An Active Learning Approach for Automated Validation of Image-based Test Inputs for Deep Learning System

by

Delaram Ghobari

Thesis submitted to the University of Ottawa

in partial fulfillment of the requirements for the degree of

Master of Applied Science

in

School of Electrical Engineering and Computer Science

University of Ottawa

Ottawa, Ontario, Canada

Abstract

Testing deep learning (DL) systems relies on the use of extensive and diverse, yet valid, test inputs. While synthetic test input generation methods, such as metamorphic testing, are widely used for DL testing, they risk introducing invalid inputs that do not align with the expected distribution of the system's training data. Invalid test inputs can lead to misleading results. Hence, there is a need for automated validation of test inputs to ensure effective assessment of DL systems. In this paper, we propose a test input validation approach for vision-based DL systems. Our approach relies on active learning to effectively balance the trade-off between accuracy and the manual effort required for the test input validation process. In addition, our approach adaptively selects image-comparison metrics for the optimal classification of valid and invalid test inputs, tailored to each specific dataset and test generation method, unlike existing methods where metrics are pre-selected. We evaluate our approach using an industrial and a public domain dataset. Our evaluation demosntrates that out human-in-the-loop test input validator (HiL-TV) achieves up to 96% validation accuracy with minimal human involvement, requiring only 20% of test inputs to be validated by a human for the industrial dataset and 50% for the public dataset. For higher accuracy levels, such as 98%, the required human involvement increases to 40% and 70%, respectively. Additionally, HiL-TV significantly outperforms existing baselines, being the only method capable of reaching 99% validation accuracy. These results highlight the effectiveness of HiL-TV in reducing manual effort while ensuring high validation accuracy, making it a robust solution for the validation of test inputs in DL systems across various domains.

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Professor Shiva Nejati, for her invaluable guidance, mentorship, and insightful feedback, which have been instrumental in the completion of this thesis. I am also grateful to Professor Mehrdad Sabetzadeh for his valuable consultations on the technical aspects of this work, which greatly contributed to the success of this research. I also want to thank Ali Mirferdows for developing the early stages of this project.

My heartfelt thanks go to my husband, Mohammad Hossein, for his unwavering support, encouragement, and patience throughout this journey. His assistance with the technical aspects of this work was invaluable and made this process much smoother.

I am also deeply thankful to my parents, Saeed and Shahrzad, for their endless love and belief in me, and to my sister, Taranom, for always being a source of inspiration.

Table of Contents

List of Tables viii

List of Figures x

1 Introduction 1

1.1 Challenges 2

1.2 Research Contribution 3

1.3 Organization 4

2 Background 6

2.1 Image Fidelity Metrics 6

2.1.1 Peak signal-to-noise ratio (PSNR) 7

2.1.2 Structural similarity index measure (SSIM) 9

2.1.3 Mean square error (MSE) 12

2.1.4 Texture similarity index (TSI) . . . 15
2.1.5 Wasserstein score (WD) . . . 18
2.1.6 Cosine similarity (CS) . . . 20
2.1.7 Kullback-Leibler Divergence (KL Divergence) . . . 23
2.1.8 Histogram intersection (Hist_i) . . . 26
2.1.9 Classifier Perceptual Loss (CPL) . . . 29
2.1.10 Semantic Segmentation Score (SSS) . . . 33
2.1.11 Visual Information Fidelity (VIF) . . . 37
2.2 Deep Neural Network Architectures . . . 41
2.2.1 Autoencoders . . . 41
2.2.2 Variational Autoencoders (VAE) . . . 44
2.3 Machine Learning Classifiers . . . 48
2.3.1 Support Vector Machine (SVM) . . . 49
2.3.2 Logistic Regression . . . 52
2.3.3 Decision Tree . . . 54
2.3.4 Random Forest . . . 57
2.4 Active Learning . . . 59

3 Related Work **64**

3.1 VAE-Based Method . . . 64

3.2 VIF-Based Method 66

4 Automated, Human-in-the-Loop Test Input Validation 72

4.1 Step 1: Image-comparison metric selection. 73

4.2 Step 2: Feature selection and best classifier identification. 76

4.3 Step 3: Active learning. 77

5 Empirical Evaluation 79

5.1 Research Questions 79

5.2 Subject Datasets 81

5.3 Experiment setup 82

5.4 RQ1 Experiments and Results. 83

5.5 RQ2 Experiments and Results. 85

5.6 RQ3 Experiments and Results. 87

5.7 RQ4 Experiments and Results. 89

5.8 Custom-Developed Labeling Tool 92

5.8.1 Setup the Tool 93

5.8.2 Tool's Output 95

6 Conclusion and Future Work 103

6.1 Conclusion 103

6.2 Threats to Validity 104

6.3 Future Work 105

References **107**

List of Tables

2.1 A list of image-comparison metrics from the literature [21,28,41]: The metric name and its source, the metric definition, and the level at which the metric compares images, i.e., at the pixel level or at feature-level where the features are obtained from the latent space of a pre-trained deep learning model. . . 63

5.1 Key characteristics of our public and industrial datasets 82

5.2 Dataset splits and their number of elements 82

5.3 The Pearson correlation coefficients between each metric in Table 2.1 and the valid labels in the correlation set, D_{cor}, of the industrial and public datasets. The values highlighted green represent the metrics with the highest correlations for each dataset. The VIF metric for the industrial dataset is removed due to being redundant. 83

5.4 The pairwise Pearson correlations between the five metrics highlighted green in Table 5.3 for the industrial dataset. The cells highlighted in yellow indicate a high level of redundancy between VIF and CS. 84

5.5 The pairwise Pearson correlations between the four metrics highlighted green in Table 5.3 for the public dataset. 84

5.6 Classifier instances with the highest accuracy for each of the four classifier types – random forest, decision tree, SVM, and logistic regression. 88

List of Figures

2.1 PSNR values for an image with different qualities. [40] 10

2.2 SSIM values for an image with different artifacts. [36] 12

2.3 Architecture of an autoencoder, illustrating the encoding, bottleneck, and decoding processes [34] 42

2.4 Architecture of a Variational Autoencoder (VAE), illustrating the probabilistic encoding, sampling from the latent space, and probabilistic decoding processes [2] 46

2.5 SVM separating 2-dimensional data points. [56] 50

2.6 An intuitive example of how SVM kernels can separate data points that are not linearly separable. [47] 51

2.7 Simple example of a Decision Tree. [4] 55

2.8 Simple example of a Random Forest. [35] 58

2.9 Workflow of Active Learning [18] 62

3.1 Invalid test inputs generated by DeepXplore [37] 65

3.2 Some invalid and valid test inputs generated by TIGs in Riccio and Tonella's experiments [41] 71

4.1 An overview of our test input validation approach for vision-based Deep learning systems (HiL-TV) 73

5.1 Accuracy for each classifier with different input features on SmartInside dataset 86

5.2 Accuracy for each classifier with different input features on Cifar10-c dataset 87

5.3 Accuracy vs human effort obtained by different executions of the active learning step of HiL-TV based on different values for parameters α and β of Algorithm 1 96

(a) The results for the industry dataset 96

(b) The results for the public dataset 96

5.4 Accuracy vs Human Effort w.r.t. β on SmartInside Dataset 97

5.5 Accuracy vs Human Effort w.r.t. α on SmartInside Dataset 97

5.6 Box plot of F1-score with respect to the human effort or SmartInside dataset 98

5.7 Box plot of accuracy with respect to the human effort or SmartInside dataset 98

5.8 Box plot of F1-score with respect to the human effort or Cifar10-c dataset 99

5.9 Box plot of accuracy with respect to the human effort or Cifar10-c dataset 99

5.10 AILA labeling tool 100

5.11 AILA labeling tool statistics window 101

5.12 Requirements.txt file for setting up the tool. 101

5.13 The output of the tool in csv format. 102

Chapter 1

Introduction

Deep learning (DL) systems have become integral to many advanced systems we build today due to their ability to automate complex tasks such as anomaly detection, object detection, and semantic segmentation. DL systems are fundamentally data-centric, making their effective testing and verification dependent on the availability of extensive and diverse test inputs. To address this need, various synthetic test input generation methods have been proposed. These include metamorphic testing approaches, which create new test inputs by systematically modifying existing ones while preserving certain properties [10, 21, 41]. Similarly, various test input generation techniques for DL image classifiers aim to create diverse test inputs by applying transformations such as rotation, scaling, or color adjustments to images [21, 37].

1.1 Challenges

Transforming inputs or synthetically generating them may lead to *invalid* test inputs. According to recent research in the software testing literature [10, 41, 44], valid test inputs are those that fall within the expected distribution and satisfy the constraints defined by the training data of the DL system under test, while invalid inputs deviate from these expectations and constraints. In particular, Riccio and Tonella [41] define invalid inputs for a DL system as those that cannot be confidently recognized and labeled by human domain experts within the input domain. Using invalid inputs for testing DL systems can lead to misleading results, such as identifying spurious errors caused by invalid inputs rather than by realistic, valid inputs. Furthermore, invalid inputs may give a false sense of confidence in the system's reliability, as the test time budget is spent assessing the system with inputs it will not encounter in real-world scenarios, which means that flaws that may arise with valid inputs could go undetected. Therefore, test inputs need to be validated before being used for testing.

Recent approaches for automated test input validation techniques target vision-based DL systems [21, 41]. These techniques rely on image-comparison metrics [28], which quantify how well a transformed image aligns with its corresponding original image. Specifically, Hu et al. [21] use visual information fidelity (VIF) [42], while Riccio and Tonella [41] rely on the loss function of a variational autoencoder (VAE) [26] trained on a set of valid test inputs. They identify an optimal threshold for their respective image-comparison metric, derived from a manually validated set of test inputs. To automatically determine the validity of a pair of original and transformed images, the metric value for the pair

is compared against the identified thresholds. Since the metric values vary considerably from one dataset to another, the thresholds do not generalize to datasets not considered in the original papers. Further, these approaches use only a single metric. It is unclear whether a single metric can effectively distinguish valid inputs for different datasets, such as industrial and specialized-domain datasets, which, in contrast to the datasets used by these approaches [21, 41], are not limited to common objects such as animals and vehicles.

1.2 Research Contribution

In this thesis, we propose HiL-TV, an automated, human-in-the-loop test input validation approach for vision-based DL systems. HiL-TV receives a set of image pairs consisting of original and transformed images, and it identifies the valid pairs – those where the transformed image is a valid transformation of the original image – from the invalid pairs. HiL-TV uses an active learning loop to continually train a classifier to distinguish between valid and invalid pairs. When the classifier's confidence in a prediction falls below a user-defined threshold, the image pair is flagged, and a human labels it as valid or invalid. The input features of the classifier are image-comparison metrics computed for each image pair.

Prior to the active-learning loop, HiL-TV uses two pre-processing steps to develop and optimize the active-learning classifier specific to a given dataset. The first step identifies a subset of image-comparison metrics from an extensive range of options collected from the literature that best correlate with valid pairs in the given dataset. The second step trains a classifier using the optimal image-comparison metrics selected in the first step. This step ensures the optimality of the trained classifier by comparing different classifiers and

considering different subsets of the optimal image-comparison metrics as input features. HiL-TV requires a labeled subset of the input dataset to compute metric correlations and to train classifiers.

We evaluate HiL-TV using two extensive datasets: one from industry and the other from the open-source domain. Our industrial dataset is developed by our industry partner, SmartInside AI Inc [43], and includes images of power grid facilities as well as transformations of these images aimed at simulating foggy, rainy, and snowy weather conditions. As for the public-domain dataset, we use the dataset developed by Hu et al. [21]. This dataset includes images from the Cifar-10 datasets as well as transformations of these datasets.

Our evaluation assesses the trade-off between accuracy of test input validation versus the effort required for manual labelling. We compare HiL-TV with two baselines from the literature that are the approaches developed by Hu et. al. [21] and by Riccio and Tonella [41]. Our results show that HiL-TV significantly outperforms the baselines in both the industrial and public-domain datasets.

1.3 Organization

This thesis is structured into seven chapters, each building upon the concepts introduced in the previous sections to provide a comprehensive overview of the research conducted.

- **Chapter 2: Background** - This chapter lays the groundwork for the research by introducing essential concepts, such as image fidelity metrics, machine learning

classifiers, and active learning techniques. These topics are crucial for understanding the methodologies applied in the thesis.

- **Chapter 3: Automated, Human-in-the-Loop Test Input Validation** - In this chapter, we introduce the proposed HiL-TV approach for the automated validation of test inputs in vision-based deep learning systems. The chapter details the three-step process of image-comparison metric selection, feature selection, classifier identification, and the integration of active learning.

- **Chapter 4: Empirical Evaluation** - This chapter presents the empirical evaluation of the HiL-TV approach. We outline the research questions, describe the datasets used, explain the experimental setup, and discuss the results obtained from various experiments.

- **Chapter 5: Custom-Developed Labeling Tool** - In this chapter, we describe the custom tool developed to facilitate the labeling process for the empirical evaluation. The chapter covers the tool's setup, functionalities, and output format.

- **Chapter 6: Related Work** - This chapter reviews the existing literature and compares our approach with related methods. It provides context and highlights the contributions of this thesis to the field.

- **Chapter 7: Conclusion and Future Work** - The final chapter summarizes the key findings of the research, discusses the implications of the results, and outlines potential directions for future research.

Chapter 2

Background

This chapter explains the important concepts needed for this research. It starts with image-comparison metrics, which help measure the quality of generated images. Then, it covers machine learning classifiers, which are used to categorize and predict data. Finally, it introduces active learning, a method to improve models by choosing the most useful data points. These topics provide the necessary background for understanding the methods used in this thesis.

2.1 Image Fidelity Metrics

We investigate image fidelity metrics designed to compare pairs of images, where one image is a transformed version of the other. These metrics assess the quality, level of similarity, or differences in various aspects, such as segments or texture, between the two images. Our

focus is on identifying image fidelity metrics that most effectively indicate the preservation of essential information from the original image in its corresponding transformations.

Each image fidelity metric quantifies the degree of similarity or difference between the original and transformed images by assigning a numerical value. This value reflects how well the transformation has maintained the critical features of the original image, such as edges, textures, and structural details. By carefully selecting appropriate fidelity metrics, we aim to ensure that the transformed images retain the necessary visual information for accurate analysis and interpretation. Table 2.1 gives an overview of the metrics along with their sources.

In the sections below, we provide a detailed explanation of 13 image fidelity metrics that we gathered from the literature. [21, 26, 28, 41, 42] These metrics are explored in depth to understand their effectiveness in preserving essential information during image transformations.

2.1.1 Peak signal-to-noise ratio (PSNR)

Peak Signal-to-Noise Ratio (PSNR) is a widely used metric in image processing and video compression to measure the quality of a reconstructed or compressed image compared to the original image. PSNR is expressed in decibels (dB) and provides an estimate of the fidelity or quality of the processed image relative to the original. A higher PSNR value indicates better quality, meaning that the processed image is closer to the original. [14, 31, 54]

Mathematical Definition

PSNR is derived from the Mean Squared Error (MSE), which is a measure of the average

squared difference between the original image and the processed image. Let us define an original image as I_{original} and a processed image as $I_{\text{processed}}$, both of size $M \times N$ pixels.

The Mean Squared Error (MSE) is given by:

$$\text{MSE} = \frac{1}{MN} \sum_{i=1}^{M} \sum_{j=1}^{N} \left[I_{\text{original}}(i,j) - I_{\text{processed}}(i,j) \right]^2$$

Where:

- M and N are the dimensions of the images.
- $I_{\text{original}}(i,j)$ and $I_{\text{processed}}(i,j)$ are the pixel values of the original and processed images at position (i,j), respectively.

Once the MSE is calculated, the PSNR is defined as:

$$\text{PSNR} = 10 \cdot \log_{10} \left(\frac{\text{MAX}^2}{\text{MSE}} \right)$$

Where:

- MAX is the maximum possible pixel value of the image. For an 8-bit image, MAX = 255.
- $\log_{10}$ is the logarithm to base 10.

Interpretation

High PSNR Indicates that the processed image is very similar to the original image, implying high fidelity. Generally, a PSNR value above 30 dB is considered good quality. Low PSNR Indicates that the processed image has significant differences from the original image, implying low fidelity.

Use Cases

PSNR is commonly used in various applications, including image compression, where it assesses the quality of images after lossy compression, such as JPEG. It is also utilized in video compression to evaluate the quality of video frames after compression, such as in H.264 or H.265 codecs. Additionally, PSNR is employed in image de-noising to measure the effectiveness of noise reduction algorithms.

Limitations

While PSNR is a useful metric for measuring image quality, it has some limitations. It is a global metric and does not always correlate well with human visual perception. Additionally, PSNR does not account for perceptual phenomena such as contrast sensitivity and masking effects.

2.1.2 Structural similarity index measure (SSIM)

The Structural Similarity Index Measure (SSIM) is a perceptual metric that quantifies the image quality degradation caused by processing such as data compression or transmission losses. Unlike traditional methods like Peak Signal-to-Noise Ratio (PSNR) that focus on pixel-wise differences, SSIM considers changes in structural information, luminance, and contrast. [16, 32, 55]

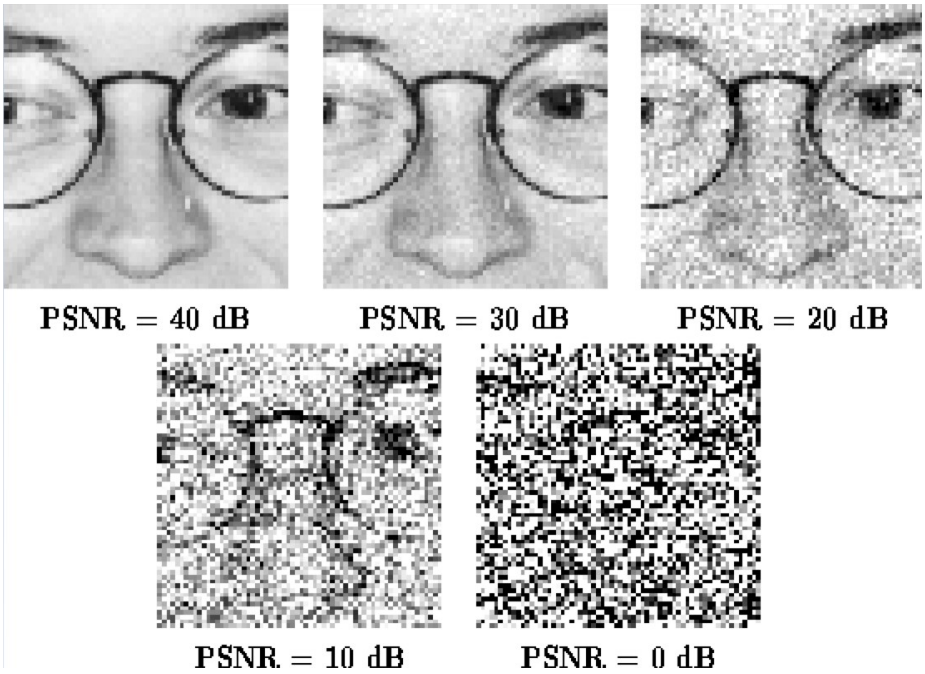


Figure 2.1: PSNR values for an image with different qualities. [40]

Mathematical Definition

The SSIM index between two images x and y is calculated as:

$$\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}$$

Where:

- μ_x and μ_y are the mean intensities of x and y,
- σ_x^2 and σ_y^2 are the variances of x and y,
- σ_{xy} is the covariance of x and y,

- C_1 and C_2 are constants to stabilize the division.

The SSIM metric ranges from -1 to 1, where:

- SSIM $= 1$ indicates perfect structural similarity between the two images,
- SSIM $= 0$ indicates no structural similarity,
- SSIM < 0 indicates that the images are structurally dissimilar.

Use Cases

SSIM is extensively used in various applications, including image and video compression, where it evaluates the perceived quality of compressed images and videos. It is also applied in image denoising to assess the quality of images after noise reduction, as well as in image reconstruction to compare the similarity between an original image and its reconstructed version.

One of the key advantages of SSIM is that it aligns more closely with human visual perception compared to pixel-based metrics like MSE or PSNR. An extension of SSIM, known as Multi-scale SSIM (MS-SSIM), computes the metric at multiple image scales, further improving its robustness and accuracy.

Limitations

While SSIM is better aligned with human vision than MSE or PSNR, it is still not perfectly attuned to human visual perception. Additionally, SSIM can be sensitive to changes in luminance and contrast that do not necessarily affect perceived quality.

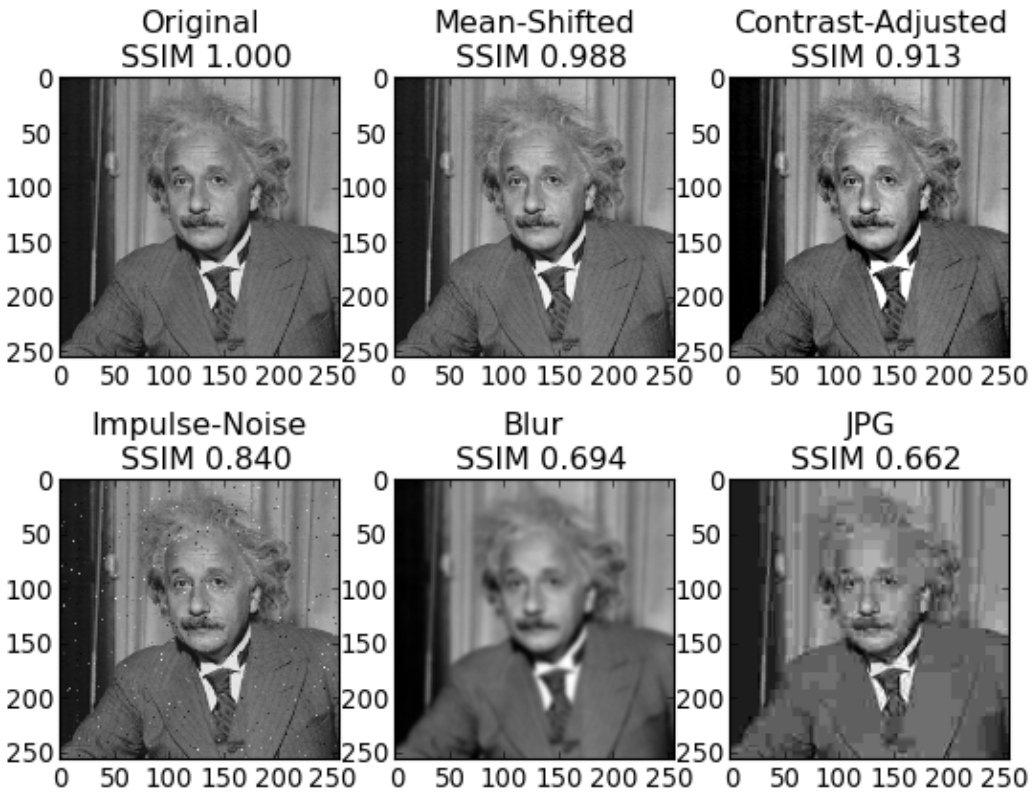


Figure 2.2: SSIM values for an image with different artifacts. [36]

2.1.3 Mean square error (MSE)

Mean Square Error (MSE) is a widely used metric in statistics, signal processing, and machine learning to measure the average squared difference between the actual values and the predicted values. A lower MSE value indicates a better fit of the model to the data or higher accuracy of the predicted values. [13, 30, 53]

Mathematical Definition

Let y_i be the actual value, $\hat{y}_i$ be the predicted value, and n be the number of data points. The Mean Square Error is defined as:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Where:

- n is the total number of observations,
- y_i is the actual value for the i-th observation,
- $\hat{y}_i$ is the predicted value for the i-th observation,
- $y_i - \hat{y}_i$ is the error or residual, representing the difference between the actual and predicted values.

The MSE metric calculates the average of the squares of these errors, providing a single value that represents the quality of the predictions made by a model.

Interpretation

Low MSE indicates that the predicted values are close to the actual values, suggesting that the model is performing well. High MSE indicates that the predicted values are far from the actual values, suggesting that the model is not performing well. MSE is a risk

function, corresponding to the expected value of the squared error loss. It is sensitive to outliers because it squares the errors, which amplifies the influence of large errors.

Use Cases

MSE is commonly used in regression analysis to assess the accuracy of a regression model's predictions, in image processing to measure the difference between original and processed images, particularly in image compression and denoising, and in signal processing to evaluate the performance of signal approximation or filtering algorithms.

Limitations

While MSE is a simple and intuitive metric, it has some limitations. MSE is highly sensitive to outliers because it squares the errors, which can distort the overall error measurement. Additionally, MSE depends on the scale of the data, meaning that it may not be directly comparable across different datasets or models with different scales.

Relationship with Root Mean Square Error (RMSE)

The Root Mean Square Error (RMSE) is another related metric that is often used in conjunction with MSE. It is defined as the square root of the MSE:

$$\text{RMSE} = \sqrt{\text{MSE}} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

RMSE provides an error metric in the same units as the original data, which can be more interpretable in some contexts.

2.1.4 Texture similarity index (TSI)

The Texture Similarity Index (TSI) is a metric used to evaluate the similarity between the textures of two images. While traditional metrics like Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index (SSIM) focus on overall image quality or structural similarity, TSI specifically targets the texture patterns within images. This makes it particularly useful in applications such as image synthesis, texture mapping, and image compression, where preserving texture details is crucial. [12, 49, 58]

Mathematical Definition

The mathematical formulation of TSI is based on comparing texture features extracted from the images. One common approach to compute TSI involves the use of statistical texture descriptors, such as the Gray-Level Co-occurrence Matrix (GLCM), or wavelet transforms.

Let's assume we have two images, I_{original} and $I_{\text{processed}}$. The general steps to compute TSI are as follows:

Extract Texture Features: For each image, texture features are extracted using a method such as GLCM or wavelet transforms. Let T_{original} and $T_{\text{processed}}$ represent the texture features of the original and processed images, respectively.

Compute Texture Similarity: The texture similarity index $\text{TSI}(I_{\text{original}}, I_{\text{processed}})$ can be calculated as:

$$\text{TSI}(I_{\text{original}}, I_{\text{processed}}) = \frac{2 \cdot \text{Cov}(T_{\text{original}}, T_{\text{processed}})}{\text{Var}(T_{\text{original}}) + \text{Var}(T_{\text{processed}})}$$

Where:

- $\text{Cov}(T_{\text{original}}, T_{\text{processed}})$ is the covariance between the texture features of the original and processed images.
- $\text{Var}(T_{\text{original}})$ and $\text{Var}(T_{\text{processed}})$ are the variances of the texture features for the original and processed images, respectively.

This formulation is inspired by the SSIM metric, but instead of using luminance and contrast, it focuses on texture features.

Interpretation

High TSI: A TSI close to 1 indicates that the textures of the original and processed images are highly similar.

Low TSI: A TSI close to 0 indicates that there is little or no similarity between the textures of the two images.

Use Cases

TSI is particularly useful in:

- **Image Synthesis**: To compare the texture similarity between synthesized images and their original counterparts.
- **Texture Mapping**: To assess the fidelity of textures applied to 3D models in graphics applications.
- **Image Compression**: To evaluate how well texture details are preserved after compression.

Advantages of TSI

Focused on Texture: TSI provides a focused measure of texture similarity, which is crucial in many image processing tasks.

Complementary to SSIM and PSNR: While SSIM and PSNR measure overall image quality, TSI complements these metrics by focusing specifically on texture.

Limitations of TSI

Complexity: The extraction of texture features and computation of TSI can be computationally intensive, especially for large images.

Sensitivity to Noise: Depending on the feature extraction method, TSI may be sensitive to noise, which can affect the accuracy of the similarity measurement.

2.1.5 Wasserstein score (WD)

The Wasserstein Distance (WD), also known as the Earth Mover's Distance (EMD), is a measure of the distance between two probability distributions. It is widely used in fields such as machine learning, particularly in Generative Adversarial Networks (GANs) for evaluating the similarity between the generated data distribution and the real data distribution. The Wasserstein Distance is part of the family of optimal transport distances, which consider the "cost" of transforming one distribution into another. [3, 48, 57]

Mathematical Definition

Let P and Q be two probability distributions defined on a metric space M (e.g., $\mathbb{R}^d$). The Wasserstein Distance of order p between these distributions is defined as:

$$W_p(P, Q) = \left(\inf_{\gamma \in \Gamma(P,Q)} \mathbb{E}_{(x,y)\sim\gamma}[d(x, y)^p] \right)^{\frac{1}{p}}$$

Where:

- $p \geq 1$ is a parameter that determines the order of the distance.
- $\Gamma(P, Q)$ is the set of all joint distributions (couplings) $\gamma(x, y)$ whose marginals are P and Q respectively.
- $d(x, y)$ is the distance between points x and y in the space M, typically using the Euclidean distance.

For the common case when $p = 1$, the Wasserstein Distance of order 1 is:

$$W_1(P, Q) = \inf_{\gamma \in \Gamma(P,Q)} \mathbb{E}_{(x,y)\sim\gamma}[d(x, y)]$$

In simple terms, $W_1(P, Q)$ measures the minimum "cost" required to transform distribution P into distribution Q, where the cost is defined as the amount of probability mass that needs to be moved multiplied by the distance it needs to be moved.

Interpretation

Small Wasserstein Distance: Indicates that the two distributions P and Q are similar, meaning that it requires a small "cost" to transform one distribution into the other.
Large Wasserstein Distance: Indicates that the distributions are dissimilar, meaning that a large "cost" is required to transform one distribution into the other.

Applications in Machine Learning

Generative Adversarial Networks (GANs): The Wasserstein Distance is used in the training of Wasserstein GANs (WGANs), where it helps to stabilize the training process and improve the quality of the generated samples by providing a more meaningful loss function compared to the original GAN.

Distributional Robustness: In scenarios where models are trained to be robust to small perturbations in the data distribution, Wasserstein Distance can be used as a measure of discrepancy between the training distribution and test distribution.

Image Retrieval and Processing: Wasserstein Distance is used to compare histograms, shapes, and other distributional features in image processing tasks.

Mathematical Example

Consider two discrete distributions P and Q on a one-dimensional space:

$$P = \{(x_1, p_1), (x_2, p_2), \ldots, (x_n, p_n)\}$$

$$Q = \{(y_1, q_1), (y_2, q_2), \ldots, (y_m, q_m)\}$$

The Wasserstein Distance $W_1(P, Q)$ can be computed by solving an optimization problem that finds the best matching between the points x_i and y_j that minimizes the cost:

$$W_1(P, Q) = \inf_{\gamma} \sum_{i=1}^{n} \sum_{j=1}^{m} \gamma_{ij} d(x_i, y_j)$$

Where γ_{ij} represents the amount of probability mass transported from x_i to y_j.

2.1.6 Cosine similarity (CS)

Cosine Similarity (CS) is a metric used to measure the similarity between two non-zero vectors in an inner product space. It is widely used in various applications, such as text analysis, natural language processing (NLP), information retrieval, and recommendation

systems. The cosine similarity measures the cosine of the angle between two vectors, giving a value between -1 and 1. [11, 29, 51, 57]

Mathematical Definition

Given two vectors $\mathbf{A}$ and $\mathbf{B}$ in an n-dimensional space, the cosine similarity is defined as:

$$\text{Cosine Similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$

Where:

- $\mathbf{A} \cdot \mathbf{B}$ is the dot product of the vectors $\mathbf{A}$ and $\mathbf{B}$.
- $\|\mathbf{A}\|$ and $\|\mathbf{B}\|$ are the magnitudes (Euclidean norms) of the vectors $\mathbf{A}$ and $\mathbf{B}$.
- θ is the angle between the two vectors.

Interpretation

- **Cosine Similarity = 1**: The vectors are identical in direction, meaning the angle $\theta = 0°$, and the vectors are perfectly similar.
- **Cosine Similarity = 0**: The vectors are orthogonal (perpendicular), meaning the angle $\theta = 90°$, and there is no similarity.
- **Cosine Similarity = -1**: The vectors are diametrically opposed, meaning the angle $\theta = 180°$, and the vectors are perfectly dissimilar.

Use Cases

Cosine similarity is particularly useful in:

- **Text Mining and NLP**: To compare the similarity between documents or sentences represented as vectors.
- **Information Retrieval**: To rank documents based on their similarity to a query in search engines.
- **Recommendation Systems**: To measure the similarity between users or items.
- **Clustering and Classification**: In machine learning, cosine similarity is used as a metric to cluster similar items or classify data points.

Advantages of Cosine Similarity

Scale Invariant: Unlike Euclidean distance, cosine similarity only considers the orientation of the vectors, not their magnitude, making it effective for comparing documents of different lengths.

Efficient Computation: Cosine similarity can be efficiently computed even for high-dimensional data, making it suitable for large-scale applications.

Limitations of Cosine Similarity

Ignores Magnitude: Cosine similarity does not account for the magnitude of the vectors, which might be important in some contexts.

Sparse Data: In cases of sparse data (e.g., text vectors), cosine similarity may become less effective, especially when dealing with high-dimensional data.

2.1.7 Kullback-Leibler Divergence (KL Divergence)

Kullback-Leibler Divergence, often referred to as KL Divergence, is a fundamental concept in information theory and statistics that measures the difference between two probability distributions. It is widely used in various fields such as machine learning, data science, and image processing to quantify the dissimilarity between distributions. In the context of images, KL Divergence can be used to compare the distribution of pixel values between a generated image and a reference image, providing insight into the fidelity and quality of the generated image.

Mathematical Definition

Given two probability distributions P (the true distribution) and Q (the approximate distribution), the Kullback-Leibler Divergence from Q to P is defined as:

$$D_{KL}(P \parallel Q) = \sum_{i} P(i) \log \left(\frac{P(i)}{Q(i)} \right)$$

Where:

- $P(i)$ represents the probability of event i under distribution P.
- $Q(i)$ represents the probability of event i under distribution Q.

- The logarithm is typically taken to base 2 or base e, depending on the context.

KL Divergence is a non-symmetric measure of the difference between the two distributions P and Q. It is non-negative, with $D_{KL}(P \parallel Q) = 0$ if and only if $P = Q$ almost everywhere, meaning that the two distributions are identical.

Interpretation

- **Low KL Divergence**: A lower value of KL Divergence indicates that the approximate distribution Q is close to the true distribution P, implying greater similarity between the two distributions. In image processing, this means that the pixel value distribution of the generated image closely matches that of the reference image, suggesting high fidelity in the generation process.

- **High KL Divergence**: A higher value of KL Divergence suggests that the approximate distribution Q diverges significantly from the true distribution P, indicating dissimilarity. In the context of images, this would imply that the pixel value distribution of the generated image is quite different from the reference image, potentially indicating artifacts or loss of important features during generation.

Use Cases

KL Divergence is extensively used in various applications, including:

- **Generative Models**: In machine learning, particularly in Variational Autoencoders

(VAEs), KL Divergence is used as a regularization term to measure how well the learned distribution approximates the true data distribution.

- **Image Processing**: In image generation and transformation tasks, KL Divergence can be used to compare the distribution of pixel values between a generated image and its reference, helping to assess the quality of the generated image.
- **Data Compression**: KL Divergence is applied in compression algorithms to measure the efficiency of encoding schemes by comparing the original distribution of data with the distribution after compression.

Advantages

- **Theoretical Foundation**: KL Divergence has a strong theoretical basis in information theory, making it a widely accepted and reliable measure of distributional differences.
- **Sensitivity to Distribution Differences**: KL Divergence effectively captures differences in distributions, making it useful for applications where precise comparison is required.

Limitations

- **Non-Symmetry**: KL Divergence is not symmetric, meaning that $D_{KL}(P \parallel Q)$ is not necessarily equal to $D_{KL}(Q \parallel P)$. This can be a limitation in contexts where a symmetric measure of dissimilarity is preferred.

- **Undefined for Zero Probabilities**: KL Divergence is undefined if $Q(i) = 0$ for any event i where $P(i) > 0$. This can pose challenges in practical applications, particularly when dealing with sparse data.

Relationship with Other Metrics

KL Divergence is closely related to other divergence measures such as Jensen-Shannon Divergence, which is a symmetrized and smoothed version of KL Divergence. In some cases, Jensen-Shannon Divergence may be preferred due to its symmetry and bounded range.

2.1.8 Histogram intersection (Hist_i)

Histogram Intersection is a similarity measure used to compare two histograms, which are representations of the distribution of data. In image processing, histograms are often used to represent the distribution of pixel intensities or color values within an image. Histogram Intersection quantifies the overlap between two histograms, providing a measure of similarity that is particularly useful when comparing images with similar content but varying brightness or contrast.

Mathematical Definition

Given two histograms H_1 and H_2, each with n bins, the Histogram Intersection $I(H_1, H_2)$ is defined as:

$$I(H_1, H_2) = \sum_{i=1}^{n} \min(H_1(i), H_2(i))$$

Where:

- $H_1(i)$ and $H_2(i)$ represent the values of the i-th bin in histograms H_1 and H_2, respectively.
- $\min(H_1(i), H_2(i))$ computes the minimum value between the corresponding bins of the two histograms.

The result of the Histogram Intersection is a single value that represents the degree of overlap between the two histograms. This value can be normalized by the total sum of the histogram bins to obtain a similarity score between 0 and 1.

Interpretation

- **High Histogram Intersection**: A higher value indicates a greater overlap between the two histograms, suggesting that the distributions of pixel values or colors in the two images are similar. This implies that the images have similar content or characteristics.
- **Low Histogram Intersection**: A lower value indicates less overlap between the histograms, suggesting that the images differ significantly in terms of their pixel value or color distributions. This could be due to differences in content, lighting, or other factors.

Use Cases

Histogram Intersection is used in various image processing and computer vision tasks, including:

- **Image Retrieval**: Histogram Intersection is commonly used in content-based image retrieval systems to compare the histograms of a query image with those in a database, retrieving images with similar content.
- **Object Recognition**: In object recognition tasks, Histogram Intersection can be used to compare color histograms of detected objects with known reference histograms, aiding in the identification of objects.
- **Image Matching**: Histogram Intersection is used in image matching to assess the similarity between different views of the same scene, even under varying lighting conditions.

Advantages

- **Robust to Lighting Variations**: Histogram Intersection is less sensitive to changes in lighting or contrast, making it effective for comparing images with different exposure levels.
- **Simple and Efficient**: The computation of Histogram Intersection is straightforward and computationally efficient, making it suitable for real-time applications.

Limitations

- **Limited Discriminative Power**: Histogram Intersection may not be effective in distinguishing images with different contents but similar histogram distributions, as it only considers the overlap between histograms without accounting for spatial information.
- **Dependent on Histogram Binning**: The effectiveness of Histogram Intersection can be influenced by the choice of histogram binning. Poor binning choices can lead to misleading similarity measures.

Relationship with Other Metrics

Histogram Intersection is often compared with other histogram-based similarity measures such as Bhattacharyya distance and Chi-square distance. While Histogram Intersection measures overlap, Bhattacharyya and Chi-square distances take into account the overall distribution and variance, providing alternative ways to compare histograms.

2.1.9 Classifier Perceptual Loss (CPL)

Classifier Perceptual Loss (CPL) is an advanced loss function used in image generation and transformation tasks. Unlike traditional pixel-wise loss functions, such as Mean Squared Error (MSE) or L1 loss, CPL focuses on the perceptual quality of the generated images. It does so by measuring the difference between the feature maps created by the convolutional layers of a pre-trained classification model. These feature maps capture high-level

representations of the input images, including textures, objects, and shapes, making CPL a powerful tool for assessing the perceptual similarity between images.

Mathematical Definition

Let Φ represent a pre-trained convolutional neural network (CNN) that is used for classification, and let $\Phi_l(x)$ denote the feature map obtained from the l-th layer of the network when processing an input image x. Given a reference image I_{ref} and a generated image I_{gen}, the Classifier Perceptual Loss is defined as:

$$\text{CPL}(I_{\text{ref}}, I_{\text{gen}}) = \sum_{l \in L} \lambda_l \|\Phi_l(I_{\text{ref}}) - \Phi_l(I_{\text{gen}})\|^2$$

Where:

- $\Phi_l(I)$ represents the feature map from layer l of the CNN for image I.
- $\| \cdot \|^2$ denotes the squared Euclidean distance between the feature maps.
- λ_l is a weighting factor for the contribution of layer l to the overall loss.
- L is the set of layers used for computing the perceptual loss.

By comparing the feature maps of I_{gen} to those of I_{ref}, CPL measures the perceptual differences between the images, focusing on high-level characteristics rather than pixel-level accuracy.

Interpretation

- **Low CPL Value**: A lower Classifier Perceptual Loss value indicates that the generated image I_{gen} shares more perceptual characteristics with the reference image I_{ref}. This suggests high similarity in terms of textures, objects, and overall image content, making the generated image perceptually closer to the reference image.
- **High CPL Value**: A higher CPL value indicates greater perceptual differences between the generated and reference images. This suggests that the generated image may have lost important perceptual features or introduced artifacts that make it less similar to the reference image.

Use Cases

Classifier Perceptual Loss is widely used in various deep learning applications, including:

- **Image Super-Resolution**: CPL is used to enhance the perceptual quality of super-resolved images by ensuring that they not only match the high-resolution reference images in terms of pixel values but also retain high-level perceptual features.
- **Image Style Transfer**: In style transfer tasks, CPL helps to preserve the content of the original image while applying the style of another image, ensuring that the generated image maintains recognizable objects and structures.
- **Generative Adversarial Networks (GANs)**: CPL is often integrated into the loss functions of GANs to improve the realism of generated images by focusing on

perceptual features rather than pixel-wise differences.

Advantages

- **Focus on Perceptual Quality**: CPL directly targets the perceptual aspects of images, making it more aligned with human visual perception compared to traditional loss functions that focus solely on pixel-wise differences.
- **Effective for Complex Visual Tasks**: By leveraging feature maps from deep convolutional layers, CPL is effective in tasks that require preserving high-level content and structural details, such as image synthesis and restoration.

Limitations

- **Dependency on Pre-Trained Networks**: The effectiveness of CPL depends on the quality and architecture of the pre-trained classification network. Poorly trained networks or inappropriate architectures can lead to suboptimal results.
- **Computational Complexity**: Calculating CPL involves passing images through deep convolutional networks, which can be computationally expensive, especially when using multiple layers or large images.

Relationship with Other Metrics

Classifier Perceptual Loss is often used in conjunction with other loss functions, such as pixel-wise losses (e.g., MSE) or adversarial losses (in the case of GANs). While CPL focuses

on high-level perceptual features, other losses can complement it by ensuring accuracy at the pixel level or improving the overall distribution of generated images.

2.1.10 Semantic Segmentation Score (SSS)

The Semantic Segmentation Score (SSS) is a metric used to evaluate the semantic consistency and structural fidelity of generated images. This score is particularly important in tasks where the preservation of high-level semantic information, such as object boundaries and class labels, is critical. SSS measures the difference between the semantic details of a generated image and a reference image by applying a pre-trained segmentation model, such as Fully Convolutional Network (FCN-8), to both images and comparing the resulting segmentation maps. The goal of SSS is to quantify how well the generated image retains the semantic content of the reference image.

Mathematical Definition

Let S_{ref} represent the ground truth segmentation map of a reference image I_{ref}, and let S_{gen} represent the segmentation map obtained from the generated image I_{gen} using a pre-trained segmentation model Φ. The Semantic Segmentation Score is defined as:

$$\text{SSS}(I_{\text{ref}}, I_{\text{gen}}) = \frac{1}{n} \sum_{i=1}^{n} \|S_{\text{ref}}(i) - S_{\text{gen}}(i)\|$$

Where:

- $S_{\text{ref}}(i)$ and $S_{\text{gen}}(i)$ represent the semantic labels of the i-th pixel in the reference and generated images, respectively.
- $\|\cdot\|$ denotes a suitable norm, often the L1 or L2 norm, to measure the difference between the semantic labels.
- n is the total number of pixels in the images.

The SSS captures the extent to which the generated image I_{gen} retains the semantic structure of the reference image I_{ref}. A high SSS error indicates significant differences in the semantic segmentation maps, suggesting that the generated image may have lost important semantic details.

Interpretation

- **Low SSS Value**: A lower Semantic Segmentation Score indicates that the semantic segmentation map of the generated image is closely aligned with the ground truth segmentation of the reference image. This suggests that the generated image preserves the structural and semantic details of the reference image, such as object boundaries and class labels.
- **High SSS Value**: A higher SSS value indicates greater differences between the segmentation maps, suggesting that the generated image fails to retain the semantic content of the reference image. This could be due to distortions, artifacts, or loss of critical image structures during generation.

Use Cases

Semantic Segmentation Score is used in various image processing and computer vision tasks, including:

- **Image Synthesis and Generation**: SSS is employed to evaluate the quality of images generated by models such as Generative Adversarial Networks (GANs) or Variational Autoencoders (VAEs), ensuring that the generated images maintain the semantic content of the reference images.
- **Image-to-Image Translation**: In tasks where images are transformed from one domain to another (e.g., turning sketches into photos), SSS helps to assess whether the transformation preserves the semantic meaning and structure of the original images.
- **Image Restoration**: SSS is used to measure the effectiveness of image restoration techniques, such as inpainting or super-resolution, by ensuring that the restored images remain faithful to the original semantic structure.

Advantages

- **Focus on Semantic Integrity**: SSS directly measures the preservation of semantic information, making it a valuable metric for applications where maintaining the meaning and structure of the original image is crucial.

- **Alignment with Human Perception**: By evaluating the semantic content, SSS often aligns more closely with human perception of image quality, as humans are sensitive to the preservation of objects and structures in images.

Limitations

- **Dependency on Pre-Trained Models**: The accuracy of SSS depends heavily on the performance of the pre-trained segmentation model used. If the model is not well-suited to the task or the images in question, the SSS may not provide reliable assessments.
- **Computationally Intensive**: Computing SSS involves running a pre-trained segmentation model on potentially large datasets, which can be computationally expensive and time-consuming.

Relationship with Other Metrics

Semantic Segmentation Score complements other perceptual and pixel-wise metrics by focusing on the semantic and structural aspects of image quality. While other metrics like Classifier Perceptual Loss (CPL) may measure high-level feature similarity, SSS provides a direct assessment of the semantic content, making it particularly useful in tasks requiring detailed structural preservation.

2.1.11 Visual Information Fidelity (VIF)

Visual Information Fidelity (VIF) is a perceptual image quality metric that quantifies the amount of visual information preserved in a distorted or generated image relative to a reference image. VIF is based on the concept that the human visual system (HVS) is sensitive to the amount of information that an image conveys, and it aims to measure the loss or preservation of this information as the image undergoes various transformations or distortions. VIF is widely used in image processing and computer vision tasks to evaluate the fidelity of images generated by different models and algorithms.

Mathematical Definition

VIF is rooted in the information-theoretic framework, where the image quality is assessed based on the mutual information between the reference and the distorted images. The mutual information is computed at different levels of wavelet decompositions to capture both the global and local visual information.

Given a reference image I_{ref} and a distorted image I_{gen}, the Visual Information Fidelity is defined as:

$$\text{VIF} = \frac{\sum_{i=1}^{N} I(I_{\text{ref}}^{i}; I_{\text{gen}}^{i})}{\sum_{i=1}^{N} I(I_{\text{ref}}^{i}; \hat{I}_{\text{ref}}^{i})}$$

Where:

- I_{ref}^{i} and I_{gen}^{i} represent the coefficients of the reference and generated images at the i-th level of the wavelet decomposition.

- $\hat{I}^i_{\text{ref}}$ represents the noise-free estimate of the reference image coefficients at the i-th level.
- $I(\cdot;\cdot)$ denotes the mutual information between the coefficients.
- N is the number of wavelet decomposition levels considered.

VIF essentially measures how much of the original image information is retained in the generated image. A higher VIF value indicates that more information from the reference image is preserved in the generated image, leading to better visual fidelity.

Interpretation

- **High VIF Value**: A higher VIF value indicates that the generated image retains a significant amount of visual information from the reference image, suggesting high fidelity and perceptual quality. This implies that the image has preserved essential details, structures, and textures from the original image.
- **Low VIF Value**: A lower VIF value indicates a significant loss of visual information in the generated image compared to the reference image, suggesting lower fidelity. This can result from various factors, such as artifacts, blurring, or compression, leading to a degraded perceptual quality.

Use Cases

Visual Information Fidelity is used in a variety of image processing and computer vision applications, including:

- **Image Compression**: VIF is commonly used to evaluate the quality of compressed images by measuring the amount of visual information retained after compression. It helps in optimizing compression algorithms to achieve a balance between file size and image quality.
- **Image Restoration**: In image restoration tasks, such as denoising or deblurring, VIF is employed to assess how well the restored image recovers the original visual information lost due to noise or blur.
- **Super-Resolution**: VIF is used to evaluate the effectiveness of super-resolution algorithms by measuring how much high-frequency information, which corresponds to details and textures, is preserved in the upscaled image.

Advantages

- **Alignment with Human Perception**: VIF is designed to align closely with human visual perception by focusing on the preservation of visual information, making it a reliable metric for assessing image quality in perceptual terms.
- **Multi-Scale Analysis**: By considering multiple levels of wavelet decomposition, VIF captures both local and global visual information, providing a comprehensive

assessment of image quality.

Limitations

- **Complexity of Computation**: The calculation of VIF involves complex operations such as wavelet decomposition and mutual information estimation, making it computationally intensive compared to simpler metrics like PSNR or SSIM.
- **Dependency on Wavelet Transform**: The effectiveness of VIF can be influenced by the choice of wavelet transform and its parameters, which may require careful tuning for different types of images.

Relationship with Other Metrics

VIF is often compared with other perceptual quality metrics such as Structural Similarity Index (SSIM) and Peak Signal-to-Noise Ratio (PSNR). While SSIM and PSNR focus on structural similarity and pixel-wise accuracy, respectively, VIF provides a more holistic measure by quantifying the preservation of visual information, making it particularly useful in applications where perceptual quality is paramount.

2.2 Deep Neural Network Architectures

2.2.1 Autoencoders

Autoencoders are a type of artificial neural network used primarily for unsupervised learning. Their main purpose is to learn a compressed, low-dimensional representation (or encoding) of input data and then reconstruct the input from this encoding as accurately as possible. Autoencoders are composed of two main parts: an encoder and a decoder, with a bottleneck in between that forces the network to compress the input into a lower-dimensional representation.

Architecture

As shown in Figure 2.3, an autoencoder consists of three key components:

- **Encoder (g_ϕ)**: The encoder maps the input $\mathbf{x}$ to a lower-dimensional latent space $\mathbf{z}$ using a function parameterized by weights ϕ. The encoder reduces the dimensionality of the input, capturing the most important features. Mathematically, the encoder is defined as:

$$\mathbf{z} = g_\phi(\mathbf{x})$$

- **Bottleneck ($\mathbf{z}$)**: The bottleneck is the compressed representation of the input data. It contains the essential information needed to reconstruct the input while discarding noise and redundant information.

- **Decoder** (f_θ): The decoder takes the latent representation $\mathbf{z}$ and reconstructs the input $\mathbf{x}'$ using a function parameterized by weights θ. The decoder aims to reconstruct the input as closely as possible to the original. Mathematically, the decoder is defined as:

$$\mathbf{x}' = f_\theta(\mathbf{z})$$

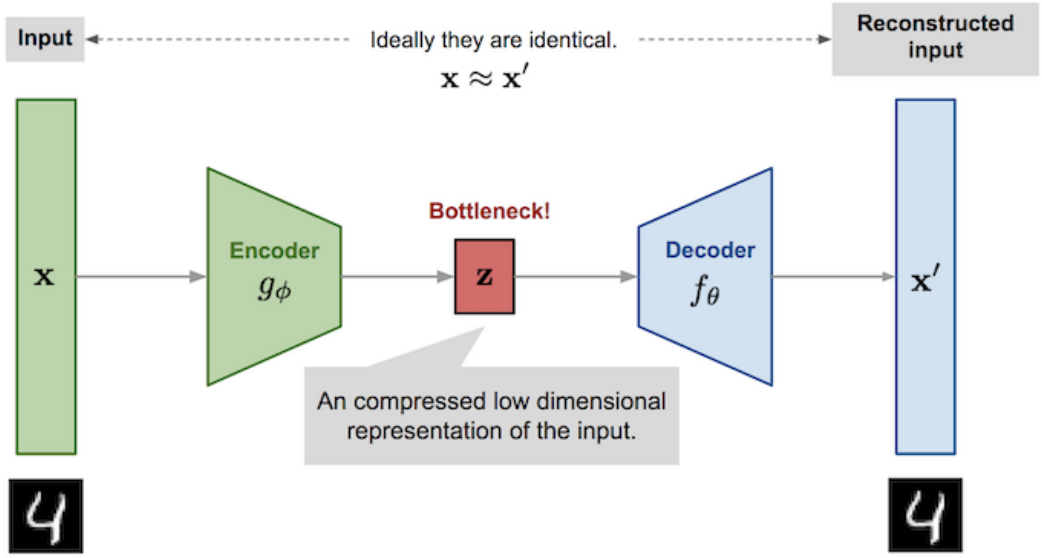


Figure 2.3: Architecture of an autoencoder, illustrating the encoding, bottleneck, and decoding processes [34]

Training

Autoencoders are trained to minimize the difference between the input $\mathbf{x}$ and the reconstructed output $\mathbf{x}'$. This is done by optimizing the parameters ϕ and θ of the encoder and decoder, respectively. The training process involves backpropagation and gradient descent to update the weights in a way that minimizes the reconstruction error.

Loss Function

The loss function used to train an autoencoder is typically the Mean Squared Error (MSE) between the input $\mathbf{x}$ and the reconstructed output $\mathbf{x}'$, defined as:

$$\mathcal{L}(\mathbf{x}, \mathbf{x}') = \frac{1}{n} \sum_{i=1}^{n} (\mathbf{x}_i - \mathbf{x}'_i)^2$$

Where:

- $\mathbf{x}_i$ represents the original input data.
- $\mathbf{x}'_i$ represents the reconstructed data.
- n is the number of data points.

The goal is to minimize this loss function so that the output $\mathbf{x}'$ is as close as possible to the input $\mathbf{x}$, indicating that the autoencoder has learned an efficient and accurate representation of the data.

Applications

Autoencoders have a wide range of applications, including:

- **Dimensionality Reduction**: Autoencoders can be used as an alternative to traditional methods like Principal Component Analysis (PCA) for reducing the dimensionality of data while preserving important features.

- **Anomaly Detection**: Since autoencoders learn to reconstruct the normal data distribution, they can be used to detect anomalies by identifying inputs that result in high reconstruction errors.
- **Image Denoising**: Autoencoders can be trained to remove noise from images by learning to map noisy images to their clean counterparts.
- **Generative Modeling**: Variants of autoencoders, such as Variational Autoencoders (VAEs), can be used for generating new data that is similar to the training data.
- **Feature Extraction**: The latent representation learned by the encoder can be used as features for other machine learning tasks, such as classification or clustering.

2.2.2 Variational Autoencoders (VAE)

Variational Autoencoders (VAEs) are a type of generative model that extends the concept of traditional autoencoders by introducing a probabilistic approach to the encoding and decoding process. VAEs are designed not only to reconstruct the input data but also to learn a latent representation that can be used to generate new data similar to the training data. This is achieved by imposing a probabilistic structure on the latent space, which allows for the generation of diverse samples.

Architecture

As depicted in Figure 2.4, a Variational Autoencoder consists of the following components:

- **Probabilistic Encoder ($q_\phi(\mathbf{z}|\mathbf{x})$)**: Unlike traditional autoencoders, the encoder in a VAE does not directly output a single latent vector. Instead, it produces parameters of a probability distribution, typically a Gaussian distribution, over the latent space. These parameters are the mean μ and standard deviation σ of the distribution:

$$\mu, \sigma = g_\phi(\mathbf{x})$$

A latent vector $\mathbf{z}$ is then sampled from this distribution:

$$\mathbf{z} = \mu + \sigma \odot \epsilon, \quad \epsilon \sim \mathcal{N}(0, \mathbf{I})$$

where ϵ is a random noise vector sampled from a standard normal distribution.

- **Latent Space ($\mathbf{z}$)**: The latent vector $\mathbf{z}$ represents a compressed, low-dimensional probabilistic encoding of the input data. It is sampled from the distribution defined by the mean and standard deviation produced by the encoder.

- **Probabilistic Decoder ($p_\theta(\mathbf{x}|\mathbf{z})$)**: The decoder in a VAE also takes a probabilistic approach. It maps the latent vector $\mathbf{z}$ back to the data space, aiming to reconstruct the input $\mathbf{x}'$. The decoder outputs the parameters of a probability distribution over the data space (e.g., the mean of a Gaussian distribution), from which the final reconstructed data is sampled:

$$\mathbf{x}' \sim p_\theta(\mathbf{x}|\mathbf{z})$$

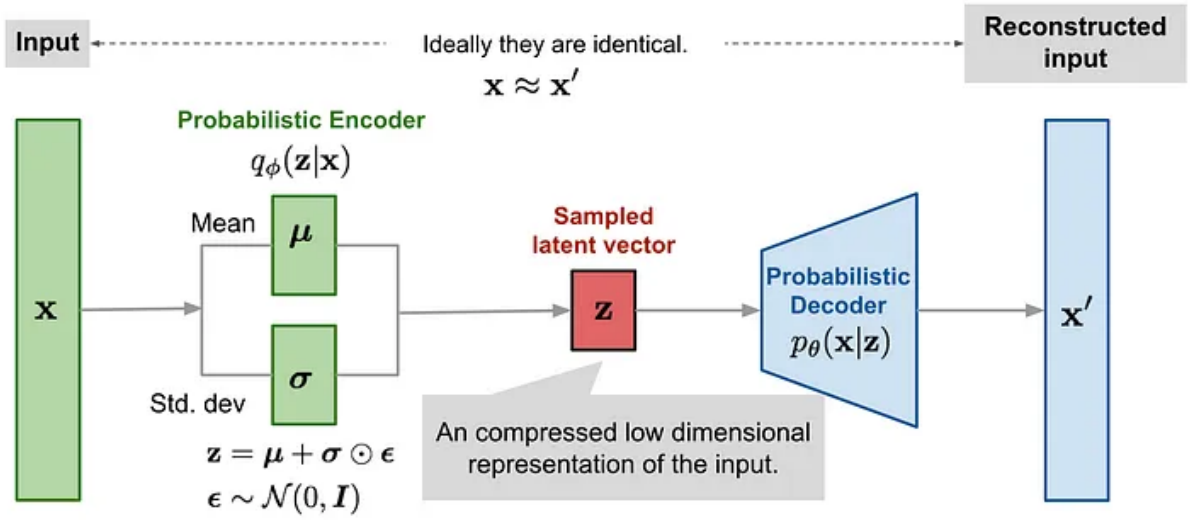


Figure 2.4: Architecture of a Variational Autoencoder (VAE), illustrating the probabilistic encoding, sampling from the latent space, and probabilistic decoding processes [2]

Training

Training a VAE involves optimizing two objectives simultaneously: reconstructing the input data and ensuring that the latent space follows a desired distribution, typically a standard normal distribution. This is achieved by minimizing the VAE loss function, which consists of two parts: the reconstruction loss and the Kullback-Leibler (KL) divergence.

Loss Function

The loss function for training a VAE is composed of the following two terms:

- **Reconstruction Loss**: This term measures how well the generated image $\mathbf{x}'$ matches the input image $\mathbf{x}$. It is often implemented as the Mean Squared Error (MSE) or

Binary Cross-Entropy (BCE) between the input and the reconstruction:

$$\mathcal{L}_{\text{reconstruction}} = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})]$$

- **KL Divergence**: This term regularizes the latent space by ensuring that the distribution of the latent variables $q_\phi(\mathbf{z}|\mathbf{x})$ is close to a prior distribution $p(\mathbf{z})$, usually a standard normal distribution $\mathcal{N}(0, \mathbf{I})$. The KL divergence between the two distributions is given by:

$$\mathcal{L}_{\text{KL}} = D_{KL}\left(q_\phi(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z})\right)$$

The total loss function for training a VAE is the sum of these two terms:

$$\mathcal{L}_{\text{VAE}} = \mathcal{L}_{\text{reconstruction}} + \mathcal{L}_{\text{KL}}$$

Minimizing this loss encourages the model to produce accurate reconstructions while maintaining a well-structured latent space.

Applications

Variational Autoencoders have a wide range of applications in various fields, including:

- **Generative Modeling**: VAEs can generate new data samples that are similar to the training data by sampling from the latent space. This makes them useful for tasks like image synthesis, text generation, and data augmentation.

- **Anomaly Detection**: By learning the normal distribution of the data, VAEs can be used to detect anomalies. Inputs that result in high reconstruction errors or unusual latent variables can be flagged as outliers.

- **Data Imputation**: VAEs can be used to fill in missing data by leveraging the learned distribution over the data space, providing plausible completions for missing values.

- **Latent Space Exploration**: The structured latent space learned by a VAE allows for exploration and interpolation between different data points, making it useful for visualizing and understanding the underlying structure of complex datasets.

- **Semi-Supervised Learning**: VAEs can be adapted for semi-supervised learning by combining the generative model with labeled data, allowing for improved performance on tasks with limited labeled data.

2.3 Machine Learning Classifiers

Classifiers are a fundamental aspect of machine learning, which is a branch of artificial intelligence that focuses on building systems that can learn from and make decisions based on data. In supervised learning, classifiers are algorithms that assign class labels to instances based on the input features. These labels are selected from a predefined set of possible outcomes. Effective classification is crucial for many applications, such as image recognition, email filtering, and medical diagnosis, where making accurate predictions based on

historical data can significantly impact performance. This section will explain four classifiers that are used in this work: Support Vector Machines (SVM), Logistic Regression, Random Forest, and Decision Trees.

2.3.1 Support Vector Machine (SVM)

Support Vector Machines (SVMs) are a robust set of supervised learning models primarily utilized for classification, though they also have applications in regression. The essence of SVMs as linear classifiers is their ability to delineate data points in an N-dimensional space with a hyperplane that distinctly classifies these points. The SVM algorithm focuses on identifying the hyperplane that maintains the greatest minimal distance to the training examples, a concept encapsulated in the term 'margin'. This margin, defined as the distance between the hyperplane and the nearest data point on either side, is integral to the model's performance; a larger margin is generally correlated with a lower generalization error of the classifier.

To address non-linear problems, SVMs employ the kernel trick. The kernel trick is a powerful technique used primarily in machine learning algorithms like Support Vector Machines (SVMs) to enable learning in high-dimensional spaces without explicitly performing the computationally expensive transformation. This section expands on the kernel trick, detailing its concept, how it works, and its advantages. The kernel trick revolves around the idea of transforming data into a higher-dimensional space to make it easier to separate using a linear classifier, such as an SVM. This is particularly useful in scenarios where the relationship between class labels and features is not linearly separable in the original

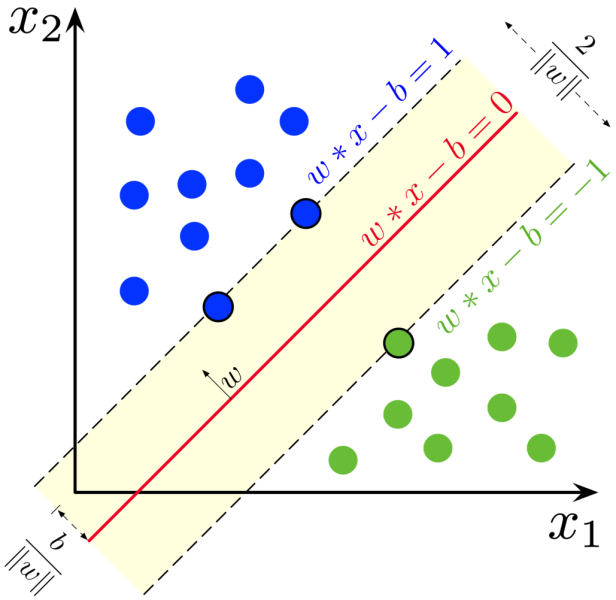


Figure 2.5: SVM separating 2-dimensional data points. [56]

feature space. In its essence, the kernel trick involves using a kernel function to compute the dot product of two vectors (data points) in a high-dimensional space without ever computing the coordinates of the points in that space. This allows the algorithm to operate in a high-dimensional feature space using the original inputs of the data, thus avoiding the direct computation of the coordinates in the high-dimensional space. Some of the famous kernels are mentioned below.

1. **Linear Kernel:** The linear kernel is the simplest form of the kernel function, represented by the inner product of two vectors. It is used when the data is linearly separable. This kernel is computationally efficient and often used in cases where the number of features is very large compared to the number of samples [17]. In

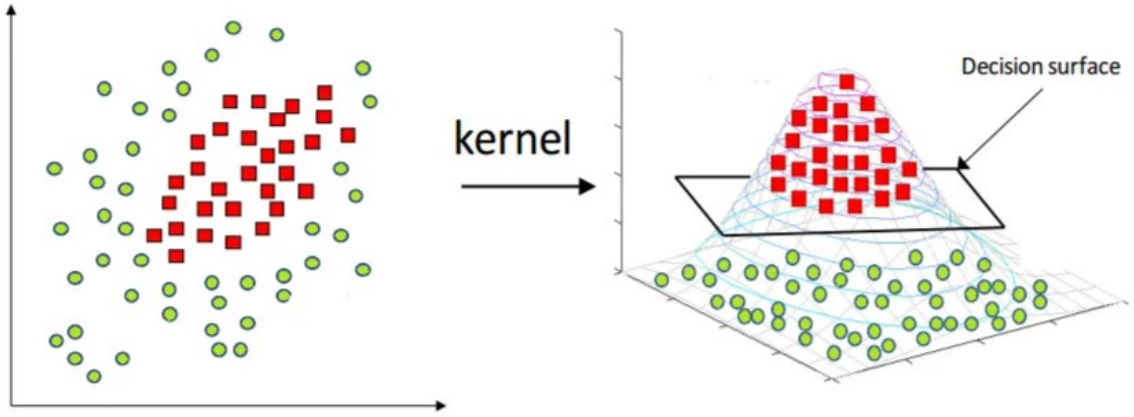


Figure 2.6: An intuitive example of how SVM kernels can separate data points that are not linearly separable. [47]

mathematical terms, it is defined as:

$$K(x, x') = \langle x, x' \rangle$$

2. **Polynomial Kernel:** The polynomial kernel computes a polynomial of the original variables, allowing for the modeling of more complex relationships between data points. It is defined as:

$$K(x, x') = (\gamma \langle x, x' \rangle + r)^d$$

where d is the degree of the polynomial, r is the coefficient term, and γ is a scale factor. This kernel can model interactions up to the specified degree, making it useful for problems where the relationship between features is not simply linear [50].

3. **Radial Basis Function (RBF) or Gaussian Kernel:** The RBF kernel is one of

the most popular kernel functions used in SVM due to its ability to handle non-linear relationships. It is defined as:

$$K(x, x') = \exp\big(-\gamma \|x - x'\|^2\big)$$

where γ is a parameter that determines the spread of the kernel. This kernel is effective in cases where the relationship between class labels and features is highly non-linear, as it maps the input features into an infinite-dimensional space.

4. **Sigmoid Kernel:** The sigmoid kernel, which mimics the activation function of a neural network, is defined as:

$$K(x, x') = \tanh(\gamma \langle x, x' \rangle + r)$$

This kernel transforms the data similarly to a two-layer, perceptron-like neural network and can model complex relationships between features. However, it is less commonly used compared to the RBF and polynomial kernels [1].

2.3.2 Logistic Regression

Logistic Regression is a statistical method primarily used for binary classification, but it can be extended to handle multi-class problems using certain frameworks such as One-vs-Rest (OvR) or Multinomial Logistic Regression. Despite its name suggesting a regression algorithm, it is actually used for classification tasks, estimating the probability that a given input belongs to a particular category.

The core idea behind Logistic Regression is to find a relationship between features and the probability of a particular outcome. In binary classification, it predicts the probability that the target variable (with two classes) belongs to one class, generally labeled as "1". This is achieved through the logistic function, which bounds the output between 0 and 1, ensuring it can be interpreted as a probability.

The mathematical model uses the logistic function or sigmoid function to express probabilities. The function is defined as:

$$P(y = 1 \mid x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n)}}$$

where:

- $P(y = 1 \mid x)$ is the probability that the target variable y is class 1 given the predictors x,
- $\beta_0, \beta_1, \ldots, \beta_n$ are the coefficients of the model,
- $x_1, \ldots, x_n$ are the feature values.

The coefficients of the logistic regression model are typically estimated using maximum likelihood estimation (MLE), a method that seeks to determine the set of coefficients that maximizes the likelihood of the observed data given the predictors [52].

Since Logistic Regression calculates a probability, a decision threshold needs to be set to determine the class assignment. By default, this threshold is often set at 0.5. If the predicted probability is greater than or equal to 0.5, the output is classified as class 1;

otherwise, it is classified as class 0. This threshold can be adjusted to tune the sensitivity and specificity of the classifier according to particular application needs [24].

Logistic Regression is highly interpretable, making it a preferred choice over more complex models such as neural networks or ensemble methods. It is easy to implement, efficient to train, and provides results that are straightforward to understand. The model performs well when the dataset linearly separates the classes and can be extended to multiclass classification using strategies such as One-vs-Rest (OvR) or Multinomial Logistic Regression [8].

Logistic Regression is extensively used in fields like medicine for predicting disease likelihood, finance for credit scoring, and marketing for predicting customer retention. For instance, it is used to predict whether a patient has a certain disease based on various predictors, such as age, sex, and blood test results [24, 52].

While versatile, Logistic Regression can struggle with non-linear relationships unless feature engineering is employed to include non-linear components. It also assumes linearity between the log-odds of the dependent variable and each predictor, which might not always hold true in real-world scenarios [8].

2.3.3 Decision Tree

Decision trees are a straightforward type of machine learning model primarily used for classification tasks, where they sort data into categories. However, they can also be used for regression to predict continuous values. Their simplicity and interpretability make them popular for tasks requiring clear decision-making steps.

A decision tree resembles a flowchart where each decision point represents a question based on data features. The paths represent the possible answers, and each endpoint (leaf) represents a prediction or outcome. This structure allows you to trace the path from the root to the leaf to understand the logic behind each prediction.

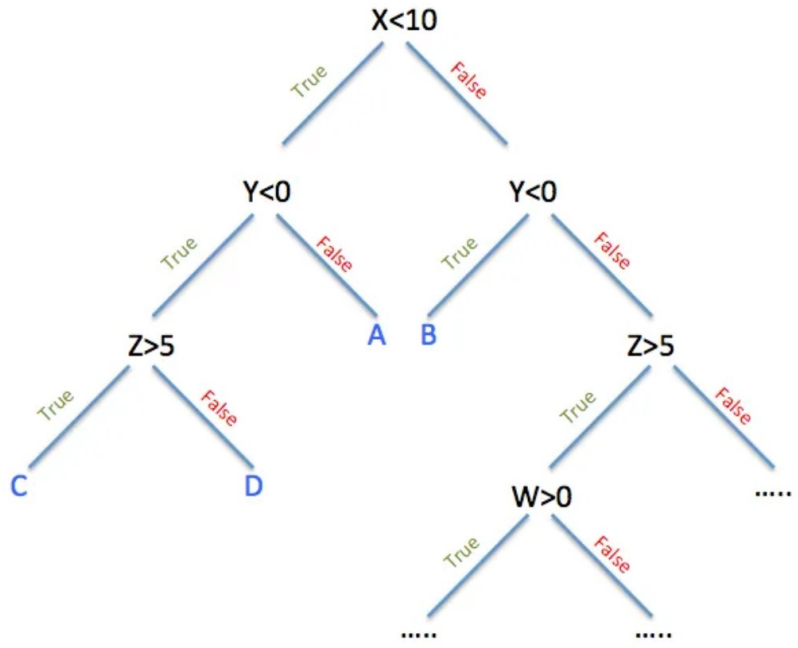


Figure 2.7: Simple example of a Decision Tree. [4]

The process of creating a decision tree involves splitting the data into smaller groups based on specific attributes. This splitting is done recursively until a stopping condition is met, such as a maximum tree depth or a minimum number of samples per leaf node. Choosing where to split the data is crucial and is typically done using methods that measure the quality of the split. Common methods include:

1. **Gini Impurity**: Measures the probability of incorrectly classifying a randomly cho-

sen element from the dataset. Lower values indicate better splits.

2. **Entropy**: From information theory, entropy measures the impurity or disorder in the data. Information gain is calculated to determine the reduction in entropy after a split.

3. **Gain Ratio**: Adjusts the information gain to account for biases in attributes with many categories, making it especially useful for dealing with such attributes.

One significant advantage of decision trees is that they require little to no data preparation. There is no need to scale or normalize your data, and they can handle both numerical and categorical data. This ability to work with various data types and uncover complex patterns makes them versatile [7, 22, 46].

However, decision trees have some limitations. They can easily create overly complex trees that do not generalize well to new data, a problem known as overfitting. They are also sensitive to small changes in the data, which can lead to different trees being generated for slightly different datasets. To mitigate these issues, techniques such as pruning (removing branches that have little importance), setting a maximum depth, and ensemble methods like Random Forests or Gradient Boosting are often used. These methods help stabilize the model and improve its generalization to new data [9, 25]

Decision trees are extensively used in various fields:

1. **Medicine**: For diagnosing diseases based on patient symptoms.

2. **Finance**: For credit risk assessment and stock price prediction.

3. **Marketing**: For identifying customer segments and guiding marketing strategies.

4. **Manufacturing**: For quality control and process optimization.

5. **Natural Language Processing**: For text classification tasks [7, 22, 46].

Despite their simplicity, decision trees provide valuable insights and clear decision-making logic, making them a robust tool in the machine learning arsenal. However, the trade-off between interpretability and complexity must be managed carefully, especially when dealing with large and complex datasets.

2.3.4 Random Forest

Random Forests are an extension of decision trees and one of the most effective machine learning models for prediction and classification tasks. They combine multiple decision trees to improve accuracy and control overfitting, making them more robust than individual decision trees [5].

A random forest is a collection of decision trees, each built on a different subset of the data and using different subsets of features chosen at random. The idea is that by averaging multiple deep decision trees, you reduce the risk of overfitting while maintaining high accuracy. The trees in a random forest run in parallel with no interaction between them, which makes the computations fast and scalable [33].

When building a random forest, each tree is constructed using a random sample of the data points, drawn with replacement from the original dataset, a method known as bootstrap aggregating or bagging. When splitting each node in the tree, instead of considering

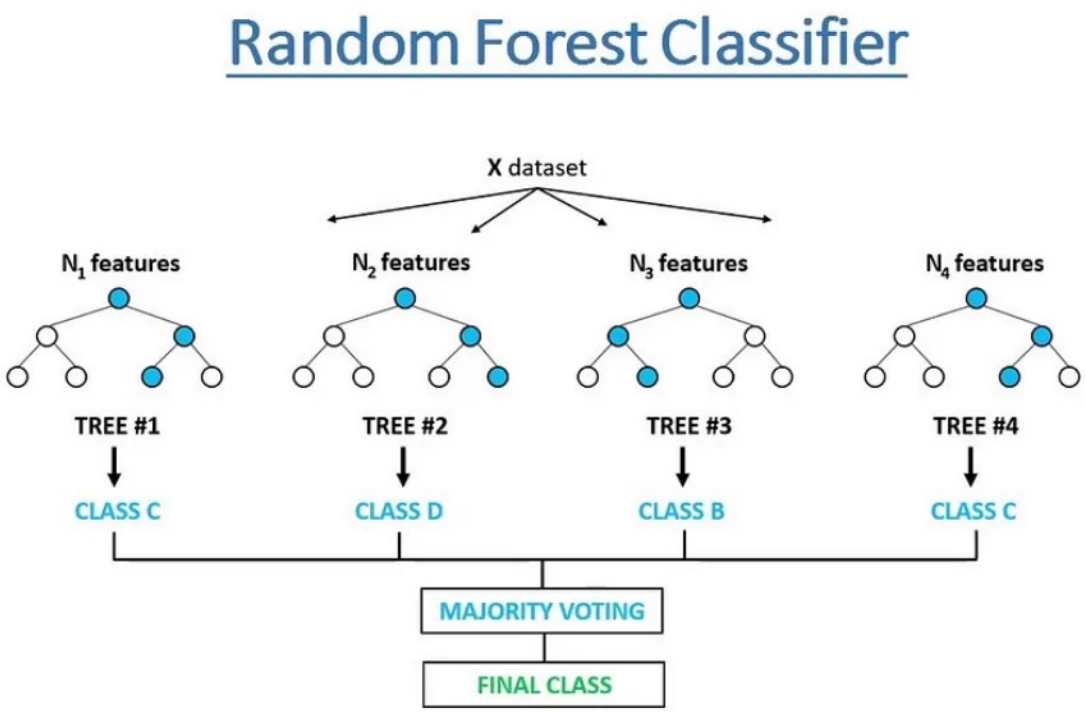


Figure 2.8: Simple example of a Random Forest. [35]

all features, a random subset of features is selected. This randomness helps in making the trees less correlated and increases the diversity among the trees, enhancing the overall performance of the model [5].

During training, each tree in the forest makes a decision based on the attributes of the input data points and votes for that decision. For classification tasks, the forest chooses the classification having the most votes over all the trees in the forest. For regression tasks, it typically averages the predictions of all the trees.

Random Forests are very versatile and can handle both regression and classification tasks. They perform well on large datasets and maintain accuracy even when a large

proportion of the data is missing. They also provide useful insights into feature importance, as they can measure how much each feature contributes to the final decision by averaging the decrease in impurity over all trees [15].

While Random Forests are powerful, they also come with their own set of challenges. They require more computational resources and can be slower to train compared to other algorithms due to the large number of trees. The model can also become complex, which might lead to longer prediction times. Moreover, if the trees are too deep, they can still suffer from overfitting, although much less so than individual decision trees [23].

2.4 Active Learning

Active Learning (AL) is a powerful machine learning technique that helps improve model performance by carefully choosing the most valuable data points to label. It's especially useful when labeled data is hard to come by or expensive to produce. The main idea is to focus on labeling the data that will give the model the most bang for its buck, so to speak—whether those data points are the most uncertain or offer the most new information. This approach allows models to get smarter with less labeled data, making it a critical strategy in data-driven AI development.

There are several strategies you can use to identify these key data points. For example, uncertainty sampling zeroes in on data where the model is most unsure, using techniques like entropy-based sampling or margin sampling. Query by Committee (QBC) takes a different route, using multiple models to predict the same data and then focusing on the

samples where these models disagree the most. Another approach, expected model change, looks for data points that, once labeled, would lead to significant updates in the model's understanding. Diversity sampling ensures that the labeled data covers a broad range of the input space, selecting samples that are different from those already labeled. Finally, adaptive sampling combines several of these methods, adjusting dynamically to pick the most informative samples based on the model's current state and the data it's working with. [38, 39]

As shown in Fig. 2.9, the Active Learning process generally involves several key steps designed to enhance the efficiency of machine learning models by selectively choosing the most informative data points for labeling. Here's a structured overview of these steps:

1. **Initial Model Training**

   The process begins with training an initial model on a small set of labeled data. This model serves as the starting point for further learning and helps in making initial predictions on the unlabeled data pool.

2. **Sample Selection**

   After training the initial model, it is used to predict labels for a large pool of unlabeled data. The critical step here is to apply a selection strategy to identify the most informative samples. Common strategies include Uncertainty Sampling, Diversity Sampling, and Query by Committee (QBC).

3. **Human Annotation**

   The selected samples are then presented to human annotators for labeling. This step

is essential because it ensures that the newly added data is of high quality, directly contributing to the model's learning process.

4. **Model Retraining**

   Once the new samples are labeled, the model is retrained using both the original and newly labeled data. This retraining helps the model improve its performance and accuracy by refining its decision boundaries.

5. **Iteration**

   The process is iterative. The newly retrained model again selects the most informative samples from the remaining unlabeled data, and the cycle of human annotation and model retraining continues until a stopping criterion is met, such as reaching a desired level of accuracy or exhausting the annotation budget.

6. **Final Model Deployment**

   Once the iterative process has sufficiently improved the model, it can be deployed for real-world applications. The final model is expected to perform well with fewer labeled instances, making Active Learning particularly useful in scenarios where labeled data is expensive or scarce.

Active Learning presents several important benefits, including improved efficiency, higher accuracy, and the integration of human expertise. By concentrating on the most informative samples, Active Learning reduces the amount of labeled data required, thus saving both time and resources. The iterative process of selecting uncertain samples and retraining the model contributes to refining the model's decision boundaries, resulting in

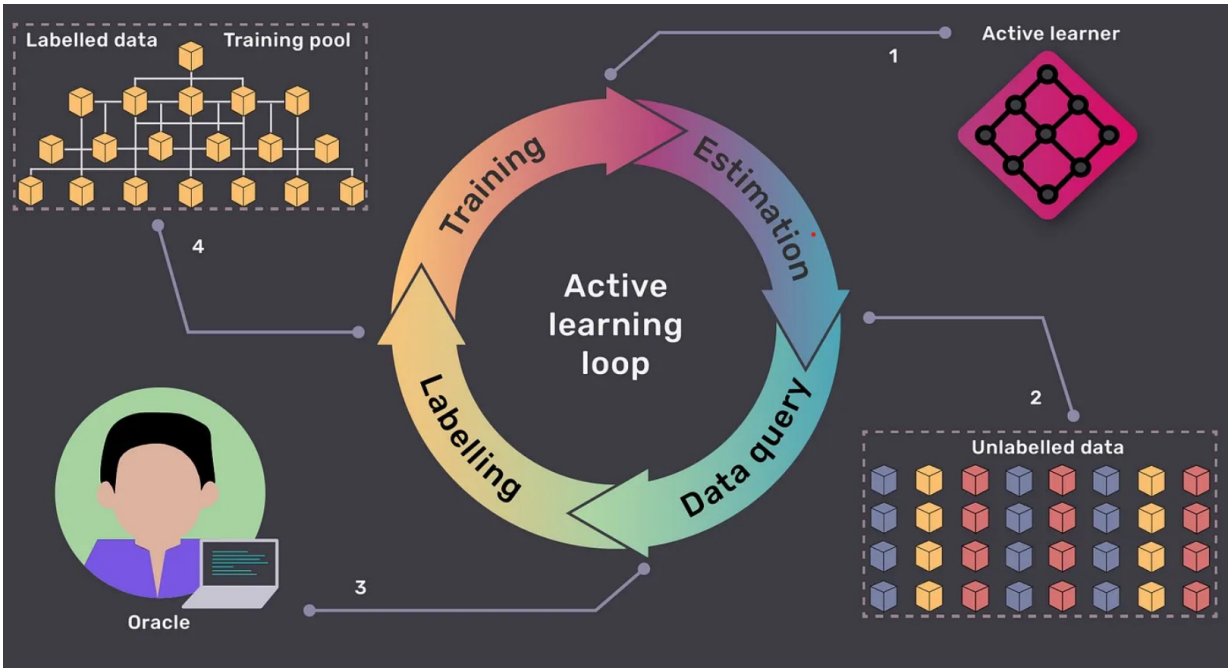


Figure 2.9: Workflow of Active Learning [18]

enhanced performance. Additionally, involving human annotators ensures that the labeled data is of high quality, which is crucial for the model's overall success.

Active Learning has proven to be highly effective in various domains, such as natural language processing, computer vision, and healthcare. For example, in computer vision, where labeling images can be especially labor-intensive, Active Learning enables models to achieve high accuracy with significantly fewer labeled images. This is particularly important in applications like medical image analysis, where expert annotations are both costly and time-consuming.

Table 2.1: A list of image-comparison metrics from the literature [21, 28, 41]: The metric name and its source, the metric definition, and the level at which the metric compares images, i.e., at the pixel level or at feature-level where the features are obtained from the latent space of a pre-trained deep learning model.

Metric	Definition	Comparison level
PSNR	Peak signal-to-noise ratio (PSNR) measures the ratio of the maximum possible pixel value to the mean squared error (MSE) between the original and transformed images.	Pixel-level
SSIM	Structural similarity index measure (SSIM) evaluates the quality of a transformed image by comparing its structural information, e.g., edges, textures, and patterns, with the original image.	Pixel-level
MSE	Mean square error (MSE) is the average squared difference between pixel values of the transformed and original images.	Pixel-level
TSI	Texture similarity index (TSI) measures the texture properties of the original and transformed images using gray-level co-occurrence matrices.	Pixel-level
WS	Wasserstein score (WS) measures the similarity between two images by calculating the minimum effort needed to transform one image into the other. It quantifies the cost of redistributing pixel values from the generated image to match the reference image.	Pixel-level
CS	Cosine similarity (CS) measures the similarity between high-level feature representations of the original and transformed images created by the convolutional layers of a pre-trained deep-learning model using cosine similarity.	Feature-level
KL	Kullback–Leibler (KL) divergence measures the difference between the pixel value distribution of the original and transformed images.	Pixel-level
$Hist_{int}$	Histogram intersection ($Hist_{int}$) measures the similarity by computing the intersection of the pixels histograms of the transformed and original images.	Pixel-level
$Hist_{cor}$	Histogram correlation ($Hist_{cor}$) measures the similarity by computing the correlation of the pixels histograms of the transformed and original images.	Pixel-level
CPL	Classifier perceptual loss (CPL) measures the difference in high-level representations obtained from a pretrained deep-learning model of the transformed and original images.	Feature-level
SSS	Semantic segmentation score (SSS) measures the semantic and structural differences by comparing segmentation outputs of the transformed and original images obtained by a pretrained segmentation model.	Feature-level
VAE-RE	Variational autoencoder reconstruction error (VAE-RE) measures how far a transformed image is from the distribution of a set of original images. To compute this measure, a VAE needs to be trained on a set of original images. VAE-RE is the loss function of the trained VAE for the transformed images.	Feature-level
VIF	Visual information fidelity (VIF) evaluates the visual quality of an image by quantifying the information shared between the transformed and original images.	Pixel-level

Chapter 3

Related Work

This chapter includes the recent related work in automated test input validation approaches.

3.1 VAE-Based Method

Dola et al. [10] propose DAIV, a distribution-aware approach to DNN testing that emphasizes the validation of test inputs using generative models, specifically Variational Autoencoders (VAEs). Existing DNN testing techniques, such as DeepXplore [37], DLFuzz [19], and DeepConcolic [45], have been widely used to assess the robustness of DNNs. However, Dola et al. identify a critical flaw in these methods: a substantial proportion of the test inputs generated by these techniques are invalid, meaning they do not fall within the distribution of the training data. Figure 3.1 shows some invalid test inputs generated by

Figure 3.1: Invalid test inputs generated by DeepXplore [37]

DeepXplore. The use of such invalid inputs not only leads to an inflated sense of test coverage but also risks generating misleading results, as these inputs do not represent realistic scenarios that the DNN is likely to encounter in deployment.

To mitigate the issue of invalid test inputs, the authors propose incorporating a VAE-based approach to validate the inputs generated by DNN testing techniques. The VAE model is trained on the same data distribution as the DNN under test, allowing it to accurately identify whether a given input falls within the valid input space. This validation process helps to ensure that only those test inputs which are relevant and realistic are used in testing, thereby improving the overall reliability of the test results. Dola et al. empirically demonstrate that their VAE-based approach significantly reduces the number of invalid inputs generated by existing techniques, leading to more accurate and reliable test outcomes. Their findings highlight the limitations of relying solely on traditional DNN testing methods without considering input validity and suggest that incorporating generative models into the testing process can enhance the robustness of DNN validation.

This work is closely related to our research, which also focuses on validating test inputs for DNNs. While Dola et al. use a VAE-based approach to ensure that test inputs are within the training data distribution, our work extends this concept by employing an active learning loop to dynamically select and validate test inputs using multiple image-comparison metrics. By addressing the limitations of single-metric validation and extending the analysis to industrial datasets, our approach aims to improve the accuracy of test input validation in specialized domains.

3.2 VIF-Based Method

Ensuring the reliability of Machine Vision Components (MVCs) in safety-critical systems, such as autonomous driving, presents significant challenges due to the potential for catastrophic failures if these systems misinterpret visual inputs. Hu et al. [21] propose a comprehensive framework for defining and verifying machine-verifiable reliability requirements for MVCs, specifically addressing safety-related image transformations.

Current deep neural network (DNN) testing approaches often lack clear reliability requirements, which can result in gaps in ensuring robustness against real-world conditions. Hu et al. argue that the reliability of MVCs should be benchmarked against human performance: if an image transformation does not impact a human's ability to interpret the image correctly, it should likewise not affect the MVC's performance. This principle aligns closely with the need in our research to validate test inputs for DNNs, ensuring they remain within expected distributions and are representative of real-world scenarios.

The authors introduce two classes of reliability requirements: *correctness-preservation* and *prediction-preservation*. The correctness-preservation requirement asserts that an MVC's predictions should remain accurate after an image has undergone a transformation that falls within a human-tolerated range. The prediction-preservation requirement ensures that an MVC's predictions remain consistent before and after such transformations. These requirements are instantiated using human performance data, collected from approximately 2000 human participants, to establish a baseline for evaluating MVC reliability across a variety of realistic image transformations.

A key contribution of Hu et al. is the formulation of the *Visual Change* metric, denoted as Δ_v, which quantifies the perceptual change in images due to transformations. This metric is defined as follows:

$$\Delta_v(x, x') = \begin{cases} 0 & \text{if } \mathrm{VSNR}(x, x') = \infty \text{ or } \mathrm{VIF}(x, x') > 1 \\ 1 - \mathrm{VIF}(x, x') & \text{otherwise} \end{cases}$$

where x is the original image, x' is the transformed image, VSNR is the Visual Signal-to-Noise Ratio, and VIF is the Visual Information Fidelity. This metric allows the reliability requirements to be framed in terms of perceptual changes that humans can tolerate, ensuring that MVCs are only tested on realistic and relevant transformations.

The framework also includes an automated method for checking whether MVCs meet these requirements, which involves generating test cases within the specified Δ_v range, executing these tests on the MVC under evaluation, and calculating the *reliability distance*—a metric that indicates the extent to which the MVC satisfies the defined reliability criteria.

Hu et al. demonstrate the effectiveness of their approach through extensive empirical evaluations using state-of-the-art image classification models. Their results reveal that the proposed method can identify reliability gaps that are not detected by other existing approaches, underscoring the importance of human performance data in defining meaningful reliability requirements.

The human effort involved in this process is considerable. Specifically, 2000 human participants were tasked with annotating original and transformed images under time-constrained conditions to simulate real-world decision-making scenarios. This annotation process is labor-intensive. The work by Hu et al. is directly applicable to our research on validating test inputs for vision-based DL systems. We use their introduced metric as a baseline to our method and show that we outperform their metric in both public and industrial datasets.

The validation of test inputs for Deep Learning (DL) systems is a critical challenge, particularly when it comes to ensuring that artificially generated inputs are valid and represent real-world scenarios that the DL system may encounter. Riccio and Tonella [41] address this challenge by conducting a comprehensive empirical study on the effectiveness of various Test Input Generators (TIGs) in producing valid test inputs for DL systems. Their work is particularly relevant to research focused on validating and testing DL systems, as it highlights the complexities involved in maintaining input validity and preserving ground-truth labels.

Riccio and Tonella identify that while TIGs are essential for generating diverse inputs to evaluate the robustness of DL systems, these generated inputs often fail to maintain

validity. The authors define valid inputs as those that are recognizable and interpretable by human experts within the specific input domain. They emphasize the importance of this validity in providing a reliable assessment of the DL system's quality, as invalid inputs can lead to misleading conclusions about the system's performance.

The study evaluates five different TIGs across three classification tasks, using both automated validators and human assessors to determine the validity of the generated inputs. Figure 3.2 shows some invalid and valid test inputs generated by TIGs. The results reveal that while automated validators can achieve a good level of accuracy (78%) in assessing input validity, they still struggle with complex, feature-rich datasets, and do not always align with human judgment. This finding underscores the limitations of relying solely on automated validators, highlighting the need for human involvement in the validation process—especially when dealing with nuanced or complex datasets.

Moreover, Riccio and Tonella also explore the challenge of label preservation, where the expected label of a generated input should remain consistent with its original ground-truth label. Their study finds that many TIGs, while generating valid inputs, fail to preserve the correct label, which can significantly impact the reliability of testing outcomes. This issue is particularly pertinent when assessing the robustness of DL systems, as mislabeling can lead to incorrect assessments of system performance and reliability.

This work aligns closely with our research to improve the validation and testing processes for DL systems by ensuring that test inputs are not only valid but also representative of realistic and meaningful scenarios. The insights provided by Riccio and Tonella about the limitations of TIGs and the need for a more integrated approach to validity assess-

ment—combining automated tools with human oversight—are invaluable for developing more robust testing methodologies.

We have used the approaches proposed by Hu et. al. [21] and Riccio and Tonella [41] as baselines in our work. We have shown that our approach, HiL-TV, significantly outperforms these baselines for both the industrial and the public datasets.

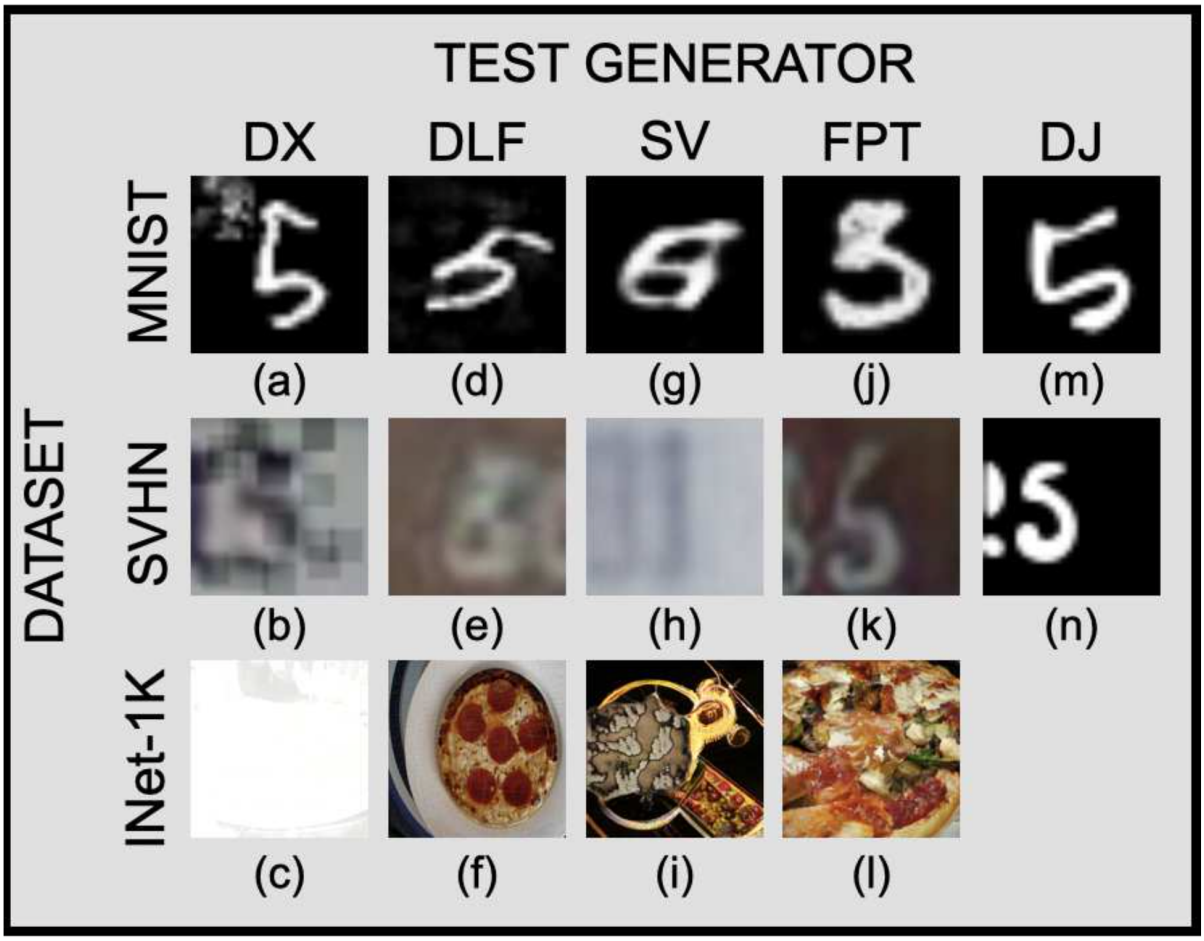


Figure 3.2: Some invalid and valid test inputs generated by TIGs in Riccio and Tonella's experiments [41]

Chapter 4

Automated, Human-in-the-Loop Test Input Validation

This chapter presents our automated, human-in-the-loop test input validation approach for vision-based DL systems (HiL-TV). Figure 4.1 provides an overview of HiL-TV. In summary, given a dataset D comprising pairs of original and transformed images, HiL-TV is designed to determine whether the transformed images are valid test inputs, while minimizing human effort.

Our approach begins with a subset of image pairs in D that have been pre-validated, meaning they are labeled as either valid or invalid. This pre-validated subset is crucial for selecting appropriate metrics and training an effective classifier for the active learning loop. HiL-TV operates through three main steps, which are detailed below:

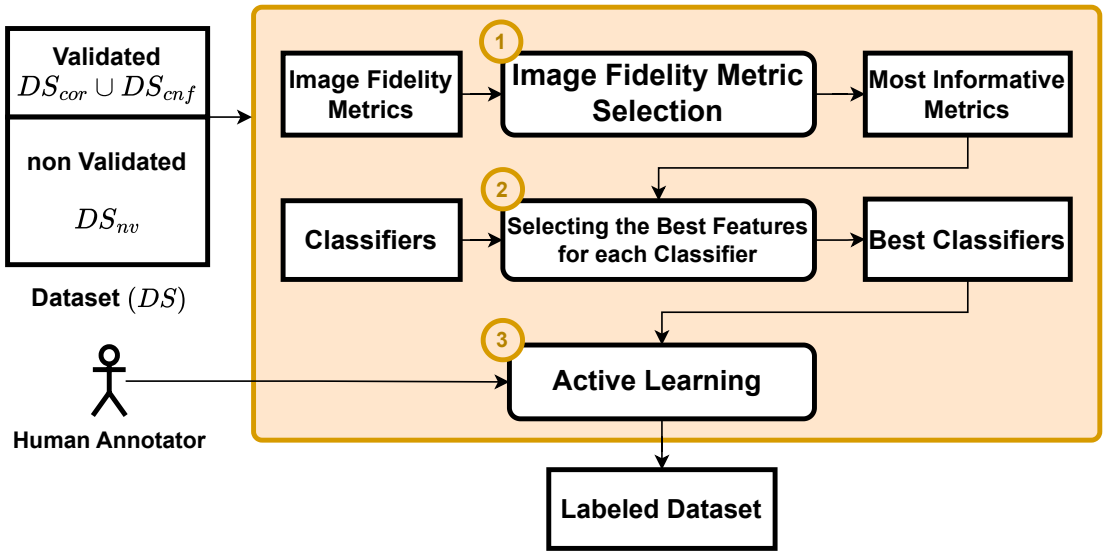


Figure 4.1: An overview of our test input validation approach for vision-based Deep learning systems (HiL-TV)

4.1 Step 1: Image-comparison metric selection.

The first step of HiL-TV is to identify the optimal image-comparison metrics for validating the test inputs in a given dataset.

We conducted a comprehensive literature review spanning both the computer vision and software testing communities to identify various image-comparison metrics suitable for evaluating the content similarity between an original image and its transformations [21, 28, 41]. In this survey, we encountered a range of metrics and carefully selected those that were most relevant to our work. Some of these metrics had been previously employed within the software testing community for similar purposes, and we included these in our set, using them as a baseline for comparison in our study.

Among the literature, a key reference was the work of Stefano Carlo Lambertenghi and Andrea Stocco [28], published in the 2024 IEEE Conference on Software Testing (ICST). In their study, Lambertenghi and Stocco conducted an extensive review, gathering existing image comparison metrics at both the distribution level and the single-image level. We incorporated several of their metrics that aligned closely with our objectives, leveraging them to strengthen the robustness and applicability of our approach. These identified metrics are discussed in Section 2.1.

The primary goal of these image-comparison metrics is to determine whether the essential content of an image is preserved in its corresponding transformation. This is achieved by quantifying the similarity or difference between the original image and its transformed version using a numeric value. Some of the metrics studied perform this comparison at the pixel level, meaning they directly use pixel values to measure the difference between the two images. These pixel-level comparisons are often straightforward and can quickly highlight any discrepancies between the original and transformed images.

On the other hand, other image-comparison metrics in Table 2.1 operate based on high-level representations. These high-level comparisons involve feeding images into a pre-trained deep-learning model and using the extracted features from the latent space of that model to compare the images. High-level metrics are particularly useful for capturing more abstract or semantic similarities and differences that might not be apparent at the pixel level.

Among the image-comparison metrics listed in Table 2.1, only VAE (Variational Autoencoder) and VIF (Visual Information Fidelity) have been previously used specifically

for assessing the validity of transformed test inputs [21, 41]. These metrics have shown promise in determining the quality and validity of transformations in vision-based deep learning systems, and they serve as a benchmark for evaluating other metrics.

Given a dataset D, HiL-TV selects the most effective image-comparison metrics from those in Section 2.1 that best distinguish valid from invalid pairs within D. To achieve this, HiL-TV utilizes a validated subset, $D_{cor} \subset D$, and systematically evaluates each metric on the image pairs within D_{cor}. The selection process involves two stages of filter selection [20]. In the first stage, HiL-TV computes the correlation between each image-comparison metric and the validity of the image pairs in D_{cor}. The top-k metrics with the highest correlation to validity are selected as the most informative metrics for distinguishing between valid and invalid test inputs.

In the second stage, HiL-TV checks for redundancy among the selected top-k image-comparison metrics. Specifically, it evaluates whether any pair (m_i, m_j) of these metrics is highly correlated, i.e., has a correlation value greater than 90%, based on their values for the image pairs in D_{cor}. If a redundant pair is identified, the metric with the lower validity correlation is replaced by the next best metric that is not already among the top-k.

The value of k is chosen based on HiL-TV's computational budget, as in the next step, classifiers will be trained on all possible subsets of the top-k image-comparison metrics, which amounts to $2^k - 1$ combinations.

At the conclusion of this step, HiL-TV provides the k most informative and distinct image-comparison metrics that are best suited for distinguishing valid test inputs from invalid ones. This careful selection ensures that the subsequent steps in the process are

based on the most relevant and non-redundant metrics, thereby improving the accuracy and efficiency of the test input validation.

4.2 Step 2: Feature selection and best classifier identification.

After identifying the set of relevant metrics in Step 1, the next critical task in Step 2 is to select a classification technique that can effectively distinguish between valid and invalid images when trained using these metrics as input features. This step is fundamental to ensuring the accuracy and reliability of the overall system. It considers a range of widely used and well-established classification techniques, including random forest, decision tree, support vector machine (SVM), and logistic regression. These methods are selected for their proven track record in handling various types of classification tasks and their ability to work well with diverse datasets.

The performance of these classifiers, however, is not solely dependent on the algorithms themselves; it also hinges significantly on the design of the input features. The metrics identified in Step 1 serve as these features, and their careful selection and arrangement are crucial for maximizing the classifier's effectiveness. This step involves a thorough evaluation of alternative input feature designs by systematically matching each classifier with all possible non-empty subsets of the informative metrics identified earlier. By exploring these different combinations, the process ensures that the classifier is equipped with the most relevant and non-redundant features, thereby optimizing its performance.

To ensure that these evaluations are both robust and unbiased, a validated subset of the dataset, denoted as D_{cnf}, is employed. This subset is distinct from D_{cor}, which was used in the first step for selecting the metrics. By using a separate dataset for this phase, the evaluation process avoids any potential bias that could arise from using the same data for both metric selection and classifier training. The use of D_{cnf} ensures that the assessments are rigorous and that the chosen classification technique and feature set are well-suited for the task of distinguishing between valid and invalid images in a reliable manner.

4.3 Step 3: Active learning.

The third step in the process involves the semi-automatic validation of the non-validated pairs in D through the use of active learning, as detailed in Algorithm 1. This algorithm is designed to streamline the validation process by combining automated predictions with user input. It takes as input the non-validated subset D_{nv} of the dataset D and the best classifier c that was developed in the second step. This classifier is a binary classifier due to the nature of our dataset which consists of valid and invalid image pairs. During each iteration, Algorithm 1 leverages classifier c to predict labels for the remaining non-validated image pairs. If the classifier's prediction confidence for a given pair exceeds the user-defined threshold α, the prediction is automatically accepted. The pair is then moved from the non-validated set D_{nv} to the validated set D_{val} (lines 2–8), effectively reducing the number of pairs requiring further validation.

However, in cases where the classifier cannot confidently validate any more pairs, the algorithm prompts the user to manually label up to β randomly selected non-validated

Algorithm 1 Active learning of HiL-TV

Input α: The confidence threshold for acceptable prediction
Input β: # of pairs to be validated by human in each iteration

1: **while** unlabeled data exists **do**
2: **for** every sample d **do**
3: $(prediction, confidence) \leftarrow \textsc{Predict}(C, d)$
4: **if** $confidence \geq \alpha$ **then**
5: Add d to D_{val}
6: **end if**
7: **end for**
8: Randomly select β samples from unlabeled dataset
9: Ask Human to validate
10: $C \leftarrow \textsc{FineTune}(C, humanLabeledSamples)$
11: **end while**
12: **return** D_{val}

pairs (lines 9–10). These user-labeled pairs are subsequently added to the validated set D_{val} (line 11), contributing to the training data. The classifier c is then fine-tuned using this expanded set D_{val}, which enhances its performance and accuracy for subsequent iterations (line 12). This iterative process allows the classifier to continuously improve, gradually reducing the workload on the user while increasing the proportion of validated pairs.

The active learning loop continues until all image pairs in the dataset have been validated, ensuring that the entire dataset D is accurately and comprehensively labeled. This method not only maximizes efficiency by minimizing manual labeling but also ensures that the classifier is optimally trained to handle future validation tasks with increased confidence and precision.

Chapter 5

Empirical Evaluation

In this chapter, we present our empirical evaluation for assessing our approach described in Chapter 3.

5.1 Research Questions

In this section, we evaluate HiL-TV using two datasets: our industrial dataset as well as a public-domain dataset. Our experiments begin with **RQ1** and **RQ2**, which, respectively, assess the capability of HiL-TV in identifying informative image-comparison metrics for each dataset and developing classifiers that are effective for validating test inputs for each dataset. **RQ3** demonstrates that HiL-TV effectively explores alternative trade-offs between accuracy and the manual effort required for the test input validation process. Finally, **RQ4** evaluates the effectiveness of HiL-TV compared to two state-of-the-art methods in the literature [21, 41] that rely on a single image-comparison metric for test input validation.

RQ1 (Image-comparison metric selection). *Which image-comparison metrics are the most informative for validating our industrial and public datasets?* We conduct experiments for both industrial and public datasets to identify the most informative metrics among those in Table 2.1 for validating each dataset.

RQ2 (Feature selection and best classifier identification). *Which subset of the informative metrics identified in RQ1, when combined with which classifier, yields the best prediction accuracy for test input validation of our industrial and public datasets?* We consider four classifiers: random forest, decision tree, SVM, and logistic regression. We evaluate the prediction accuracy of each of these classifiers using all the non-empty subsets of the metrics identified in RQ1 as the classifier's input features. For each of the industrial and public datasets, we choose the combination of classifier and input features that shows the highest accuracy for predicting valid test inputs.

RQ3 (Optimizing the trade-off between accuracy and effort). *How well does our active learning approach optimize the trade-off between accuracy and human effort for test input validation across each of our industrial and public datasets?* We evaluate the active learning step of HiL-TV as presented in Algorithm 1 based on the best instances of the random forest, decision tree, SVM, and logistic regression classifiers from RQ2 for both datasets. To explore different trade-offs between accuracy and human effort in the test input validation process, we vary the values of the active learning loop parameters, namely the confidence threshold α and the number of pairs to be manually validated at each iteration β. We then identify the parameter values that lead to the desired trade-offs, which are selected based on feedback from the industry partner.

RQ4 (State-of-the-art comparison). *How well does our test input validation framework perform compared with state-of-the-art test input validation frameworks in the literature?* We examine two baselines, the best classifier, and active learning methods with different levels of human effort (25%, 50%, 100%) and compare the results in terms of accuracy, F1-score, and human effort.

5.2 Subject Datasets

Industrial dataset. This dataset, provided by our industry partner SmartInside AI (SIA), includes 847 unique images of power facilities mounted on utility poles in South Korea. These images were used to train a DL-based anomaly detection system. To simulate Canada's climatic conditions, SIA transformed the images using InstructPix2Pix [6], a fine-tuned diffusion model that applies localized changes while maintaining the images' overall structure. SIA created nine transformations representing rain, fog, and snow at three intensity levels: light, moderate, and heavy, resulting in a total of $847 \times 9 = 7,623$ transformed images.

Public dataset. We use the CIFAR10-C dataset as a public dataset [21], which was generated by applying eight safety-related image transformations to the *Car* class of the CIFAR10 dataset [27]. The CIFAR10-C dataset consists of two classes, "Car" and "Not Car," comprising 1,273 pairs of original and transformed images. Similar to our industrial dataset discussed earlier, the validity of each pair is indicated by a label with values of "Valid" or "Invalid". We use the labels provided by Hu et al. [21]. It is worth mentioning that only a portion of the data was available online. Table 5.1 shows properties of the

Table 5.1: Key characteristics of our public and industrial datasets

Dataset	Size	Valid Pairs	Image Size (Pixels)
SmartInside (industrial)	7623	54.2 %	512 x 320
cifar10-c (public)	1273	16.4 %	32 x 32

Table 5.2: Dataset splits and their number of elements

Dataset	Correlation set for RQ1 (D_{cor})	Configuration set for RQ2 (D_{cnf})	Test set for RQ3 and RQ4 (D_{test})
Industrial	1298 (20%)	1298 (20%)	3897 (60%)
Public	254 (20%)	254 (20%)	764 (60%)

industrial and public datasets.

5.3 Experiment setup

For our experiments, we randomly divide each dataset into three disjoint subsets: D_{cor}, D_{cnf}, and D_{test}. The D_{cor} and D_{cnf} sets are defined in Figure 4.1, and the D_{test} set is the same as the non-validated set, i.e., D_{nv}, in Figure 4.1. In the remainder of this section, we refer to D_{cor}, D_{cnf}, and D_{test} as *correlation, configuration and test* sets, respectively. Specifically, D_{cor} is used to compute the best image-comparison metrics in RQ1, D_{cnf} is used for feature selection and training the best classifier in RQ2, and D_{test} is used for evaluating the active-learning loop of HiL-TV in RQ3 and RQ4. Table 5.2 shows the partitioning of each of our datasets into these three sets. Note that HiL-TV only requires the D_{cor} and D_{cnf} sets to be validated, i.e., to be labelled as valid and invalid. However, to answer RQ3 and RQ4, we require the labels for the D_{test} set to be able to evaluate the accuracy of HiL-TV.

Table 5.3: The Pearson correlation coefficients between each metric in Table 2.1 and the valid labels in the correlation set, D_{cor}, of the industrial and public datasets. The values highlighted green represent the metrics with the highest correlations for each dataset. The VIF metric for the industrial dataset is removed due to being redundant.

	PSNR	CPL	CS	SSIM	MSE	WD	KL	SSS	TSI	HIST_C	HIST_I	VAE-RE	VIF
Industrial	0.47	-0.58	0.78	0.54	-0.23	-0.26	-0.07	-0.36	-0.42	0.16	0.18	0.02	0.70
Public	0.10	0.16	0.21	0.16	-0.10	-0.07	0.13	-0.11	-0.05	0.01	0.12	0.21	0.12

5.4 RQ1 Experiments and Results.

To answer RQ1, we perform the first step of HiL-TV discussed in Section 4.1. We compute the Pearson correlation between each of the 13 metrics in Section 2.1 and the valid labels in the correlation set, D_{cor}. After selecting the top metrics, we compute the Pearson correlation between each pair of these selected metrics and then eliminate the redundant ones. This experiment is performed on the correlation sets for the industrial and public datasets.

Results. Table 5.3 shows the Pearson correlation coefficients between each metric and the valid labels in the correlation sets of our two datasets. Note that the correlations are generally higher for the industrial dataset than for the public one. This is because the images in our industrial dataset have significantly higher resolutions than those in the public dataset (see Table 5.1). Images of our public dataset are from CIFAR10 and their size is 32×32 pixels while the size of samples for our industrial dataset is 512×320. For the industrial dataset, we select the metrics with an absolute correlation coefficient above 0.5, resulting in the selection of the following five metrics: PSNR, CPL, CS, SSIM, and VIF. Table 5.4 shows the pairwise correlations assessing potential redundancies between these five metrics. The pairwise comparison reveals a strong correlation between CS and

Table 5.4: The pairwise Pearson correlations between the five metrics highlighted green in Table 5.3 for the industrial dataset. The cells highlighted in yellow indicate a high level of redundancy between VIF and CS.

Metric	PSNR	CPL	CS	SSIM	VIF
PSNR	1.0	-0.60	0.63	0.74	0.67
CPL	-0.60	1.0	-0.74	-0.76	-0.71
CS	0.63	-0.74	1.0	0.74	0.93
SSIM	0.74	-0.76	0.74	1.0	0.76
VIF	0.67	-0.71	0.93	0.76	1.0

Table 5.5: The pairwise Pearson correlations between the four metrics highlighted green in Table 5.3 for the public dataset.

Metric	CPL	CS	SSIM	VAE
CPL	1.00	-0.48	-0.09	0.56
CS	-0.48	1.00	0.58	0.07
SSIM	-0.09	0.58	1.00	0.43
VAE	0.56	0.07	0.43	1.00

VIF, with a value of 0.93. Since the other pairwise correlations are all below 0.76, they do not indicate a high degree of redundancy between the compared pairs. To eliminate redundancy, between VIF and CS, we remove the one with a lower correlation value in Table 5.1, i.e., VIF. Hence, the PSNR, CPL, CS, and SSIM metrics are selected for the industrial dataset.

As for the public dataset, we select the top four metrics highlighted in Table 5.3 to be consistent with the industrial dataset. The pairwise comparison between these four metrics does not indicate a high degree of overlap between any pair as their pairwise correlations are well below 0.9. Hence, CPL, CS, SSIM, and VAE-RE are selected for the industrial dataset.

RQ1 Answer: Among the image-comparison metrics in Section 2.1, CPL, CS, PSNR and SSIM are identified as the most important metrics for our industrial dataset. For our public domain dataset, CPL, CS, SSIM, and VAE-RE are identified to be the most important metrics.

5.5 RQ2 Experiments and Results.

To answer RQ2, we perform the second step of HiL-TV in Figure 4.1. As discussed in Section 4.2, we consider the following classification techniques: random forest, decision tree, SVM, and logistic regression. To identify effective input features for these classifiers, we train them with all non-empty subsets of the metrics identified for each dataset in RQ1. Since four different metrics are selected for each dataset in RQ1, we train 15, i.e., $2^4 - 1$, different instances of each classifier for each dataset, where each instance is trained with input features that are a subset of the metrics identified in RQ1. We train these classifier instances on the configuration sets, D_{cnf}, of our industry and public datasets. Specifically, for each classifier instance, we perform a five-fold cross validation on the configuration set, by training and tuning the classifier instance on four training folds and testing it on the test fold. We assess the performance of each classifier instance using the accuracy, precision and recall metrics. In total, for RQ2, we trained and tested $60 = 15(\textit{instances}) \times 4(\textit{classifiers})$ different classifiers.

Results. Figure 5.1 and 5.2 show the accuracy for each classifier with different subsets of image-comparison metrics for each dataset. Table 5.6 shows the classifier instances with

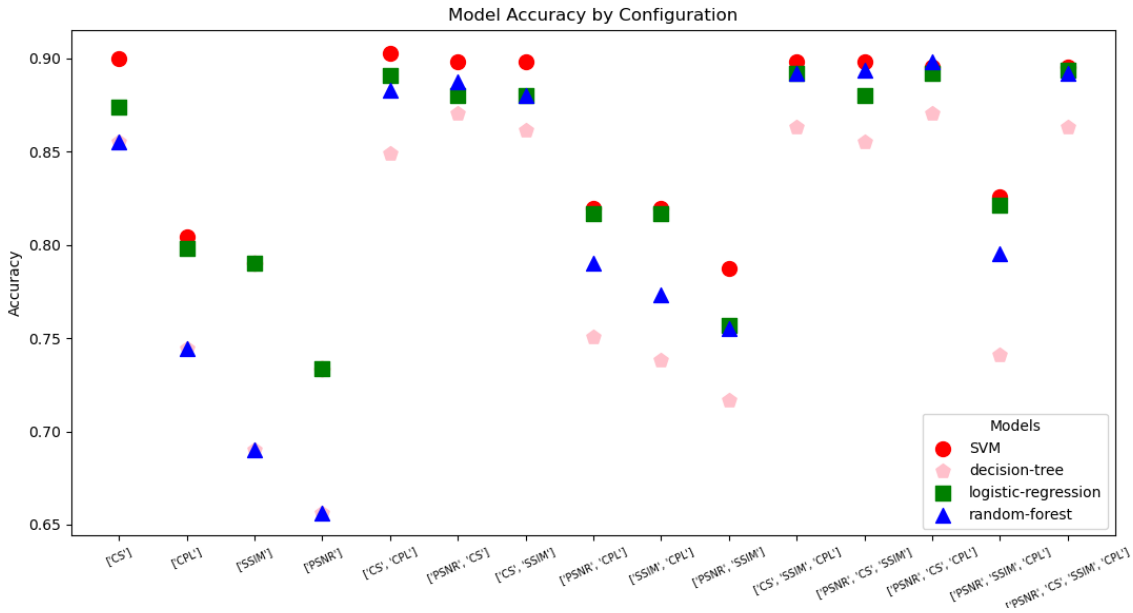


Figure 5.1: Accuracy for each classifier with different input features on SmartInside dataset

the highest accuracy trained on the industry datasets. Specifically, the table presents, for each of the four classifier types – random forest, decision tree, SVM, and logistic regression – the instance with the highest accuracy, the set of metrics used as the input features for that instance, and the accuracy, precision, and recall values for that instance. As shown in the table, SVM with PSNR, CS and CPL input features is the most accurate classifier for the industry dataset. We note that for the industry dataset, the most accurate instance does not require the use of all four input features identified in RQ1, indicating that considering subsets of the optimal metrics may lead to increased classifier accuracy. Overall, the best classifier instances for the industry dataset achieve an accuracy ranging from 86.68% to 90.26%, while those for the public dataset achieve an accuracy ranging from 84.11% to 85.75%.

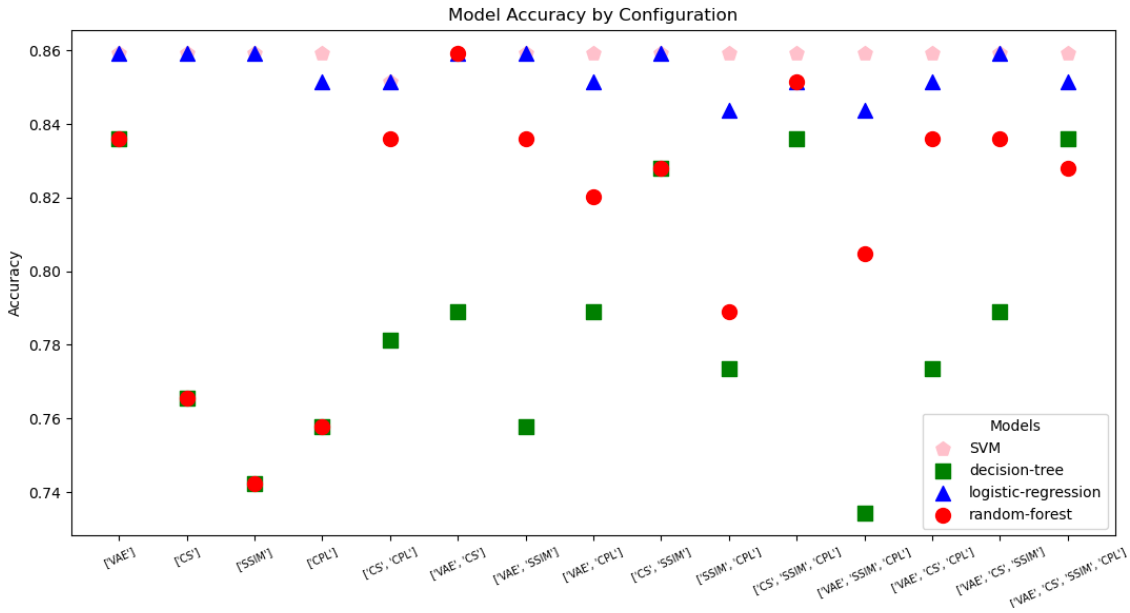


Figure 5.2: Accuracy for each classifier with different input features on Cifar10-c dataset

RQ2 Answer: For the industry dataset, SVM and, for the public dataset, random forest yield the highest accuracy. For both the industrial and public datasets, random forest, logistic regression, and decision tree classifiers exhibit their best performance when trained with the four image-comparison metrics identified in RQ1, while SVM yields its best performance with a subset, but not all, of these metrics.

5.6 RQ3 Experiments and Results.

To answer RQ3, we execute Algorithm 1 using the best classifier instances from RQ2. The purpose of RQ3 is to identify optimal accuracy versus effort trade-offs. To explore different

Table 5.6: Classifier instances with the highest accuracy for each of the four classifier types – random forest, decision tree, SVM, and logistic regression.

Classifier Type	Input Features	Accuracy %	Precision %	Recall %
SVM	PSNR, CS, CPL	90.26	90.59	93.80
Random Forest	PSNR, CS, CPL, SSIM	89.93	90.89	92.81
Logistic Regression	PSNR, CS, CPL, SSIM	88.92	88.23	94.48
Decision Tree	PSNR, CS, CPL, SSIM	86.68	89.06	89.18

trade-offs Algorithm 1 can generate, we execute it with various input values as follows: (1) the D_{nv} input is the D_{test} set specified in Table 5.2 for the industry dataset; (2) the classifier input is replaced by each of the best classifier instances in Table 5.6 for each dataset; (3) the confidence threshold α is, respectively, set to 0.8, 0.85, 0.9, 0.95 and 0.99; (4) the number of pairs to be manually validated at each iteration, β, is, respectively, set to 1%, 3%, 5%, 8%, 10%, and 15%, and (5) the percentage of validated set $D_{cor} \cup D_{cnf}$ that is used for the initial training, respectively, set to 10%, 15%, 20%, 25%, 30%, and 40%.

Different portions of the datasets are specified in Table 5.2. In total, the number of times Algorithm 1 is executed for RQ3 and for each dataset is: $4(\textit{classifiers}) \times 5(\alpha\ \textit{values}) \times 6(\textit{initialdata}\ \textit{values}) \times 6(\beta\ \textit{values}) = 720$.

For each execution, we measure the human effort as the size of the initially validated pairs used as an input of Algorithm 1, i.e., subset of $|DS_{cor} \cup DS_{cnf}|$, and the number of pairs that Algorithm 1 requires to be manually validated. We measure the accuracy of the validated set produced by each execution of Algorithm 1 as the percentage of correctly validated pairs relative to the size of the set.

Results. Figure 5.3 shows the accuracy versus the human effort for the 720 different

executions of Algorithm 1 for our industry and public datasets. As the figure shows, as human effort increases, i.e., as more pairs are manually labeled, the accuracy of the validation process also improves.

Figure 5.4 presents a similar analysis where β varies while other variables remain fixed. This figure demonstrates that while changes in β have a minor impact on improving accuracy, they do lead to an increase in human effort. Figure 5.5 illustrates the relationship between accuracy and human effort as a function of α, with all other variables held constant. The trend indicates that increasing α enhances accuracy, but it also requires greater human effort. It is important to note that data from the Decision Tree model has been excluded from the plot since it was not within the same range as the other classifiers.

RQ3 Answer: For the industrial dataset, HIL-TV can validate test inputs with up to 96% accuracy with 20% of test inputs being validated by a human. For accuracy of 98%, it needs at least 40% of test inputs being validated by a human. For the public dataset, HIL-TV can validate test inputs with up to 96% accuracy with 50% of test inputs being validated by a human. For accuracy of 98%, it needs at least 70% of test inputs being validated by a human.

5.7 RQ4 Experiments and Results.

This research question compares our methodology with the two baselines discussed in Section **??**. For our methodology, we consider two separate methods: (i) The output of step two in the HiL-TV discussed in Section 4.2, i.e., only using the best classifier with

no active learning. In our results, we refer to this method as *Classifier*. (ii) The output of step three in the HiL-TV discussed in Section 4.3, i.e., the active learning output. In our results, we refer to this method as *Active-Learning*. For the *Classifier* and the *Active-Learning* methods, we utilize the best classifier identified for each dataset in RQ2, as shown in Table 5.6. We refer to the baseline from Hu et. al. [21] and the baseline from Riccio and Tonella [41], respectively, as *B-VIF* and *B-VAE*.

We evaluate these methods by assessing the accuracy of the test inputs they validate and the human effort needed in their training and validation process. To compare the methods, we fix the human effort at 25%, 50%, and 75%, and then evaluate the accuracy of each method at each level. To evaluate the *Classifier* method with human effort $x\%$, we randomly pick $x\%$ test inputs of the dataset D, denoted by D_{train}, and train the classifier of this method on these test inputs. We then evaluate the *Classifier* method's accuracy on the remaining part of D. Note that for the *Classifier* method, the human effort will be the size of D_{train} since it needs to be manually validated by the human annotator.

To evaluate the *Active-Learning* method with human effort $x\%$, we randomly pick $\frac{x}{2}\%$ test inputs of the dataset D, denoted by D_{train}. We train the best classifier from RQ2 on D_{train} and execute Algorithm 1. To evaluate the *Active-Learning* method at the human effort level $x\%$, we track the overall human effort, i.e., the total amount of manual test input validation while initially training the classifier and inside the loop of Algorithm 1. Once the overall human effort is $x\%$, we stop further training the classifier inside the loop and validate the remaining test inputs with the classifier and evaluate the overall accuracy.

To evaluate *B-VIF* and *B-VAE* methods with human effort $x\%$, randomly pick $x\%$ test

inputs of the dataset D, denoted by D_{train}. We then compute the corresponding metric for each of the baselines over these test inputs. We then split the range of each threshold to 1000 equidistance points. For each point, we compute the validation accuracy of the baseline using that point as the threshold. For each baseline, we select the point yielding the highest accuracy on these test inputs as the optimal threshold. We then evaluate the accuracy of each baseline with its optimal threshold over the remaining part of the dataset D. Similar to the *Classifier* method, the human effort for both the baseline methods is the size of D_{train} since it needs to be manually validated by the human annotator.

We conduct experiments for these four methods using three levels of human effort: 25%, 50%, and 75%. This allows for a more comprehensive comparison of their performance. We run experiments 20 times for each method and randomly shuffle D each time to ensure consistent results. In total, we run $20 \times 4 \times 3 = 240$ experiments for the four methods with three levels of human effort.

Results. Figures 5.6, 5.7, 5.8, and 5.9 show box plots of the accuracy and F1-score with respect to the human efforts over the 20 runs for each method and each level of human effort for the datasets. As we can see in these figures, even using the *Classifier* method without any active learning, outperforms the baselines. Furthermore, using the *Active-Learning* method demonstrates a significant improvement over the other methods. According to 5.6 and 5.7, *B-VAE* is not performing well on the industrial dataset and according to 5.8, *B-VIF* is not performing well on the public dataset. However, both our methods *Classifier* and *Active-Learning* are outperforming the baselines in both the industrial and public datasets. Finally, as shown in Figures 5.6, 5.7, 5.8, and 5.9, the more the human effort, the more the overall accuracy and F1-score for our methods *Classifier* and *Active-Learning*.

This is not the case for the baselines, which can be another confirmation of the fact that a single image-comparison metric is not enough for a reliable test input validation.

RQ4 Answer: Our proposed methods *Classifier* and *Active-Learning* outperform the baselines for both the industrial and the public datasets with *Active-Learning* significantly outperforming the other methods. The *Active-Learning* method is the only method among our four investigated methods that can reach to 99% test input validation accuracy.

5.8 Custom-Developed Labeling Tool

In this chapter, we present the tool developed in this research.

We created a tool called the Active Image Labeling App (AILA) to help with a specific labeling task. AILA is designed for situations where you need to compare an original image with its modified version and accurately label the changes. The "Active" part of the name comes from the active learning system that's built into the app. This system makes the labeling process more efficient by using a machine-learning model that learns from the annotator's work.

As the annotator labels some of the data manually, the model gets better at understanding what to look for. This means AILA can start automatically labeling other images more accurately, which reduces the amount of work the human annotator has to do. By doing this, AILA makes the whole process faster and less repetitive, while also ensuring

that the data is labeled consistently and accurately. Figure 5.10 shows the main page of the application where the labeling takes place. Figure 5.11 shows the statistics of the labeling procedure.

This tool is developed using Python, with a graphical user interface (GUI) created using the Streamlit module. Streamlit is an open-source Python library that simplifies the process of building and sharing custom web applications, particularly for machine learning and data science. By leveraging Streamlit, we can develop and deploy basic applications with minimal effort.

The backend of the application is powered by an active learning algorithm that we developed from scratch. This algorithm is designed to iteratively improve the model's performance by selectively choosing data points for labeling. We utilize Numpy for performing mathematical operations and Pandas for managing datasets, including handling CSV files.

In addition, we integrate Scikit-learn (Sklearn), a widely used Python module, to provide the classifiers for our active learning algorithm. This combination of tools allows our application to efficiently manage and analyze data while continuously improving its predictive accuracy through active learning.

5.8.1 Setup the Tool

To use the application, follow these steps:

1. **Clone the repository.**

```
git clone https://github.com/delaramGh/active-learning-app/tree/main
```

2. **Install the required dependencies:**

   Run the following command to install all necessary libraries:

   ```
   pip install -r requirements.txt
   ```

3. **Configure the application settings:**

   Open the `AL-Config.txt` file and set the following parameters:

   - `split1`: The portion of the dataset used for initial training. (default = 0.2)
   - `split2`: The portion of the dataset used for tuning the model. (default = 0.05)
   - `confidence_threshold`: If the model's prediction confidence exceeds this threshold, the app will automatically label the image. (default = 0.9)
   - `org_img_path`: The directory path where the original images are stored.
   - `gen_img_path`: The directory path where the generated images are stored.

4. **Run the application:**

   Use the following command to start the application:

   ```
   streamlit run app.py
   ```

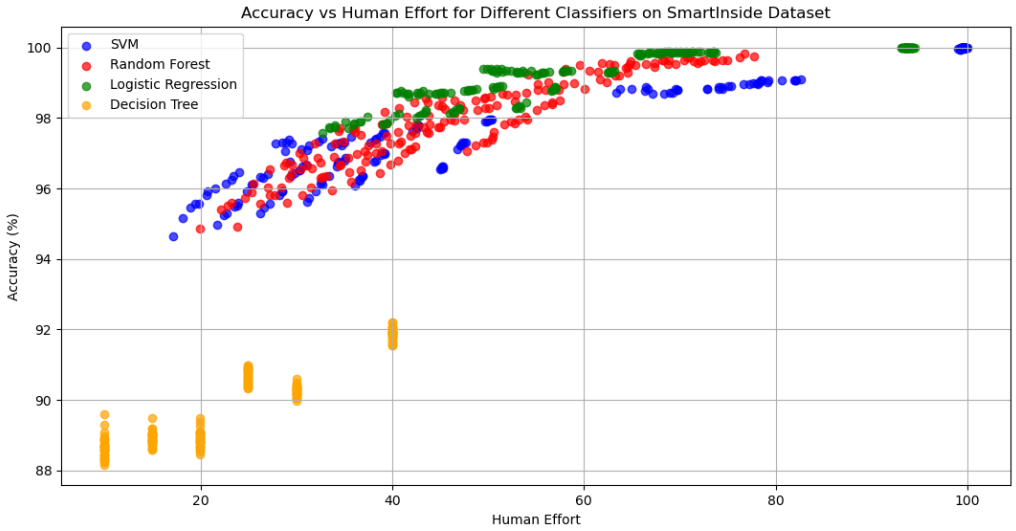
Note: The first time you run the application, it may take some extra time. This is because the app needs to generate a `.CSV` file for the dataset and perform preprocessing on the image pairs.

5.8.2 Tool's Output

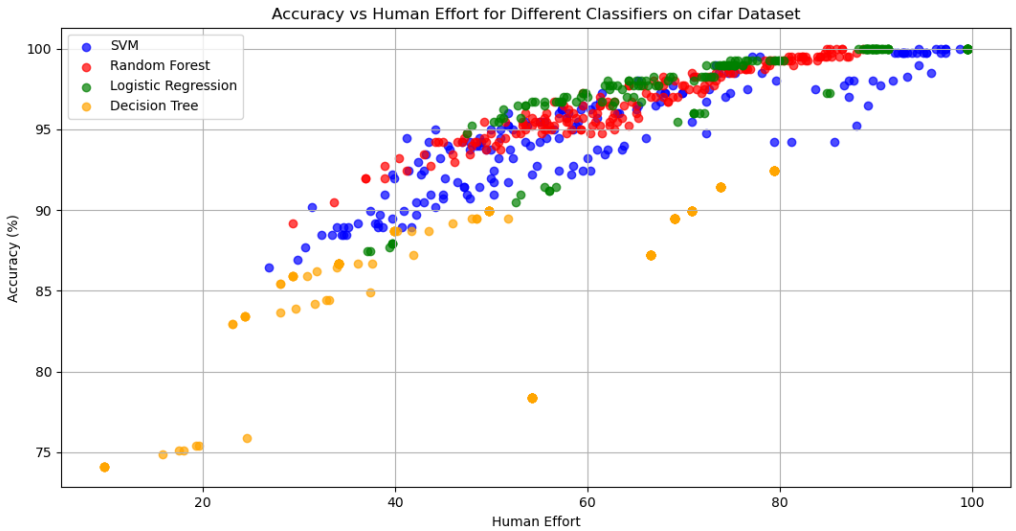
The results are stored in the `AILA_Dataset.csv` file, which contains eight columns:

- **original**: This column lists the names of the original images.
- **generated**: This column lists the names of the generated images.
- **Label**: This column records the labels for each image pair. The label "1" indicates that the pair is correct, "0" indicates that the pair is incorrect or lost, and "-1" signifies that the pair is unlabeled.
- **human-machine**: This column indicates whether the image was labeled by a human annotator or automatically by the machine learning model. "h" denotes labeling by a human, while "m" indicates automatic labeling.
- **Feature Columns**: The remaining columns contain extracted features from the image pairs. These features are used for training the machine learning model and are crucial in determining the labels for new image pairs.

This structure allows for a clear and organized dataset, facilitating efficient training and evaluation of the machine learning model while maintaining traceability of human and machine contributions to the labeling process.



(a) The results for the industry dataset



(b) The results for the public dataset

Figure 5.3: Accuracy vs human effort obtained by different executions of the active learning step of HiL-TV based on different values for parameters α and β of Algorithm 1

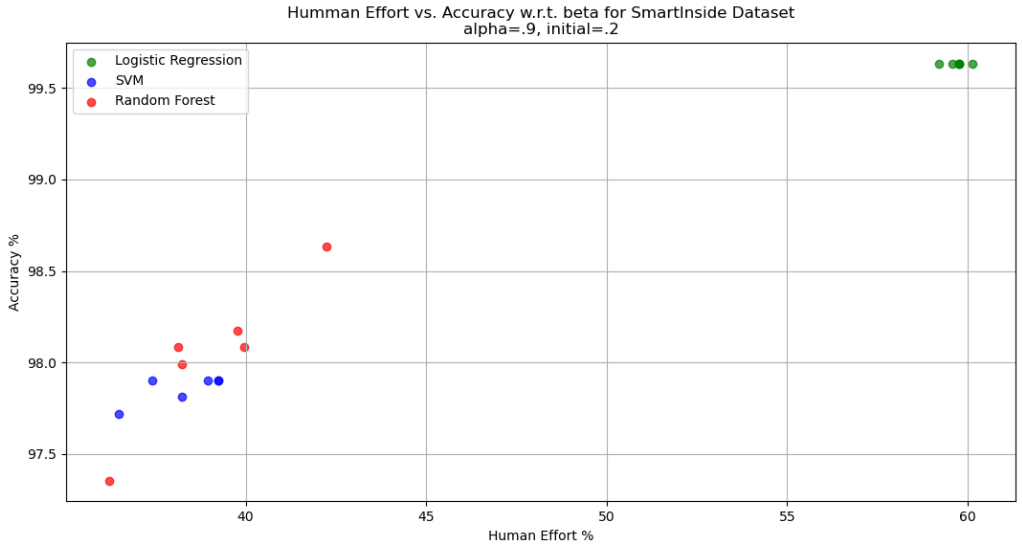


Figure 5.4: Accuracy vs Human Effort w.r.t. β on SmartInside Dataset

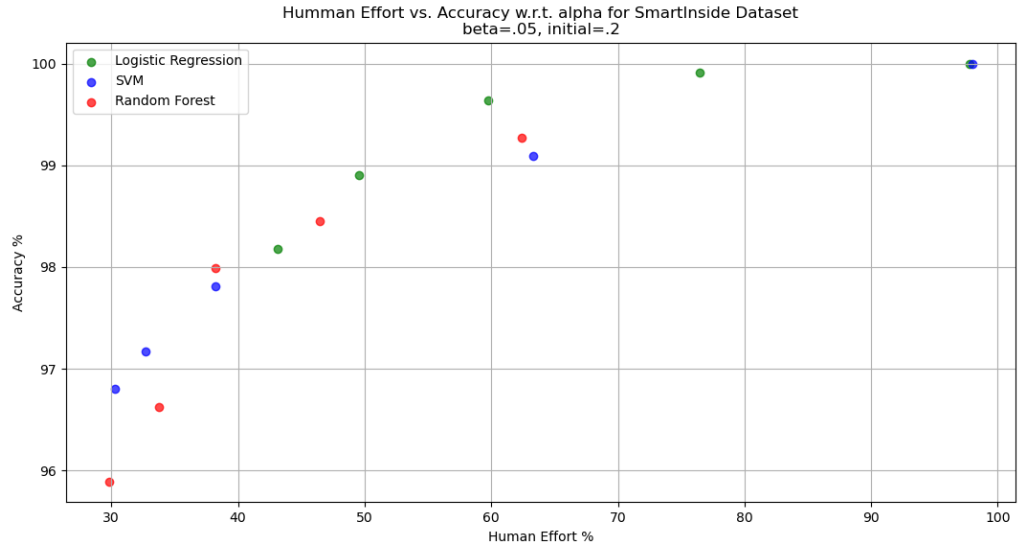


Figure 5.5: Accuracy vs Human Effort w.r.t. α on SmartInside Dataset

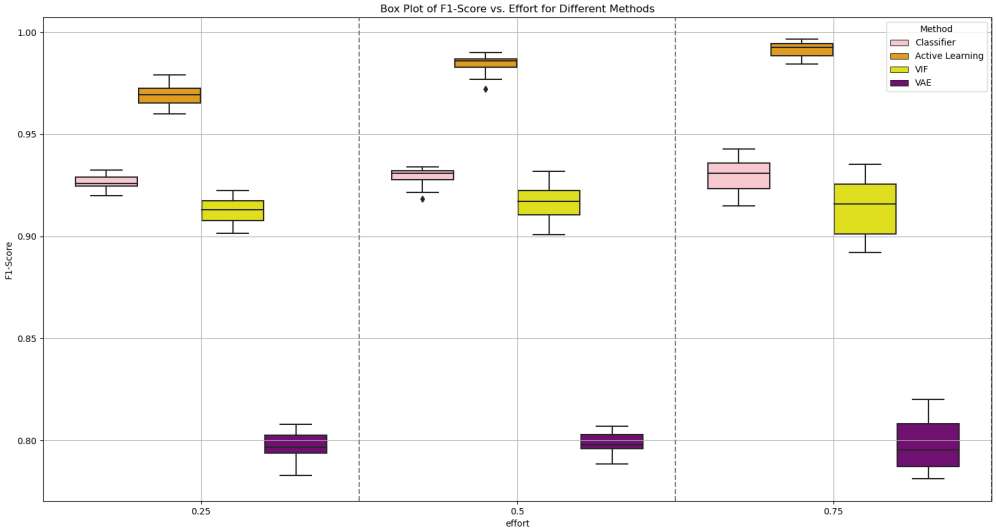


Figure 5.6: Box plot of F1-score with respect to the human effort or SmartInside dataset

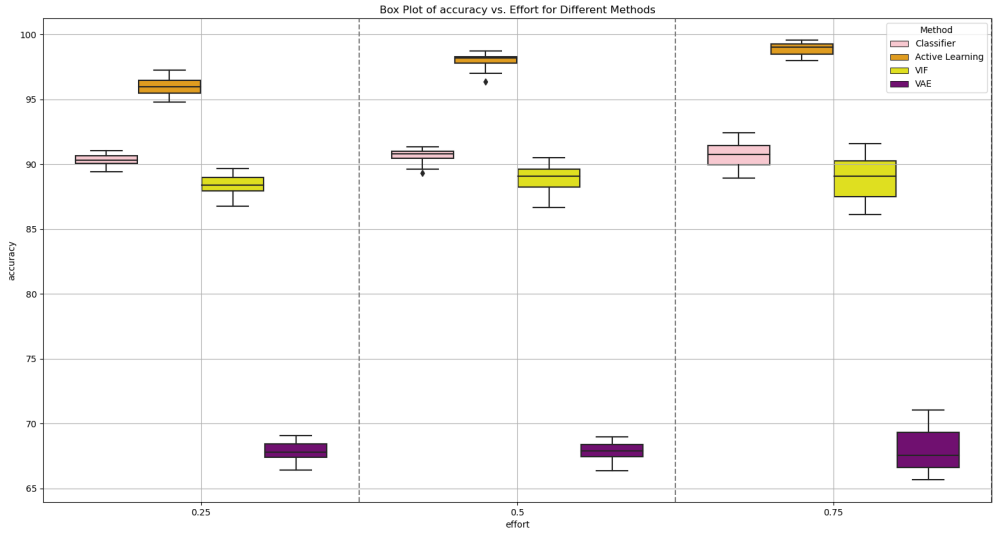


Figure 5.7: Box plot of accuracy with respect to the human effort or SmartInside dataset

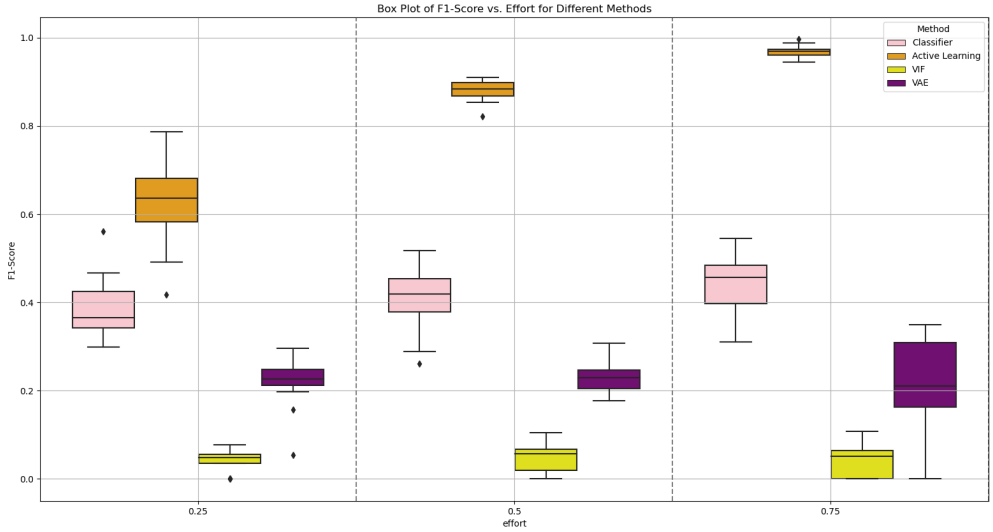


Figure 5.8: Box plot of F1-score with respect to the human effort or Cifar10-c dataset

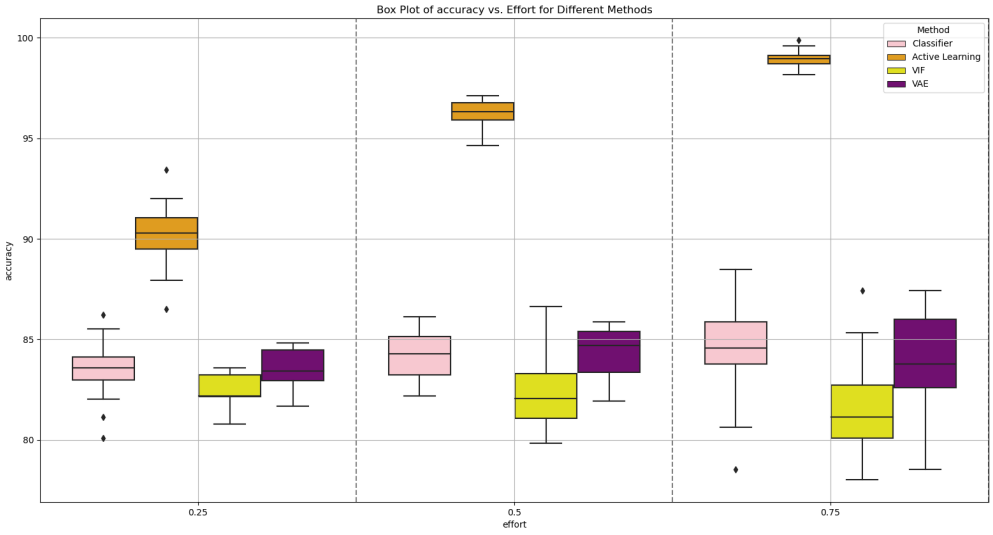


Figure 5.9: Box plot of accuracy with respect to the human effort or Cifar10-c dataset

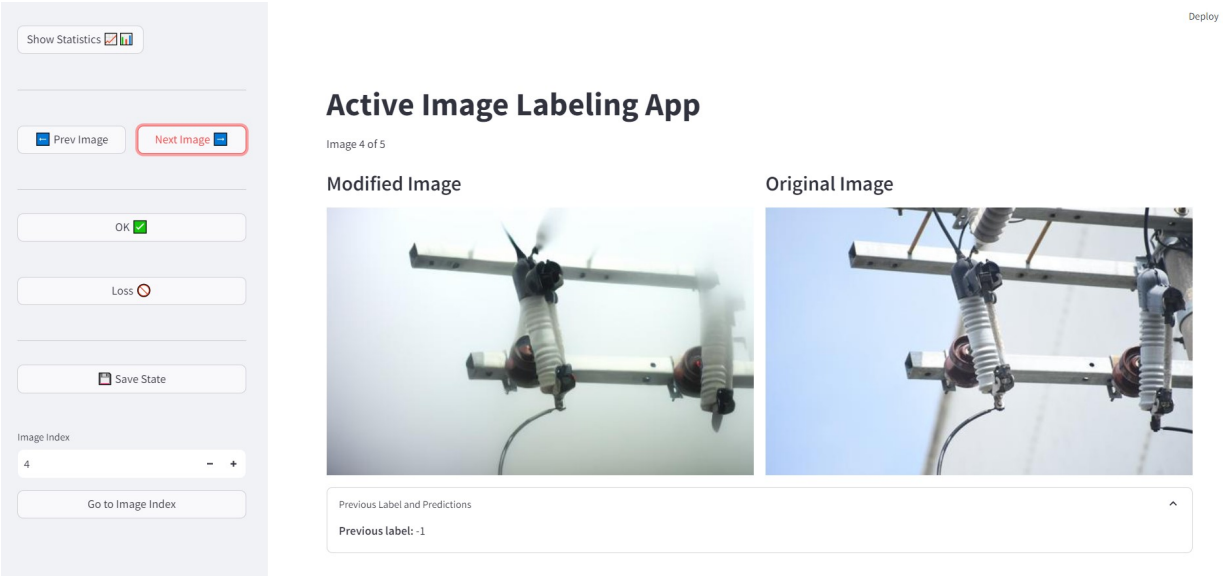


Figure 5.10: AILA labeling tool

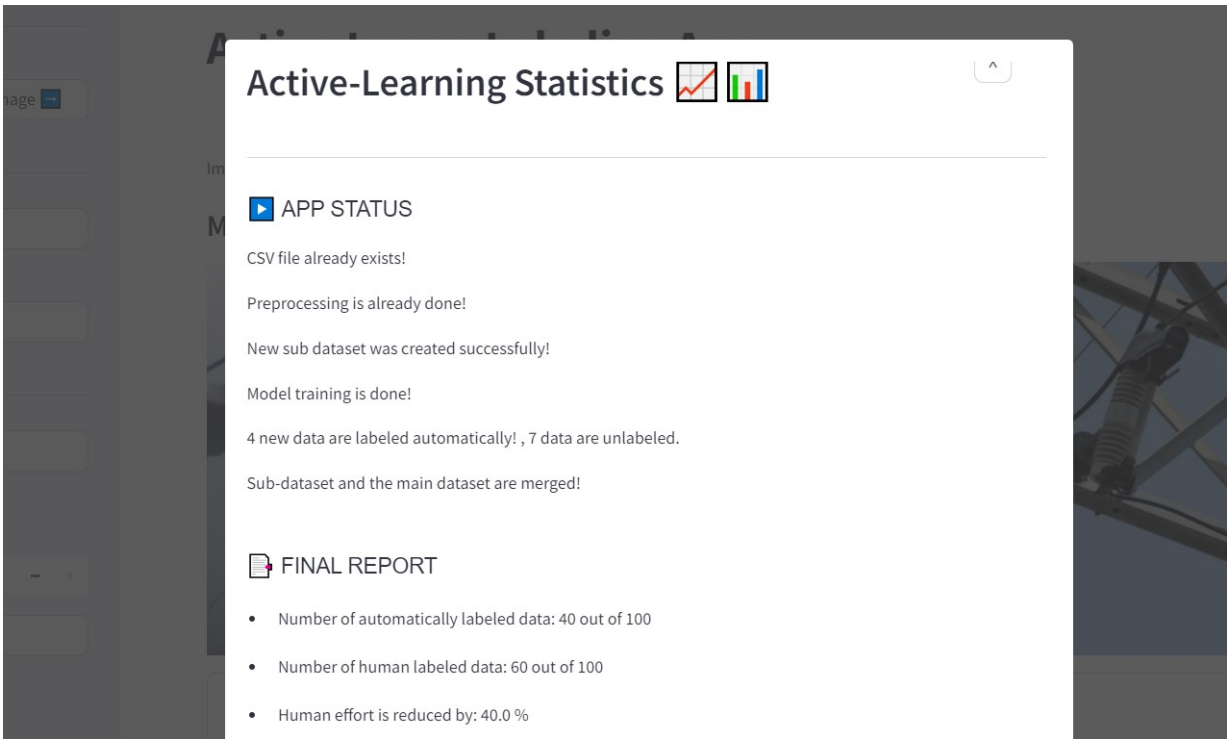


Figure 5.11: AILA labeling tool statistics window

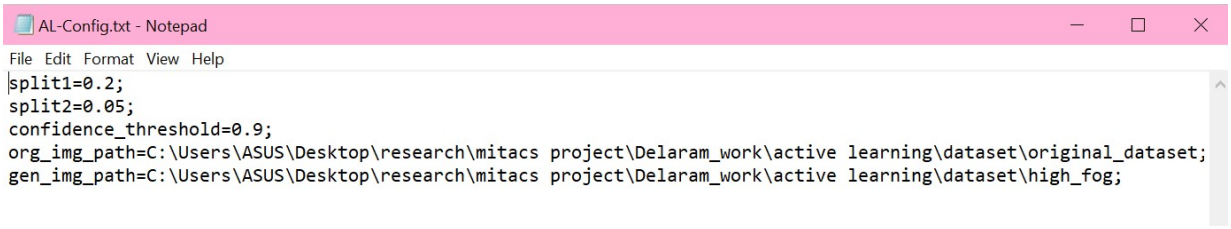


Figure 5.12: Requirements.txt file for setting up the tool.

	A	B	C	D	E	F	G	H	I
1		original	generated	Label	human-machine	PSNR	SSIM	CPL	CS
2	0	4520_2079W642	4520_2079W64	1	m	17.03941	0.840138	35.69857	0.831451
3	1	4520_2079X191_	4520_2079X191	1	h	15.30947	0.839766	21.6362	0.90715
4	2	4520_2079X731_	4520_2079X731	1	h	12.59412	0.846829	17.56169	0.89004
5	3	4520_2079X891_	4520_2079X891	0	m	9.714615	0.624842	90.76119	0.192004
6	4	4520_2080E752_	4520_2080E752	0	h	9.313954	0.399759	110.6514	0.043593
7	5	4520_2080H201_	4520_2080H20	0	m	10.0769	0.578552	119.2476	0.500053
8	6	4520_2080X442_	4520_2080X442	0	m	12.12565	0.630516	69.10123	0.121207
9	7	4520_2080X491_	4520_2080X491	1	m	16.20053	0.837449	30.29215	0.848038
10	8	4520_2080X623_	4520_2080X623	0	m	10.37197	0.506869	122.8158	0.096604
11	9	4520_2081H482_	4520_2081H48	1	h	12.57316	0.500455	12.79028	0.465082
12	10	4520_2168Q362_	4520_2168Q36	1	h	17.39947	0.906686	28.85989	0.685244
13	11	4520_2168Q751_	4520_2168Q75	1	m	17.16257	0.888975	19.26782	0.881795
14	12	4520_2169S642_	4520_2169S642	0	m	10.37391	0.475418	58.48456	0.057551
15	13	4520_2171S981_	4520_2171S981	1	h	16.06926	0.808731	45.93046	0.655469
16	14	4520_2171Y184_	4520_2171Y184	0	m	14.98603	0.856983	33.8115	0.4967
17	15	4520_2171Z541_	4520_2171Z541	1	m	13.08485	0.675346	74.81309	0.349082
18	16	4520_2175E771_	4520_2175E771	0	h	13.7307	0.758198	53.7131	0.613173
19	17	4520_2175F051_	4520_2175F051	0	h	10.5459	0.778399	38.13913	0.42165
20	18	4520_2175F581_	4520_2175F581	1	h	13.325	0.753593	43.99302	0.797088
21	19	4520_2175F902_	4520_2175F902	0	m	9.415364	0.543933	87.99365	0.171677
22	20	4520_2176E201_	4520_2176E201	0	h	12.14825	0.57831	68.70911	0.234798

Figure 5.13: The output of the tool in csv format.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this work, we proposed HiL-TV (Human-in-the-Loop Test Validation), an approach for semi-automatic validation of test inputs for vision-based deep learning systems. HiL-TV addresses the critical challenge of ensuring the validity of synthetic test inputs by incorporating an active learning loop that balances the trade-off between validation accuracy and manual effort. Unlike existing methods, which rely on pre-selected image-comparison metrics, HiL-TV dynamically selects the most suitable metrics based on the specific characteristics of the dataset and the transformations applied. This adaptability allows our approach to maintain high validation accuracy across various datasets, including those from specialized industrial domains.

Our extensive evaluation, conducted on both industrial and public-domain datasets,

demonstrated that HiL-TV significantly outperforms state-of-the-art baselines in terms of validation accuracy while reducing the manual effort required for labeling. The results underscore the effectiveness of our approach in validating test inputs for complex and diverse DL systems, particularly in scenarios where traditional methods may fall short.

Furthermore, our approach provides valuable insights into the potential of active learning for automated test validation, offering a flexible framework that can be tailored to different testing contexts. The lessons learned from our experiments suggest that integrating human expertise into the validation process can lead to more reliable and robust testing outcomes, particularly in domains where the cost of errors is high.

6.2 Threats to Validity

To prevent data leakage, we used separate, non-overlapping training and test sets in each research question. In RQ4, to ensure a fair comparison between HiL-TV and the two baselines, we used the same training and test sets for all methods. To counteract random variation in RQ4, we randomly selected 20 subsets from our datasets for each effort level and repeated the experiments for each subset. Regarding the mitigation of bias in our datasets, we note the following: The ground-truth labels for the public dataset were taken directly from Hu et al. [21] (via majority voting), with no modifications made by us. For our industry dataset, labeling was performed by two independent human labelers who are not co-authors of this paper. The ground-truth labels were subsequently reviewed by domain experts.

The public dataset used in our evaluation is derived from CIFAR-10 [27], a well-known benchmark in the research community that has also been used previously for evaluating test input validators [?, ?, 21]. The consistency observed between the results of our experiments on the public dataset and our domain-specific industry dataset provides a degree of confidence in the generalizability of our findings. To our knowledge, we are the first to evaluate and compare test input validation methods using an industry dataset. Nevertheless, as with any case study-based research, further evaluation across a broader range of study subjects is necessary to draw more definitive conclusions about generalizability.

6.3 Future Work

While HiL-TV has shown promising results in the validation of test inputs for vision-based deep learning systems, several avenues for future research remain.

Firstly, one potential extension of our work involves broadening the scope of HiL-TV to handle more complex and diverse transformations beyond those considered in our current evaluation. This includes exploring transformations that involve multiple concurrent changes to input data, such as combined geometric and color transformations, as well as those that simulate more sophisticated environmental conditions. We can also add a multi-class classification option to the algorithm.

Secondly, although our approach has been applied successfully to image classification tasks, we believe that HiL-TV could be adapted for other types of deep learning systems, such as those used in natural language processing, speech recognition, or reinforcement

learning. Future research could investigate how our dynamic metric selection and active learning framework can be tailored to these domains, potentially uncovering new challenges and solutions specific to different types of data and tasks.

Another area for future work is the integration of more advanced machine learning techniques within the active learning loop of HiL-TV. For example, incorporating reinforcement learning or deep active learning could further reduce the reliance on human intervention, thereby improving the scalability and efficiency of the validation process. Additionally, experimenting with unsupervised or semi-supervised learning methods could help to reduce the need for labeled data, making the approach more feasible for large-scale industrial applications where labeled data may be scarce.

These future directions highlight the potential of HiL-TV to evolve and adapt to the ever-growing challenges in DL testing, offering a robust and flexible solution for ensuring the validity and reliability of deep learning systems across a wide range of applications.

References

[1] Stack Abuse. Implementing svm and kernel svm with python's scikit-learn, 2023. Accessed: 2023-08-08.

[2] AI Mind. Understanding and implementing variational autoencoders (vaes). https://pub.aimind.so/understanding-and-implementing-variational-autoencoders-vaes-ebf74b6241e7, 2024. Accessed: 2024-08-23.

[3] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017. Accessed: 2024-08-08.

[4] Machine Learning Bites. Machine learning: Decision tree classifier, 2024. Accessed: 2024-08-08.

[5] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[6] Tim Brooks, Aleksander Holynski, and Alexei A. Efros. Instructpix2pix: Learning to follow image editing instructions. In *IEEE/CVF Conference on Computer Vision and*

Pattern Recognition, CVPR 2023, Vancouver, BC, Canada, June 17-24, 2023, pages 18392–18402. IEEE, 2023.

[7] Coursera. Decision trees in machine learning: Two types (+ examples), 2024. Accessed: 2024-08-08.

[8] Coursera. Logistic regression: An overview, 2024. Accessed: 2024-08-08.

[9] DataHeroes. Decision trees: An overview and practical guide, 2024. Accessed: 2024-08-08.

[10] Swaroopa Dola, Matthew B. Dwyer, and Mary Lou Soffa. Distribution-aware testing of neural networks using generative models. In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*, pages 226–237. IEEE, 2021.

[11] GeeksforGeeks. Cosine similarity, 2024. Accessed: 2024-08-08.

[12] GeeksforGeeks. Gray-level co-occurrence matrix (glcm), 2024. Accessed: 2024-08-08.

[13] GeeksforGeeks. Mean squared error (mse), 2024. Accessed: 2024-08-08.

[14] GeeksforGeeks. Peak signal to noise ratio (psnr), 2024. Accessed: 2024-08-08.

[15] GeeksforGeeks. Random forest algorithm, 2024. Accessed: 2024-08-08.

[16] GeeksforGeeks. Structural similarity index (ssim), 2024. Accessed: 2024-08-08.

[17] GeeksforGeeks. Support vector machine algorithm, 2024. Accessed: 2024-08-08.

[18] Farnaz Gholami. Ultimate guide for active learning: Main approaches, 2023. Accessed: 2024-08-22.

[19] Jianmin Guo, Yu Jiang, Yue Zhao, Quan Chen, and Jiaguang Sun. Dlfuzz: differential fuzzing testing of deep learning systems. In Gary T. Leavens, Alessandro Garcia, and Corina S. Pasareanu, editors, *Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2018, Lake Buena Vista, FL, USA, November 04-09, 2018*, pages 739–743. ACM, 2018.

[20] Trevor Hastie, Jerome H. Friedman, and Robert Tibshirani. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Series in Statistics. Springer, 2001.

[21] Boyue Caroline Hu, Lina Marsso, Krzysztof Czarnecki, Rick Salay, Huakun Shen, and Marsha Chechik. If a human can see it, so should your system: Reliability requirements for machine vision components. In *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022*, pages 1145–1156. ACM, 2022.

[22] IBM. What is a decision tree?, 2024. Accessed: 2024-08-08.

[23] IBM. What is a random forest?, 2024. Accessed: 2024-08-08.

[24] IBM. What is logistic regression?, 2024. Accessed: 2024-08-08.

[25] Keboola. The ultimate guide to decision trees for machine learning, 2024. Accessed: 2024-08-08.

[26] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[27] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[28] Stefano Carlo Lambertenghi and Andrea Stocco. Assessing quality metrics for neural reality gap input mitigation in autonomous driving testing. In *17th IEEE International Conference on Software Testing, Verification and Validation (ICST 2024)*, 2024. To appear. Available at https://arxiv.org/abs/2404.18577.

[29] MathWorks. Cosine similarity, 2024. Accessed: 2024-08-08.

[30] MathWorks. Mse (mean squared error), 2024. Accessed: 2024-08-08.

[31] MathWorks. Psnr (peak signal-to-noise ratio), 2024. Accessed: 2024-08-08.

[32] MathWorks. Ssim (structural similarity index), 2024. Accessed: 2024-08-08.

[33] Medium. Introduction to random forest classification by example, 2024. Accessed: 2024-08-08.

[34] Mohsen Mollahyari. Autoencoder tutorial. https://mallahyari.github.io/ml_tutorial/autoencoder/, 2024. Accessed: 2024-08-23.

[35] MrMaster907. Introduction to random forest classification by example, 2024. Accessed: 2024-08-08.

[36] Mubeta06. Ssim in python, 2024. Accessed: 2024-08-08.

[37] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. Deepxplore: automated whitebox testing of deep learning systems. *Commun. ACM*, 62(11):137–145, 2019.

[38] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2211.14819*, 2022.

[39] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. *arXiv preprint arXiv:2012.04225*, 2021.

[40] ResearchGate. Illustration of the psnr measure, 2024. Accessed: 2024-08-08.

[41] Vincenzo Riccio and Paolo Tonella. When and why test generators for deep learning produce invalid inputs: an empirical study. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 1161–1173. IEEE, 2023.

[42] H.R. Sheikh and A.C. Bovik. Image information and visual quality. *IEEE Transactions on Image Processing*, 15(2):430–444, 2006.

[43] SmartInside AI. Smartinside ai - AI Solutions for Smart Industries. https://smartinside.ai/en/, 2024. Accessed: 2024-08-23.

[44] Andrea Stocco, Michael Weiss, Marco Calzana, and Paolo Tonella. Misbehaviour prediction for autonomous driving systems. In Gregg Rothermel and Doo-Hwan Bae,

editors, *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020*, pages 359–371. ACM, 2020.

[45] Youcheng Sun, Xiaowei Huang, Daniel Kroening, James Sharp, Matthew Hill, and Rob Ashmore. Deepconcolic: testing and debugging deep neural networks. In Joanne M. Atlee, Tevfik Bultan, and Jon Whittle, editors, *Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, pages 111–114. IEEE / ACM, 2019.

[46] Machine Learning How To. Decision trees in machine learning: A comprehensive overview, 2024. Accessed: 2024-08-08.

[47] Analytics Vidhya. How to classify non-linear data to linear data, 2024. Accessed: 2024-08-08.

[48] Cédric Villani. *Optimal Transport: Old and New*. 2008. Accessed: 2024-08-08.

[49] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: From error visibility to structural similarity. 2004. Accessed: 2024-08-08.

[50] Wikipedia. Support vector machine, 2023. Accessed: 2023-08-08.

[51] Wikipedia. Cosine similarity, 2024. Accessed: 2024-08-08.

[52] Wikipedia. Logistic regression, 2024. Accessed: 2024-08-08.

[53] Wikipedia. Mean squared error, 2024. Accessed: 2024-08-08.

[54] Wikipedia. Peak signal-to-noise ratio, 2024. Accessed: 2024-08-08.

[55] Wikipedia. Structural similarity index, 2024. Accessed: 2024-08-08.

[56] Wikipedia. Support vector machine margin image, 2024. Accessed: 2024-08-08.

[57] Wikipedia. Wasserstein metric, 2024. Accessed: 2024-08-08.

[58] Y. Zhang and X. Wu. Texture similarity for image quality assessment. 2011. Accessed: 2024-08-08.