among the models tested. The DenseNet121 model had a testing uncertainty standard deviation that was higher by a margin of 0.03. In addition, it has a lower threshold value makes it hard to recommend as a model. The MobileNetV2 model was especially sensitive to noise as the uncertainty standard deviation increase by 0.08 when it was added to Inter-Transformation. Based on the results of this algorithm, VGG16 is the best CNN encoder evaluated, with DenseNet121 and MobileNetV2 being slightly worse.

#### *5.2.5.2 Best loss function revisited*

Based on our results, it seems that the entropy values for the MobileNetV2 model are outliers and do not relate to this model's uncertainty. Therefore, we want to assess how uncertain the models trained using the BCE Loss are. To do so, we calculated the uncertainty of each of the 10 models trained with the BCE and WRL loss functions. The results can be found in Table 10. The BCE models' uncertainties are not significantly lower than those of the WRL models at a 95% level of significance. This means that the sensitivity to noise and changes is most likely caused by the choice of encoder, MobileNetV2 in this case, and only slightly reduced by the choice of the loss functions.

*Table 10 - Testing Results (with uncertainty) of Multiple Models trained on BCE loss and WRL with MobileNetV2. The random uncertainty allowed us to test for statistical significance of the performance resulting from training on the two loss functions. Highlighted cells represent the best results on each metric with recall and Double F1-Score being the most important.*

| Loss Function | Learning Rate | Loss Weight for negative class | Threshold | F1-Score | Double F1-Score | Precision | Recall | Entropy | Uncertainty Estimation Standard Error (Rotation and Flip) | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | Without Noise | With Noise |
| Binary Cross-entropy | $1 * 10^{-5}$ | 0 | 0.5 | 0.8278 | 0.8695 | 0.8529 | 0.8329 | 0.1360 | 0.1866 | 0.2529 |
| | | | | 0.7909 | 0.8355 | 0.8432 | 0.7860 | 0.2006 | 0.1683 | 0.2348 |
| | | | | 0.8136 | 0.8458 | 0.8714 | 0.8049 | 0.1860 | 0.1712 | 0.2220 |
| | | | | 0.7885 | 0.8440 | 0.8187 | 0.7977 | 0.1727 | 0.1732 | 0.2423 |
| | | | | 0.8165 | 0.8472 | 0.8633 | 0.8221 | 0.2089 | 0.1742 | 0.2349 |
| | Mean | | | 0.8075 | 0.8484 | 0.8499 | 0.8087 | 0.1808 | 0.1747 | 0.2374 |
| | Standard Deviation | | | 0.0153 | 0.0113 | 0.0183 | 0.0168 | 0.0256 | 0.0063 | 0.0102 |
| Weighted Recall Loss | $1 * 10^{-5}$ | 0.30 | 0.5 | 0.8448 | 0.8786 | 0.8326 | 0.8989 | 0.0513 | 0.1518 | 0.2340 |
| | | | | 0.8440 | 0.8807 | 0.8300 | 0.9052 | 0.0521 | 0.2533 | 0.2617 |
| | | | | 0.8449 | 0.8813 | 0.8380 | 0.8938 | 0.0582 | 0.1751 | 0.2585 |
| | | | | 0.8396 | 0.8779 | 0.8192 | 0.9112 | 0.0665 | 0.1645 | 0.2453 |
| | | | | 0.8181 | 0.8684 | 0.8155 | 0.8672 | 0.0576 | 0.1614 | 0.2543 |
| | Mean | | | 0.8383 | 0.8774 | 0.8271 | 0.8953 | 0.0572 | 0.1812 | 0.2508 |
| | Standard Deviation | | | 0.0103 | 0.0047 | 0.0084 | 0.0152 | 0.0055 | 0.0368 | 0.0100 |
| Independent T-test p-values | | | | 0.01227 | 0.00438 | 0.06652 | 6.462e-05 | 0.00046 | 0.7436 | 0.09749 |

*Table 11 - Uncertainty Estimation of VGG16 models with BCE loss and WRL on the testing set. In this case, we tried all the uncertainty estimation methods presented in Mi et al. [61].*

| Loss Function | Uncertainty Estimation Standard Error | | | |
|---|---|---|---|---|
| | Rotations and Flips | Rotations, Flips and Noise | Infer-dropout $[p = 0.5]$ | Infer-Noise $[\sigma = 0.20]$ |
| Binary Cross-entropy | 0.1248 | 0.1348 | 0.0591 | 0.0863 |
| Weighted Recall Loss | 0.1410 | 0.1505 | 0.1171 | 0.1301 |

To further compare both loss function and their uncertainty distribution, we trained a classification model with a VGG16 encoder for both loss functions. The uncertainty of both these functions is then estimated using all four methods of Mi et al. [61]. The Infer-dropout and Infer-Noise from Mi et al. [61] are implemented by adding a Dropout or Gaussian Noise layers, respectively after the CNN encoder and before the Global Pooling Average layer, as we experimentally found it to be the ideal spot for these layers. Since the model is implemented with TensorFlow Keras Application Programming Interface (API), these added layers are customized to reuse the API random dropout and noise layer, but only enabling it during inference instead of training [189]. The dropout probability and Gaussian standard error were found by increasing their value until the uncertainty became unrealistically high as proposed in Mi et al. [61]. The chosen value is then

the last one before the unrealistic increase. The results of this experimentation can be found in Table 11 where we can see that BCE loss does in fact lead to lower uncertainty than WRL. The distribution of the testing uncertainty for the entire dataset and the set of TP, TN, FP and FN results are plotted in Figure 3 and Figure 4.

In Table 10 and Table 11, we can see that the WRL models performed better in all testing metrics except precision and uncertainty. The difference in the uncertainty standard deviation on the testing set (0.0065) is quite small when compared to the difference for the testing precision, recall, F1-Score and Double F1-Score metrics. Moreover, as discussed in section 5.1, these metrics are more important to describe a successful model for gore classification. The addition of uncertainty in the evaluation metrics will therefore not change the choice of the best loss function.

Looking at the distribution of uncertainty for both models in Figure 3 and Figure 4, we notice that the errors (FP and FN) generally have a higher uncertainty as at least 50% of the outputs have more uncertainty than correct predictions (TP and TN). The WRL model has a lower uncertainty for TP while the BCE model has a lower uncertainty for TN and FN. This suggests that the WRL model is better at distinguishing images containing gore (positives) from images without (negatives). However, the BCE model seems better at distinguishing negative from positive results. This could explain why the latter model's precision reduced significantly when maximizing recall.

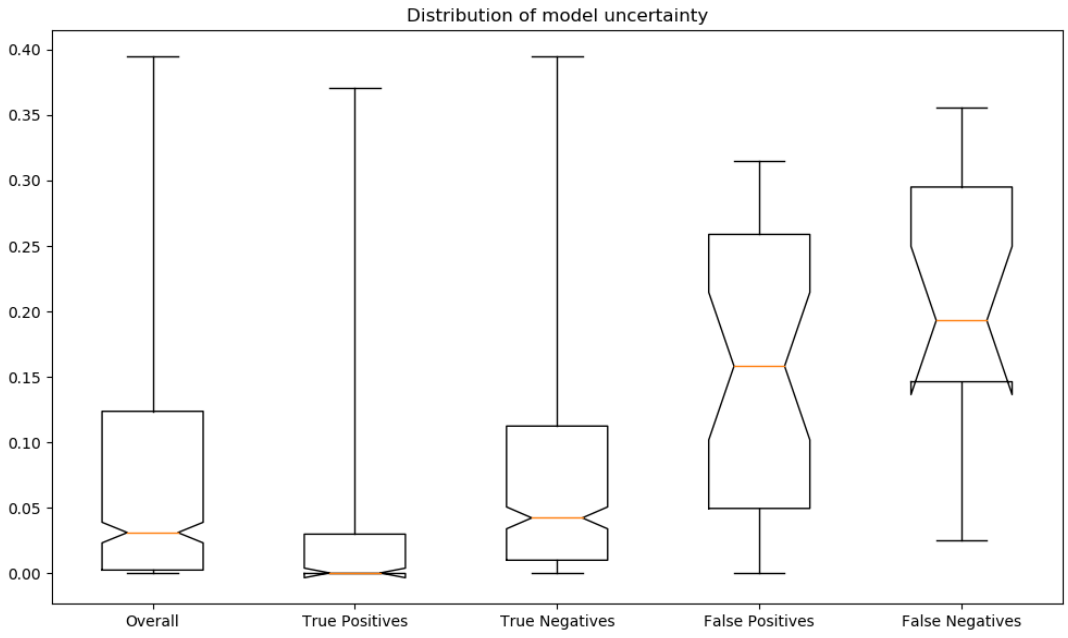


*Figure 3 - Distribution of testing uncertainty for VGG16 model trained with WRL. The uncertainty was computed using the infer-dropout method of Mi et al. [61]. The uncertainty for correct predictions is for the most part lower than the uncertainty of wrong predictions.*

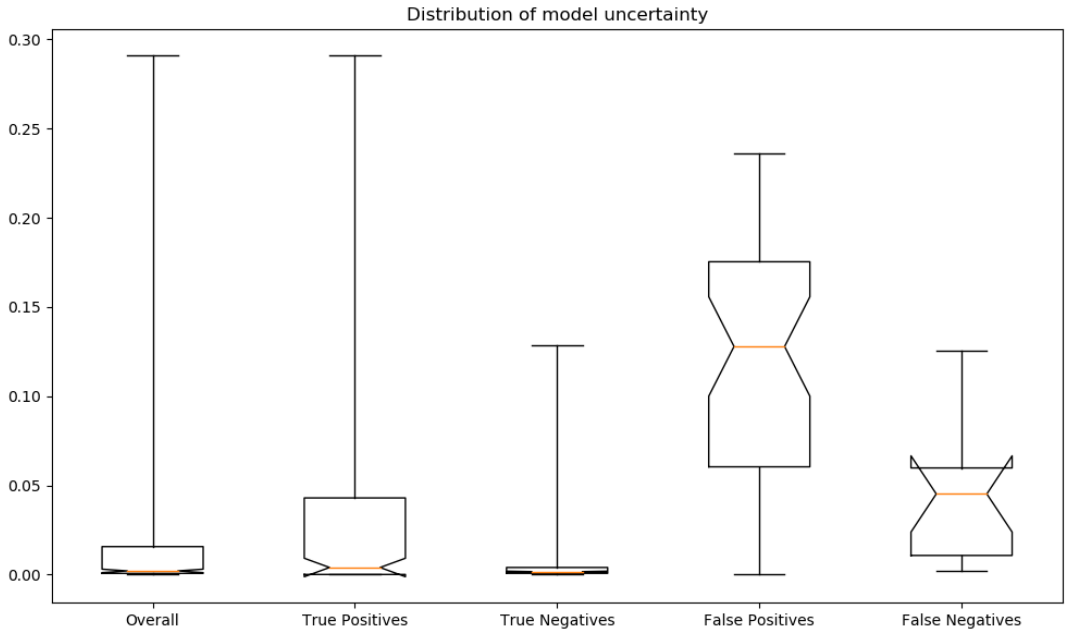


*Figure 4 - Distribution of testing uncertainty for VGG16 model trained with BCE loss. The uncertainty was computed using the infer-dropout method of Mi et al. [61]. The uncertainty for correct predictions is for the most part lower than uncertainty of wrong predictions. This model is more certain than WRL model when making negative predictions.*

### 5.2.6 Phase 6 – Ensembles

The VGG16 models trained with BCE and WRL showed lower uncertainty for different classes as seen in Figure 3 and Figure 4. We also built ensembles with the DenseNet121 and MobileNetV2 encoders trained with WRL. One could then ask if this could be leveraged in a combined model. There are multiple ways to do so, and this section will explore multiple solutions.

#### *5.2.6.1 Cascading Ensemble*

As previously stated, models with high recall and precision are the ones preferred for gore classification. However, the model that performs best on these metrics (VGG16 trained with WRL) has a higher uncertainty for non-gore images in the testing set. In cases where this model is uncertain of its prediction, we will like to 'consult' the BCE model which will help where the image does not contain gore, as it can be seen in Figure 5. In the case where both models have high uncertainty for a given image, the prediction is the average of the output of both models. In other words, the consultation or cascading occurs when the threshold (0.5) is contained in Equation (5.3) where $\bar{x}$ is the mean prediction and $\sigma_x$ the standard deviation of the model's prediction on the given input.

$$[\bar{x} - \sigma_x, \bar{x} + \sigma_x] \tag{5.3}$$

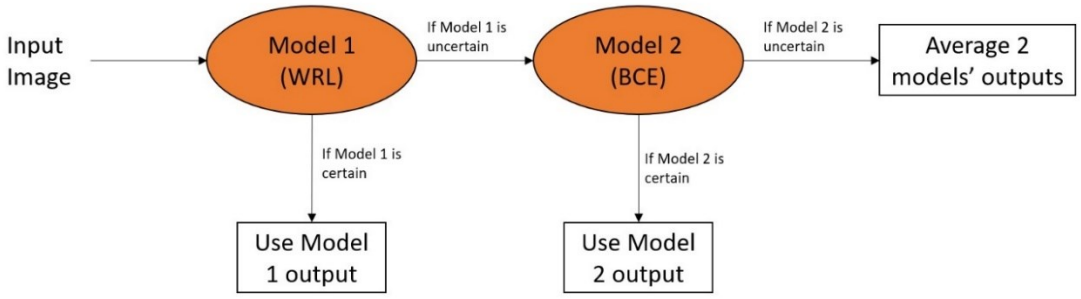


*Figure 5 - Cascading Ensemble based on model uncertainty. A model is uncertain if the decision threshold is within a standard deviation of the mean prediction (Equation (5.3)).*

From Table 12, we notice that the Cascading model performs slightly better when using the threshold of 0.5 for both models instead of the values which produced the best F1-Score for each individual model. The same table also shows that this model can achieve the same recall value as the VGG16 WRL model with a higher testing precision score of 0.8454 vs 0.8367. There is a cost to this model performance as it has a higher average uncertainty.

Comparing Figure 6 to Figure 3 and Figure 4, we see that the cascading model's testing uncertainty distribution is similar to the WRL model with the BCE model achieving much lower uncertainty overall. One of the most striking differences between Figure 3 and Figure 6 is that the FP first quartile is much lower for the cascading model. The third quartile and median are also slightly lower which makes this distribution closer to the BCE model. However, it does not have the latter's very low uncertainty distribution of FNs. This suggests that the cascading model is slightly better at finding gore in images than the WRL model.

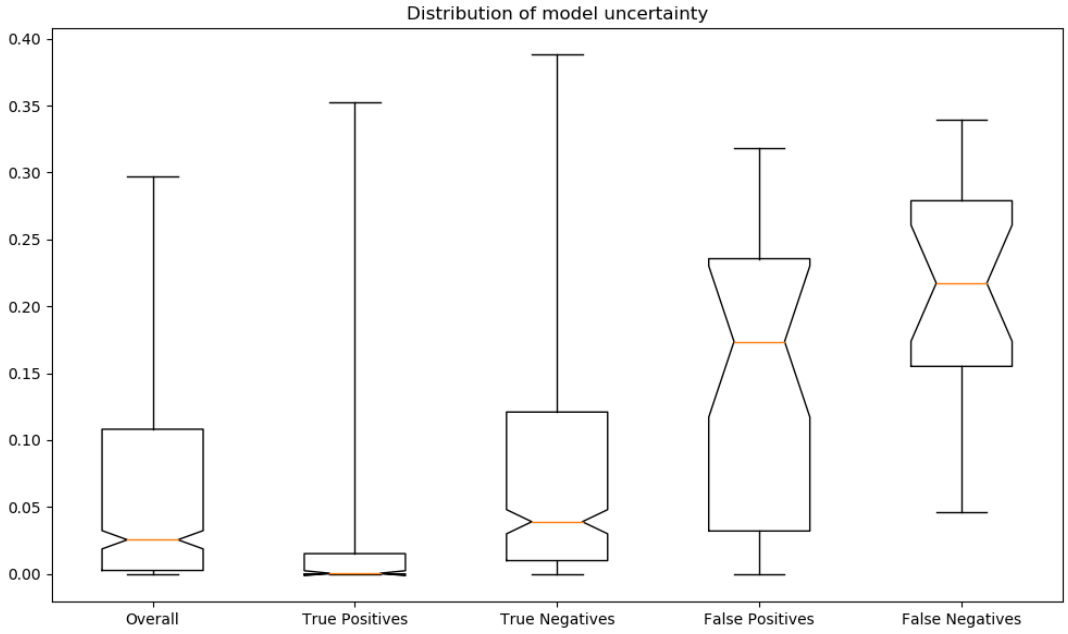


*Figure 6 - Distribution of testing uncertainty for Cascading VGG16 model. The uncertainty was computed using the infer-dropout method of Mi et al. [61]. The first quartile for FPs predictions is lower than in Figure 3 and Figure 4.*

*Table 12 - Comparison of testing results of the Cascading model and the models used to make it. Highlighted cells represent the best results on each metric with recall and Double F1-Score being the most important. The cascading ensemble was able to achieve a very slight improvement in terms of Double F1-Score compared to the standalone VGG16 WRL model. The cascading model did not achieve lower uncertainty as hypothesized.*

| Model name | Threshold | F1-Score | Double F1-Score | Precision | Recall | Entropy | Uncertainty Estimation (Infer-dropout [$p = 0.5$]) | |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | Variance | Standard deviation |
| VGG16 (BCE) | 0.5 | 0.7949 | 0.8504 | 0.7980 | 0.8403 | 0.1425 | 0.0035 | 0.0591 |
| | 0.1779 | 0.8463 | 0.8838 | 0.7814 | 0.9231 | 0.1806 | 0.0035 | 0.0591 |
| VGG16 (WRL) | 0.5 | 0.8500 | 0.8803 | 0.8500 | 0.8896 | 0.1237 | 0.0137 | 0.1171 |
| | 0.4173 | 0.8677 | 0.9024 | 0.8367 | 0.9011 | 0.1259 | 0.0137 | 0.1171 |
| VGG16 Cascading | 0.5 for both models | 0.8723 | 0.9060 | 0.8454 | 0.9011 | 0.1927 | 0.0947 | 0.1266 |
| | 0.4173 (WRL), 0.1779 (BCE) | 0.8608 | 0.8960 | 0.8107 | 0.9176 | 0.2014 | 0.0873 | 0.1145 |

#### *5.2.6.2 Voting Ensemble*

Another way of using the output of multiple models to obtain a single prediction is to perform voting once all the models made their individual predictions. This is what Figure 7 shows. For the combination of different models to be effective, the individual models need to be different enough to increase the odds of making a correct prediction. In our case, we will do so by providing models trained with different CNN encoders and loss functions. The voting can be implemented by averaging the predictions of each model in the ensemble. A weighted sum can be determined by adding a FC layer that receives predictions as input as presented in Figure 7. The average is then obtained by dividing the layer's outputs by the sum of the weights. This approach is useful when we want the best model(s) to weigh more in the decision-making process [190, 191] and so it will be adopted in this experiment.

For each voting ensemble, up to 4 models are used. Three were all trained with WRL and used the following encoders respectively: VGG16, MobileNetV2, DenseNet121. The other model used is the VGG16 based model trained with BCE loss as used in the Cascading model. To avoid confusion, the latter will be referred as 'VGG16 (BCE)' in this section while the other model with the same encoder will be 'VGG16 (WRL)'. Since all models used had inference dropout layers, the results below are based on the average output after giving the same input 30 times. The uncertainty is calculated as described in section 5.2.5.2 using the Infer-dropout method.

When combining the models, we were unsure if the voting layer will perform better with a vote threshold. To determine this, we created two versions of each model, one with thresholding prior to vote and one without. The results are presented in Table 13. From this table, we notice that at the threshold value of

0.5, models with thresholding tended to have a higher recall than those without. However, this came at the cost of precision as these models have lower F1-Scores than the ones without thresholding. Thus, we determined that not using threshold results in a better overall model.

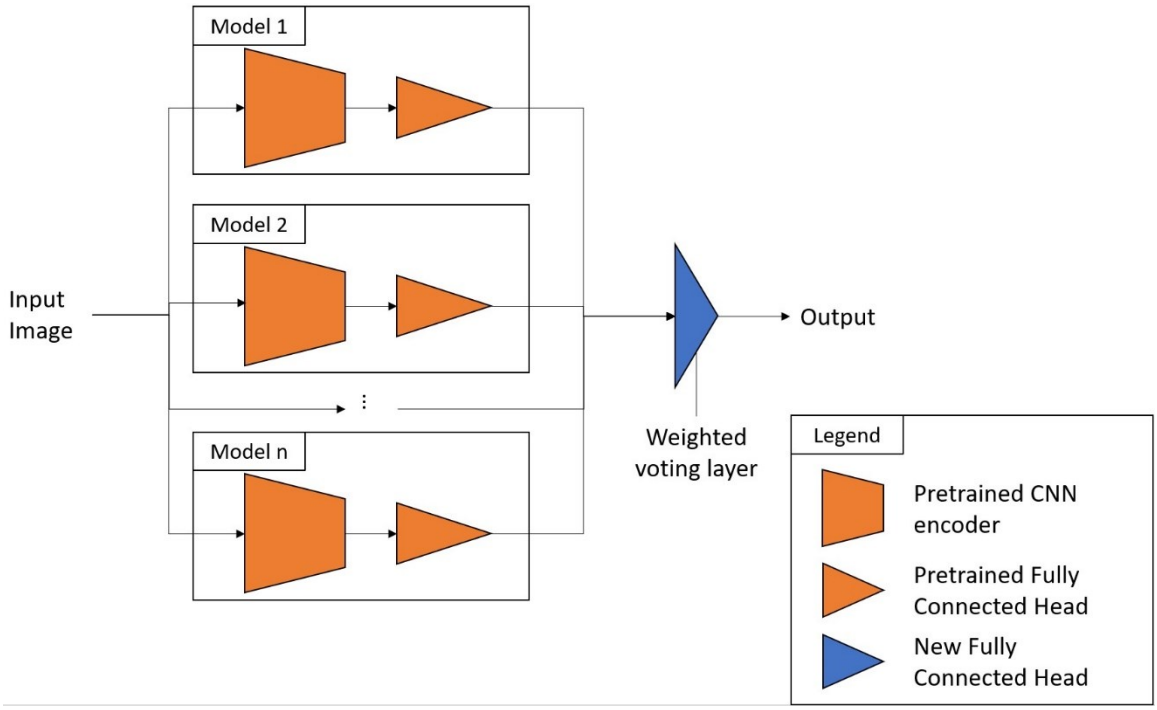


*Figure 7 – Diagram of a Voting Ensemble with learned weighted voting layer.*

In Table 13, the cells highlighted in gold show the best results at the threshold of 0.5. The cells in green show the best results at the threshold that produces the highest recall for this model. Of the 8 ensembles trained, only 2 achieved the best testing results on multiple metrics. The other ensembles either performed worse overall or achieved higher recalls at the cost of performance on the other metrics. The two best performing models are the Voting_WRL model trained on all three of the WRL models and the Voting_Mixed model trained with all four models. The latter performed the best at the threshold value of 0.5. It achieved a testing precision of 87.30% and testing recall of 90.66% giving it the best F1-Score of 88.95%. Voting_WRL produced 87.70% precision and 90.11% recall for a F1-Score of 88.89%. Hence, the performance of Voting_Mixed and Voting_WRL is nearly equivalent. On the other hand, when moving the threshold to obtain a 90% recall on the validation set, the testing scores of Voting_WRL are less affected. It achieved a precision of 84.69% and a recall of 87.83% for a F1-Score of 87.83% at the threshold value of 0.3939. Meanwhile, the Voting_Mixed model achieved a testing 82.35% precision and a testing 92.31 recall at a threshold value of 0.3409. This gave it a lower F1-Score of 87.05% when compared to its F1-Score at 0.5 which was 88.95%.

Another impact of the voting ensembles is the distribution of uncertainty. Comparing Figure 8 to Figure 3, Figure 4 and Figure 6, we notice a higher amount of testing uncertainty for most of the TPs and TNs.

However, the interquartile range (IQR) of correct and incorrect predictions' uncertainty do not overlap. For the most part, the ensemble has the desirable property of being wrong when it is less certain of its results, a sign of epistemic uncertainty [192]. It is possible that this is because the model was less (or even not) exposed to similar examples during training. It could be possible to correct these errors by finding what is missing in the training set and adding it [192]. On the other hand, an IQR overlap between correct and incorrect predictions could be a sign of bias within the pretrained model or it could come from the experimental setup, which will be harder to correct. Experimental setup issues are possible as the overlap is not consistent between models. Thus, it cannot be seen as uncertainty coming from the training set: its an aleatoric uncertainty [192]. This refers to uncertainty that comes from conditions outside of the model control.

Voting ensembles are a useful way to obtain better performance using pre-trained models. In the case of gore classification, this resulted in almost a 3% increase in precision, at a similar recall to the previous best performing models. This increase in performance also results in an increase in computing time as the end model is multiple times larger than that of each included model, individually. The MobileNetV2, DenseNet121 and VGG16 have 2,921,539, 7,568,961 and 14,981,953 parameters, respectively. However, the best voting models, Voting_WRL and Voting_Mixed, have 25,472,454 and 40,454,408 parameters.

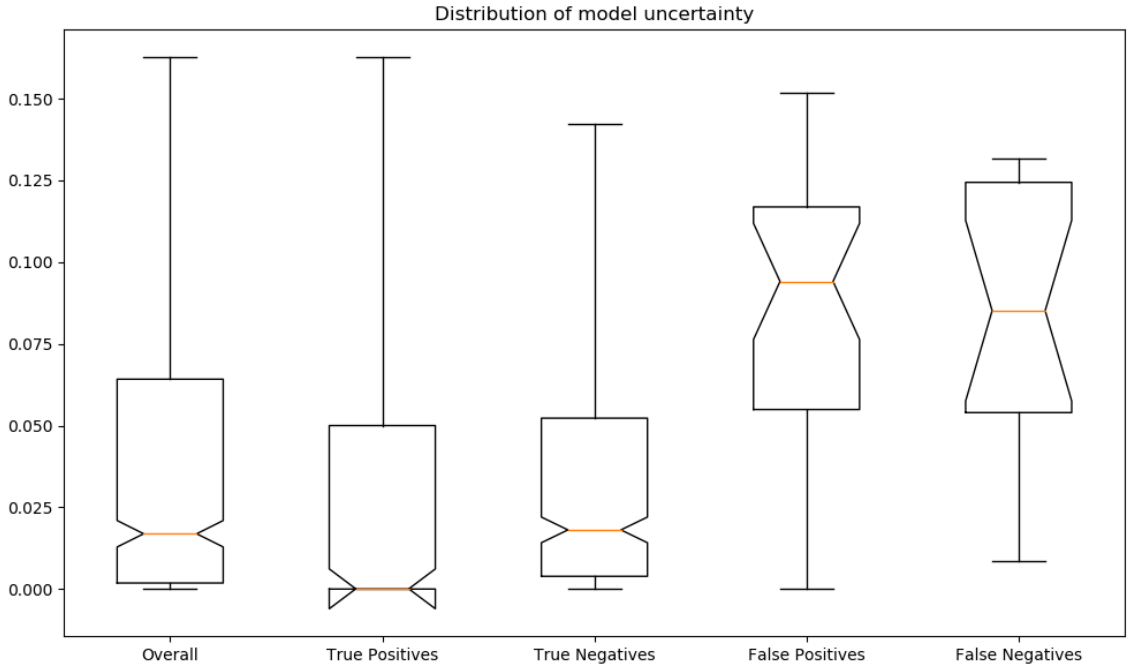


*Figure 8 – Testing uncertainty distribution for the 'Voting_WRL' ensemble without no thresholding prior to voting. The uncertainty was computed using the infer-dropout method of Mi et al. [61]. The uncertainty of wrong predictions is much more consistent than models presented in Figures Figure 3, Figure 4 and Figure 6.*

*Table 13 - Comparison of testing results of different Voting Ensembles for Gore Classification. Highlighted cells in gold represent best performance at the threshold of 0.5. Green coloured cells represent the best performance at the threshold which achieved the highest validation F1-Score. The recall and Double F1-Score being the most important.*

| Model name | Models used | Threshold | F1-Score | Double F1-Score | Precision | Recall | Uncertainty Estimation (Infer-dropout [$p = 0.5$]) | |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | Variance | Standard deviation |
| Voting_VGG16 (No threshold prior to voting) | VGG16 (WRL), VGG16 (BCE) | 0.5 | 0.8602 | 0.8976 | 0.8421 | 0.8791 | 0.0681 | 0.0913 |
| Voting_VGG16 | VGG16 (WRL), VGG16 (BCE) | 0.5 | 0.8579 | 0.8958 | 0.8377 | 0.8791 | 0.1022 | 0.1514 |
| Voting_WRL (No threshold prior to voting) | VGG16 (WRL), MobileNetV2 (WRL), DenseNet121 (WRL) | 0.5 | 0.8889 | 0.9189 | 0.8770 | 0.9011 | 0.0530 | 0.0700 |
| | | 0.3929 | 0.8783 | 0.9102 | 0.8469 | 0.9121 | 0.0543 | 0.0705 |
| Voting_WRL | VGG16 (WRL), MobileNetV2 (WRL), DenseNet121 (WRL) | 0.5 | 0.8714 | 0.9047 | 0.8342 | 0.9121 | 0.0760 | 0.1110 |
| | | 0.3793 | 0.8769 | 0.9078 | 0.8221 | 0.9396 | 0.0763 | 0.1097 |
| Voting_WRL_2 Models (No threshold prior to voting) | VGG16 (WRL), DenseNet121 (WRL) | 0.5 | 0.8841 | 0.9152 | 0.8677 | 0.9011 | 0.0674 | 0.0872 |
| | | 0.3159 | 0.8747 | 0.9060 | 0.8182 | 0.9396 | 0.0667 | 0.0875 |
| Voting_WRL_2 Models | VGG16 (WRL), DenseNet121 (WRL) | 0.5 | 0.8653 | 0.8996 | 0.8186 | 0.9176 | 0.0981 | 0.1391 |
| Voting_Mixed (No threshold prior to voting) | VGG16 (WRL), MobileNetV2 (WRL), DenseNet121 (WRL), VGG16 (BCE) | 0.5 | 0.8895 | 0.9192 | 0.8730 | 0.9066 | 0.0493 | 0.0643 |
| | | 0.3409 | 0.8705 | 0.9034 | 0.8235 | 0.9231 | 0.0492 | 0.0651 |
| Voting_Mixed | VGG16 (WRL), MobileNetV2 (WRL), DenseNet121 (WRL), VGG16 (BCE) | 0.5 | 0.8691 | 0.9029 | 0.8300 | 0.9121 | 0.0615 | 0.0872 |
| | | 0.4982 | 0.8773 | 0.9089 | 0.8358 | 0.9231 | 0.0608 | 0.0850 |

#### *5.2.6.3 Stacking Ensembles*

The two previous types of ensembles tested kept each of the models' encoders and fully connected head. Thus, only the end results of each model are considered in an ensemble setting. The stacking ensemble combines the results of the different models at the encoder level. Accordingly, a new classification head must be trained with the new combined features as shown in Figure 9. Conveniently, the three best encoders for

gore classification (MobileNetV2, DenseNet121 and VGG16) have an output size of 7x7; only the number of channels differs. MobileNetV2 has 1280 channels in the output while DenseNet121 has 1024 and VGG16, 512. In this experiment, the output of each encoder's has its number of channels reduced to 256 using 1x1 convolutional layers before it is passed to a fully connected head with two layers. The hidden layer has 256 nodes.

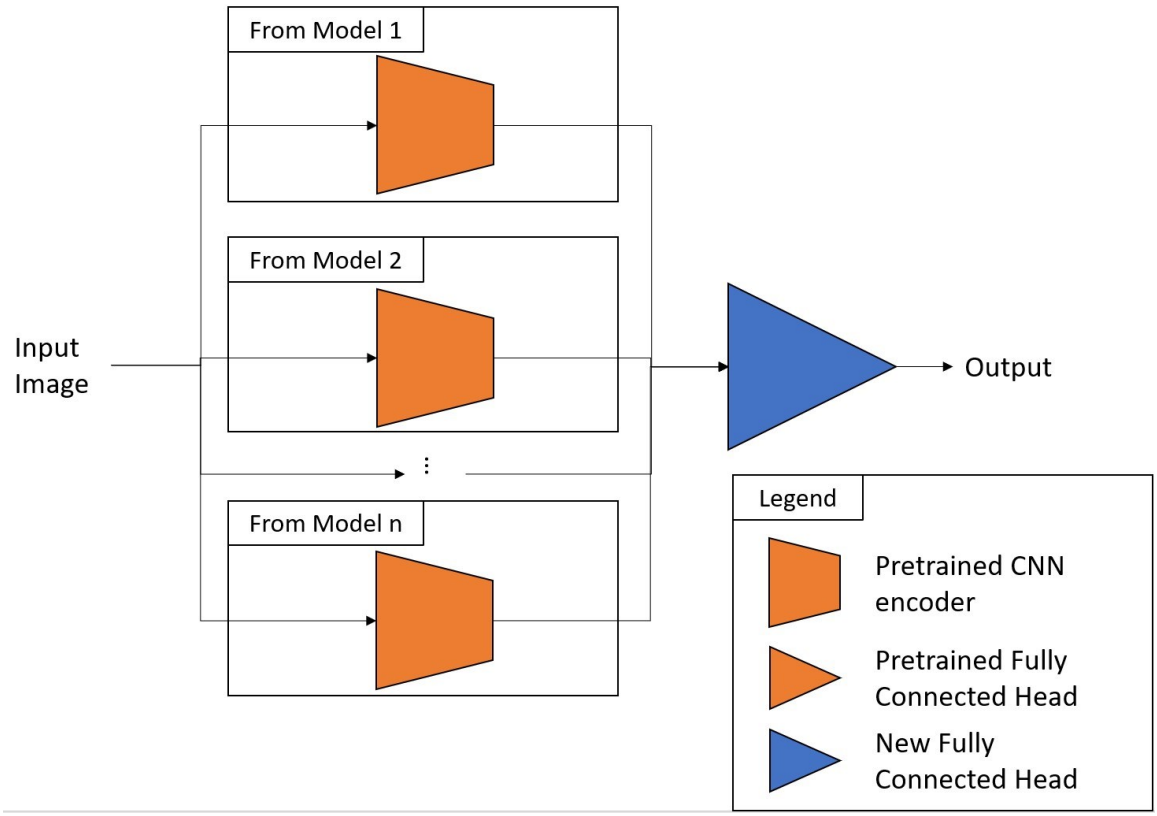


*Figure 9 - Diagram of a Stacking ensemble for Image classification*

The testing results of training stacking ensembles are presented in Table 14. We notice that the model trained using the encoders pretrained on gore classification (from Phase 4) was able to achieve better results on all testing metrics when compared to the one trained with encoders pre-trained on the ImageNet task. However, the gore classification pre-trained model was not able to improved on the 88.95% testing F1-Score and 91.92% Double F1-Score of the best voting ensemble of the previous section. This model performance was also slightly worse in terms of these metrics than the VGG16 and DenseNet121 models from section 5.2.4. Given our results, we conclude that the performance of the stacking ensemble models we tested is inferior to that of other ensemble approaches we investigated for the gore classification task.

*Table 14 - Stacking Ensembles Gore Classification Testing Results. Highlighted cells represent the best results on each metric with recall and Double F1-Score being the most important.*

| Model name | Models used | Threshold | F1-Score | Double F1-Score | Precision | Recall | Uncertainty Estimation (Infer-dropout [$p = 0.5$]) | |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | Variance | Standard deviation |
| classification_RecallLoss_030 (Encoder with ImageNet weights) | VGG16 (WRL), MobileNetV2 (WRL), DenseNet121 (WRL) | 0.5 | 0.8493 | 0.8862 | 0.8342 | 0.9050 | 0.0590 | 0.1195 |
| classification_RecallLoss_030_best_start (Encoder weights from best individual model) | VGG16 (WRL), MobileNetV2 (WRL), DenseNet121 (WRL) | 0.5 | 0.8642 | 0.8940 | 0.8836 | 0.8827 | 0.0488 | 0.0974 |
| | | 0.3793 | 0.8631 | 0.8929 | 0.8644 | 0.9034 | 0.0473 | 0.0931 |

### 5.2.7 Phase 7 – MIL for Gore Classification

Given the successful use of MIL in Pornography and Nudity detection in Jin et al. [94], we decided to test this approach to possibly improve our gore classification models. Therefore, we converted the input images into bags of instances where each bag is formed of 5 instances. The first is the full image at a resolution of 224x224. The other 4 are 4 224x224 patches taken from the image at a resolution of 448x448. The resulting input is a multi-scale representation of the original image. To be able to use pre-trained encoders included in TensorFlow, the 5 images are passed as individual RGB images through the encoder and are then concatenated together. Once concatenated, the output is further convolved to combine local features across images and reduce the number of channels. It is then passed to a fully connected head to obtain the gore classification result, as with previous models. The best encoders from previous phases (VGG16, DenseNet121 and MobileNetV2) were used for this experiment.

The MIL results presented in Table 15 show that this experiment has not performed as intended. The Double F1-Scores for all the used encoders have decrease when the MIL is applied. The worst performing encoder in this setup is MobileNetV2 which incurred a 14.64% decrease in this testing metric at 0.5 threshold. The VGG16 and DenseNet121 models' performance decreased by 1.59% and 1.5% respectively. The performance decrease could be due to several causes. For instance, the MIL models are much larger in terms of parameters and are thus more likely to overfit [193, 194]. Another possible cause is the fact that the encoder weights are shared for all the provided instances to save memory and so the SGD update of the encoder weights takes all instances into consideration. This may lead to issues during backpropagation since the gradient could be artificially reduced or increased. For example, if the gore within the image is in the top left corner, we expect a positive response only in two of the instances, the full image, and the top left patch. This means that three other patches contain no gore. However, since the loss function does not specify where the

gore is located, equal importance will be given to all 5 instances. Thus, the weight being updated learns more from non-gore inputs than from gore inputs, reducing the odds it learns a useful feature over time.

*Table 15 - MIL Experimental Results for Gore Classification. Highlighted cells in gold represent best performance at the threshold of 0.5. Green coloured cells represent the best performance at the threshold which achieved the highest validation F1-Score. The recall and Double F1-Score being the most important.*

| Encoder | Loss function | Threshold | F1-Score | Double F1-Score | Precision | Recall | Uncertainty Estimation (Infer-dropout [$p = 0.5$]) | |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | Variance | Standard deviation |
| VGG16 | Weighted Recall Loss (0.30) | 0.5 | 0.8392 | 0.8865 | 0.8504 | 0.8611 | 0.0639 | 0.0974 |
| | | 0.4711 | 0.8392 | 0.8865 | 0.8504 | 0.8611 | 0.0655 | 0.0999 |
| VGG16 | Weighted Recall Loss (0.20) | 0.5 | 0.8244 | 0.8668 | 0.7732 | 0.9137 | - | - |
| | | 0.5813 | 0.8338 | 0.8809 | 0.8133 | 0.8864 | - | - |
| VGG16 | Binary Cross-entropy | 0.5 | 0.7814 | 0.8494 | 0.8031 | 0.8018 | - | - |
| | | 0.2805 | 0.8013 | 0.8572 | 0.7442 | 0.9029 | - | - |
| DenseNet121 | Weighted Recall Loss (0.30) | 0.5 | 0.8431 | 0.8866 | 0.7855 | 0.9535 | 0.0767 | 0.1426 |
| | | 0.5642 | 0.8446 | 0.8886 | 0.7915 | 0.9445 | 0.0785 | 0.1445 |
| DenseNet121 | Weighted Recall Loss (0.35) | 0.5 | 0.8422 | 0.8853 | 0.7888 | 0.9384 | 0.0785 | 0.1410 |
| | | 0.6930 | 0.8467 | 0.8887 | 0.8083 | 0.9293 | 0.0830 | 0.1539 |
| MobileNetV2 | Weighted Recall Loss (0.30) | 0.5 | 0.6305 | 0.7490 | 0.7094 | 0.6194 | - | - |
| | | 0.1451 | 0.6508 | 0.7516 | 0.6524 | 0.7327 | - | - |

### 5.2.8 Phase 8 – Further Uncertainty Experimentation

In previous experiments, we used uncertainty for model evaluation and the mean value for prediction. In this section, we will run two different experiments to further assess the value of estimating uncertainty. One aim is to see if the standard deviation obtained during uncertainty estimation can be used during inference to increase recall. Another aim is to determine if the increased processing cost of computing a distribution of the models' results, given a specific input image, leads to significantly better testing results.

#### *5.2.8.1 Using Uncertainty Standard Deviation to reduce prediction errors*

When exploring uncertainty previously, only the mean result was used when making predictions. However, the variance and standard deviation can be computed in this context as well. The latter could then be used to ensure that an example is put in the right class. In the case of gore classification, the worst outcomes are FNs. To reduce such cases, we could maximise the recall of the model by adding the standard deviation multiplied by a scalar, to the mean prediction. This is similar to lowering the threshold as was done previously. However, the threshold reduction is proportional to the model uncertainty for a given input. The results in Table 16 were gathered in two ways. In the first, the standard deviation scalar values marked with an asterix (*) were found using the same search techniques as the one used for the threshold values (see Section 5.1). In the second, the scalar values were found by manually searching scalar values to achieve comparable results to the ones obtained with threshold fine-tuning (note that in Table 16, rows with similar results are coloured using the same colour for readability).

The results in Table 16 show that for the most part the two methods of maximizing recall are equivalent. For example, when using a scalar of 4.0 on the Voting_WRL model, the threshold of 0.1852 has a better precision for the same recall. However, changing the scalar to 3.48 achieved an even better precision. A similar effect can be seen when lowering or raising the threshold. This effect also occurred on the Voting_Mixed model with the threshold of 0.3409 and the scalar values of 1.9 and 2.0.

*Table 16 - Testing results of voting models when using the standard deviation during inference. Rows with similar results are coloured using the same colour. Optimal thresholding is able to achieve better precision at higher recall values than when using the standard deviation at a fixed threshold (0.5).*

| Model name | Models used | Threshold | Standard deviation scalar | F1-Score | Double F1-Score | Precision | Recall |
|---|---|---|---|---|---|---|---|
| Voting_WRL (No threshold prior to voting) | VGG16 (WRL), MobileNetV2 (WRL), DenseNet121 | 0.5 | 0 | 0.8883 | 0.9187 | 0.8811 | 0.8956 |
| | | 0.5 | 1.6* | 0.8682 | 0.9016 | 0.8195 | 0.9231 |
| | | 0.5 | 2.5 | 0.8737 | 0.9047 | 0.8084 | 0.9505 |
| | | 0.5 | 3.2* | 0.8509 | 0.8855 | 0.7665 | 0.9560 |
| | | 0.5 | 3.48 | 0.8454 | 0.8805 | 0.7743 | 0.9615 |
| | | 0.5 | 4.0 | 0.8274 | 0.8650 | 0.7261 | 0.9615 |
| | | 0.5 | 6.0 | 0.8036 | 0.8421 | 0.6820 | 0.9780 |
| | | 0.6844 | 0 | 0.8782 | 0.9128 | 0.9064 | 0.8516 |
| | | 0.3929 | 0 | 0.8819 | 0.9125 | 0.8442 | 0.9231 |
| | | 0.1852 | 0 | 0.8537 | 0.8875 | 0.7675 | 0.9615 |
| Voting_Mixed (No threshold prior to voting) | VGG16 (WRL), MobileNetV2 (WRL), DenseNet121, VGG16 (BCE) | 0.5 | 0 | 0.8919 | 0.9210 | 0.8777 | 0.9066 |
| | | 0.5 | 1.7* | 0.8750 | 0.9070 | 0.8317 | 0.9231 |
| | | 0.5 | 1.9 | 0.8689 | 0.9019 | 0.8164 | 0.9286 |
| | | 0.5 | 2.0 | 0.8645 | 0.8983 | 0.8086 | 0.9286 |
| | | 0.5 | 3.3* | 0.8593 | 0.8925 | 0.7803 | 0.9560 |
| | | 0.5 | 6.0 | 0.7991 | 0.8396 | 0.6836 | 0.9615 |
| | | 0.4483 | 0 | 0.8824 | 0.9136 | 0.8594 | 0.9066 |
| | | 0.3409 | 0 | 0.8667 | 0.9001 | 0.8125 | 0.9286 |
| | | 0.1645 | 0 | 0.8441 | 0.8790 | 0.7489 | 0.9670 |

A drawback of the standard deviation method is that there seems to be a limit to the scalar where this method ceases to be useful. This is probably because most results are polarized to either end of the spectrum (around 0.0 or 1.0). By making the scalar too large, TNs results become FPs thus lowering the precision of the model. This can be seen when setting the scalar value at 6.0 for the Voting Mixed model in Table 16. For this scalar value, the model achieves a lower recall than the same model with a threshold of 0.1645. However, it also achieves a lower precision and, by extension, a lower F1-Score.

From this, we gather that using the uncertainty standard deviation to boost recall cannot achieve very high recall (like 99%) with the same level precision as threshold lowering. While combining both these methods could yield better results, it is still unclear if computing the model's uncertainty is worth it in comparison to inference, which is the subject of the next experiment.

#### *5.2.8.2 Uncertainty Inference Performance Analysis*

While uncertainty proved useful to assess the level of confidence different models have in their predictions, it is still unclear if it is useful during inference. To obtain the uncertainty metrics (mean and standard deviation), we need to pass the image through the model multiple times with a new random dropout layer values each time. We refer the number of passes through the model as the sample size. Doing so, comes at an increasing computing cost. This might be of value if it provides a significant performance gain. However, it is unlikely to be the case as uncertainty is most possibly normally distributed when using a sufficiently large sample size (at least 30 samples based on the Central Limit Theorem [195]).

In this experiment, a version of the best voting model has been created without the two inference dropout layers. This is the appropriate model to use for a sample size of 1 (no uncertainty), because we do not want the randomness of the dropout to affect the results. The voting models (with dropout) will also be used, but with different samples size to assess performance on the classification testing set. The prediction time for the whole sample set is captured to infer the cost of uncertainty computation. For samples sizes larger than 1, the distribution is computed one sample at a time. This means that the cost is directly related to the sample size. The use of batching will reduce the prediction time. Nonetheless, the prediction time will still be a linear function of the sample size. Furthermore, batching will prevent the paralysation of calls in the Image Processing Pipeline once deployed, by preventing the pipeline to provide multiple different images to the models at the same time.

*Table 17 - Performance analysis of Voting ensembles with different number of uncertainty samples. For a sample size of one, no random dropout is used to ensure no randomness in the results. Highlighted cells show the best results for the evaluation metrics for a given threshold and model.*

| Model name | Models used | Threshold | Number of samples | Prediction time | F1-Score | Double F1-Score | Precision | Recall |
|---|---|---|---|---|---|---|---|---|
| Voting_WRL (No threshold prior to voting) | VGG16 (WRL), MobileNetV2 (WRL), DenseNet121 | 0.5 | 1 | 0.0558 | 0.8889 | 0.9189 | 0.8770 | 0.9011 |
| | | | 10 | 0.4004 | 0.8889 | 0.9189 | 0.8770 | 0.9011 |
| | | | 30 | 1.2914 | 0.8859 | 0.9168 | 0.8763 | 0.8956 |
| | | | 100 | 4.4017 | 0.8913 | 0.9208 | 0.8817 | 0.9011 |
| | | 0.25 | 1 | 0.0538 | 0.8615 | 0.8952 | 0.7953 | 0.9396 |
| | | | 10 | 0.4197 | 0.8536 | 0.8884 | 0.7783 | 0.9451 |
| | | | 30 | 1.2888 | 0.8628 | 0.8958 | 0.7900 | 0.9505 |
| | | | 100 | 4.4453 | 0.8672 | 0.8993 | 0.7972 | 0.9505 |
| | | 0.05 | 1 | 0.0554 | 0.8337 | 0.8692 | 0.7265 | 0.9780 |
| | | | 10 | 0.4086 | 0.7834 | 0.8225 | 0.6509 | 0.9835 |
| | | | 30 | 1.2911 | 0.7920 | 0.8307 | 0.6630 | 0.9835 |
| | | | 100 | 4.4231 | 0.7920 | 0.8307 | 0.6630 | 0.9835 |
| Voting_Mixed (No threshold prior to voting) | VGG16 (WRL), MobileNetV2 (WRL), DenseNet121, VGG16 (BCE) | 0.5 | 1 | 0.0563 | 0.8943 | 0.9229 | 0.8824 | 0.9066 |
| | | | 10 | 0.4386 | 0.8841 | 0.9152 | 0.8677 | 0.9011 |
| | | | 30 | 1.3085 | 0.8865 | 0.9171 | 0.8723 | 0.9011 |
| | | | 100 | 4.3049 | 0.8895 | 0.9192 | 0.8730 | 0.9066 |
| | | 0.24 | 1 | 0.0538 | 0.8579 | 0.8920 | 0.7854 | 0.9451 |
| | | | 10 | 0.4302 | 0.8614 | 0.8942 | 0.7838 | 0.9560 |
| | | | 30 | 1.2772 | 0.8586 | 0.8922 | 0.7828 | 0.9505 |
| | | | 100 | 4.3855 | 0.8586 | 0.8922 | 0.7828 | 0.9505 |
| | | 0.03 | 1 | 0.0539 | 0.8174 | 0.8542 | 0.6992 | 0.9835 |
| | | | 10 | 0.4269 | 0.7427 | 0.7815 | 0.5967 | 0.9835 |
| | | | 30 | 1.2853 | 0.7627 | 0.8014 | 0.6207 | 0.9890 |
| | | | 100 | 4.3815 | 0.7601 | 0.7994 | 0.6194 | 0.9835 |

Table 17 shows that the prediction time is linearly related to the sample size. Interestingly, the Voting_Mixed model uncertainty estimation led to a performance decrease at a sample size of 100 when compared to the same model without uncertainty for two threshold values (0.5 and 0.03). For the other threshold value of this experiment (0.24), using the same model, we are unable to make any conclusions as the uncertainty results achieve higher recall at lower precision, following the expected precision recall curve. The results of Table 17 convey a different outcome for the Voting_WRL model. For two thresholds (0.5 and 0.25), the sample size of 100 achieved a performance increase when compared to the same model without uncertainty. At 0.5, the model produced a 0.47% precision increase at the same recall value. While at 0.25, the model with uncertainty showed a 0.19% precision increase and a 1.09% increase compared to the one without it. However, this slight performance improvement only occurred with a very large sample size (> 30) and thus comes at a heavy computational cost, making it impractical.

This experiment allowed us to reach two conclusions. The first is that the computational cost of uncertainty estimation outweighs the possible performance increase. The second is that the Voting_Mixed

model is more robust in its prediction than Voting_WRL because uncertainty estimation had less effect on this model's results This makes it the model with the least uncertainty (i.e., "more certain").

### 5.2.9 Comparison with Google Cloud's SafeSearch

During our research, the only publicly callable commercial solution for an Image classification task similar to gore detection was Google Cloud's Vision API SafeSearch [196]. The "medical" and "violence" categories defined by this API are related to gore as they frequently contain images with medical or violent content. Therefore, this API can provide a good comparison to the best gore classification models.

To gather results, the images from the gore classification are sent to the API and its response are saved [197]. The API returns a likelihood bucked for each image category [196]. These buckets have 6 different values (UNKNOWN, VERY_UNLIKELY, UNLIKELY, POSSIBLE, LIKELY and VERY_LIKELY) [196]. To use a numerical threshold, we converted these qualitative values into ones ranging from 0.0 to 1.0. As none of the responses contained the "unknown" value, 0.0 represents "very unlikely" and 1.0 represents "very likely". An image is said to be containing gore if the SafeSearch results of either the "medical" or "violence" categories are above a set threshold.

Table 18 shows the results of both the best gore classification model previously presented in this thesis and Google Cloud's SafeSearch results at varying thresholds. The latter was able to achieve a recall of 90.66% for detecting gore when the threshold is set to 0.5 ("Likely" likelihood and above). However, it does so with a much lower precision (56.90%) than the two voting ensembles (87.70 and 88.24%). Hence, we can state that the voting ensembles outperform Google Cloud's SafeSearch for gore classification.

*Table 18 - Google SafeSearch's Gore Classification model comparison on the testing set. Highlighted cells represent the best results on each metric with recall and Double F1-Score being the most important.*

| Model name | Threshold | F1-Score | Double F1-Score | Precision | Recall | Negative Precision | Negative Recall |
|---|---|---|---|---|---|---|---|
| Voting_WRL (No threshold prior to voting) | 0.5 | 0.8889 | 0.9189 | 0.8770 | 0.9011 | 0.9549 | 0.9431 |
| Voting_Mixed (No threshold prior to voting) | 0.5 | 0.8943 | 0.9229 | 0.8824 | 0.9066 | 0.9574 | 0.9455 |
| Google SafeSearch | 1.0 (Very Likely) | 0.6997 | 0.7877 | 0.7453 | 0.6593 | 0.8541 | 0.8985 |
| | 0.75 (Likely and above) | 0.7065 | 0.7766 | 0.6455 | 0.7802 | 0.8907 | 0.8069 |
| | 0.50 (Possible and above) | 0.6992 | 0.7481 | 0.5690 | 0.9066 | 0.9426 | 0.6906 |

### 5.2.10 Analysis of Voting Ensembles Results

The previous experimental results and analysis have shown that the voting ensembles are the best performing models, based on the metrics. In this section, we will describe the patterns we noticed while analysing the images classified. For this, we will consider FPs and FNs. The reviews also include the set of images used when training the cascading model dataset in 5.2.6.1. These images were not seen by the voting ensembles. This mean the images can be used to infer how the voting ensemble will perform on new images.

Looking at the FPs and FNs, it becomes clear that the model has a strong response to blood and skin. When it comes to wounds, the model seems to be more likely to classify it as gore if the wound includes blood or a darker region. Wounds covered with lighter substances, such as infected wounds, and blisters are sometimes not classified as gore when they should. Conversely, scabs and bruises are misclassified as gore seemingly because of their redness and contrast with the surrounding skin. This also translates to misclassification of red fabrics and objects when shown in a context with high contrast. The strong response to these features leads to many FPs with food related pictures (e.g., pies, pastries, fried foods, ketchup and hot dogs).

The detection of skin and wound also seems to work on darker skin tones. However, there is a very limited number of these images in the dataset, so definitive conclusions on this matter will require future study. There are also some FPs which seem to support this possibility, such as a Batman Beyond cosplay image where the red logo contrasts with the black full body suit, and an image of a black individual cosplaying with red face paint. There are also several images of charred meats which closely resembles burned skin or subcutaneous wounds on dark skin. A feature sometimes seen in wounds of individual with darker skin is a lighter region around the red wound. This might also be why certain images of cooked meat are classified as gore. However, the models tend to classify meat as gore due of its redness (pork or beef) and similarity to human muscles which can appear in certain types of injuries and wounds.

The models also have a strong response to holes. This is perhaps, because they closely resemble open wounds which the models have seen as part of their datasets. Certain examples of such FPs are when objects are stacked together leaving dark cavities in the structures such as a school of fish or a pile of apples. The strong response to holes does not translate to human skulls as the model seemed to have learned it not to be gore unless it shows abnormal structures or growths.

The models also seem to have some comprehension of human anatomy. This is because the models consider some injuries and wounds that deform the facial or body structure as gore. These features also lead to the misclassification of art pieces, especially when the goal of the piece is to be creepy. To get such a response from the viewers, these art pieces will sometimes deform, elongate the human anatomy which is likely to be detected by the models. Examples of this is the classification of images of the movie

representation of the character Venom[1]. However, because of the nature of classification, it is unclear if the models truly detect these elements or detect one of the previously mentioned patterns.

There are cases where the models' understanding of facial structures leads to misclassification. Such as when the facial features are changed by gravity (e.g., when upside down) or due to accumulation of fat. Future effort should be used to address these issues. Anatomical representations showing muscles, nerves or other subdermal structures are also classified as gore. The models do not seem to misclassify smiles or open mouths as gore, which means putting such images in the dataset seems to have worked.

The models perform better when the gore is in focus and of a somewhat large size. It thus performs better on medical images where the wounds are the focus of the image and take up a large portion of the frame. In cases where the gore is out of focus or very small, the models tend not to classify it correctly. An example of this is a frame of the movie "Jigsaw" where a body is in the foreground, but out of focus. Another is a frame of "Saving Private Ryan" showing the effect of a long-range sniper shot. This is, perhaps, because the filters that need to activate for the correct classification do not do so on pixelated or small regions.

### 5.2.11 Summary of Chapter 5

In this chapter, we systematically built a model to solve the problem of gore classification. We started in section 5.2.1 by searching for the evaluation metric for early stopping. Our objective was to find the metric that will yield the best precision-recall values with a particular emphasis on recall due to the nature of the application. We compared using the Best Loss, F1-score, Double F1-Score, and Weighted Double F1-Score on the validation set. We found that the Double F1-Score achieved higher testing precision (0.8109) and high testing recall (0.8956) as can be seen in Table 3. We interpreted that this meant that the precision recall curve of the model had a higher area under the curve and, as such, was a better overall choice.

To further determine the best training setup for gore classification, we tested different data augmentation setups and a multitude of loss functions to see which one leads to a better overall model. Table 4 shows that our data augmentation proved beneficial in training models for this task. Of all the loss functions tested, 2 of them (BCE and WRL) achieved the best results as shown in Table 5. Since their results were very close, we decided to fit 5 models with both functions to ensure the difference was not due to chance. These results are compiled in Table 6. They show that WRL achieved significantly better F1-Score, Double F1-Score and Recall with comparable precision. Hence, we decided to use this loss function for the rest of this chapter. This decision was revisited in section 5.2.5.2 where we found that BCE loss leads to slightly lower model uncertainty, but this difference is minimal. Moreover, we felt that this improvement did not outweigh the benefits of the WRL loss which produces higher recall, F1-Score and Double F1-Score.

---

[1] Example image can be seen on https://www.rollingstone.com/movies/movie-features/venom-everything-you-need-to-know-732533/ (Image courtesy of Sony Pictures).

Since we found an experimental setup that works well for gore classification, we shifted our focus to finding the best CNN encoder for the task. As we were using Tensorflow Keras to make the models, we had access to the pretrained models of this library [43]. During our experimentation, we found that larger models in terms of layers and parameters performed worse on the task, as it can be seen in Table 7. Hence, the larger models included in Tensorflow Keras were not tested [43]. From Table 5, we noticed that VGG16, DenseNet121 and MobileNetV2 encoders were all able to achieve 90% or above testing recall at 80% or more testing precision, which was the best performance of the tested encoders. VGG16 achieved a slightly higher testing Double F1-Score, but most probably not by a significant margin. Hence, we decided to use all these models when creating ensembles. Table 9 shows that the VGG16 is the least uncertain of these 3 models. This led us to conclude that the VGG architecture was best suited for gore classification on our testing dataset.

To further improve the testing results, we experimented with different ensemble types. The first type we tested is the cascading ensemble. We performed this investigation since we found that the VGG16 model trained WRL had higher uncertainty on non-gore images than the one trained with BCE. We built an ensemble model that uses the BCE prediction when the WRL model prediction is uncertain (see Equation (5.3)). This ensemble achieved a slightly higher testing F1-Score and Double F1-Score than its member models individually. It also had much higher uncertainty as it can be seen in Table 12.

We also experimented with Voting ensembles. In this ensemble type, the predictions of each model are combined in a weighted sum (or voting layer) that is learned using the training set. We trained and tested multiple permutations of the following 4 models: VGG16_WRL, VGG16_BCE, DenseNet121_WRL and MobileNetV2_WRL. In Table 13, we noticed that two of these trained models achieved the best testing results up to this point. Both models do not perform thresholding prior to voting. One of these models, Voting_Mixed, contains all four models while the other, Voting_WRL, contains all three WRL trained models. Voting_Mixed achieved the best testing F1-Score (0.8895) and Double F1-Score (0.9192) at a threshold of 0.5. Voting_WRL performed only slightly worse at the same threshold with an F1-Score (0.8889) and Double F1-Score (0.9189). Hence, further experiments will be needed to assess which of these two models performs best on this task.

Another set of experiments used the stacking ensembles. In these models, only the CNN encoders are kept from each model and a new head is added on top combining the extracted features from the encoders. The results in Table 14 show that this ensemble type did not work for gore classification as the models were unable to improve on the results of the cascading and voting ensembles. While not technically an ensemble, we tested a similar concept: MIL. Instead of stacking different encoders that are fed the same image, MIL models feed the same encoder multiple subregions of the image before feeding it to a single head using all the feature for classification. This experiment also did not yield improvements in testing results.

In our eigth experimental phase, we reviewed our use of uncertainty estimation. We started by seeing if we can use the standard deviation obtained through this analysis as an alternative to optimal thresholding.

This experiment worked to an extent. However, in order to obtain the high recall values we want for gore classification, the threshold needed to be changed as well using the uncertainty standard deviation method which makes it unnecessary. In the second experiment of this phase, we determined that the cost of computing the model's uncertainty is linear in terms of the number of sample we want to estimate the mean and standard deviation. Furthermore, we were only able to achieve a small performance increased in certain situations and with large sample sizes of 30 or more. Thus, we conclude that uncertainty estimation is not worth doing in this application.

Based on the experiments on gore classification up to this point, the two voting ensemble models, Voting_WRL and Voting_Mixed are the best performing models on the testing set. These models will thus be the only gore classification models used in further experiments. These include being included in an Image Processing Pipeline in Chapter 7. However, before doing so, we will try to censor the gore within the image by training ML models on the gore segmentation tasks.

# Chapter 6. Gore Segmentation

Segmentation models are quite useful when censoring content as they give the location of every pixel in the original image which needs to be censored in their output feature map. There is a wide variety of architectures that have shown success on a multitude of segmentation tasks. These architectures will be tested for gore segmentation. Implementations of state-of-the-art architectures tend to be custom-made, which can make it difficult to adapt to other applications or frameworks. To simplify the initial research, we searched for a package made for quick prototyping of segmentation models and found one used by Yakubovskiy [114] for that purpose. This package is compatible with the Keras and TensorFlow Keras APIs. It offers 4 different segmentation architectures (U-Net [113], Linknet [116], FPN [121] and Pyramid scene parsing network (PSPNet) [198]). It also has a few CNN backbones which are pretrained on ImageNet [114]. The architectures available date from a few years ago [113, 116, 121, 198]. However, they have been used for an extended period and have shown great success on a wide variety of segmentation tasks [199]. Moreover, Google Scholar search results shows that Ronneberger et al. [113] has been cited 29226 times, Lin et al. [121], 8108 times, and Chaurasia and Culurciello [116], 476 times. In this chapter, we will also attempt to improve the results obtained with the previously mentioned architectures by using GAN-like training - based on the Pix2Pix architecture - and optimal thresholding. We will also test the object detection and instance segmentation model Mask R-CNN as another possible way of solving the gore segmentation problem.

## 6.1 Training Methodology

IOU is the evaluation metric used to compare models, and for early stopping, as it is widely used in the literature [104, 99, 168, 167, 164, 200]. The other metrics that we will used to compare trained models are MSE, F1-Score and Double F1-Score. The F1-Score of the negative class is also tracked, although we expect good results for most models as it is a majority class. This is because most pixels in the images do not contain gore. entropy will also be used to estimate model uncertainty during training and initial testing, as presented in Jukna [201]. This is because dropout layers cannot be easily added into a pretrained model. Since entropy is maximal at 0.5 [202], the threshold for segmentation will be set at this value to simplify training and testing.

It is important to use Data Augmentation for this task as the dataset is much smaller. The use of augmentation should lessen the model's tendency to overfit to the images in the training set. Transfer Learning will also help alleviate overfitting as the model is trained for a limited number of epochs. However, as only the CNN backbones are pre-trained, the number of epochs need to be large enough for the rest of the model to train. The ImageDataGenerator class from TensorFlow cannot be used for a segmentation task as the feature maps are not rotated or flipped along with the image. Hence, we created our own image pipeline that use the TensorFlow image module [203] to change the image and the patches. The alterations that can happen on a given image are random hue, brightness, contrast, gamma, and saturation change. A maximum of two of these alterations can be performed on a given image. Figure 10 and Figure 11 show the upper and

lower bounds of the augmentation on an image. From these figures, it becomes clear that the gamma augmentation did not have the intended effect and should not have been used. The upper bound of contrast augmentation led to some atypical artifacts that occur when the value is set to more than 1.0. Thus, in hindsight, the value for contrast should have been limited to 1.0 not 1.5. Flips are also possible, and they are performed on both the image and the gore map to ensure the ground truths matches the image.

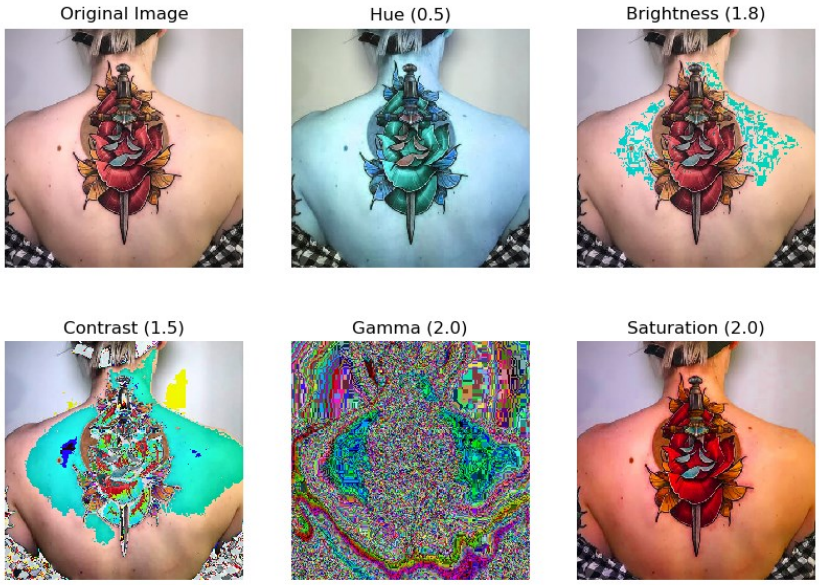


*Figure 10 - Segmentation Data Augmentation (Upper bound). This image shows the result of the 5 alteration types using the upper bound values of the random range used when making the augmentation during training. Image used in this figure [204] is in the public domain.*

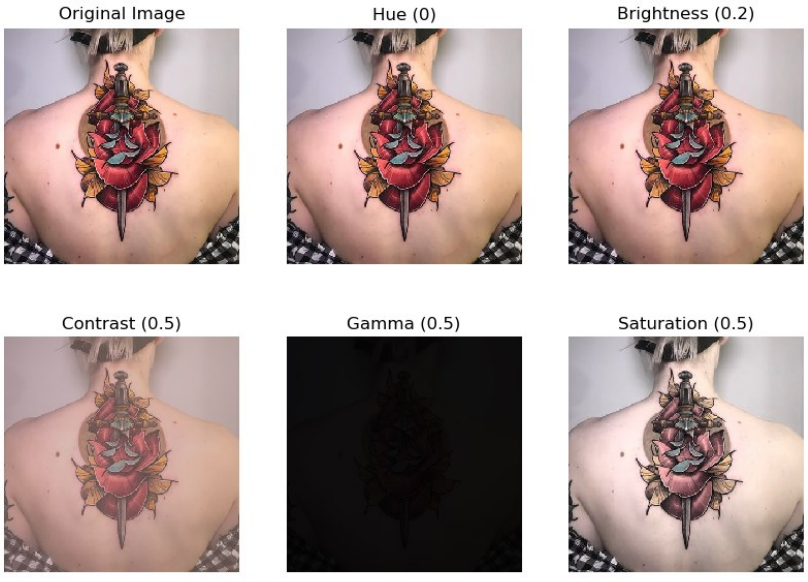


*Figure 11 - Segmentation Data Augmentation (Lower bound). This image shows the result of the 5 alteration types using the lower bound values of the random range used when making the augmentation during training. Image used in this figure [204] is in the public domain.*

## 6.2 Experimental Results

Using the same approach as the one presented in Chapter 5, we aim to systematically train the best model for the gore segmentation task. In the first 4 experimental phases, we will train a traditional image segmentation model to find the loss functions, CNN encoders and architectures best suited for the task. The fifth and sixth phases will introduce new paradigms, used successfully for pornographic image censoring and medical image segmentation, and test them for gore segmentation. This will be performed to determine if these paradigms can further improve the testing results achieved in the previous phases. Phase 5 will use a GAN-like training setup with a new discriminator and segmentation model acting as generator. The object detection and instance segmentation architecture Mask R-CNN will be investigated in Phase 6. In the seventh phase, we will use optimal thresholding for gore segmentation as we did in Chapter 4 for gore classification. Finally, we will perform an ablation study of the experimental setup in the eight phase to ensure the segmentation model was optimally trained.

### 6.2.1 Phase 1 – Loss function experimentation

The goal of this phase is to find the best loss function to train the gore segmentation models. The testing setup uses a U-Net model architecture along with a pretrained ResNet34 CNN backbone. The initial set of loss functions tested were the ones presented in the article by Nieradzik [109]. These functions include BCE loss, Binary FL, Dice Loss, and its generalization Tversky Loss. Additionally, IoU loss from [104, 99] is also tested. The test results, as seen in Table 19, show that the model trained with Binary FL achieved the lowest MSE. Meanwhile, the one trained using IoU Loss model achieved the best IoU, F1-Score and Double F1-Score test results. The IoU Loss model was therefore selected as the to be use in the remaining experiments.

*Table 19 - Comparison of the testing performance of Loss Functions for Gore Segmentation on U-Net Architecture. Highlighted cells show the best performance on each metric.*

| Loss Function | MSE | IoU | F1-Score | Double F1-Score |
|---|---|---|---|---|
| Binary Cross-entropy | 0.0470 | 0.3361 | 0.4035 | 0.6805 |
| Binary Focal Loss | 0.0401 | 0.4689 | 0.5279 | 0.7480 |
| Dice Loss | 0.0963 | 0.2520 | 0.3232 | 0.6318 |
| Tversky Loss (Beta = 0.66) | 0.0693 | 0.3816 | 0.3816 | 0.6699 |
| IoU Loss | 0.0454 | 0.5065 | 0.5636 | 0.7670 |

### 6.2.2 Phase 2 – CNN encoder experimentation for U-Net

In this phase, we assess the effect of depth in the CNN backbone on the gore segmentation results. To do so, a U-Net segmentation model architecture is used. This model is trained using IoU Loss; the validation set`s IoU metric is also used for early stopping. The models are trained for 400 to 500 epochs, but the training

stops if the best IoU score on the validation set has not improved after 100 epochs. This last hyperparameter is known as the model`s patience in the Keras framework [187].

*Table 20 - Comparison of the testing performance of CNN Encoders for Gore Segmentation on U-Net Architecture. Highlighted cells show the best performance on each metric. When two cells are highlighted for the same metric, it is because the difference is too small to tell the difference between both model ($< 5 * 10^{-4}$).*

| CNN Encoder | Number of Parameters | Training epochs | MSE | IOU | F1-Score | Negative F1-Score | Double F1-Score |
|---|---|---|---|---|---|---|---|
| ResNet34 | 24,456,154 | 500 | 0.0454 | 0.5065 | 0.5636 | 0.9703 | 0.7670 |
| ResNet50 | 32,561,114 | 400 | 0.0448 | 0.4784 | 0.5470 | 0.9692 | 0.7581 |
| ResNet101 | 51,605,466 | 400 | 0.0479 | 0.4881 | 0.5576 | 0.9693 | 0.7634 |
| ResNet152 | 67,295,194 | 200 | 0.0416 | 0.4975 | 0.5633 | 0.9708 | 0.7671 |

Table 20 shows that the two best performing models on the testing set, ResNet34 and ResNet152, have near identical performance on the F1-Score, Negative F1-Score and Double F1-Score metrics. ResNet34 has a better testing IoU value compared to ResNet152, but the latter beats it in terms of MSE. Since the ResNet34 model was trained for 2.5 times more epochs, we presume that ResNet152 will achieve better performance if it was trained for the same number of epochs. This is because, apart from ResNet34 which was trained for longer, the ResNet models' performance in Table 20 otherwise positively correlates with the number of parameters. Hence, ResNet152 will be used as our encoder of choice in subsequent phases.

### 6.2.3 Phase 3 – Segmentation model architecture experimentation

In the previous phases, the segmentation model architecture used was always U-Net. It is an architecture that performs well on Binary segmentation tasks which made it a good choice to start with. Moreover, settling on an architecture allowed us to see the effects of changes to other facets of the model, and how they affect the resulting predictions. However, it is still unclear if U-Net is the best choice as an architecture for gore segmentation. The goal of this phase it to make that determination.

Based on the results of previous phases, the model trained in the current phase will use a ResNet152 CNN backbone pretrained on ImageNet using IoU loss. The architectures that will be tested were those implemented by Yakubovskiy [114]. These include U-Net, Linknet and FPN. PSPNet is also included in Yakubovskiy [114], but this model architecture is incompatible with the image input size which is set to 224 by 224 pixels [204]. This input size is the one most ImageNet pretrained CNN encoder use, including ResNet152 [205]. Hence, we decided not to train models using the PSPNet architecture. The FPN architecture has a hyperparameter that allows us to change the number of channels in each level of the pyramid representation. Therefore, multiple models have been trained with a different value for this hyperparameter to find the one that is best suited for gore classification.

Table 21 presents the testing results achieved when training models with the different architectures. The encoder-decoder models, U-Net and Linknet, performed worse than most FPN networks. U-Net and Linknet did not achieve an IoU percentage higher than 50%. These models also seemed to have some difficulty with identifying gore as the testing F1-Score is 56.33% for U-Net and 57.93% for Linknet. For the FPN architecture, there seems to be too few (or too many) channels in the pyramid representation. The models with 12 and 64 channels performed worse than U-Net and Linknet on every metric. The model with 32 channels achieved the lowest testing Negative F1-Score in Table 21 and was only marginally better than the 24-channel model (by 0.02%). This model achieved the best testing values on all the other tracked metrics of Table 21. We will thus use the FPN architecture with 24 channels going forward.

*Table 21 - Comparison of the testing performance of segmentation Model Architectures for Gore Segmentation. Highlighted cells show the best performance on each metric. When two cells are highlighted for the same metric, it is because the difference is too small to tell the difference between both model ($< 5 * 10^{-4}$).*

| Model Architecture | Number of Parameters | MSE | IOU | F1-Score | Negative F1-Score | Double F1-Score |
|---|---|---|---|---|---|---|
| U-Net | 67,295,194 | 0.0416 | 0.4975 | 0.5633 | 0.9708 | 0.7671 |
| Linknet | 63,517,466 | 0.0456 | 0.5105 | 0.5793 | 0.9693 | 0.7743 |
| FPN (12 pyramid ch) | 58,332,500 | 0.0590 | 0.4436 | 0.5108 | 0.9655 | 0.7382 |
| FPN (16 pyramid ch) | 58,352,258 | 0.0421 | 0.5336 | 0.6025 | 0.9702 | 0.7864 |
| FPN (24 pyramid ch) | 58,395,230 | 0.0393 | 0.5539 | 0.6179 | 0.9724 | 0.7952 |
| FPN (32 pyramid ch) | 58,442,810 | 0.0412 | 0.5366 | 0.5977 | 0.9722 | 0.7850 |
| FPN (64 pyramid ch) | 58,679,210 | 0.0457 | 0.4926 | 0.5550 | 0.9686 | 0.7618 |

### 6.2.4 Phase 4 – CNN encoder experimentation for FPN24

Up until now, the CNN backbones used were all from the ResNet family of models from He et al. [45]. While widely used, there is no empirical data to suggest that this model family is the most appropriate for gore segmentation. Hence, more backbones should be tested. As part of Yakubovskiy's package [114], pretrained CNN encoders are ready to be used with the four architectures included. This is particularly useful as the best layers at each size are already identified and are automatically used in the decoders or pyramid representations. The CNNs backbones included in this package are ResNet with "Squeeze-and-excitation" blocks [57], ResNext [54] (both with and without SE blocks [57]), Inception [41], DenseNet [51] and EfficientNet [206] models.

In Table 22, we notice that the ResNet152 and SE-ResNext101 models both achieved nearly equal MSE scores, which were the best of the models tested. The ResNet152 and DenseNet169 models achieved very similar leading results on all other metrics (IoU, F1-Score, Negative F1-Score, Double F1-Score). The SE-

Resnet152 and EfficientNetB7 also performed well on these metrics, achieving an IoU around 2% lower than ResNet152 and DenseNet169. These two models Double F1-Score is also around 1.5% lower than the latter models. Since we do not yet know how well the gore segmentation models will work with gore classification in the Image Processing pipeline, we decided to keep assessing these 4 models. The Image Pipeline analysis will be discussed in section 7.1.

*Table 22 - Comparison of the testing performance of CNN Encoders for Gore Segmentation on FPN24 Architecture. Highlighted cells show the best performance on each metric. When two cells are highlighted for the same metric, it is because the difference is too small to tell the difference between both model ($< 5 * 10^{-4}$).*

| CNN Encoder | Number of Parameters | MSE | IoU | F1-Score | Negative F1-Score | Double F1-Score |
|---|---|---|---|---|---|---|
| ResNet152 | 58,395,230 | 0.0393 | 0.5539 | 0.6179 | 0.9724 | 0.7952 |
| SE-ResNet152 | 58,395,230 | 0.0437 | 0.5327 | 0.5933 | 0.9697 | 0.7815 |
| ResNext101 | 42,381,662 | 0.0402 | 0.4897 | 0.5572 | 0.9710 | 0.7641 |
| SE-ResNext101 | 47,159,365 | 0.0394 | 0.5136 | 0.5756 | 0.9730 | 0.7743 |
| InceptionV3 | 21,904,821 | 0.0469 | 0.4708 | 0.5366 | 0.9658 | 0.7512 |
| EfficientNetB7 | 64,235,813 | 0.0423 | 0.5320 | 0.5942 | 0.9691 | 0.7816 |
| EfficientNetB6 | 41,086,365 | 0.0447 | 0.5223 | 0.5872 | 0.9674 | 0.7773 |
| DenseNet169 | 12,730,325 | 0.0403 | 0.5553 | 0.6175 | 0.9717 | 0.7946 |
| DenseNet201 | 18,421,717 | 0.0495 | 0.4824 | 0.5402 | 0.9666 | 0.7534 |
| InceptionResNetV2 | 54,434,933 | 0.0440 | 0.4907 | 0.5549 | 0.9705 | 0.7627 |

### 6.2.5 Phase 5 – Experimentation with GANs

The use of GANs in pornographic image detection by More et al. [20] and Simoes et al. [160] suggested it could be used for gore as well. More et al. [20] defines the task as an image translation from one domain of images (X) to another (Y). Both More et al. [20] and Simoes et al. [160] use a cyclical generator architecture. This means that the network trains two generator networks: one to go from X to Y and one to go from Y to X. The discriminators are then used to differentiate the images that are from the domain (either X or Y) from images translated using the generator. Simoes et al. [160] only uses one discriminator to differentiate censored nude images from images of models wearing bikinis. The other discriminator is replaced with the difference between the original image and the one that passed through both generators.

Looking at the results of different GANs on different applications such as the ones presented in Brownlee's article [207], we noticed that these networks can sometimes create unwanted artifacts in the generated image. Since the censored images are meant to be used by humans as part of their work, we decided that the generator should output a segmentation map instead of a censored image. Because of this decision,

a cyclical training setup such as the one presented in More et al. [20] and Simoes et al. [160] is not appropriate. Thus, to train a segmentation GAN, we decided to use the Pix2Pix approach [208]. This approach is known as a conditional adversarial training as it blends supervised learning and adversarial training. Consequently, during training, the model receives a pair (X, Y) where X is the input image and Y, the expected results for that image. For the segmentation map generator, we can reuse the best segmentation architecture from the previous experiments which was the EfficientNetB7 FPN24 network. The discriminator architecture is the one presented in Pix2Pix, PatchGAN, [208]. To simplify the implementation, we used the one presented in the following TensorFlow tutorial [209].

In the Pix2Pix architecture, the generator learns using a weighted loss function. The first part of the loss function is discriminator loss. In this function, the loss is 1 if the generated image is classified as 'fake' by the discriminator and 0 otherwise. The second part is a loss function calculated using the expected results and the generated image. Hence, we can reuse the best loss function of the previous experiments: IoU loss. In this experiment, we can view the discriminator loss as a regularization component for IoU loss. By doing so, we believe that this regularization could lead the segmentation out of a potential local minimum, which will improve the results.

In this first part of this experiment, the generator's encoder was pretrained using the ImageNet weights (as before for segmentation). This was performed to determine the best loss function weighing parameter, lambda, to balance the discriminator loss and the IoU loss. The model was trained for 300 epochs, the same number of epochs as Phase 4. However, the training was done in three increments of 100 epochs as the training program will stop on its own. This experiment allows us to assess if the generative training setup could be beneficial on its own for gore Censoring. Because of the high level of computing resources required for this experiment, we decided to only use one CNN encoder. We chose the one which performed the best as part of the Image Processing Pipeline: EfficientNetB7 (See Table 30 and Table 31). The results of the GAN experiment are shown in Table 23. This table shows that smaller values of lambda (10 and 50) lead to a larger value for IoU and Double F1-Score. The maximal values were achieved at lambda equal to 50; this model getting a testing IoU value of 38.48% and a testing Double F1-Score of 71.12%. Interestingly, these testing values are much lower than the ones reached in Phase 4 by the EfficientNetB7 FPN24 model trained with only IoU loss.

In the second part of the experiment, the best model from section 6.2.4 is used as the starting point. It is fine-tuned using the weighted discriminator IoU loss. This is to determine if the addition of a discriminator loss could be used to move the trained model out of a possible local minimum and converge to a better and, potentially, global minimum. The training in this part will be shorter and with a smaller learning rate to ensure that as little information as possible is lost from the initial model at each epoch. Otherwise, the results will be similar to Part 1, which were not ideal. During this training, the first 20 epochs are performed with the generator's weights frozen to train the discriminator. During the next 80 epochs, the generator and discriminator are trained simultaneously.

*Table 23 – Testing results of Segmentation Pix2Pix GAN Experimentation's models using EfficientNetB7 FPN24 Generator with ImageNet Weights. Highlighted cells show the best performance on each metric.*

| Lambda | Learning rate | MSE | IOU | F1-Score | Negative F1-Score | Double F1-Score |
|---|---|---|---|---|---|---|
| 100 | $1e^{-5}$ | 0.0460 | 0.3473 | 0.4163 | 0.9689 | 0.6926 |
| 100 | $1e^{-6}$ | 0.0499 | 0.2780 | 0.3529 | 0.9639 | 0.6584 |
| 10 | $5e^{-6}$ | 0.0439 | 0.3792 | 0.4481 | 0.9697 | 0.7089 |
| 50 | $5e^{-6}$ | 0.0462 | 0.3848 | 0.4541 | 0.9683 | 0.7112 |
| 100 | $5e^{-6}$ | 0.0499 | 0.3655 | 0.4293 | 0.9668 | 0.6980 |
| 1000 | $5e^{-6}$ | 0.0490 | 0.2779 | 0.3514 | 0.9662 | 0.6588 |

*Table 24 - Testing results of Segmentation Pix2Pix GAN Experimentation's models using pretrained EfficientNetB7 FPN24 Generator. Highlighted cells in gold show the best performance on each metric. The IoU values highlighted in blue are when the GAN fine-tuned model improved its performance due to the fine-tuning.*

| Name | Lambda | Learning rate | MSE | IOU | F1-Score | Negative F1-Score | Double F1-Score |
|---|---|---|---|---|---|---|---|
| Run 1 | 100 | $1e^{-7}$ | 0.0440 | 0.5631 | 0.6242 | 0.9702 | 0.7972 |
| Run 2 | 100 | $1e^{-7}$ | 0.5675 | 0.0618 | 0.1014 | 0.4663 | 0.2838 |
| Run 3 | 100 | $1e^{-7}$ | 0.0449 | 0.5625 | 0.6220 | 0.9698 | 0.7959 |
| Run 4 | 100 | $1e^{-7}$ | 0.0441 | 0.5675 | 0.6286 | 0.9702 | 0.7994 |
| Run 5 | 100 | $1e^{-7}$ | 0.0439 | 0.5068 | 0.5712 | 0.9699 | 0.7706 |
| Run 6 | 100 | $1e^{-7}$ | 0.0473 | 0.5165 | 0.5774 | 0.9684 | 0.7729 |
| Run 1 | 100 | $1e^{-8}$ | 0.0485 | 0.5220 | 0.5795 | 0.9679 | 0.7737 |
| Run 2 | 100 | $1e^{-8}$ | 0.0477 | 0.5306 | 0.5889 | 0.9682 | 0.7786 |
| Run 1 | 50 | $1e^{-7}$ | 0.0447 | 0.5643 | 0.6244 | 0.9699 | 0.7971 |
| Run 2 | 50 | $1e^{-7}$ | 0.0435 | 0.5017 | 0.5675 | 0.9699 | 0.7687 |
| Run 3 | 50 | $1e^{-7}$ | 0.0501 | 0.5268 | 0.5791 | 0.9673 | 0.7732 |

In Table 24, we see that only 4 of the 11 GAN fine-tuning runs were able to improve the starting model. Using a lambda value of 100 and learning rate of $1e^{-7}$ was the best setup, as 3 of the 6 runs led to an improvement. The 4$^{th}$ run using this setup led to the best improvement, achieving a testing IoU score of 56.75% and Double F1-Score of 79.94%. The resulting model is the best performing model to date. Its

performance surpasses that of the models from Table 22. Accordingly, we conclude that using GAN fine-tuning improves gore segmentation testing results and, by extension, the model itself for the given task.

Training using GAN seems to be heavily affected by the discriminator's quality. This is highlighted by the stark difference between runs 2 and 4 with a lambda value of 100 and learning rate of $1e^{-7}$. The former model achieved the worse testing IoU and Double F1-Score of any of the trained models, while the latter achieved the best values. The most likely cause for this is the set and the order of training images on which the discriminator was initially trained. If the discriminator weight started to converge towards the minimum, it will challenge the generator and lead to learning improvement. However, if it were not able to do so, the results could negatively affect the generator when the weights are unfrozen.

### 6.2.6 Phase 6 – Mask R-CNN

In this phase, the Object Detection model Mask R-CNN is compared to the previously trained segmentation models. This is possible because Mask R-CNN outputs a segmentation mask for the identified object along with the traditional output of the detection model, which includes the class of the object and its bounding box. A Tensorflow 2.0 port of the Matterport Tensorflow implementation [210] was used. This implementation gives two choices of base models: ResNet50 and ResNet101. Both are tested in this phase.

The results from Table 25 show that Mask R-CNN is heavily influenced by changes in the learning rate. The learning rate of $1e^{-4}$ worked best during training. Another important hyperparameter is the number of regions of interest (ROIs) generated by the model during training. Setting it to 64 gave the best results.

The best training results were achieved with the ResNet50 base model. This is probably because the batch size could be set higher with this model during training. The ResNet50 base model achieved a testing IOU score of 54.03% which is better than the non-GAN refined EfficientNetB7 FPN24 model (53.20%). However, it achieved inferior results compared to the ResNet152 and DenseNet169 models from Table 22. Thus, it seems like a viable alternative to train a gore segmentation model. However, it requires a longer training time (approximately 10 to 12 hours per model on a personal computer with a GPU with 8 GB of VRAM, as opposed to the 4 hours required to train models in previous phases). Considering that the additional training time was incurred on a smaller dataset, we conclude that this model is not viable option for the purpose of this thesis.

*Table 25 – Testing results of Mask R-CNN Gore Segmentation Models. Highlighted cells show the best performance on each metric. When two cells are highlighted for the same metric, it is because the difference is too small to tell the difference between both model ($< 5 * 10^{-4}$). The asterisk in this table denotes a model which the training was restarted after the initial 600 epochs were over.*

| Base Model | Training Epochs | Learning rate | Weight decay | Training Region of Interests (ROIs) | MSE | IOU | F1-Score | Negative F1-Score | Double F1-Score |
|---|---|---|---|---|---|---|---|---|---|
| ResNet50 | 400 | $1e^{-4}$ | $1e^{-4}$ | 92 | 0.0580 | 0.4894 | 0.5651 | 0.9607 | 0.7629 |
| ResNet50 | 300 | $1e^{-3}$ | $1e^{-3}$ | 64 | 0.0580 | 0.4911 | 0.5619 | 0.9613 | 0.7616 |
| ResNet50 | 300 | $1e^{-5}$ | $1e^{-3}$ | 64 | 0.0693 | 0.4649 | 0.5404 | 0.9543 | 0.7474 |
| ResNet50 | 300 | $1e^{-4}$ | $1e^{-3}$ | 64 | 0.0564 | 0.5291 | 0.6057 | 0.9617 | 0.7837 |
| ResNet50 | 600 | $1e^{-4}$ | $1e^{-3}$ | 64 | 0.0587 | 0.5391 | 0.6169 | 0.9602 | 0.7886 |
| ResNet50 | 624* | 1e-4 (1e-6 from restart) | $1e^{-3}$ | 64 | 0.0583 | 0.5403 | 0.6185 | 0.9604 | 0.7895 |
| ResNet50 | 624* | 1e-4 (1e-7 from restart) | $1e^{-3}$ | 64 | 0.0583 | 0.5403 | 0.6184 | 0.9604 | 0.7894 |
| ResNet101 | 300 | $1e^{-4}$ | $1e^{-4}$ | 92 | 0.1026 | 0.1472 | 0.1675 | 0.9371 | 0.5523 |
| ResNet101 | 300 | $1e^{-5}$ | $1e^{-4}$ | 64 | 0.0806 | 0.3090 | 0.3901 | 0.9483 | 0.6692 |
| ResNet101 | 300 | $1e^{-4}$ | $1e^{-3}$ | 64 | 0.1043 | 0.0999 | 0.1149 | 0.9362 | 0.5255 |
| ResNet101 | 300 | $1e^{-3}$ | $1e^{-3}$ | 64 | 0.1043 | 0.0999 | 0.1149 | 0.9362 | 0.5255 |
| ResNet101 | 300 | $1e^{-5}$ | $1e^{-3}$ | 64 | 0.0648 | 0.4620 | 0.5391 | 0.9570 | 0.7480 |

### 6.2.7 Phase 7 – Optimal Thresholding

In all previous phases, the segmentation map was thresholded at 0.5. However, this value is arbitrary and may not yield the best results. Therefore, just as for classification, the validation set is used to find the threshold that optimizes a given metric. For segmentation, we want to primarily optimize the IoU score. If we obtain the same IoU score (for up to 3 decimal places) for two thresholds, then Double F1-Score is used to break the tie. Table 26 shows the results of the optimal thresholding on the models from Phase 5. We notice that the optimal threshold based on the validation set did not translate to better testing results on all models. Accordingly, the threshold value of 0.5 is still a good choice for gore segmentation.

*Table 26 – Effect of validation Optimal Thresholding for Gore Segmentation on the testing results. Highlighted cells show the best performance on each metric.*

| Name | Lambda | Learning rate | MSE | Threshold | IOU | F1-Score | Negative F1-Score | Double F1-Score |
|---|---|---|---|---|---|---|---|---|
| FPN24 EfficientNetB7 (Phase 4 Best Model Retest) | - | $5e^{-6}$ | 0.0423 | 0.50 | 0.5320 | 0.5942 | 0.9691 | 0.7816 |
| | | | | 0.45 | 0.5320 | 0.5943 | 0.9690 | 0.7817 |
| FPN24 EfficientNetB7 GAN Lambda 100, $1e^{-7}$ LR (Run 1) | 100 | $1e^{-7}$ | 0.0440 | 0.50 | 0.5631 | 0.6242 | 0.9702 | 0.7972 |
| | | | | 0.90 | 0.5597 | 0.6188 | 0.9694 | 0.7946 |
| FPN24 EfficientNetB7 GAN Lambda 100, $1e^{-7}$ LR (Run 3) | 100 | $1e^{-7}$ | 0.0449 | 0.50 | 0.5625 | 0.6220 | 0.9698 | 0.7959 |
| | | | | 0.60 | 0.5591 | 0.6180 | 0.9697 | 0.7940 |
| FPN24 EfficientNetB7 GAN Lambda 100, $1e^{-7}$ LR (Run 4) | 100 | $1e^{-7}$ | 0.0441 | 0.50 | 0.5675 | 0.6286 | 0.9702 | 0.7994 |
| | | | | 0.15 | 0.5630 | 0.6223 | 0.9696 | 0.7963 |
| FPN24 EfficientNetB7 GAN Lambda 50, $1e^{-7}$ LR (Run 1) | 50 | $1e^{-7}$ | 0.0447 | 0.50 | 0.5643 | 0.6244 | 0.9699 | 0.7971 |
| | | | | 0.90 | 0.5626 | 0.6254 | 0.9708 | 0.7972 |

### 6.2.8 Phase 8 - Ablation study

When training the gore segmentation model, we used Data Augmentation and Image Patching to attempt to maximize performance with our limited dataset. However, these techniques were not tested to verify whether they have the desired effect on the end models. Accordingly, in this section, to assess the effectiveness of the pre-processing steps, the FPN24 EfficientNetB7 models are trained with and without them. All models are trained with the same learning rate ($5e^{-6}$) for the same number of epochs (400). The number of training images/patches per epoch is 2000 when patching is done and 200 without. This is to avoid overfitting the models without patching.

Table 27 shows the results of the training with different variations of training preprocessing. The importance of patching during training is evident as the models trained without it have an IOU (0.4218 and 0.4558) that is significantly lower than the ones with it (0.5320, 0.5377, 0.5242 and 0.5412). The usefulness of Data Augmentation is not clear from Table 27 as the second model trained without it achieved the best IOU and Double F1-Score for a model trained from scratch. However, these results did not translate into fine-tuning the model with GANs as it can be seen in Table 28. The 5 fine-tuning runs only worsened the results of the model. This contrasts with the models with Data Augmentation which produced significant improvement using the same fine-tuning technique in 3 of the 6 runs with the same hyperparameters (see Table 24 and Table 26). This shows that Data Augmentation is useful to converge to a more robust minimum which allows for further fine-tuning.

*Table 27 – Testing results of models of our Gore Segmentation Ablation Study of our Training setup. Highlighted cells show the best performance on each metric.*

| Name | Image Patching | Data Augmentation | MSE | Threshold | IOU | F1-Score | Negative F1-Score | Double F1-Score |
|---|---|---|---|---|---|---|---|---|
| FPN24 EfficientNetB7 (Table 22) | Yes | Yes | 0.0423 | 0.50 | 0.5320 | 0.5942 | 0.9691 | 0.7816 |
| | | | | 0.45 | 0.5320 | 0.5943 | 0.9690 | 0.7817 |
| FPN24 EfficientNetB7 NoAug 1 | Yes | No | 0.0453 | 0.50 | 0.5046 | 0.5726 | 0.9674 | 0.7700 |
| | | | | 0.95 | 0.5242 | 0.5909 | 0.9690 | 0.7779 |
| FPN24 EfficientNetB7 NoAug 2 | Yes | No | 0.0417 | 0.50 | 0.5410 | 0.6039 | 0.9700 | 0.7870 |
| | | | | 0.95 | 0.5412 | 0.6047 | 0.9704 | 0.7869 |
| FPN24 EfficientNetB7 No Patch | No | Yes | 0.0419 | 0.50 | 0.3910 | 0.4633 | 0.9686 | 0.7160 |
| | | | | 0.85 | 0.4218 | 0.4906 | 0.9675 | 0.7293 |
| FPN24 EfficientNetB7 No Aug and Patch | No | No | 0.0423 | 0.50 | 0.3909 | 0.4648 | 0.9682 | 0.7165 |
| | | | | 0.95 | 0.4558 | 0.5230 | 0.9668 | 0.7291 |

*Table 28 - Testing results of Fine-Tuning Augmentation-less models with Pix2Pix GAN. Highlighted cells show the best performance on each metric.*

| Name | Lambda | Learning rate | MSE | Threshold | IOU | F1-Score | Negative F1-Score | Double F1-Score |
|---|---|---|---|---|---|---|---|---|
| FPN24 EfficientNetB7 NoAug 2 (Starting model) | - | $5e^{-6}$ | 0.0417 | 0.50 | 0.5410 | 0.6039 | 0.9700 | 0.7870 |
| | | | | 0.95 | 0.5412 | 0.6047 | 0.9704 | 0.7869 |
| FPN24 EfficientNetB7 GAN Lambda 100, $1e^{-7}$ LR (Run 1) | 100 | $1e^{-7}$ | 0.0428 | 0.50 | 0.5064 | 0.5703 | 0.9705 | 0.7704 |
| | | | | 0.90 | 0.5113 | 0.5740 | 0.9702 | 0.7722 |
| FPN24 EfficientNetB7 GAN Lambda 100, $1e^{-7}$ LR (Run 2) | 100 | $1e^{-7}$ | 0.0429 | 0.50 | 0.5089 | 0.5725 | 0.9705 | 0.7715 |
| | | | | 0.80 | 0.5100 | 0.5727 | 0.9703 | 0.7716 |
| FPN24 EfficientNetB7 GAN Lambda 100, $1e^{-7}$ LR (Run 3) | 100 | $1e^{-7}$ | 0.0414 | 0.50 | 0.4638 | 0.5298 | 0.9709 | 0.7504 |
| | | | | 0.95 | 0.4915 | 0.5560 | 0.9709 | 0.7630 |
| FPN24 EfficientNetB7 GAN Lambda 100, $1e^{-7}$ LR (Run 4) | 100 | $1e^{-7}$ | 0.0432 | 0.50 | 0.5051 | 0.5681 | 0.9703 | 0.7692 |
| | | | | 0.70 | 0.5051 | 0.5677 | 0.9702 | 0.7690 |
| FPN24 EfficientNetB7 GAN Lambda 100, $1e^{-7}$ LR (Run 5) | 100 | $1e^{-7}$ | 0.0411 | 0.50 | 0.4725 | 0.5417 | 0.9709 | 0.7563 |
| | | | | 0.85 | 0.4958 | 0.5644 | 0.9708 | 0.7675 |

### 6.2.9 Summary of Chapter 6

In this chapter, we trained and tested a multitude of models to identify the one best suited for gore segmentation. Since this is a novel task, we needed to find the segmentation loss function that leads to the best testing IoU metric. Through experimentation, we found that IoU loss, a loss function derived from the IoU formula, yielded the best results using a U-Net architecture and ResNet34 encoder (see Table 19). We then tested pretrained encoders from the ResNet family and found that ResNet152 (the deepest in terms of number of layers and largest in terms of parameters) led to the best results (see Table 20).

In the third experimental phase, we trained models using different segmentation model architectures included in the "segmentation Models" Python package by Yakubovskiy [114]. The architectures used were the package's implementation of U-Net, Linknet and FPN [114]. All models used the ResNet152 as the CNN backbone and were trained with IoU loss function. The results, shown in Table 21, demonstrated that the FPN architecture with 24 channels per level in the pyramid achieved the best results for gore segmentation.

Once the segmentation architecture was set, we then trained and tested models using a wider variety of CNN encoders or backbones. We found that the ResNet152, DenseNet169, SE-ResNet152 and EfficientNetB7 ImageNet pretrained encoders achieved the best testing results on the tasks (see Table 22). Because the performance gap between these models is relatively close, we decided to test all four of these models in the Image Processing pipeline (the subject of the next chapter). Testing these models in that context, we noticed the EfficientNetB7 encoder led to the best results. This model achieved a testing IoU value of 53.20% and a testing Double F1-Score of 78.16% when tested outside the processing pipeline, the fourth best result in that context. However, the EfficientNetB7 FPN24 segmentation model was used for the rest of the experiments when applicable because it worked best in conjunction with the classification models.

To test GAN-like training for gore segmentation, we applied a Pix2Pix experimental setup, including the PatchGAN discriminator, to generate and discriminate segmentation mask. Training an FPN24 EfficientNetB7 only starting with the ImageNet weights led to poor results (see Table 23). However, using this setup to fine-tune the previously trained model on gore segmentation improved the results. One such model was able to achieve a testing IoU score of 56.75% and Double F1-Score of 79.94%, (see Table 24). This represents a small, but noticeable, improvement over the previously trained segmentation model.

Next, we experimented with the object detection and instance segmentation model architecture Mask R-CNN. The Tensorflow 2.0 port of the Matterport Tensorflow implementation [210] was used, including two COCO pretrained base models: ResNet50 and ResNet101. We trained models with both. From Table 25, we noticed that only the ResNet50 model achieves the best gore segmentation results with a testing IoU of 54.03% and a Double F1-Score of 78.95%. The ResNet101 models underperformed, only achieving a testing IoU of 46.20% and a Double F1-Score of 74.80%. We hypothesise that the Mask R-CNN models were limited

by the relatively small size of our gore segmentation dataset, and perhaps, in a smaller part, by the lack of computing power of our training setup.

Finally, we performed experiments to ensure the training setup, used throughout this chapter, was optimal. The first experiment consisted of testing optimal thresholding for gore segmentation. Up to that point, we always used the threshold value of 0.5. The results of Table 26 show that using the validation set to set the threshold leads to worse testing results than continuing the use of 0.5. Thus, optimal thresholding is not useful on our gore segmentation dataset. We then performed an ablation study of the training setup. To do so, we trained FPN24 EfficientNetB7 models with and without Image Patching and Data Augmentation. Table 27 shows that Image Patching was essential to obtain optimal testing results. Conversely, we obtained better testing results without performing Data Augmentation. However, when we tried to use GAN-like fine-tuning to improve on these results as was done with the model from Phase 4, the resulting models achieved worse results, as it can be seen in Table 28. Hence, we conclude that Data Augmentation makes the model converge to a more robust loss function minimum. Thus, it should be included as part of an optimal training setup. An augmentation-less model will be further tested as part of the Image Processing Pipeline in the following chapter.

Contrary to Chapter 5, we will keep a variety of models to test in the next chapter. This is because as part of our Image Processing Pipeline, the segmentation models will work in conjunction with the classification models. This could lead to different behaviours and testing results. The four best FPN segmentation networks will be tested. These models used ResNet152, DenseNet169, SE-ResNet152 and EfficientNetB7 as their CNN encoders. The GAN fine-tuned and the Data Augmentation-less EfficientNetB7 models will also be tested as part of the Image Processing Pipeline. We will also test the two best performing Mask R-CNN models of both architectures (ResNet50 and ResNet101). Table 29 summarizes the selected models' standalone testing performance on the gore segmentation task.

*Table 29 - Gore Segmentation Models standalone testing performance before the Image Processing Pipeline. Highlighted cells show the best performance on each metric.*

| Segmentation Model | Threshold | MSE | IoU | F1-Score | Double F1 |
|---|---|---|---|---|---|
| FPN24 ResNet152 (Table 22) | 0.5 | 0.0393 | 0.5539 | 0.6179 | 0.7952 |
| FPN24 DenseNet169 (Table 22) | | 0.0403 | 0.5553 | 0.6175 | 0.7946 |
| FPN24 SE-ResNet152 (Table 22) | | 0.0437 | 0.5327 | 0.5933 | 0.7815 |
| FPN24 EfficientNetB7 (Table 22) | | 0.0423 | 0.5320 | 0.5942 | 0.7816 |
| GAN Improved FPN24 EfficientNetB7 (Table 24 – Run 4) | | 0.0441 | 0.5675 | 0.6286 | 0.7994 |
| FPN24 EfficientNetB7 (Table 27 - Without Training Augmentation) | | 0.0417 | 0.5410 | 0.6039 | 0.7869 |
| Mask R-CNN ResNet50 (Table 25) | | 0.0583 | 0.5403 | 0.6185 | 0.7895 |
| Mask R-CNN ResNet101 (Table 25) | | 0.0648 | 0.4620 | 0.5391 | 0.7480 |

# Chapter 7. Image Processing Pipeline Testing, Deployment and Use

For the trained models to fulfill their purpose, they need to be deployed in a way that is useful to the end user and uses resources efficiently. To that end, the gore segmentation model should only create a censored image if the gore classification has determined it contains gore. This is what is shown in Figure 12. While this will result in reduced processing time and storage space requirements, it is an untested approach for this application. Accordingly, this section will perform testing for the entire pipeline for both tasks to ensure the cascading of the two models is beneficial in terms of segmentation results. We will also look at the effect on the classification results by looking at whether an image was censored as a classification results. We will then use these results to make our recommendation in terms of use of these models to minimize as much as possible the amount of gore that could possibly be show to end-users. Details on how the models were implemented for use using BentoML can be found in Appendix C - BentoML Services Creation.

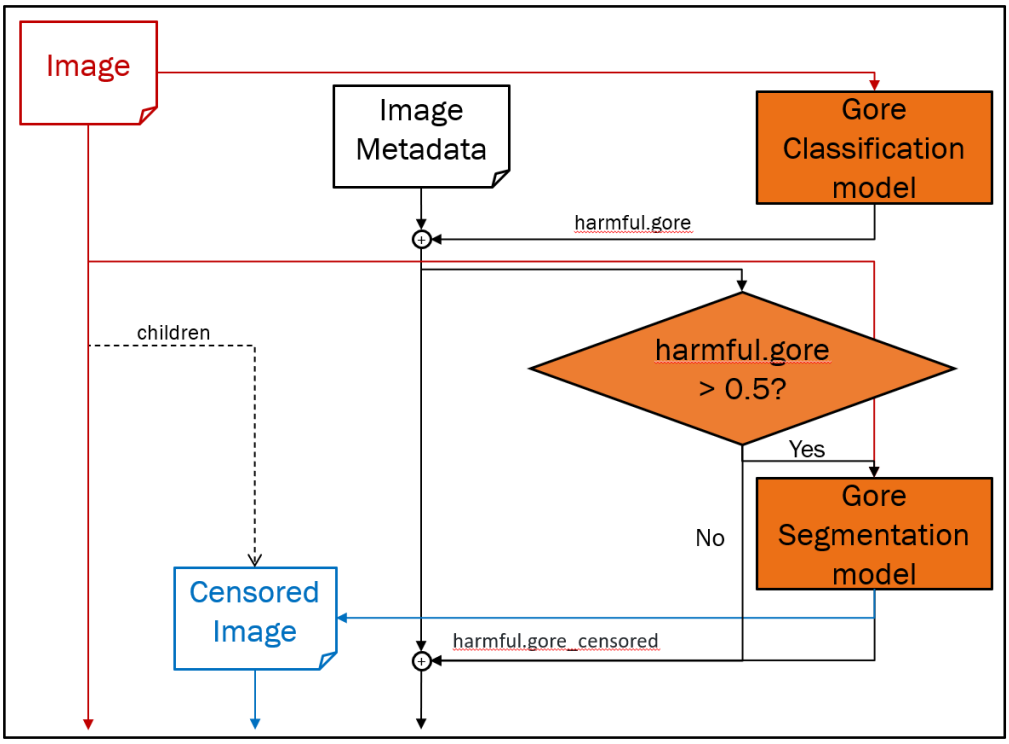


*Figure 12 - Gore Censoring Image Processing Services Workflow*

## 7.1 Pipeline Testing

Our hypothesis on using the Image Processing Pipeline is that it will allow us to maximize the gore censored in images. In order to test it, we decided to test the whole Image Processing Pipeline results on the gore classification and gore segmentation tasks. They will be tested using the same testing sets used in Chapter 5 and Chapter 6. Thus, the following results can be compared to testing results presented in these chapters. The testing results in Chapter 5 and Chapter 6 also serve as the standalone performance of each service.

### 7.1.1 Gore Classification

In this section, we test the entire pipeline as a classification model. Since the classification models trained in Chapter 5 were able to achieve very high recall with high precision and were trained on a larger dataset, they will be used to remove accidental exposure such as within search results and gallery views. Thus, this test looks at the ability of the pipeline to censor the right images when necessary.

Table 30 shows the classification results of the entire pipeline at a 0.5 threshold (results at other threshold values can be found in Appendix B - B1 - Classification). In these results, we notice that the precision increased, and the recall decreased drastically. This is because the segmentation model does not censor some of the images that the classification model considered as containing gore. This means that it corrects FPs but generates FNs. Since there is significant class imbalance, this effect is much more pronounced on the recall values even though more FPs are corrected than FNs generated. The model for which this effect was less pronounced was the Mask R-CNN ResNet101 which makes it the best suited for the Pipeline classification. However, its poor results in standalone segmentation evaluation means it might not be the best overall model with this dataset. It is an encouraging result for future work where a larger segmentation dataset will be created and used. Of the 4 segmentation models from section 6.2.4, the EfficientNetB7 achieved the best F1-Score (0.8485), Double F1-Score (0.8946) and Recall (0.7692) when working with both classification models, on the classification testing set. This is somewhat surprising as it was the fourth best on the segmentation testing set. The performance DenseNet169 is only slightly worse achieving a 0.8373 testing F1-Score and a 0.8865 testing Double F1-Score. The EfficientNetB7 model without training augmentation is close behind with a 0.8344 testing F1-Score and a 0.8853 testing Double F1-Score.

For gore classification, we wanted to maximize recall to ensure the least amount of false positives. The results of Table 30 being worse than the Voting_WRL and Voting_Mixed standalone classification performance from Table 13, this confirms our recommendation of using the gore classification models' results directly to reduce the exposure as much as possible.

*Table 30 - Pipeline Classification Testing Results at 0.5 classification threshold. Highlighted cells in gold show the best performance on each metric. The cells in blue are the best performance of the 4 models of Phase 4.*

| Classification model | Segmentation model | Threshold | F1-Score | Double F1-Score | Precision | Recall |
|---|---|---|---|---|---|---|
| Voting_WRL (No threshold prior to voting) | FPN24 ResNet152 (Table 22) | 0.5 | 0.7857 | 0.8547 | 0.9603 | 0.6648 |
| | FPN24 DenseNet169 (Table 22) | | 0.8373 | 0.8865 | 0.9267 | 0.7637 |
| | FPN24 SE-ResNet152 (Table 22) | | 0.8188 | 0.8753 | 0.9493 | 0.7198 |
| | FPN24 EfficientNetB7 (Table 22) | | 0.8485 | 0.8946 | 0.9459 | 0.7692 |
| | GAN Improved segmentation model (Table 24 – Run 4) | | 0.8101 | 0.8700 | 0.9552 | 0.7033 |
| | FPN24 EfficientNetB7 (Table 27 - Without Training Augmentation) | | 0.8344 | 0.8853 | 0.9444 | 0.7473 |
| | Mask R-CNN ResNet50 (Table 25) | | 0.4898 | 0.6775 | 0.9524 | 0.3297 |
| | Mask R-CNN ResNet101 (Table 25) | | 0.8652 | 0.9032 | 0.8851 | 0.8462 |
| Voting_Mixed (No threshold prior to voting) | FPN24 ResNet152 (Table 22) | 0.5 | 0.7871 | 0.8553 | 0.9531 | 0.6703 |
| | FPN24 DenseNet169 (Table 22) | | 0.8468 | 0.8930 | 0.9338 | 0.7747 |
| | FPN24 SE-ResNet152 (Table 22) | | 0.8162 | 0.8734 | 0.9424 | 0.7198 |
| | FPN24 EfficientNetB7 (Table 22) | | 0.8554 | 0.8991 | 0.9467 | 0.7802 |
| | GAN Improved segmentation model (Table 24 – Run 4) | | 0.8139 | 0.8724 | 0.9556 | 0.7088 |
| | FPN24 EfficientNetB7 (Table 27 - Without Training Augmentation) | | 0.8379 | 0.8876 | 0.9448 | 0.7527 |
| | Mask R-CNN ResNet50 (Table 25) | | 0.4898 | 0.6775 | 0.9524 | 0.3297 |
| | Mask R-CNN ResNet101 (Table 25) | | 0.8715 | 0.9075 | 0.8864 | 0.8571 |

### 7.1.2 Gore Segmentation

As in the previous section, the entire gore Image Processing Pipeline is tested on the segmentation testing set. The results in Table 31 show the testing results, when using the classification threshold of 0.5 (Appendix B - B2 – Segmentation show results at other thresholds). In this table, we see that the two best models are FPN24 EfficientNetB7 trained both with and without Data Augmentation. Interestingly, these segmentation models achieved the same results with both voting classification models (Voting_WRL and Voting_Mixed). The FPN24 DenseNet169 model is the third best performing model on the segmentation testing set in this context. The fourth best is the ResNet101 Mask R-CNN which is again encouraging considering its poor performance as a standalone segmentation model.

Surprisingly, the FPN24 EfficientNetB7 model fine-tuned using GAN-like training was only the sixth best performing model even though it was the best standalone gore segmentation model as seen in Table 29. This may be because this model is better than the other FPN24 EfficientNetB7 model at not segmenting non-gore images and comes at the cost of being less capable of segmenting gore in gory images. This

performance gain is thus nullified using the classification model prior to segmentation. This is probably the reason behind the drop in performance for the FPN24 ResNet152 and SE-ResNet152 models as well.

*Table 31 - Pipeline Segmentation Testing Results at 0.5 classification threshold. Highlighted cells in gold show the best performance on each metric.*

| Classification model | Segmentation model | Threshold | MSE | IoU | F1-Score | Double F1-Score |
|---|---|---|---|---|---|---|
| Voting_WRL (No threshold prior to voting) | FPN24 ResNet152 (Table 22) | 0.5 | 0.0380 | 0.6506 | 0.7190 | 0.8439 |
| | FPN24 DenseNet169 (Table 22) | | 0.0375 | 0.6785 | 0.7477 | 0.8590 |
| | FPN24 SE-ResNet152 (Table 22) | | 0.0368 | 0.6695 | 0.7336 | 0.8523 |
| | FPN24 EfficientNetB7 (Table 22) | | 0.0330 | 0.6995 | 0.7658 | 0.8693 |
| | GAN Improved segmentation model (Table 24 – Run 4) | | 0.0417 | 0.6392 | 0.7026 | 0.8351 |
| | FPN24 EfficientNetB7 (Table 27 - Without Training Augmentation) | | 0.0345 | 0.6929 | 0.7587 | 0.8655 |
| | Mask R-CNN ResNet50 (Table 25) | | 0.0684 | 0.4700 | 0.5045 | 0.7298 |
| | Mask R-CNN ResNet101 (Table 25) | | 0.0521 | 0.6512 | 0.7283 | 0.8460 |
| Voting_Mixed (No threshold prior to voting) | FPN24 ResNet152 (Table 22) | 0.5 | 0.0380 | 0.6506 | 0.7190 | 0.8439 |
| | FPN24 DenseNet169 (Table 22) | | 0.0375 | 0.6785 | 0.7477 | 0.8590 |
| | FPN24 SE-ResNet152 (Table 22) | | 0.0368 | 0.6695 | 0.7336 | 0.8523 |
| | FPN24 EfficientNetB7 (Table 22) | | 0.0330 | 0.6995 | 0.7658 | 0.8693 |
| | GAN Improved segmentation model (Table 24 – Run 4) | | 0.0417 | 0.6392 | 0.7026 | 0.8351 |
| | FPN24 EfficientNetB7 (Table 27 - Without Training Augmentation) | | 0.0345 | 0.6929 | 0.7587 | 0.8655 |
| | Mask R-CNN ResNet50 (Table 25) | | 0.0684 | 0.4700 | 0.5045 | 0.7298 |
| | Mask R-CNN ResNet101 (Table 25) | | 0.0502 | 0.6602 | 0.7373 | 0.8510 |

### 7.2 Analysis of the Pipeline's Segmentation results

Like the analysis performed in Section 5.2.10, the pipeline was presented the unseen images of both the classification and segmentation datasets. While the previous analysis could only conclude from seeing the similarity between positive results, the segmentation results allow us to clearly see what features in the original image led to censoring. The analysis will be done on the FPN24 EfficientNetB7 model from Table 22. This model is chosen for consistently performing well in the previous section, when used as part of the image pipeline. An argument could be made to use the FPN24 DenseNet169 model instead, because it achieved only slightly worse results with much less parameters (12,730,325 vs 64,235,813). However, we decided to give a priority to the model with the best testing metrics. We note that further study is needed to complete an in-depth assessment of which of these two CNN models and architectures is best suited for gore segmentation.

This analysis confirmed the pipeline's tendency to censor red or rosy regions, especially when contrasted with a skin-like colour (white, light-brown, or brown). The tendency to censor holes and cavities when present in regions of skin tone colours or red is also evident. We can confidently make this conclusion as these features were consistently censored in images. The notable FNs included ketchup; red, brown, or dark sauces; pies and pastries (especially with a red or dark filling); and fried/charred foods. This seems to confirm the hypothesis related to these features presented in Section 5.2.10. Shadows have a significant effect on the pipeline's results. An example of this are nearly identical cubes of red meat where only one is censored, because it was the only one that had a shadow which enhanced the blood-like colour of the meat. Another example is a pile of green and red peppers where the shadow and gap between two peppers was censored as if it was a gaping wound.

In the previous analysis, we thought the model seemed to respond to deformation of the facial structure or human anatomy. The segmentation results do not support this hypothesis as the censored part of these images were not the ones with the deformed features. Instead, the areas that contained blood, cavities, or subcutaneous anatomy (bones and muscles) were censored.

The one part of anatomy the segmentation model seems to understand is the mouth. The model did not seem to censor a smiling or open human mouth unless it showed the throat or signs of injury. On the other hand, the model did censor the mouths of animals (such as dogs) and mouths with sharp teeth in creepy images, consistently.

### 7.3 Summary of Chapter 7

In this chapter, we built and tested an image processing pipeline for gore Censoring using the models trained in chapters 4 and 5. We started by assessing how well the gore classification and gore segmentation models worked together. Surprisingly, we found that the model which achieved the best segmentation results in this pipeline context (see Table 31) was the fourth best segmentation model of section 6.2.4, FPN24 EfficientNetB7. In Table 30, this segmentation model also achieved the second-best pipeline gore classification results (the best model being Mask R-CNN ResNet101). Since Mask R-CNN ResNet101 was much worse on the gore segmentation task, we concluded that FPN24 EfficientNetB7 was the best choice of gore segmentation model, in our specific context. The results of Table 30 also show that the Voting_Mixed model achieved better classification results than the Voting_WRL model. Since these models achieved equivalent segmentation results when use with FPN24 EfficientNetB7, we decided that Voting_Mixed was the best choice of classification model for the Image Processing Pipeline.

We also found in this chapter that using segmentation models after classification models led to an important number of FNs (although less than when we used the gore classification models alone). However, since the objective of this thesis is to reduce the exposure to gore, we suggest only using the segmentation results when the image absolutely needs to be displayed to the user so they can perform their work. Otherwise, the image should be blurred or not displayed, based on the classification results.

# Chapter 8. Conclusion

This thesis proposed a novel gore and Graphic classification and Censoring task. The goal of this task is to shield adult professional users from gory content to which they might be exposed to as part of their work. This is novel as previous work primarily looked at detecting violent or graphic content in movies for a different target audience: children. Hence, their definition of gore is broader as children are more sensible to such content than adults. To solve this, we had to create a dataset for two tasks (classification and Censoring/segmentation). The latter required a lengthy annotation process as it contained 922 images from varied sources. The training, validation and testing set for Censoring/segmentation contain 737, 74 and 111 images, respectively. The classification dataset contains 3830 training, 391 validation and 586 testing images. The classification dataset was larger for two reasons: the annotation process was easier, and more images were needed to alleviate poor initial results obtained with a smaller set.

This thesis also showed that these novel tasks could be accomplished using Image classification and segmentation architectures and techniques from the literature. The solutions for these tasks where systematically developed by identifying and tracking the best evaluation metric, training loss function, architecture, and CNN encoder for a given task. Moreover, an ablation study of the training setup was performed to ensure it led to optimal results.

The best performing classification model was a voting ensemble with three models trained with a novel WRL, and one trained with BCE. The three WRL models used ImageNet pretrained MobileNetV2, DenseNet121 and VGG16 CNN to extract visual information from the images. The BCE model used the VGG16 CNN model. The voting ensemble was able to achieve a 90.66% recall, 87.30% precision, 88.95% F1-Score and a 91.92% Double F1-Score on the classification testing set at a threshold value of 0.5. This solution outperformed the only publicly available Graphic content detection model, Google SafeSearch, by a margin of 31.34% for precision for the same recall value.

The best performing gore segmentation solution of the ones tested was a FPN using an ImageNet pretrained EfficientNetB7 as its backbone. This model achieved a 53.20% IoU testing results. Using a Pix2Pix GAN-like training, this model achieved a higher 56.75% IoU. A ResNet50 Mask R-CNN model achieved a 54.03% IoU. However, when the classification model was added before the segmentation model to form an Image Processing Pipeline, these two models performed worse than the initial FPN EfficientNetB7 model. In this setting, this model achieved a 69.95% IoU compared to 63.92% and 47% for the Pix2Pix and ResNet50 Mask R-CNN respectively.

Additionally, a means of deploying the previously trained models was presented where the classification model assesses the image first and a censored image is only generated for images with the highest odds of containing gore. Another advantage of this approach is that the classification model can recognize gore in images that the segmentation model is not able to censor. We know this because the recall value dropped by 13.74% on the classification testing set when using images censored by the best segmentation model (given

the results of the best classification model compared to the best segmentation on its own). This difference is most likely due to the smaller number of images in the segmentation training set when compared to the classification one but could also be due to the more complex nature of the segmentation task. Because of this significant reduction, we proposed blurring the images using the classification results whenever the image will be displayed to workers in contexts where the image is displayed for convenience or non-analytical purpose. An example of this context is search results or gallery views. This will reduce the risk of accidental exposure to gore. The censored images should only be displayed when a worker needs to perform specific analysis. We believe this will be the most efficient way of using the proposed models.

### 8.1 Limitations

The most significant limitation of this thesis is the dataset. Its small size means it can merely be used for fine-tuning, only allowing the use of models and architecture from the literature which have been released or are compatible with pre-trained CNNs encoders/backbones. Additionally, the segmentation annotations, while adequate, are far from perfect especially on images with many small gore regions leading to the fact that even a perfect model will obtain a non-perfect IoU score. The subjectivity of gore and the fact that only a single annotator worked on the dataset means that he annotations for both tasks do not have any inter-annotator agreement and may not be ideal for all situations they could be used in. Moreover, the fact that the classification dataset only includes high level annotation (Blood, Wound, gore) means the dataset splitting is not guaranteed to have an even split of various situations in all sets. Lastly, the fact that copyrighted and sometimes deleted images were used to create the dataset means it cannot be released to the public. This all means that this thesis must be considered more as a proof of concept as opposed to a full-scale solution to the presented problem.

Since we based our decision making on testing results in each experimental phases in Chapter 5 and Chapter 6 and then used the same testing set to evaluate the end models, we introduced a bias in said results which becomes a major limitiation. As such, they are not indicative of results on unseen data which is usually what is expected of testing results since the model were chosen for their good performance on the testing set. We still fell the testing results show that solving gore classification and segmentation is doable using DL which is our main contribution. However, future work on these tasks may not achieve the same performance as the one obtained in this work, because of the bias we introduced in our results. In hindsight, we should have based our decision on validation results and kept the testing set when the models were settled.

Another limitation of this thesis is its broad scope which limited the scope of our experimentation. For instance, we never revisited the results of a previous experiments even after we made changes to the model architecture. Thus, it is possible that we obtained sub-optimal results in our end models, because we did not revisit our choice of loss function or evaluation metric with the best encoder and architectures. We also only used solutions compatible with TensorFlow Keras to simplify implementation. This is especially true for segmentation where the 3 tested architectures were the ones available in the segmentation models package [114] that was compatible with our input configuration. The Mask R-CNN results also suffered from this

limitation as the original and maintained implementation is in PyTorch [211] and will have required a redesign of the datasets and training code. Accordingly, a port of a TensorFlow 1.0 implementation [210] was used instead, which was probably not as efficient as the PyTorch one.

## 8.2 Future Work

To further research this domain, a larger and better annotated dataset should be created. The sourcing of gore images is non-trivial and is a major hurdle to overcome in this domain. The annotation process of this new dataset should include multiple annotators. This will be much more realistic for classification annotation as the exposure to gore needed to annotate is minimal. However, the annotators should still be supervised by a therapist to ensure the work does not negatively affect them. This means that traditional methods of crowdsourcing annotation are not ideal for gore classification. The more tedious annotation process required for gore segmentation means that it will be much more difficult to responsibly scale up the annotation process and thus, the dataset.

Another shortfall of the current solution, which will need to be corrected in future work, is trying to remove the censoring of blatant FPs such as food while maintaining the ability of censoring gore. Adding such cases in the dataset, as it was done in our classification dataset, will help. However, other solutions and architectures, not evaluated in this thesis, might prove more resistant to these cases.

Finally, analysis to ensure the models can censor gore on people of varied skin colour and genders, as well as animals, needs to be performed. This analysis is needed as most of the data identified for this thesis involved injuries and wounds on Caucasian individuals. Therefore, there is no guarantee that the model will be able to effectively generalise to other situations with the same accuracy.

# Bibliography

[1] D. Mishra, "Convolution Vs Correlation," Towards Data Science, 13 November 2019. [Online]. Available: https://towardsdatascience.com/convolution-vs-correlation-af868b6b4fb5. [Accessed 20 May 2021].

[2] Wikipedia, "Data augmentation," 28 March 2021. [Online]. Available: https://en.wikipedia.org/wiki/Data_augmentation. [Accessed 20 May 2021].

[3] Wikipedia, "Entropy (information theory)," 9 May 2021. [Online]. Available: https://en.wikipedia.org/wiki/Entropy_(information_theory). [Accessed 20 May 2021].

[4] D. Opitz and R. Maclin, "Popular ensemble methods: An empirical study," *Journal of Artificial Intelligence Research,* no. 11, pp. 169-198, 1999.

[5] Deep AI, "Evaluation Metrics," [Online]. Available: https://deepai.org/machine-learning-glossary-and-terms/evaluation-metrics. [Accessed 20 May 2021].

[6] "Gore," Merriam-Webster, [Online]. Available: https://www.merriam-webster.com/dictionary/gore. [Accessed 17 February 2021].

[7] Wikipedia, "Image segmentation," 7 June 2020. [Online]. Available: https://en.wikipedia.org/wiki/Image_segmentation. [Accessed 3 July 2020].

[8] Deep AI, "Max Pooling," [Online]. Available: https://deepai.org/machine-learning-glossary-and-terms/max-pooling. [Accessed 20 May 2021].

[9] Wikipedia, "Precision and recall," 1 July 2020. [Online]. Available: https://en.wikipedia.org/wiki/Precision_and_recall. [Accessed 3 July 2020].

[10] Wikipedia, "Transfer learning," 23 June 2020. [Online]. Available: https://en.wikipedia.org/wiki/Transfer_learning. [Accessed 3 July 2020].

[11] Y. Gal and Z. Ghahramani, "Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning," 4 October 2016. [Online]. Available: https://arxiv.org/pdf/1506.02142.pdf. [Accessed 17 September 2020].

[12] "Wound," Merriam-Webster, [Online]. Available: https://www.merriam-webster.com/dictionary/wound. [Accessed 17 February 2021].

[13] B. Marr, "How Much Data Do We Create Every Day? The Mind-Blowing Stats Everyone Should Read," Forbes, 21 May 2018. [Online]. Available: https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/?sh=539cb1fd60ba. [Accessed 18 May 2021].

[14] B. Vuleta, "How Much Data Is Created Every Day? [27 Staggering Stats]," SeedScientific, 28 January 2021. [Online]. Available: https://seedscientific.com/how-much-data-is-created-every-day/. [Accessed 18 May 2021].

[15] BBC News, "Facebook to pay $52m to content moderators over PTSD," BBC News, 13 May 2020. [Online]. Available: https://www.bbc.com/news/technology-52642633#:~:text=Facebook%20has%20agreed%20to%20pay,issues%20developed%20on%20the

%20job.&text=The%20moderators%20alleged%20that%20reviewing,traumatic%20stress%20disorder%20(PTSD).. [Accessed 20 June 2020].

[16] A. Hern, "Ex-Facebook worker claims disturbing content led to PTSD," The Guardian, 4 December 2019. [Online]. Available: https://www.theguardian.com/technology/2019/dec/04/ex-facebook-worker-claims-disturbing-content-led-to-ptsd. [Accessed 02 July 2020].

[17] C. Newton, "The Terror Queue," The Verge, 16 December 2019. [Online]. Available: https://www.theverge.com/2019/12/16/21021005/google-youtube-moderators-ptsd-accenture-violent-disturbing-content-interviews-video. [Accessed 02 July 2020].

[18] J. Guynn, "Mark Zuckerberg: Facebook is transparent about cleaning up hate speech and violence, others aren't," USA Today, 13 November 2019. [Online]. Available: https://www.usatoday.com/story/tech/2019/11/13/facebook-making-some-progress-curbing-harmful-content/4180109002/. [Accessed 2 July 2020].

[19] C. Demarty, C. Penet, M. Soleymani and G. Gravier, "VSD, a public dataset for the detection of violent scenes in movies: design, annotation, analysis and evaluation," May 2014. [Online]. Available: http://link.springer.com/article/10.1007/s11042-014-1984-4. [Accessed 2 July 2020].

[20] M. D. More, D. M. Souza, J. Wehrmann and R. C. Barros, "Seamless Nudity Censorship: an Image-to-Image Translation Approach based on Adversarial Training," in *International Joint Conference on Neural Networks (IJCNN)*, Rio de Janeiro, 2018.

[21] J.-L. Shih, C.-H. Lee and C.-S. Yang, "An adult image identification system employing image retrieval technique," *Pattern Recognition Letters,* vol. 28, no. 16, pp. 2367-2374, 2007.

[22] A. Gangwar, E. Fidalgo, E. Alegre and V. González-Castro, "Pornography and Child Sexual Abuse Detection in Image and Video: A Comparative Evaluation," in *International Conference on Imaging for Crime Detection and Prevention (ICDP)*, Madrid, 2017.

[23] M. F. A. Fauzi, I. Khansa, K. Catignani, G. Gordillo, C. K. Sen and M. N. Gurcan, "Computerized segmentation and measurement of chronic wound images," *Computers in Biology and Medicine,* vol. 60, pp. 74-85, 2015.

[24] A. Abadpour and S. Kasaei, "Pixel-Based Skin Detection for Pornography Filtering," *Iranian journal of electrical and electronic engineering,* vol. 1, no. 3, pp. 21-41, 2005.

[25] B. e. al., "AUTOMATED DETECTION OF PORNOGRAPHIC IMAGES". United States Patent 6,751,348 B2, 15 June 2004.

[26] B. Garcia-Zapirain, A. Shalaby, A. El-Baz and A. Elmaghraby, "Automated framework for accurate segmentation of pressure ulcer images," *Computers in Biology and Medicine,* vol. 90, pp. 137-145, 2017.

[27] "Gruesomeness," Merriam-Webster, [Online]. Available: https://www.merriam-webster.com/dictionary/gruesomeness. [Accessed 5 November 2021].

[28] C. L. Badour and M. T. Feldner, "The Role of Disgust in Posttraumatic Stress: A Critical Review of the Empirical Literature," *Journal of Experimental Psychopathology,* vol. 9, no. 3, 2018.

[29] Y. LeCun., Y. Bengio and G. Hinton, "Deep learning," 27 May 2015. [Online]. Available: https://www.nature.com/articles/nature14539. [Accessed 3 July 2020].

[30] LeCun, Y. et al., "Backpropagation Applied to Handwritten Zip Code Recognition," *Neural Computation,* no. 1, pp. 541-551, 1989.

[31] LeCun, Y. et al., "Handwritten digit recognition with a back-propagation network," in *Advances in Neural Information Processing Systems*, Denver, 1990.

[32] A. Al-Masri, "How Does Back-Propagation in Artificial Neural Networks Work?," Towards Data Science, 29 January 2019. [Online]. Available: https://towardsdatascience.com/how-does-back-propagation-in-artificial-neural-networks-work-c7cad873ea7. [Accessed 23 May 2021].

[33] G. Zhang, C. Wang, B. Xu and R. Grosse, "Three mechanisms of weight decay regularization," 2019. [Online]. Available: https://openreview.net/pdf?id=B1lz-3Rct7. [Accessed 3 July 2020].

[34] S. Ruder, "An overview of gradient descent optimization algorithms," 15 June 2017. [Online]. Available: https://arxiv.org/pdf/1609.04747.pdf. [Accessed 3 July 2020].

[35] J. B. Diederik P. Kingma, "Adam: A Method for Stochastic Optimization," in *3rd International Conference for Learning Representations*, San Diego, 2015.

[36] S. Verma, "Multi-Label Image Classification with Neural Network | Keras," Medium, 30 September 2019. [Online]. Available: https://towardsdatascience.com/multi-label-image-classification-with-neural-network-keras-ddc1ab1afede. [Accessed 3 July 2020].

[37] V. Nair and G. E. Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines," in *27th International Conference on Machine Learning*, Haifa, 2010.

[38] X. Glorot, A. Bordes and Y. Bengio, "Deep Sparse Rectifier Neural Networks," in *14th International Conference on Artificial Intelligence and Statistics (AISTATS)*, Fort Lauderdale, 2011.

[39] Tensorflow, "Image classification," 12 June 2020. [Online]. Available: https://www.tensorflow.org/tutorials/images/classification. [Accessed 3 July 2020].

[40] A. Krizhevsky, I. Sutskever and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing System*, Lake Tahoe, 2012.

[41] C. Szegedy et al., "Going deeper with convolutions," 17 September 2014. [Online]. Available: https://arxiv.org/pdf/1409.4842.pdf. [Accessed 7 July 2020].

[42] K. Simonyan and A. Zisserman, "VERY DEEP CONVOLUTIONAL FOR LARGE-SCALE IMAGE RECOGNITION," 10 April 2015. [Online]. Available: https://arxiv.org/pdf/1409.1556.pdf. [Accessed 08 May 2021].

[43] TensorFlow, "Module: tf.keras.applications," 12 September 2020. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/applications. [Accessed 20 September 2020].

[44] Pytorch Team, "VGG-NETS," Pytorch, [Online]. Available: https://pytorch.org/hub/pytorch_vision_vgg/. [Accessed 08 May 2021].

[45] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," 10 December 2015. [Online]. Available: https://arxiv.org/pdf/1512.03385.pdf. [Accessed 9 July 2020].

[46] K. Zhang, W. Zuo, Y. Chen, D. Meng and L. Zhang, "Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising," 1 February 2017. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7839189. [Accessed 9 July 2020].

[47] Y. Song, C. Ma, L. Gong, J. Zhang, R. W. H. Lau and M.-H. Yang, "CREST: Convolutional Residual Learning for Visual Tracking," in *IEEE International Conference on Computer Vision*, Venise, 2017.

[48] D. Pakhomov, V. Premachandran, M. Allan, M. Azizian and N. Navab, "Deep Residual Learning for Instrument Segmentation in Robotic Surgery," in *International Workshop on Machine Learning in Medical Imaging*, Shenzhen, 2019.

[49] C. Liu et al., "Clouds Classification from Sentinel-2 Imagery with Deep Residual Learning and Semantic Image Segmentation," *Remote Sens.,* vol. 11, no. 2, p. 119, 2019.

[50] J. Brownlee, "Overfitting and Underfitting With Machine Learning Algorithms," 12 August 2019. [Online]. Available: https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/. [Accessed 9 July 2020].

[51] G. Huang, Z. Liu, L. van der Maaten and K. Q. Weinberger, "Densely Connected Convolutional Networks," 28 January 2018. [Online]. Available: https://arxiv.org/pdf/1608.06993.pdf. [Accessed 08 May 2021].

[52] Pytorch Team, "DENSENET," [Online]. Available: https://pytorch.org/hub/pytorch_vision_densenet/. [Accessed 08 May 2021].

[53] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov and L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," 21 March 2019. [Online]. Available: https://arxiv.org/pdf/1801.04381.pdf. [Accessed 9 July 2020].

[54] S. Xie, R. Girshick, P. Dollár, Z. Tu and K. He, "Aggregated Residual Transformations for Deep Neural Networks," 11 April 2017. [Online]. Available: https://arxiv.org/pdf/1611.05431. [Accessed 22 July 2020].

[55] B. Zoph, V. Vasudevan, J. Shlens and Q. V. Le, "Learning Transferable Architectures for Scalable Image Recognition," 11 April 2018. [Online]. Available: https://arxiv.org/pdf/1707.07012.pdf. [Accessed 09 May 2021].

[56] "NEURAL ARCHITECTURE SEARCH WITH REINFORCEMENT LEARNING," 15 February 2017. [Online]. Available: https://arxiv.org/pdf/1611.01578.pdf. [Accessed 09 May 2021].

[57] J. Hu, L. Shen, S. Albanie, G. Sun and E. Wu, "Squeeze-and-Excitation Networks," 16 May 2019. [Online]. Available: https://arxiv.org/pdf/1709.01507.pdf. [Accessed 22 July 2020].

[58] M. Tan and Q. V. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," [Online]. Available: https://arxiv.org/pdf/1905.11946.pdf. [Accessed 26 July 2021].

[59] M. Tan and Q. V. Le, "EfficientNet: Improving Accuracy and Efficiency through AutoML and Model Scaling," Google AI Blog, 29 May 2019. [Online]. Available: https://ai.googleblog.com/2019/05/efficientnet-improving-accuracy-and.html. [Accessed 22 July 2020].

[60] Y. Huang et al., "GPipe: Easy Scaling with Micro-Batch Pipeline Parallelism," 25 July 2019. [Online]. Available: https://arxiv.org/pdf/1811.06965.pdf. [Accessed 22 July 2020].

[61] L. Mi, H. Wang, Y. Tian and N. Shavit, "TRAINING-FREE UNCERTAINTY ESTIMATION," 24 December 2019. [Online]. Available: https://openreview.net/pdf?id=Sye9lyHKwB. [Accessed 17 September 2020].

[62] OpenReview.net, "Training-Free Uncertainty Estimation for Neural Networks," 24 December 2019. [Online]. Available: https://openreview.net/forum?id=Sye9lyHKwB. [Accessed 17 September 2020].

[63] Deng, J. et al., "ImageNet: A Large-Scale Hierarchical Image Database," in *IEEE Conference on Computer Vision and Pattern Recognition*, Miami, 2009.

[64] K. Simonyan and A. Zisserman, "Very Depp Convolutional Networks For Large-Scale Image Recognition," in *International Conference on Learning Representations*, San Diego, 2015.

[65] TensorFlow, "Transfer learning with TensorFlow Hub," 12 June 2020. [Online]. Available: https://www.tensorflow.org/tutorials/images/transfer_learning_with_hub. [Accessed 14 July 2020].

[66] Keras, "Keras Applications," [Online]. Available: https://keras.io/api/applications/. [Accessed 14 July 2020].

[67] Torch Contributors, "TORCHVISION.MODELS," 2019. [Online]. Available: https://pytorch.org/docs/stable/torchvision/models.html. [Accessed 14 July 2020].

[68] R. F. Gunst and R. L. Mason, "Biased Estimation in Regression: An Evaluation Using Mean Squared Error," *Journal of the American Statistical Association,* vol. 72, no. 359, pp. 616-628, 1977.

[69] C. W. Helstrom, "Minimum Mean-Squared Error of Estimates in Quantum Statistics," *Physics Letters,* vol. 25A, no. 2, pp. 101-102, 31 July 1967.

[70] S. Hashem and B. Schmeiser, "Approximating a Function and its Derivatives Using MSE-Optimal Linear Combinations of Trained Feedforward Neural Networks," in *World Congress on Neural Networks*, Portland, 1993.

[71] Wikipedia, "Mean squared error," 18 June 2020. [Online]. Available: https://en.wikipedia.org/wiki/Mean_squared_error#Proof_of_variance_and_bias_relationship. [Accessed 14 July 2020].

[72] M. Zavershynskyi, "MSE and Bias-Variance decomposition," 21 May 2017. [Online]. Available: https://towardsdatascience.com/mse-and-bias-variance-decomposition-77449dd2ff55. [Accessed 14 July 2020].

[73] M. Buckland and F. Gey, "The Relationship between Recall and Precision," *Journal of the American Society for Information Science,* vol. 45, no. 1, pp. 12-19, 1994.

[74] K. P. Shung, "Accuracy, Precision, Recall or F1?," 2018. [Online]. Available: https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9. [Accessed 10 July 2020].

[75] C. Goutte and E. Gaussier, "A Probabilistic Interpretation of Precision, Recall and F-Score, with Implication for Evaluation," in *27th European conference on Advances in Information Retrieval Research*, Santiago de Compostela, 2005.

[76] C. J. v. Rijsbergen, "Evaluation," in *Information Retrieval*, London, Butterworths, 1979, pp. 112-140.

[77] Y. Sasaki, "The truth of the F-measure," 26 October 2007. [Online]. Available: https://www.toyota-ti.ac.jp/Lab/Denshi/COIN/people/yutaka.sasaki/F-measure-YS-26Oct07.pdf. [Accessed 10 July 2020].

[78] N. Chinchor, "MUC-4 EVALUATION METRICS," in *4th conference on Message understanding*, 1992.

[79] R. Al-Otaibi, M. Kull and P. Flach, "Declaratively Capturing Local Label Correlations and Multi-Label Trees," in *European Conference on Artificial Intelligence*, The Hague, 2016.

[80] Wikipedia, "F1 score," 8 July 2020. [Online]. Available: https://en.wikipedia.org/wiki/F1_score#Definition. [Accessed 10 July 2020].

[81] B. Shmueli, "Multi-Class Metrics Made Simple, Part II: the F1-score," Towards Data Science, 3 July 2019. [Online]. Available: https://towardsdatascience.com/multi-class-metrics-made-simple-part-ii-the-f1-score-ebe8b2c2ca1. [Accessed 18 June 2021].

[82] J. Brownlee, "A Gentle Introduction to Cross-Entropy for Machine Learning," 20 December 2019. [Online]. Available: https://machinelearningmastery.com/cross-entropy-for-machine-learning/. [Accessed 14 July 2020].

[83] Wikipedia, "Gibbs' inequality," 11 July 2019. [Online]. Available: https://en.wikipedia.org/wiki/Gibbs%27_inequality. [Accessed 14 July 2020].

[84] J. Shore and R. Johnson, "Properties of cross-entropy minimization," *IEEE Transactions on Information Theory,* vol. 27, no. 4, pp. 472-482, July 1981.

[85] I. Csiszar, "I -Divergence Geometry of Probability Distributions and Minimization Problems," *Annals of Probability,* vol. 3, no. 1, pp. 146-158, 1975.

[86] Read the Docs, "Loss Functions," 2017. [Online]. Available: https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html. [Accessed 14 July 2020].

[87] Z. Zhang and M. R. Sabuncu, "Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy Labels," in *Neural Information Processing Systems*, Montréal, 2018.

[88] D. M. Kline and V. L. Berardi, "Revisiting squared-error and cross-entropy functions for training neural network classifiers," *Neural Computing & Applications,* no. 14, pp. 310-318, 2005.

[89] A. Maiza, "The Unknown Benefits of using a Soft-F1 Loss in Classification Systems," 4 December 2019. [Online]. Available: https://towardsdatascience.com/the-unknown-benefits-of-using-a-soft-f1-loss-in-classification-systems-753902c0105d. [Accessed 13 July 2020].

[90] K. Dembczynski, A. Jachnik, W. Kotlowski, W. Waegeman and E. Hullermeier, "Optimizing the F-Measure in Multi-Label Classification: Plug-in Rule Approach versus Structured Loss Minimization," in *International Conference on Machine Learning*, Atlanta, 2013.

[91] N. Ye, K. M. A. Chai, W. S. Lee and H. L. Chieu, "Optimizing F-Measures: A Tale of Two Approaches," in *International Conference on Machine Learning*, Edinburgh, 2012.

[92] K. Dembczynski, W. Waegeman, W. Cheng and E. Hullermeier, "An Exact Algorithm for F-Measure Maximization," in *Neural Information Processing Systems*, Granada, 2011.

[93] M. Sun, T. X. Han, M.-C. Liu and A. Khodayari-Rostamabad, "Multiple Instance Learning Convolutional Neural Networks for Object Recognition," 11 October 2016. [Online]. Available: https://arxiv.org/pdf/1610.03155.pdf. [Accessed 5 April 2021].

[94] X. Jin, Y. Wang and X. Tan, "Pornographic Image Recognition via Weighted Multiple Instance Learning," *IEEE Transactions on Cybernetics,* vol. 99, pp. 1-9, 2018.

[95] W. Skarbek and A. Koschan, "Colour Image Segmentation - A Survey," October 1994. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=254252F3BC4F31D76F744F572E7E847C?doi=10.1.1.28.1325&rep=rep1&type=pdf. [Accessed 20 July 2020].

[96] M. H. Hesamian, W. Jia, X. He and P. Kennedy, "Deep Learning Techniques for Medical Image Segmentation: Achievements and Challenges," *Journal of Digital Imaging,* no. 32, p. 582–596, 19 May 2019.

[97] Papers with code, "Real-Time Semantic Segmentation," [Online]. Available: https://paperswithcode.com/task/real-time-semantic-segmentation/latest. [Accessed 20 July 2020].

[98] S. Minaee, Y. Boykov, F. Porikli, A. Plaza, N. Kehtarnavaz and D. Terzopoulos, "Image Segmentation Using Deep Learning: A Survwy," 10 April 2020. [Online]. Available: https://arxiv.org/pdf/2001.05566.pdf. [Accessed 15 May 2021].

[99] M. A. Rahman and Y. Wang, "Optimizing Intersection-Over-Union in Deep Neural Networks for Image Segmentation," December 2016. [Online]. Available: http://cs.umanitoba.ca/~ywang/papers/isvc16.pdf. [Accessed 16 July 2020].

[100] C. L. e. al., "Auto-DeepLab: Hierarchical Neural Architecture Search for Semantic Image Segmentation," in *Conference on Computer Vision and Pattern Recognition*, Long Beach, 2019.

[101] T. Birgui Sekou, M. Hidane, J. Olivier and H. Cardot, "From Patch to Image Segmentation using Fully Convolutional Networks - Application to Retinal Images," 18 June 2019. [Online]. Available: https://arxiv.org/pdf/1904.03892.pdf. [Accessed 15 May 2021].

[102] Wikipedia, "Jaccard index," 19 June 2020. [Online]. Available: https://en.wikipedia.org/wiki/Jaccard_index. [Accessed 16 July 2020].

[103] H. Rezatofigh et al., "Generalized Intersection over Union: A Metric and A Loss for Bounding Box Regression," 15 April 2019. [Online]. Available: https://arxiv.org/pdf/1902.09630.pdf. [Accessed 16 July 2020].

[104] F. v. Beers, A. Lindstrom, E. Okafor and M. A. Wiering, "Deep Neural Networks with Intersection over Union Loss for Binary Image Segmentation," in *International Conference on Pattern Recognition Applications and Methods*, Prague, 2019.

[105] p. (https://stats.stackexchange.com/users/157173/pietz), "F1/Dice-Score vs IoU," StackExchange, 25 May 2019. [Online]. Available: https://stats.stackexchange.com/q/273537. [Accessed 15 May 2021].

[106] E. Tiu, "Metrics to Evaluate your Semantic Segmentation Model," Towards Data Science, 10 August 2019. [Online]. Available: https://towardsdatascience.com/metrics-to-evaluate-your-semantic-segmentation-model-6bcb99639aa2. [Accessed 15 May 2021].

[107] w. (https://stats.stackexchange.com/users/159052/willem), "F1/Dice-Score vs IoU," StackExchange, 13 November 2017. [Online]. Available: https://stats.stackexchange.com/questions/273537/f1-dice-score-vs-iou/276144#276144. [Accessed 15 May 2021].

[108] C. H. Sudre, W. Li, T. Vercauteren, S. Ourselin and M. J. Cardoso, "Generalised Dice Overlap as a Deep Learning Loss Function for Highly Unbalanced Segmentations," in *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support*, Québec, 2017.

[109] L. Nieradzik, "Losses for Image Segmentation," 16 August 2019. [Online]. Available: https://lars76.github.io/neural-networks/object-detection/losses-for-segmentation/. [Accessed 20 July 2020].

[110] F. Milletari, N. Navab and S.-A. Ahmadi, "V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation," 15 June 2016. [Online]. Available: https://arxiv.org/abs/1606.04797. [Accessed 20 July 2020].

[111] S. S. M. Salehi, D. Erdogmus and A. Gholipour, "Tversky loss function for image segmentation using 3D fully convolutional deep networks," 18 June 2017. [Online]. Available: https://arxiv.org/abs/1706.05721. [Accessed 20 July 2020].

[112] T.-Y. Lin, P. Goyal, R. Girshick, K. He and P. Dollár, "Focal Loss for Dense Object Detection," 7 February 2018. [Online]. Available: https://arxiv.org/abs/1708.02002. [Accessed 20 July 2020].

[113] O. Ronneberger, P. Fischer and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," 18 May 2015. [Online]. Available: https://arxiv.org/pdf/1505.04597.pdf. [Accessed 15 July 2020].

[114] P. Yakubovskiy, *Segmentation Models,* GitHub, 2019.

[115] A. Budhiraja, "Dropout in (Deep) Machine learning," 15 December 2016. [Online]. Available: https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5. [Accessed 15 July 2020].

[116] A. Chaurasia and E. Culurciello, "LinkNet: Exploiting Encoder Representations for Efficient Semantic Segmentation," 14 June 2017. [Online]. Available: https://arxiv.org/pdf/1707.03718.pdf. [Accessed 15 July 2020].

[117] J. Long, E. Shelhamer and T. Darrel, "Fully Convolutional Networks for Semantic Segmentation," 8 March 2015. [Online]. Available: https://arxiv.org/pdf/1411.4038.pdf. [Accessed 15 July 2020].

[118] P. Sermanet et al., "OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks," 24 February 2014. [Online]. Available: https://arxiv.org/pdf/1312.6229.pdf. [Accessed 15 July 2020].

[119] nickhitsai, *LinkNet-Keras,* Github, 2017.

[120] D. Silva, *Keras-LinkNet,* Github, 2018.

[121] T.-Y. Lin et al., "Feature Pyramid Networks for Object Detection," in *Computer Vision and Pattern Recognition*, Honolulu, 2017.

[122] A. Kirillov, K. He, R. Girshick and P. Dollár, "A Unified Architecture for Instance and Semantic Segmentation," [Online]. Available: http://presentations.cocodataset.org/COCO17-Stuff-FAIR.pdf. [Accessed 16 July 2020].

[123] K. He, G. Gkioxari, P. Dollar and R. Girshick, "Mask R-CNN," 24 January 2018. [Online]. Available: https://arxiv.org/pdf/1703.06870.pdf. [Accessed 4 April 2021].

[124] S. Ren, K. He, R. Girshick and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," 6 January 2016. [Online]. Available: https://arxiv.org/pdf/1506.01497.pdf. [Accessed 4 April 2021].

[125] kaggle, "Toxic Comment Classification Challenge," 20 March 2018. [Online]. Available: https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge. [Accessed 23 July 2020].

[126] Facebook AI, "Hateful Memes Challenge and data set for research on harmful multimodal content," 12 May 2020. [Online]. Available: https://ai.facebook.com/blog/hateful-memes-challenge-and-data-set/. [Accessed 23 July 2020].

[127] S. Avila, E. Valle and A. d. A. Araújo, "Pornography Database," February 2018. [Online]. Available: https://sites.google.com/site/pornographydatabase/. [Accessed 23 July 2020].

[128] C. Demarty, C. Penet, M. Soleymani and G. Gravier, "The MediaEval 2014 Affect Task: Violent Scenes Detection," in *MediaEval 2014 Workshop*, Barcelona, Spain, 2014.

[129] C.-H. Demarty, C. Penet, M. Schedl, I. Bogdan, V. L. Quang and Y.-G. Jiang, "The MediaEval 2013 Affect Task: Violent Scenes Detection," *MediaEval 2013 Working Notes,* p. 2, October 2013.

[130] S. e. al., "WORKFLOW SYSTEM FOR DETECTION AND CLASSIFICATION OF IMAGES SUSPECTED AS PORNOGRAPHIC". United States Patent 6,904,168 B1, 7 June 2005.

[131] J. Rondeau, "Deep Learning of Human Apparent Age for the Detection of Sexually Exploitative Imagery of Children," *Open Access Master's Theses,* vol. Paper 1438, 2019.

[132] H. Zuo, W. Hu and O. Wu, "Patch-Based Skin Color Detection and Its Application to," 2010. [Online]. Available: https://www.researchgate.net/publication/221023098_Patch-based_skin_color_detection_and_its_application_to_pornography_image_filtering. [Accessed 5 January 2021].

[133] A. Zaidan, N. Ahmad, H. A. Karim, M. Larbani, B. Zaidan and A. Sali, "On the multi-agent learning neural and Bayesian methods," *Neurocomputing,* no. 131, pp. 397-418, 2014.

[134] S. Dewantono and I. Supriana, "Development of A Real-Time Nudity Censorship System on Images," in *International Conference on Information and Communication Technology (ICoICT)*, Bandung, 2014.

[135] H. A. Nugroho, F. Rahadian, T. B. Adji, W. K. Oktoeberza and R. L. Budiani Buana, "Texture Analysis for Skin Classification in Pornography Content Filtering Based on Support Vector Machine," *Journal of Engineering and Technological Sciences,* vol. 48, no. 5, pp. 584-596, 2016.

[136] J. Zhang, L. Sui, L. Zhuo, Z. Li and Y. Yang, "An approach of bag-of-words based on visual attention model for pornographic images recognition in compressed domain," *Neurocomputing,* vol. 110, p. 145–152, 2013.

[137] N. Sae-Bae, X. Sun, H. T. Sencar and N. D. Memon, "Towards automatic detection of child pornography," in *IEEE International Conference on Image Processing*, Paris, 2014.

[138] W. Xu, H. Parvin and H. Izadparast, "Deep Learning Neural Network for Unconventional Images Classification," *Neural Processing Letters,* vol. 52, p. 169–185, 2020.

[139] F. Nian, T. Li, Y. Wang, M. Xu and J. Wu, "Pornographic image detection utilizing deep convolutional neural networks," *Neurocomputing,* vol. 210, pp. 283-293, 2016.

[140] M. Sun, T. X. Han, M.-C. Liu and A. Khodayari-Rostamabad, "Multiple Instance Learning Convolutional Neural Networks for Object Recognition," in *International Conference on Pattern Recognition*, Cancún, 2016.

[141] C. Jansohn, A. Ulges and T. M. Breuel, "Detecting Pornographic Video Content by Combining Image Features with Motion Information," in *International Conference on Multimedia 2009*, Vancouver, 2009.

[142] M. Perez et al., "Video pornography detection through deep learning techniques and motion information," *Neurocomputing,* vol. 230, pp. 279-293, 2017.

[143] C. Caetano, S. Avila, W. R. Schwartz, S. J. F. Guimarães and A. d. A. Araújo, "A mid-level video representation based on binary descriptors: A case study for pornography detection," *Neurocomputing,* vol. 213, pp. 102-114, 2016.

[144] S. V. Varghese, T. Thomas and A. Vijayan, "Immoral Scene Censoring System," *International Journal of Science and Research,* vol. 4, p. 1, 2015.

[145] M. Gruosso, N. Capece, U. Erra and N. Lopardo, "A Deep Learning approach for the Motion Picture Content Rating," in *IEEE International Conference on Cognitive Infocommunications*, Naples, 2019.

[146] C.-H. Demarty, B. Ionescu, Y.-G. Jiang, V. L. Quang, M. Schedl and C. Penet, "Benchmarking Violent Scenes Detection in movies," in *International Workshop on Content-Based Multimedia Indexing*, Klagenfurt, 2014.

[147] M. Constantin et al., "Affect in Multimedia: Benchmarking Violent Scenes Detection," *IEEE Transactions on Affective Computing (Early Access),* pp. 1-1, 2020.

[148] Motion Picture Association, Inc., "Film ratings," 2021. [Online]. Available: https://www.motionpictures.org/film-ratings/. [Accessed 19 May 2021].

[149] MOTION PICTURE ASSOCIATION – CANADA, "Film ratings," 2021. [Online]. Available: https://www.mpa-canada.org/film-ratings/. [Accessed 19 May 2021].

[150] S. U. Khan, I. Ul Haq, S. Rho, S. W. Baik and M. Y. Lee, "Cover the Violence: A Novel Deep-Learning-Based Approach Towards Violence-Detection in Movies," *Applied Sciences,* vol. 9, no. 22, p. 4963, 2019.

[151] E. Nievas, O. Suarez, G. García and R. Sukthankar, "Violence detection in video using computer vision," in *International conference on Computer Analysis of Images and Patterns*, Berlin/Heidelberg, 2011.

[152] T. Hassner, Y. Itcher and O. Kliper-Gross, "Violent flows: Real-time detection of violent crowd behavior," in *Proceedings of the 2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Providence, RI, USA, 2012.

[153] W. Song, D. Zhang, X. Zhao, J. Yu, R. Zheng and A. Wang, "A Novel Violent Video Detection Scheme Based on Modified 3D Convolutional Neural Networks," *IEEE Access,* vol. 7, pp. 39172 - 39179, 2019 .

[154] C. Gu, X. Wu and S. Wang, "Violent Video Detection Based on Semantic Correspondence," *IEEE Access,* vol. 8, pp. 85958 - 85967, 2020.

[155] G. Mu, H. Cao and Q. Jin, "Violent Scene Detection Using Convolutional Neural Networks and Deep Audio Features," in *Chinese Conference Pattern Recognition*, Chengdu, China, 2016.

[156] Pedro V. A. de Freitas et al., "A Multimodal CNN-based Tool to Censure Inappropriate Video Scenes," 10 November 2019. [Online]. Available: https://arxiv.org/abs/1911.03974. [Accessed 17 January 2020].

[157] J. Mallmann, A. O. Santin, E. Kugler Viegas, R. R. dos Santos and J. Geremias, "PPCensor: Architecture for real-time pornography detection in video streaming," *Future Generation Computer Systems,* vol. 112, pp. 945-955, 2020.

[158] A. Joseph, T. A. Chacko, J. Jose and D. Sunny, "Automatic Image Detection and Censoring on Web Pages," *International Journal of Innovative Science and Research Technology,* vol. 5, no. 6, pp. 891-894, 2020.

[159] A. Dolgushev and J. A. Brown, "Deep Puff: Censoring via Machine Learning Cigarette Use," in *International Conference on Developments in eSystems Engineering*, Kazan, 2019.

[160] G. S. Simoes, J. Wehrmann and R. C. Barros, "Attention-based Adversarial Training for Seamless Nudity Censorship," in *International Joint Conference on Neural Networks*, Budapest, 2019.

[161] F. J. Veredas, R. M. Luque-Baena, F. J. Martín-Santos, J. C. Morilla-Herrera and L. Morente, "Wound image evaluation with machine learning," *Neurocomputing,* no. 164, pp. 112-122, 2015.

[162] D. P. Ortiz, D. Sierra-Sosa and B. García Zapirain, "Pressure ulcer image segmentation technique through synthetic frequencies generation and contrast variation using toroidal geometry," *BioMedical Engineering OnLine,* vol. 16, no. 1, 2017.

[163] Chuanbo Wang et al., "Fully automatic wound segmentation with deep convolutional neural networks," *Scientific Reports,* vol. 10, no. 1, 2020.

[164] Daniel Y.T. Chino et al., "Segmenting skin ulcers and measuring the wound area using deep convolutional networks," *Computer Methods and Programs in Biomedicine,* vol. 191, 2020.

[165] G. Litjens et al., "A survey on deep learning in medical image analysis," *Medical Image Analysis,* vol. 42, pp. 60-88, 2017.

[166] B. Song and A. Sacan, "Automated Wound Identification System Based on Image Segmentation and Artificial Neural Networks," in *IEEE International Conference on Bioinformatics and Biomedicine*, Philadelphia, 2012.

[167] F. Li, C. Wang, X. Liu, Y. Peng and S. Jin, "A Composite Model of Wound Segmentation Based on Traditional Methods and Deep Neural Networks," *Computational Intelligence and Neuroscience,* pp. 1-12, 2018.

[168] X. Liu, C. Wang, F. Li, X. Zhao, E. Zhu and Y. Peng, "A Framework of Wound Segmentation Based on Deep Convolutional Networks," in *International Congress on Image and Signal Processing, BioMedical Engineering and Informatics*, Shanghai, 2017.

[169] Q. Xiang, X. Wang, R. Li, G. Zhang, J. Lai and Q. Hu, "Fruit Image Classification Based on MobileNetV2 with Transfer Learning Technique," in *International Conference on Computer Science and Application Engineering*, Sanya, 2019.

[170] M. A. e. al., "Deep Learning Classification of Systemic Sclerosis Skin Using the MobileNetV2 Model," *IEEE Open Journal of Engineering in Medicine and Biology,* vol. 2, pp. 104-110, 2021.

[171] TensorFlow, "Transfer learning and fine-tuning," 11 November 2021. [Online]. Available: https://www.tensorflow.org/tutorials/images/transfer_learning. [Accessed 11 November 2021].

[172] C. H. Karadal et al., "Automated classification of remote sensing images using multileveled MobileNetV2 and DWT techniques," *Expert Systems with Applications,* vol. 185, 15 December 2021.

[173] D. Godoy, "Understanding binary cross-entropy / log loss: a visual explanation," Towards Data Science, 21 November 2018. [Online]. Available: https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a. [Accessed 11 11 2021].

[174] O. Sheremet, "Intersection over union (IoU) calculation for evaluating an image segmentation model," Towards Data Science, 24 July 2020. [Online]. Available: https://towardsdatascience.com/intersection-over-union-iou-calculation-for-evaluating-an-image-segmentation-model-8b22e2e84686. [Accessed 11 November 2021].

[175] C. Liu et al., "Auto-DeepLab: Hierarchical Neural Architecture Search for Semantic Image Segmentation," in *Computer Vision and Pattern Recognition*, Long Beach, 2019.

[176] C. Wang et al., "Fully automatic wound segmentation with deep convolutional neural networks," *Scientific Reports,* vol. 10, no. 21897, 2020.

[177] Iyadurai, L. et al., "Preventing intrusive memories after trauma via a brief intervention involving Tetris computer game play in the emergency department: a proof-of-concept randomized controlled trial," *Molecular Psychiatry,* no. 23, p. 674–682, 2018.

[178] M. A. Hagenaars, E. A. Holmes, F. Klaassen and B. Elzinga, "Tetris and Word games lead to fewer intrusive memories when applied several days," *European Journal of Psychotraumatology,* vol. 8, no. sup1, 2017.

[179] A. Dutta, "VGG Image Annotator," Oxford University, 2018. [Online]. Available: http://www.robots.ox.ac.uk/~vgg/software/via/via-1.0.6.html. [Accessed 08 September 2020].

[180] A. C. a. Contributors, "ImageDraw Module," Pillow, [Online]. Available: https://pillow.readthedocs.io/en/stable/reference/ImageDraw.html. [Accessed 19 May 2021].

[181] J. Brownlee, "How to use Data Scaling Improve Deep Learning Model Stability and Performance," Machine Learning Mastery, 4 February 2019. [Online]. Available: https://machinelearningmastery.com/how-to-improve-neural-network-stability-and-modeling-performance-with-data-scaling/. [Accessed 19 May 2021].

[182] W. Abdulla, "Mask R-CNN for object detection and instance segmentation on Keras and TensorFlow," Github, 2017. [Online]. Available: https://github.com/matterport/Mask_RCNN. [Accessed 17 February 2021].

[183] Scholarly Communications and Copyright Office, "Theses and Dissertations," The University of British Columbia, [Online]. Available: https://copyright.ubc.ca/theses-and-dissertations/. [Accessed 15 May 2021].

[184] S. Thomas, "Medetec Wound Database: stock pictures of wounds," [Online]. Available: http://www.medetec.co.uk/files/medetec-image-databases.html. [Accessed 17 January 2021].

[185] A. Nguyen, A. Oberföll and M. Färber, "Right for the Right Reasons: Making Image Classification Intuitively Explainable," 12 January 2021. [Online]. Available: https://arxiv.org/pdf/2007.11924.pdf. [Accessed 18 May 2021].

[186] TensorFlow, "tf.keras.preprocessing.image.ImageDataGenerator," 12 September 2020. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator. [Accessed 12 September 2020].

[187] B. Chen, "Early Stopping in Practice: an example with Keras and TensorFlow 2.0," Towards Data Science, 28 July 2020. [Online]. Available: https://towardsdatascience.com/a-practical-introduction-to-early-stopping-in-machine-learning-550ac88bc8fd. [Accessed 19 May 2021].

[188] M. S. Nikulin, "Student test. Encyclopedia of Mathematics.," EMS Press, 2001. [Online]. Available: http://encyclopediaofmath.org/index.php?title=Student_test&oldid=48883. [Accessed 12 June 2021].

[189] TensorFlow, "core.py," Github repository, 23 March 2021. [Online]. Available: https://github.com/tensorflow/tensorflow/blob/v2.5.0/tensorflow/python/keras/layers/core.py. [Accessed 18 May 2021].

[190] L. I. Kuncheva, L. I. Kuncheva, J. J. Rodríguez and J. J. Rodríguez, "A weighted voting framework for classifiers ensembles," *Knowledge and information systems,* vol. 38, no. 2, pp. 259-275, 2014.

[191] A. Dogan and D. Birant, "A Weighted Majority Voting Ensemble Approach for Classification," in *International Conference on Computer Science and Engineering*, Samsun, 2019.

[192] M. Kana, "Uncertainty in Deep Learning. How To Measure?," 26 April 2020. [Online]. Available: https://towardsdatascience.com/my-deep-learning-model-says-sorry-i-dont-know-the-answer-that-s-absolutely-ok-50ffa562cb0b. [Accessed 26 October 2020].

[193] B. Everitt and A. Skrondal, Cambridge Dictionary of Statistics, Cambridge: Cambridge University Press, 2010.

[194] A. Kausar, "Dealing with Small Datasets — Get More From Less — TensorFlow 2.0 — Part 1," 27 April 2020. [Online]. Available: https://aqsakausar30.medium.com/dealing-with-small-datasets-get-more-from-less-tensorflow-2-0-10472e65a7f6. [Accessed 18 May 2021].

[195] W. W. LaMorte, "Central Limit Theorem," Boston University School of Public Health, 24 June 2016. [Online]. Available: https://sphweb.bumc.bu.edu/otlt/MPH-Modules/BS/BS704_Probability/BS704_Probability12.html#:~:text=The%20central%20limit%20theorem%20states,will%20be%20approximately%20normally%20distributed.. [Accessed 11 May 2021].

[196] Google Cloud, "Package google.cloud.vision.v1," 12 March 2021. [Online]. Available: https://cloud.google.com/vision/docs/reference/rpc/google.cloud.vision.v1. [Accessed 07 May 2021].

[197] Google Cloud, "Detect explicit content (SafeSearch)," 04 May 2021. [Online]. Available: https://cloud.google.com/vision/docs/detecting-safe-search. [Accessed 2021 May 07].

[198] H. Zhao, J. Shi, X. Qi, X. Wang and J. Jia, "Pyramid Scene Parsing Network," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, 2017.

[199] N. Siddique, S. Paheding, C. Elkin and V. Devabhaktuni, "U-Net and its variants for medical image segmentation: theory and applications," 2 November 2020. [Online]. Available: https://arxiv.org/ftp/arxiv/papers/2011/2011.01118.pdf. [Accessed 3 August 2021].

[200] C. Wang et al., "A unified framework for automatic wound segmentation and analysis with deep convolutional neural networks," in *International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, Milan, 2015.

[201] S. Jukna, "Entropy Function," in *Extremal Combinatorics*, Berlin , Springer-Verlag, 2011, pp. 313-326.

[202] Wikipedia, "Binary entropy function," 20 May 2019. [Online]. Available: https://en.wikipedia.org/wiki/Binary_entropy_function. [Accessed 17 January 2021].

[203] TensorFlow, "Module: tf.image," 14 December 2020. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/image. [Accessed 17 January 2021].

[204] P. Yakubovskiy, "Segmentation Models Python API," 2018. [Online]. Available: https://segmentation-models.readthedocs.io/en/latest/api.html#pspnet. [Accessed 16 June 2021].

[205] TensorFlow, "tf.keras.applications.resnet.ResNet152," 15 June 2021. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/applications/resnet/ResNet152. [Accessed 16 June 2021].

[206] M. Tan and Q. V. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," [Online]. Available: https://arxiv.org/pdf/1905.11946.pdf.

[207] J. Brownlee, "18 Impressive Applications of Generative Adversarial Networks (GANs)," Machine Learning Mastery, 14 June 2019. [Online]. Available:

https://machinelearningmastery.com/impressive-applications-of-generative-adversarial-networks/. [Accessed 18 May 2021].

[208] P. Isola, J.-Y. Zhu, T. Zhou and A. A. Defros, "Image-to-Image Translation with Conditional Adversarial Networks," 26 November 2018. [Online]. Available: https://arxiv.org/pdf/1611.07004.pdf. [Accessed 18 May 2021].

[209] TensorFlow, "Pix2Pix," 31 March 2021. [Online]. Available: https://www.tensorflow.org/tutorials/generative/pix2pix. [Accessed 18 May 2021].

[210] Immersive Limit LLC, "Mask R-CNN with TensorFlow 2 on Windows 10," 19 June 2020. [Online]. Available: https://www.immersivelimit.com/tutorials/mask-rcnn-for-windows-10-tensorflow-2-cuda-101. [Accessed 13 May 2021].

[211] Facebook Research, "Mask R-CNN Benchmark," Github repository., 20 November 2019. [Online]. Available: https://github.com/facebookresearch/maskrcnn-benchmark. [Accessed 13 May 2021].

[212] Image Processing, "Nearest Neighbor Interpolation," November 2017. [Online]. Available: https://www.imageeprocessing.com/2017/11/nearest-neighbor-interpolation.html. [Accessed 18 May 2021].

[213] "DIGITAL IMAGE INTERPOLATION," Cambridge in Colour, [Online]. Available: https://www.cambridgeincolour.com/tutorials/image-interpolation.htm. [Accessed 18 May 2021].

[214] BentoML, "Getting Started," 2021. [Online]. Available: https://docs.bentoml.org/en/latest/quickstart.html. [Accessed 23 May 2021].

[215] BentoML, "Deployment Guides," 2021. [Online]. Available: https://docs.bentoml.org/en/latest/deployment/index.html#. [Accessed 23 May 2021].

[216] BentoML, "API InputAdapters," 2021. [Online]. Available: https://docs.bentoml.org/en/latest/api/adapters.html#multifileinput. [Accessed 23 May 2021].

[217] TensorFlow, "tf.keras.layers.MaxPool2D," 30 June 2020. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/layers/MaxPool2D. [Accessed 7 July 2020].

[218] TensorFlow, "tf.keras.layers.Conv2DTranspose," 30 June 2020. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2DTranspose. [Accessed 7 July 2020].

[219] TensorFlow, "tf.keras.layers.Conv2D," 30 June 2020. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2D. [Accessed 7 July 2020].

[220] TensorFlow, "tf.keras.layers.Activation," 30 June 2020. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Activation. [Accessed 7 July 2020].

[221] J. Brownlee, "How to use the UpSampling2D and Conv2DTranspose Layers in Keras," 12 July 2019. [Online]. Available: https://machinelearningmastery.com/upsampling-and-transpose-convolution-layers-for-generative-adversarial-networks/. [Accessed 7 July 2020].

[222] J. Brownlee, "A Gentle Introduction to Pooling Layers for Convolutional Neural Networks," 5 July 2019. [Online]. Available: https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/. [Accessed 7 July 2020].

[223] J. Brownlee, "A Gentle Introduction to Padding and Stride for Convolutional Neural Networks," 19 April 2019. [Online]. Available: https://machinelearningmastery.com/padding-and-stride-for-convolutional-neural-networks/. [Accessed 7 July 2020].

[224] M. Guerzhoy, "Upsampling and Interpolation," [Online]. Available: https://www.cs.toronto.edu/~guerzhoy/320/lec/upsampling.pdf. [Accessed 7 July 2020].

[225] A. Lerch, "Appendix A Convolution Properties," Wiley, 30 July 2012. [Online]. Available: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781118393550.app1. [Accessed 7 July 2020].

[226] C. Tomasi, "Image Correlation, Convolution and Filtering," 2015. [Online]. Available: https://www2.cs.duke.edu/courses/fall15/compsci527/notes/convolution-filtering.pdf. [Accessed 7 July 2020].

[227] Wikipedia, "Kernel (image_processing)," 23 June 2020. [Online]. Available: https://en.wikipedia.org/wiki/Kernel_(image_processing). [Accessed 7 July 2020].

[228] Wikipedia, "Convolution," 2 July 2020. [Online]. Available: https://en.wikipedia.org/wiki/Convolution. [Accessed 7 July 2020].

[229] D. Mishra, "Convolution Vs Correlation," 13 November 2019. [Online]. Available: https://towardsdatascience.com/convolution-vs-correlation-af868b6b4fb5. [Accessed 7 July 2020].

[230] C.-F. Wang, "A Basic Introduction to Separable Convolutions," 13 August 2018. [Online]. Available: https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728. [Accessed 9 July 2020].

[231] S. Kullback, Information Theory and Statistics, New York: Wiley, 1959.

[232] N. R. Pal, "On Minimum Cross-Entropy Thresholding," *Pattern Recognition,* vol. 29, no. 4, pp. 575-580, 1996.

[233] Y. Ioannou, "A Tutorial on Filter Groups (Grouped Convolution)," 10 August 2017. [Online]. Available: https://blog.yani.io/filter-group-tutorial/. [Accessed 22 July 2020].

[234] A. Kim, "NSFW Data Scraper," 07 May 2020. [Online]. Available: https://github.com/alex000kim/nsfw_data_scraper. [Accessed 08 September 2020].

[235] O. Ronneberger, P. Fischer and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*, Munich, 2015.

[236] P. Korus, W. Szmuc and A. Dziech, "A scheme for censorship of sensitive image content with high-quality reconstruction ability," in *IEEE International Conference on Multimedia and Expo*, Suntec City, 2010.

# Appendix A – Proof of equivalence of the F-Score

$$F_\beta := \frac{(1+\beta^2) * precision * recall}{\beta^2 * precision + recall} \tag{2.7}$$

$$= \frac{(1+\beta^2) * {tp}/{tp+fp} * {tp}/{tp+fn}}{\beta^2 * {tp}/{tp+fp} + {tp}/{tp+fn}}$$

$$= \frac{(1+\beta^2) * {tp^2}/{(tp+fp)*(tp+fn)}}{\beta^2 * tp * (tp+fn) + tp * (tp * fp) \,/\, (tp+fp)*(tp+fn)}$$

$$= \frac{(tp+fp)*(tp+fn)}{(tp+fp)*(tp+fn)} * \frac{(1+\beta^2) * tp^2}{\beta^2 * tp * (tp+fn) + tp * (tp * fp)}$$

$$= \frac{tp}{tp} * \frac{(1+\beta^2) * tp}{\beta^2 * (tp+fn) + (tp+fp)}$$

$$= \frac{(1+\beta^2) * tp}{(1+\beta^2) * tp + \beta^2 * fn + fp} \tag{2.8}$$

$$= \frac{1+\beta^2}{1/tp * (\beta^2 * (tp+fn) + (tp+fp))}$$

$$= \frac{1+\beta^2}{\beta^2 * \frac{tp+fn}{tp} + \frac{tp+fp}{tp}}$$

$$= \frac{1+\beta^2}{\beta^2 * \left(\frac{tp}{tp+fn}\right)^{-1} + \left(\frac{tp}{tp+fp}\right)^{-1}}$$

$$= \frac{1+\beta^2}{\beta^2 * recall^{-1} + precision^{-1}} \tag{2.9}$$

■

# Appendix B – Pipeline Testing Results

### B1 - Classification

| Classification model | Segmentation model | Threshold | F1-Score | Double F1-Score | Precision | Recall | Note |
|---|---|---|---|---|---|---|---|
| Voting_WRL (No threshold prior to voting) | FPN24 ResNet152 (Table 22) | 0.5 | 0.8883 | 0.9187 | 0.8811 | 0.8956 | Control (0.5 threshold no seg) |
| | | 0.5 | 0.7857 | 0.8547 | 0.9603 | 0.6648 | 0.5 threshold (w/ seg @ same class threshold) |
| | | 0.5 (+1.6 std) | 0.7796 | 0.8496 | 0.9313 | 0.6703 | With uncertainty std dev used to improve recall |
| | | 0.5 (+3.2 std) | 0.7785 | 0.8484 | 0.9179 | 0.6758 | With uncertainty std dev used to improve recall |
| | | 0.25 | 0.7810 | 0.8502 | 0.9248 | 0.6758 | 0.95 recall (w/ seg @ same class threshold) |
| | | 0.05 | 0.7688 | 0.8409 | 0.8913 | 0.6758 | 0.99 recall (w/ seg @ same class threshold) |
| Voting_Mixed (No threshold prior to voting) | | 0.5 | 0.8919 | 0.9210 | 0.8777 | 0.9066 | Control (0.5 threshold no seg) |
| | | 0.5 | 0.7871 | 0.8553 | 0.9531 | 0.6703 | 0.5 threshold (w/ seg @ same class threshold) |
| | | 0.5 (+1.7 std) | 0.7771 | 0.8477 | 0.9242 | 0.6703 | With uncertainty std dev used to improve recall |
| | | 0.5 (+3.3 std) | 0.7722 | 0.8440 | 0.9104 | 0.6703 | With uncertainty std dev used to improve recall |
| | | 0.24 | 0.7785 | 0.8484 | 0.9179 | 0.6758 | 0.95 recall (w/ seg @ same class threshold) |
| | | 0.03 | 0.7688 | 0.8409 | 0.8913 | 0.6758 | 0.99 recall (w/ seg @ same class threshold) |
| Voting_WRL (No threshold prior to voting) | FPN24 DenseNet169 (Table 22) | 0.5 | 0.8883 | 0.9187 | 0.8811 | 0.8956 | Control (0.5 threshold no seg) |
| | | 0.5 | 0.8373 | 0.8865 | 0.9267 | 0.7637 | 0.5 threshold (w/ seg @ same class threshold) |
| | | 0.5 (+1.6 std) | 0.8319 | 0.8817 | 0.8981 | 0.7747 | With uncertainty std dev used to improve recall |
| | | 0.5 (+3.2 std) | 0.8290 | 0.8788 | 0.8773 | 0.7857 | With uncertainty std dev used to improve recall |
| | | 0.25 | 0.8280 | 0.8784 | 0.8820 | 0.7802 | 0.95 recall (w/ seg @ same class threshold) |
| | | 0.05 | 0.8295 | 0.8782 | 0.8588 | 0.8022 | 0.99 recall (w/ seg @ same class threshold) |
| | | 0.5 | 0.8919 | 0.9210 | 0.8777 | 0.9066 | Control (0.5 threshold no seg) |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Voting_Mixed (No threshold prior to voting) | | 0.5 | 0.8468 | 0.8930 | 0.9338 | 0.7747 | 0.5 threshold (w/ seg @ same class threshold) |
| | | 0.5 (+1.7 std) | 0.8402 | 0.8877 | 0.9103 | 0.7802 | With uncertainty std dev used to improve recall |
| | | 0.5 (+3.3 std) | 0.8266 | 0.8770 | 0.8720 | 0.7857 | With uncertainty std dev used to improve recall |
| | | 0.24 | 0.8256 | 0.8766 | 0.8765 | 0.7802 | 0.95 recall (w/ seg @ same class threshold) |
| | | 0.03 | 0.8272 | 0.8764 | 0.8538 | 0.8022 | 0.99 recall (w/ seg @ same class threshold) |
| Voting_WRL (No threshold prior to voting) | FPN24 SE-ResNet152 (Table 22) | 0.5 | 0.8883 | 0.9187 | 0.8811 | 0.8956 | Control (0.5 threshold no seg) |
| | | 0.5 | 0.8188 | 0.8753 | 0.9493 | 0.7198 | 0.5 threshold (w/ seg @ same class threshold) |
| | | 0.5 (+1.6 std) | 0.8111 | 0.8696 | 0.9291 | 0.7198 | With uncertainty std dev used to improve recall |
| | | 0.5 (+3.2 std) | 0.8097 | 0.8674 | 0.8993 | 0.7363 | With uncertainty std dev used to improve recall |
| | | 0.25 | 0.8146 | 0.8711 | 0.9116 | 0.7363 | 0.95 recall (w/ seg @ same class threshold) |
| | | 0.05 | 0.8071 | 0.8646 | 0.8774 | 0.7473 | 0.99 recall (w/ seg @ same class threshold) |
| Voting_Mixed (No threshold prior to voting) | | 0.5 | 0.8919 | 0.9210 | 0.8777 | 0.9066 | Control (0.5 threshold no seg) |
| | | 0.5 | 0.8162 | 0.8734 | 0.9424 | 0.7198 | 0.5 threshold (w/ seg @ same class threshold) |
| | | 0.5 (+1.7 std) | 0.8123 | 0.8701 | 0.9231 | 0.7253 | With uncertainty std dev used to improve recall |
| | | 0.5 (+3.3 std) | 0.8097 | 0.8674 | 0.8993 | 0.7363 | With uncertainty std dev used to improve recall |
| | | 0.24 | 0.8085 | 0.8669 | 0.9048 | 0.7308 | 0.95 recall (w/ seg @ same class threshold) |
| | | 0.03 | 0.8071 | 0.8646 | 0.8774 | 0.7473 | 0.99 recall (w/ seg @ same class threshold) |
| Voting_WRL (No threshold prior to voting) | FPN24 EfficientNetB7 (Table 22) | 0.5 | 0.8883 | 0.9187 | 0.8811 | 0.8956 | Control (0.5 threshold no seg) |
| | | 0.5 | 0.8485 | 0.8946 | 0.9459 | 0.7692 | 0.5 threshold (w/ seg @ same class threshold) |
| | | 0.5 (+1.6 std) | 0.8353 | 0.8840 | 0.8987 | 0.7802 | With uncertainty std dev used to improve recall |
| | | 0.5 (+3.2 std) | 0.8357 | 0.8833 | 0.8788 | 0.7967 | With uncertainty std dev used to improve recall |
| | | 0.25 | 0.8406 | 0.8870 | 0.8896 | 0.7967 | 0.95 recall (w/ seg @ same class threshold) |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | 0.05 | 0.8329 | 0.8804 | 0.8596 | 0.8077 | 0.99 recall (w/ seg @ same class threshold) |
| Voting_Mixed (No threshold prior to voting) | | 0.5 | 0.8919 | 0.9210 | 0.8777 | 0.9066 | Control (0.5 threshold no seg) |
| | | 0.5 | 0.8554 | 0.8991 | 0.9467 | 0.7802 | 0.5 threshold (w/ seg @ same class threshold) |
| | | 0.5 (+1.7 std) | 0.8462 | 0.8919 | 0.9167 | 0.7857 | With uncertainty std dev used to improve recall |
| | | 0.5 (+3.3 std) | 0.8391 | 0.8856 | 0.8795 | 0.8022 | With uncertainty std dev used to improve recall |
| | | 0.24 | 0.8348 | 0.8829 | 0.8834 | 0.7912 | 0.95 recall (w/ seg @ same class threshold) |
| | | 0.03 | 0.8329 | 0.8804 | 0.8596 | 0.8077 | 0.99 recall (w/ seg @ same class threshold) |
| Voting_WRL (No threshold prior to voting) | GAN Improved segmentation model (Table 24 – Run 4) | 0.5 | 0.8883 | 0.9187 | 0.8811 | 0.8956 | Control (0.5 threshold no seg) |
| | | 0.5 | 0.8101 | 0.8700 | 0.9552 | 0.7033 | 0.5 threshold (w/ seg @ same class threshold) |
| | | 0.5 (+1.6 std) | 0.8012 | 0.8630 | 0.9214 | 0.7088 | With uncertainty std dev used to improve recall |
| | | 0.5 (+3.2 std) | 0.8049 | 0.8645 | 0.9041 | 0.7253 | With uncertainty std dev used to improve recall |
| | | 0.25 | 0.8073 | 0.8664 | 0.9103 | 0.7253 | 0.95 recall (w/ seg @ same class threshold) |
| | | 0.05 | 0.8036 | 0.8632 | 0.8926 | 0.7308 | 0.99 recall (w/ seg @ same class threshold) |
| Voting_Mixed (No threshold prior to voting) | | 0.5 | 0.8919 | 0.9210 | 0.8777 | 0.9066 | Control (0.5 threshold no seg) |
| | | 0.5 | 0.8139 | 0.8724 | 0.9556 | 0.7088 | 0.5 threshold (w/ seg @ same class threshold) |
| | | 0.5 (+1.7 std) | 0.8100 | 0.8691 | 0.9353 | 0.7143 | With uncertainty std dev used to improve recall |
| | | 0.5 (+3.3 std) | 0.8085 | 0.8669 | 0.9048 | 0.7308 | With uncertainty std dev used to improve recall |
| | | 0.24 | 0.8012 | 0.8622 | 0.9034 | 0.7198 | 0.95 recall (w/ seg @ same class threshold) |
| | | 0.03 | 0.8036 | 0.8632 | 0.8926 | 0.7308 | 0.99 recall (w/ seg @ same class threshold) |
| Voting_WRL (No threshold prior to voting) | FPN24 EfficientNetB7 (Table 27 - Without Training Augmentation) | 0.5 | 0.8883 | 0.9187 | 0.8811 | 0.8956 | Control (0.5 threshold no seg) |
| | | 0.5 | 0.8344 | 0.8853 | 0.9444 | 0.7473 | 0.5 threshold (w/ seg @ same class threshold) |
| Voting_Mixed (No threshold prior to voting) | | 0.5 | 0.8919 | 0.9210 | 0.8777 | 0.9066 | Control (0.5 threshold no seg) |
| | | 0.5 | 0.8379 | 0.8876 | 0.9448 | 0.7527 | 0.5 threshold (w/ seg @ same class threshold) |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Voting_WRL (No threshold prior to voting) | Mask R-CNN ResNet50 (Table 25) | 0.5 | 0.8883 | 0.9187 | 0.8811 | 0.8956 | Control (0.5 threshold no seg) |
| | | 0.5 | 0.4898 | 0.6775 | 0.9524 | 0.3297 | 0.5 threshold (w/ seg @ same class threshold) |
| | | 0.5 (+1.6 std) | 0.4858 | 0.6743 | 0.9231 | 0.3297 | With uncertainty std dev used to improve recall |
| | | 0.5 (+3.2 std) | 0.4880 | 0.6746 | 0.8971 | 0.3352 | With uncertainty std dev used to improve recall |
| | | 0.25 | 0.4900 | 0.6762 | 0.9104 | 0.3352 | 0.95 recall (w/ seg @ same class threshold) |
| | | 0.05 | 0.4882 | 0.6733 | 0.8611 | 0.3407 | 0.99 recall (w/ seg @ same class threshold) |
| Voting_Mixed (No threshold prior to voting) | | 0.5 | 0.8919 | 0.9210 | 0.8777 | 0.9066 | Control (0.5 threshold no seg) |
| | | 0.5 | 0.4898 | 0.6775 | 0.9524 | 0.3297 | 0.5 threshold (w/ seg @ same class threshold) |
| | | 0.5 (+1.7 std) | 0.4878 | 0.6759 | 0.9375 | 0.3297 | With uncertainty std dev used to improve recall |
| | | 0.5 (+3.3 std) | 0.4900 | 0.6762 | 0.9104 | 0.3352 | With uncertainty std dev used to improve recall |
| | | 0.24 | 0.4839 | 0.6727 | 0.9091 | 0.3297 | 0.95 recall (w/ seg @ same class threshold) |
| | | 0.03 | 0.4825 | 0.6686 | 0.8267 | 0.3407 | 0.99 recall (w/ seg @ same class threshold) |
| Voting_WRL (No threshold prior to voting) | Mask R-CNN ResNet101 (Table 25) | 0.5 | 0.8883 | 0.9187 | 0.8811 | 0.8956 | Control (0.5 threshold no seg) |
| | | 0.5 | 0.8652 | 0.9032 | 0.8851 | 0.8462 | 0.5 threshold (w/ seg @ same class threshold) |
| | | 0.5 (+1.6 std) | 0.8509 | 0.8912 | 0.8396 | 0.8626 | With uncertainty std dev used to improve recall |
| | | 0.5 (+3.2 std) | 0.8364 | 0.8782 | 0.7931 | 0.8846 | With uncertainty std dev used to improve recall |
| | | 0.25 | 0.8488 | 0.8886 | 0.8205 | 0.8791 | 0.95 recall (w/ seg @ same class threshold) |
| | | 0.05 | 0.8157 | 0.8588 | 0.7378 | 0.9121 | 0.99 recall (w/ seg @ same class threshold) |
| Voting_Mixed (No threshold prior to voting) | | 0.5 | 0.8919 | 0.9210 | 0.8777 | 0.9066 | Control (0.5 threshold no seg) |
| | | 0.5 | 0.8715 | 0.9075 | 0.8864 | 0.8571 | 0.5 threshold (w/ seg @ same class threshold) |
| | | 0.5 (+1.7 std) | 0.8587 | 0.8970 | 0.8495 | 0.8681 | With uncertainty std dev used to improve recall |
| | | 0.5 (+3.3 std) | 0.8593 | 0.8925 | 0.7803 | 0.9560 | With uncertainty std dev used to improve recall |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | 0.24 | 0.8466 | 0.8868 | 0.8163 | 0.8791 | 0.95 recall (w/ seg @ same class threshold) |
| | | 0.03 | 0.7952 | 0.8404 | 0.7017 | 0.9176 | 0.99 recall (w/ seg @ same class threshold) |

**B2 – Segmentation**

| Classification model | Segmentation model | Threshold | MSE | IoU | F1-Score | Double F1 | Note |
|---|---|---|---|---|---|---|---|
| Voting_WRL (No threshold prior to voting) | FPN24 ResNet152 (Table 22) | 0.5 | 0.0423 | 0.5320 | 0.5942 | 0.7816 | Control (0.5 threshold no class) |
| | | 0.5 | 0.0380 | 0.6506 | 0.7190 | 0.8439 | 0.5 threshold (w/ seg @ same class threshold) |
| | | 0.5 (+1.6 std) | 0.0396 | 0.6326 | 0.7010 | 0.8344 | With uncertainty std dev used to improve recall |
| | | 0.5 (+3.2 std) | 0.0418 | 0.5965 | 0.6649 | 0.8158 | With uncertainty std dev used to improve recall |
| | | 0.25 | 0.0403 | 0.6165 | 0.6861 | 0.8268 | 0.95 recall (w/ seg @ same class threshold) |
| | | 0.05 | 0.0437 | 0.5714 | 0.6410 | 0.8033 | 0.99 recall (w/ seg @ same class threshold) |
| Voting_Mixed (No threshold prior to voting) | | 0.5 | 0.0423 | 0.5320 | 0.5942 | 0.7816 | Control (0.5 threshold no class) |
| | | 0.5 | 0.0380 | 0.6506 | 0.7190 | 0.8439 | 0.5 threshold (w/ seg @ same class threshold) |
| | | 0.5 (+1.7 std) | 0.0418 | 0.5965 | 0.6649 | 0.8158 | With uncertainty std dev used to improve recall |
| | | 0.5 (+3.3 std) | 0.0418 | 0.5965 | 0.6649 | 0.8158 | With uncertainty std dev used to improve recall |
| | | 0.24 | 0.0417 | 0.5984 | 0.6681 | 0.8174 | 0.95 recall (w/ seg @ same class threshold) |
| | | 0.03 | 0.0437 | 0.5714 | 0.6410 | 0.8033 | 0.99 recall (w/ seg @ same class threshold) |
| Voting_WRL (No threshold prior to voting) | FPN24 DenseNet169 (Table 22) | 0.5 | 0.0423 | 0.5320 | 0.5942 | 0.7816 | Control (0.5 threshold no class) |
| | | 0.5 | 0.0375 | 0.6785 | 0.7477 | 0.8590 | 0.5 threshold (w/ seg @ same class threshold) |
| | | 0.5 (+1.6 std) | 0.0390 | 0.6605 | 0.7297 | 0.8495 | With uncertainty std dev used to improve recall |
| | | 0.5 (+3.2 std) | 0.0411 | 0.6155 | 0.6847 | 0.8264 | With uncertainty std dev used to improve recall |
| | | 0.25 | 0.0400 | 0.6425 | 0.7117 | 0.8402 | 0.95 recall (w/ seg @ same class threshold) |
| | | 0.05 | 0.0429 | 0.5794 | 0.6486 | 0.8079 | 0.99 recall (w/ seg @ same class threshold) |
| | | 0.5 | 0.0423 | 0.5320 | 0.5942 | 0.7816 | Control (0.5 threshold no class) |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Voting_Mixed (No threshold prior to voting) | | 0.5 | 0.0375 | 0.6785 | 0.7477 | 0.8590 | 0.5 threshold (w/ seg @ same class threshold) |
| | | 0.5 (+1.7 std) | 0.0411 | 0.6245 | 0.6937 | 0.8309 | With uncertainty std dev used to improve recall |
| | | 0.5 (+3.3 std) | 0.0411 | 0.6245 | 0.6937 | 0.8309 | With uncertainty std dev used to improve recall |
| | | 0.24 | 0.0411 | 0.6155 | 0.6847 | 0.8264 | 0.95 recall (w/ seg @ same class threshold) |
| | | 0.03 | 0.0429 | 0.5794 | 0.6486 | 0.8079 | 0.99 recall (w/ seg @ same class threshold) |
| Voting_WRL (No threshold prior to voting) | FPN24 SE-ResNet152 (Table 22) | 0.5 | 0.0423 | 0.5320 | 0.5942 | 0.7816 | Control (0.5 threshold no class) |
| | | 0.5 | 0.0368 | 0.6695 | 0.7336 | 0.8523 | 0.5 threshold (w/ seg @ same class threshold) |
| | | 0.5 (+1.6 std) | 0.0390 | 0.6515 | 0.7155 | 0.8427 | With uncertainty std dev used to improve recall |
| | | 0.5 (+3.2 std) | 0.0404 | 0.6092 | 0.6748 | 0.8218 | With uncertainty std dev used to improve recall |
| | | 0.25 | 0.0389 | 0.6363 | 0.7018 | 0.8357 | 0.95 recall (w/ seg @ same class threshold) |
| | | 0.05 | 0.0408 | 0.5822 | 0.6478 | 0.8081 | 0.99 recall (w/ seg @ same class threshold) |
| Voting_Mixed (No threshold prior to voting) | | 0.5 | 0.0423 | 0.5320 | 0.5942 | 0.7816 | Control (0.5 threshold no class) |
| | | 0.5 | 0.0368 | 0.6695 | 0.7336 | 0.8523 | 0.5 threshold (w/ seg @ same class threshold) |
| | | 0.5 (+1.7 std) | 0.0403 | 0.6183 | 0.6838 | 0.8263 | With uncertainty std dev used to improve recall |
| | | 0.5 (+3.3 std) | 0.0403 | 0.6183 | 0.6838 | 0.8263 | With uncertainty std dev used to improve recall |
| | | 0.24 | 0.0404 | 0.6092 | 0.6748 | 0.8218 | 0.95 recall (w/ seg @ same class threshold) |
| | | 0.03 | 0.0409 | 0.5822 | 0.6478 | 0.8081 | 0.99 recall (w/ seg @ same class threshold) |
| Voting_WRL (No threshold prior to voting) | FPN24 EfficientNetB7 (Table 22) | 0.5 | 0.0423 | 0.5320 | 0.5942 | 0.7816 | Control (0.5 threshold no class) |
| | | 0.5 | 0.0330 | 0.6995 | 0.7658 | 0.8693 | 0.5 threshold (w/ seg @ same class threshold) |
| | | 0.5 (+1.6 std) | 0.0361 | 0.6634 | 0.7297 | 0.8504 | With uncertainty std dev used to improve recall |
| | | 0.5 (+3.2 std) | 0.0388 | 0.6189 | 0.6857 | 0.8277 | With uncertainty std dev used to improve recall |
| | | 0.25 | 0.0369 | 0.6460 | 0.7128 | 0.8417 | 0.95 recall (w/ seg @ same class threshold) |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | 0.05 | 0.0403 | 0.5919 | 0.6587 | 0.8137 | 0.99 recall (w/ seg @ same class threshold) |
| Voting_Mixed (No threshold prior to voting) | | 0.5 | 0.0423 | 0.5320 | 0.5942 | 0.7816 | Control (0.5 threshold no class) |
| | | 0.5 | 0.0330 | 0.6995 | 0.7658 | 0.8693 | 0.5 threshold (w/ seg @ same class threshold) |
| | | 0.5 (+1.7 std) | 0.0381 | 0.6279 | 0.6947 | 0.8324 | With uncertainty std dev used to improve recall |
| | | 0.5 (+3.3 std) | 0.0381 | 0.6279 | 0.6947 | 0.8324 | With uncertainty std dev used to improve recall |
| | | 0.24 | 0.0388 | 0.6189 | 0.6857 | 0.8277 | 0.95 recall (w/ seg @ same class threshold) |
| | | 0.03 | 0.0403 | 0.5919 | 0.6587 | 0.8137 | 0.99 recall (w/ seg @ same class threshold) |
| Voting_WRL (No threshold prior to voting) | GAN Improved segmentation model (Table 24 – Run 4) | 0.5 | 0.0441 | 0.5709 | 0.6318 | 0.8010 | Control (0.5 threshold no class) |
| | | 0.5 | 0.0417 | 0.6392 | 0.7026 | 0.8351 | 0.5 threshold (w/ seg @ same class threshold) |
| | | 0.5 (+1.6 std) | 0.0435 | 0.6212 | 0.6846 | 0.8255 | With uncertainty std dev used to improve recall |
| | | 0.5 (+3.2 std) | 0.0436 | 0.5943 | 0.6579 | 0.8121 | With uncertainty std dev used to improve recall |
| | | 0.25 | 0.0435 | 0.6123 | 0.6759 | 0.8212 | 0.95 recall (w/ seg @ same class threshold) |
| | | 0.05 | 0.0447 | 0.5583 | 0.6218 | 0.7937 | 0.99 recall (w/ seg @ same class threshold) |
| Voting_Mixed (No threshold prior to voting) | | 0.5 | 0.0441 | 0.5709 | 0.6318 | 0.8010 | Control (0.5 threshold no class) |
| | | 0.5 | 0.0417 | 0.6392 | 0.7026 | 0.8351 | 0.5 threshold (w/ seg @ same class threshold) |
| | | 0.5 (+1.7 std) | 0.0435 | 0.6033 | 0.6669 | 0.8167 | With uncertainty std dev used to improve recall |
| | | 0.5 (+3.3 std) | 0.0435 | 0.6033 | 0.6669 | 0.8167 | With uncertainty std dev used to improve recall |
| | | 0.24 | 0.0436 | 0.5943 | 0.6579 | 0.8121 | 0.95 recall (w/ seg @ same class threshold) |
| | | 0.03 | 0.0447 | 0.5583 | 0.6218 | 0.7937 | 0.99 recall (w/ seg @ same class threshold) |
| Voting_WRL (No threshold prior to voting) | FPN24 EfficientNetB7 (Table 27 - Without Training Augmentation) | 0.5 | 0.0417 | 0.5410 | 0.6039 | 0.7870 | Control (0.5 threshold no class) |
| | | 0.5 | 0.0345 | 0.6929 | 0.7587 | 0.8655 | 0.5 threshold (w/ seg @ same class threshold) |
| | | 0.5 (+1.6 std) | 0.0359 | 0.6568 | 0.7227 | 0.8470 | With uncertainty std dev used to improve recall |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | 0.5 (+3.2 std) | 0.0385 | 0.6118 | 0.6776 | 0.8238 | With uncertainty std dev used to improve recall |
| | | 0.25 | 0.0368 | 0.6388 | 0.7046 | 0.8378 | 0.95 recall (w/ seg @ same class threshold) |
| | | 0.05 | 0.0391 | 0.5758 | 0.6416 | 0.8056 | 0.99 recall (w/ seg @ same class threshold) |
| Voting_Mixed (No threshold prior to voting) | | 0.5 | 0.0417 | 0.5410 | 0.6039 | 0.7870 | Control (0.5 threshold no class) |
| | | 0.5 | 0.0345 | 0.6929 | 0.7587 | 0.8655 | 0.5 threshold (w/ seg @ same class threshold) |
| | | 0.5 (+1.7 std) | 0.0379 | 0.6208 | 0.6866 | 0.8284 | With uncertainty std dev used to improve recall |
| | | 0.5 (+3.3 std) | 0.0379 | 0.6208 | 0.6866 | 0.8284 | With uncertainty std dev used to improve recall |
| | | 0.24 | 0.0385 | 0.6118 | 0.6776 | 0.8238 | 0.95 recall (w/ seg @ same class threshold) |
| | | 0.03 | 0.0391 | 0.5848 | 0.6506 | 0.8101 | 0.99 recall (w/ seg @ same class threshold) |
| Voting_WRL (No threshold prior to voting) | Mask R-CNN ResNet50 (Table 25) | 0.5 | 0.0583 | 0.5403 | 0.6185 | 0.7895 | Control (0.5 threshold no class) |
| | | 0.5 | 0.0684 | 0.4700 | 0.5045 | 0.7298 | 0.5 threshold (w/ seg @ same class threshold) |
| | | 0.5 (+1.6 std) | 0.0684 | 0.4700 | 0.5045 | 0.7298 | With uncertainty std dev used to improve recall |
| | | 0.5 (+3.2 std) | 0.0684 | 0.4700 | 0.5045 | 0.7298 | With uncertainty std dev used to improve recall |
| | | 0.25 | 0.0684 | 0.4700 | 0.5045 | 0.7298 | 0.95 recall (w/ seg @ same class threshold) |
| | | 0.05 | 0.0705 | 0.4520 | 0.4865 | 0.7202 | 0.99 recall (w/ seg @ same class threshold) |
| Voting_Mixed (No threshold prior to voting) | | 0.5 | 0.0583 | 0.5403 | 0.6185 | 0.7895 | Control (0.5 threshold no class) |
| | | 0.5 | 0.0684 | 0.4700 | 0.5045 | 0.7298 | 0.5 threshold (w/ seg @ same class threshold) |
| | | 0.5 (+1.7 std) | 0.0684 | 0.4700 | 0.5045 | 0.7298 | With uncertainty std dev used to improve recall |
| | | 0.5 (+3.3 std) | 0.0684 | 0.4700 | 0.5045 | 0.7298 | With uncertainty std dev used to improve recall |
| | | 0.24 | 0.0684 | 0.4700 | 0.5045 | 0.7298 | 0.95 recall (w/ seg @ same class threshold) |
| | | 0.03 | 0.0705 | 0.4520 | 0.4865 | 0.7202 | 0.99 recall (w/ seg @ same class threshold) |
| Voting_WRL (No threshold prior to voting) | Mask R-CNN ResNet101 (Table 25) | 0.5 | 0.0648 | 0.4620 | 0.5391 | 0.7480 | Control (0.5 threshold no class) |
| | | 0.5 | 0.0521 | 0.6512 | 0.7283 | 0.8460 | 0.5 threshold (w/ seg @ same class threshold) |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | 0.5 (+1.6 std) | 0.0532 | 0.6241 | 0.7013 | 0.8322 | With uncertainty std dev used to improve recall |
| | | 0.5 (+3.2 std) | 0.0564 | 0.5791 | 0.6562 | 0.8088 | With uncertainty std dev used to improve recall |
| | | 0.25 | 0.0542 | 0.6061 | 0.6833 | 0.8229 | 0.95 recall (w/ seg @ same class threshold) |
| | | 0.05 | 0.0621 | 0.5160 | 0.5932 | 0.7758 | 0.99 recall (w/ seg @ same class threshold) |
| Voting_Mixed (No threshold prior to voting) | | 0.5 | 0.0648 | 0.4620 | 0.5391 | 0.7480 | Control (0.5 threshold no class) |
| | | 0.5 | 0.0502 | 0.6602 | 0.7373 | 0.8510 | 0.5 threshold (w/ seg @ same class threshold) |
| | | 0.5 (+1.7 std) | 0.0557 | 0.5881 | 0.6653 | 0.8135 | With uncertainty std dev used to improve recall |
| | | 0.5 (+3.3 std) | 0.0557 | 0.5881 | 0.6653 | 0.8135 | With uncertainty std dev used to improve recall |
| | | 0.24 | 0.0564 | 0.5791 | 0.6562 | 0.8088 | 0.95 recall (w/ seg @ same class threshold) |
| | | 0.03 | 0.0621 | 0.5160 | 0.5932 | 0.7758 | 0.99 recall (w/ seg @ same class threshold) |

# Appendix C - BentoML Services Creation

During a work term in the Canadian government, the author of this thesis worked with an Applied Research team revamping their image analysis system. As part of this work, an early version of our model was integrated into the system. This system uses BentoML for ML and DL model deployment; a file management system to store the images and metadata; and a web application to provide the results to analysts. This section explains how the two models were deployed and how they worked together in the final solution.

### Services Overview

BentoML provides a framework to deploy and integrate Data Science models in Client Applications using a standardized API. The models are integrated as part of a service which also includes the pre- and post-processing steps as well as the Python environment dependencies required to run the system. The service is then saved as a file which can then be loaded in a Docker environment on the server when a request comes in to use the service. A flowchart showing this process is shown in Figure 13. BentoML "just in time" model deployment ensures a more efficient use of computing resources by only loading services when needed and can even batch requests to reduce the wait times due to services loading and unloading.

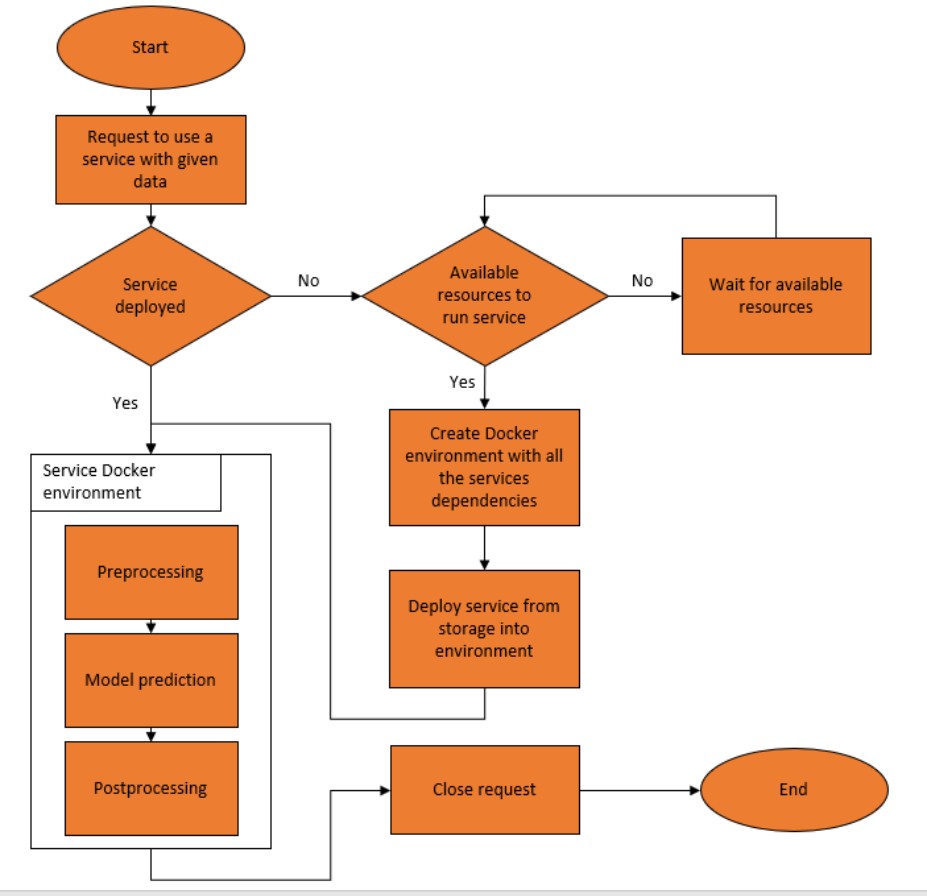


*Figure 13 - Flowchart of BentoML Service use. It shows how the services are deployed only when a request for its used has been made.*

**Gore Classification Service**

The goal of the gore classification service is to identify the bulk of the images that do not contain gore to avoid as many FPs as possible stemming from the segmentation model. The main reason for this choice is that the classification model achieved very high recall with high precision on the testing set while the segmentation model did not. The testing results of 7.1 have shown this approach maximized the amount of gore that is censored overall. Another reason for the use of the classifier prior to censoring is that we do not want to duplicate images in the file system unless we need to (i.e., to censor them).

The input into the gore classification service is the image file. The image is resized to the input size to the model (224 by 224). This image is then served to the model which outputs the probability that the image contains gore. This is then added to the image metadata file in the file system.

In the future, model uncertainty will be added to the service. To do so, the model will be changed to one with an inference dropout layer. Then, the model will be run several times using the same image as input. The resulting mean and standard deviation will then be calculated. The gore probability is the mean, while the model uncertainty is the standard deviation.

**Gore Segmentation Service**

As stated in the previous section, the result of gore classification can be used to assess if an image contains gore. Thus, the gore segmentation model is only served an image if the classification service marked that the image contains gore (i.e., if gore probability >= 0.5). Otherwise, the classification service sets a Boolean flag value to False in the image's metadata. This signifies no censored image was generated therefore client applications know the service has executed without generating a censored image.

Resizing the image to the network input size is the only preprocessing step. The post processing is a little more involved as this is when the image is censored. However, the network outputs a segmentation map of the same size as the network input (as opposed to the size of the original image which means that the segmentation map must be upsampled). Prior to upsampling, the map is binarized into a binary mask by setting any pixel with a gore probability higher than the threshold to 0 and otherwise to 1. This mask censors the image where the model found gore and leaves the other pixels unchanged. The upsampling is done using a nearest neighbour interpolation [212, 213]. The resulting mask is boxy and noisy. To solve this issue, a median filter is used to smooth out the upsampled mask. Morphological operations (closing and opening) are then applied to reduce noise. Using matrix multiplication, the mask is applied on the three colour channels (RGB) of the original image to obtain the censored image. The censored image is then saved as a child image of the original and the "gore" flag value is set to True. This is done so that the client application can look for a censored image.

### BentoML Service Use in Client Applications

Once created, the models are saved in a custom format that can be quickly loaded in memory for deployment on a server [214, 215]. Since the required packages are included in the service, the server simply needs to get the required packages installed in the environment where the service will run in for it to work. For the application at the Government of Canada, the results of each service were saved in json files. To obtain the gore classification results, the gore censoring service simply must read the json file instead of requiring the two services to run concurrently [216].

The client application also uses the results saved in the json files to serve them to the users. Therefore, we could use the gore classification results to blur images in the search results gallery if this result was higher than the threshold of 0.5. If a user wants more information on an image, they can refer to the image information page. On this page, a dropdown section allows the user to see the full resolution image. If gore is found on the image, a warning is added to the dropdown section to ensure the user knows the image may contain gore. If the gore censoring service created a censored version of the image, a new dropdown option is added allowing the user to see this version of the image instead.