Adversarial Framework with Temperature as a Regularizer for Semantic Segmentation

Chanho Kim

Thesis submitted to the University of Ottawa
in partial Fulfillment of the requirements for the
Master of Applied Science (M.A.Sc.)

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Abstract

Semantic Segmentation processes RGB scenes and classifies pixels collectively as an object. Recent deep learning methods have shown promising results in the accuracy and the speed of semantic segmentation. However, it is inevitable for the deep learning models to fall in overfitting to data used in training due to its nature of data-centric approaches.

There have been numerous Regularization methods to overcome an overfitting problem, such as data augmentation, additional loss methods such as Euclidean or Least-Square terms, and structure-related methods by adding or modifying layers like Dropout and DropConnect in a network. Among those methods, penalizing a model via an additional loss or a weight constraint does not require memory increase.

With this sight, our work purposes to improve a given segmentation model through temperatures and a lightweight discriminator. Temperatures have the role of generating different versions of probability maps through the division in softmax calculations. On top of probability maps from temperatures, we concatenate a simple discriminator after the segmentation network for the competition between groundtruth feature maps and modified feature maps. We pass the additional loss calculated from those probability maps into the principal network.

Our contribution consists of two parts. Firstly, we use the adversarial loss as the regularization loss in the segmentation networks and validate that it can substitute the L2 regularization loss with better validation results. Also, we apply temperatures in segmentation probability maps for providing different information without using additional convolutional layers.

The experiments indicate that the spiking temperature in a generator with keeping an original probability map in a discriminator provides the model improvement in terms of pixel accuracy and mean Intersection-of-Union (mIoU). Our framework shows that the segmentation model can be improved with a small increase in training time and the number of parameters.

Acknowledgement

I would like to give thanks to many people who have helped my research for the completion of the thesis. First of all, I would like to appreciate my parents, sister, other family members who have provided great support throughout my M.A.Sc. study. Also, I would like to express my sincere gratitude to my fiancée, Younggeun, for her encouragement and granting me comfort.

I want to give words to appreciate to my supervisor professor Won-Sook Lee for the guidance during my study and research. Without her valuable advice, I could not have achieved any progress in my research and this thesis. I also would like to thank Professor James Green and Professor Jiying Zhao for the thesis examination and valuable feedback.

It is my pleasure to be with LIII lab members during the study. Discussions and meetings with colleagues give research ideas and shed a light on my knowledge. Thanks to colleges from LIII lab. Last but not least, I would like to thank my friends for their concern and support.

Table of Contents

List of Figures

List of Tables

1

Introduction

1.1 Motivation

Semantic Segmentation recognizes a given image and classifies a set of pixels as an object category considering semantics and recent deep models have achieved prominent progress in the area. Deep learning generates probability maps for each pixel and extracts a corresponding object label. In supervised learning, we train a machine learning model with given input images and labels. Because a typical dataset has a finite amount of data, the model has a probability to be stuck in overfitting on the trained model, very often, resulting that it works well on only training sets of a

dataset while performs worse on validation sets or test sets. If the data is from another source, it is even more difficult to have high performance.

There have been various methods to overcome overfitting such as data augmentation, adding an additional loss to the main loss, and manipulating model structures by adding functional layers or more convolutional layers in extra paths. Data augmentation gives variations on a given training dataset for preventing a network from converging to the narrow point which only fits the training dataset rather than other data. For example, we can apply scaling, rotation, cropping in the case of image datasets. Also, regularization terms like a Euclidean or Least-Square loss from layers [1] can be added to the optimization area with the main network loss. Beyond these terms, methods applied to classification probabilities such as Label Smoothing [2] contribute a small additional loss to the main network. It is necessary to search optimal parameters when we apply those methods on training models. When considering adding layers, we can add several functional layers such as DropOut [3] and DropConnect [4]. Those layers randomly select and ignore subsets of nodes and weights during training, respectively, resulting in various versions of a given model. Beyond functional layers, there have been methods which add more layers including convolutional ones and create additional routes on a backbone model, like Shake-Shake Regularization [5]. A network can be expected to perform better by adding layers, however, we need to consider the trade-off between the performance and the computation resources.

Our work starts from the desire to improve the existing models instead of simplifying models for reducing the overhead or modifying the structures for better performance and the desire to mitigate overfitting.

1.2 Outline of Our Work

A choice of deep learning models has a trade-off between a performance and a computation power in training. It means that we need more memory and parameters for better performance with longer training time. Because the computation resources are limited, we search the way to improve a segmentation network without increasing large amounts of memory – Discriminator and Temperature.

We get the inspiration from Generative Adversarial Network (GAN) [6]. GAN is one of the most popular approaches to semantic segmentation which has 2 parties in dispute: a generator and a discriminator. In GAN models, a discriminator gets a chance to penalize a generator by measuring fake samples against real ones. We imitate the GAN structure by concatenating a discriminator after a given segmentation model which generates the feature maps like a generator in GAN. However, the segmentation model cannot be same as any generator in GAN due to the difference of input data. Input data of a general segmentation model are RGB images to be segmented whereas those of a generator in GAN models are noise latent spaces. In the case of a discriminator, real and fake samples in our approach are corresponding to the groundtruth and the feature maps from the segmentation model, respectively.

Also, Knowledge Distillation [7] gives us an idea to utilize temperatures as a regularizer. Knowledge Distillation requires a source network which typically has the larger number of parameters with better performance and a target network to obtain the knowledge from the source. To transfer the knowledge from a high-performance network to a target network to improve, a constant value, which is called as ‘temperature’, is used for the distillation loss between probability

distributions from both networks. Despite the simpleness of this concept, it is challenging to process two networks for distillation at the same time, where a higher memory consumption is inevitable. Therefore, our framework brings the concept of temperatures and saves the computation resources by concatenating a lightweight discriminator after the segmentation network, with an additional softmax layer as the input probability information from the main network.

With the GAN-inspired structure and the concept of temperatures, we propose the Adversarial Framework for semantic segmentation. The segmentation network and the discriminator network work as the generator and the discriminator, respectively. The differences between the GAN and our framework are the input data and additional parameters in probability distributions. GAN starts from random noise spaces for the generative training whereas segmentation networks in our framework calculate probabilities from RGB images in annotated datasets. Also, we use two temperature parameters in softmax calculations for generating two corresponding softmax probability maps in both the generator and the discriminator. Our work eventually aims to prevent overfitting by improving the validation accuracy without using multiple segmentation models or modifying the segmentation model structure.

Like many regularization methods, our work also aims to solve the overfitting problem where a trained deep network is not generalized to perform intended tasks in other datasets than a training dataset. Although Least-Square (L2) regularization loss [1] is commonly used in training models by passing an additional loss in backpropagation, it is necessary to find the optimal parameter of the L2 regularization equation. Therefore, we firstly search the optimal configurations in given

networks for semantic segmentation in the first step of our experiments. In this search, we find that segmentation models perform better with L2 regularization parameters different from baseline configurations of each segmentation network for given datasets. On top of the best L2 regularization values including zero, we use temperatures in softmax layers as a new form of regularization methods with the discriminator network concatenated to the segmentation network.

Through adding the discriminator with the segmentation network, our framework resembles the structure of GAN, but the segmentation network is not same as a generator in GAN due to the difference of input data. We applied temperatures $T = \{0.5, 1, 2\}$ on softmax calculations on both the segmentation model and the discriminator model. With the fixed $T = 1$ in the either side for preserving the original probability information, we search the optimal configuration where the segmentation performance is the best.

With two concepts, we calculate the segmentation loss and the adversarial loss with temperatures, resulting that the adversarial one can take a role as another kind of regularizers by providing an additional loss to the segmentation network. This loss can be obtained via only adding the discriminator parameters and a softmax layer and improves the segmentation performance of each model compared with the baseline configurations.

1.3 Contributions

Our innovation lies in two folds;

(i) We use temperatures as a regularizer. For our best knowledge, it is the first time to utilize the temperatures to mitigate overfitting as an auxiliary loss, resulting that the loss has a similar role as regularization loss terms.

(ii) We concatenate a lightweight DCGAN discriminator after a given segmentation network with the consideration of computation resources. We apply Least-Square losses from LSGAN to the discriminator due to its convergent characteristic and the loss passed to the main network eventually helps it perform better.

1.4 Thesis Organization

In the thesis, **Chapter 2** covers Deep Learning overview, Semantic Segmentation, GAN, and Knowledge Distillation.

Chapter 3 focuses on our proposed framework – Adversarial Framework with Temperatures. We also introduce backbone models which we select for our framework experiments.

Chapter 4 present the evaluation and results of our methods. We aim to search the optimal configurations for our framework which improves base segmentation networks by a new type of regularization methods.

Finally, **Chapter 5** summarizes our works and discloses limitations that we should overcome and investigate further for future works.

2

Literature Review

In **Chapter 2**, a literature review covers Deep Learning components, GAN, and Knowledge Distillation. **Section 2.1** introduces deep learning and related components which are utilized in learning CNN models. Training common CNN models require activation functions, normalization layers, loss functions, optimizers (Section 2.1.1), and regularization methods (Section 2.1.2) which are thoroughly described in this section. **Section 2.2** briefly introduces Semantic Segmentation with related methods before and after machine learning and datasets commonly used. **Section 2.3** describes the GAN architecture and its derivatives such as DCGAN, LSGAN, and so on. Finally, **Section 2.4** covers Knowledge Distillation with temperatures and its applications.

2.1 Overview of Deep Learning

2.1.1 Deep Learning Components

i. Convolutional Neural Network (CNN)

Multi-Layer Perceptron (MLP) [8] consists of multiple nodes and activation functions. Layer stages between the inputs and the outputs have hidden layers which have nonlinear activation functions. A perceptron can be trained through backpropagation [9] using gradient descent which calculates weight changes of each hidden layer. Those nonlinear activation functions enable a MLP to discriminate data that linear or high-order equations cannot distinguish easily.

Beyond perceptron, CNN emerged with convolutional layers which have less trainable parameters but can extend receptive fields, resulting that the layer can include the adjacent pixel information in processing feature maps in stages. Thus, CNN has been applied to computer vision tasks such as object detection, semantic segmentation, and so on. From this characteristic, CNN reduced computational costs with convolutional layers rather than fully-connected layers used in MLP. LeNet [10] introduces a simple convolutional neural network structure. ConvNet [11] was trained with ImageNet dataset [12] through GPU parallelization.

From ConvNet, CNN models have been evolved in terms of the number of parameters and the structure, resulting in many variations such as VGG [13], ResNet [14], Inception [15], DenseNet [16]. Moreover, there have been trials to build more efficient models such as MobileNet [17] [18] [19] and EfficientNet [20] [21].

ii. Activation functions

Activation functions have a role of converting input values to nonlinear output values mainly after the product calculation of a weight vector and the input data, adding a bias vector in some cases. By passing activation functions, each stage has an ability to convert linear data to nonlinear data. The derivativity of activation functions enable a network to be trained through backpropagation. **Figure 2-1** illustrates several kinds of activation functions.

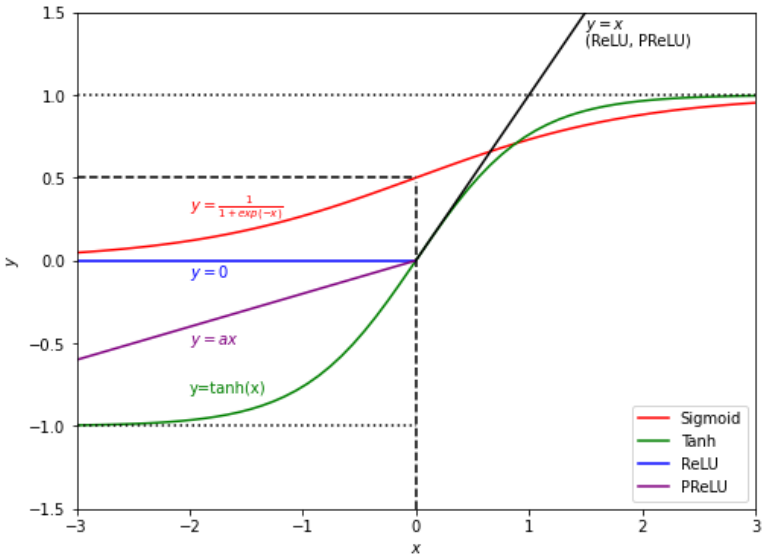

Figure 2-1. Description of several activation functions.

Sigmoid is one of the most common activation functions. As the figure above describes that the activation value goes 0 where $x \to -\infty$ and 1 where $x \to \infty$. The function is differentiable and bounded to (0,1). Because the sigmoid function is one of soft saturation functions, a gradient from an input value $|x| \to \infty$ goes to 0. Tanh is similar to sigmoid in terms of the convergence

in $|x| \rightarrow \infty$. However, the gradient is higher due to the higher boundary values (-1, 1). Both sigmoid function and Tanh function have the vanishing gradient problem in backpropagation [22] which means that a very small gradient from a high x value prevent the neural network from updating the weights.

Rectified Linear Units (ReLU) [23] mitigates the vanishing gradient problem due to its linearity [24]. It allows a faster learning and a less computations due to the absence of exponential functions. However, there is "dying ReLU" problem from no gradient at $x < 0$, resulting that a network can be stuck in dead states. Leaky ReLU [25] was proposed to grant a small and positive gradient in $x < 0$ for resolving the "dying ReLU" problem. Later, Parametic ReLU (PReLU) [26] generalizes the slope in a negative x where Leaky ReLU has $a = 0.01$.

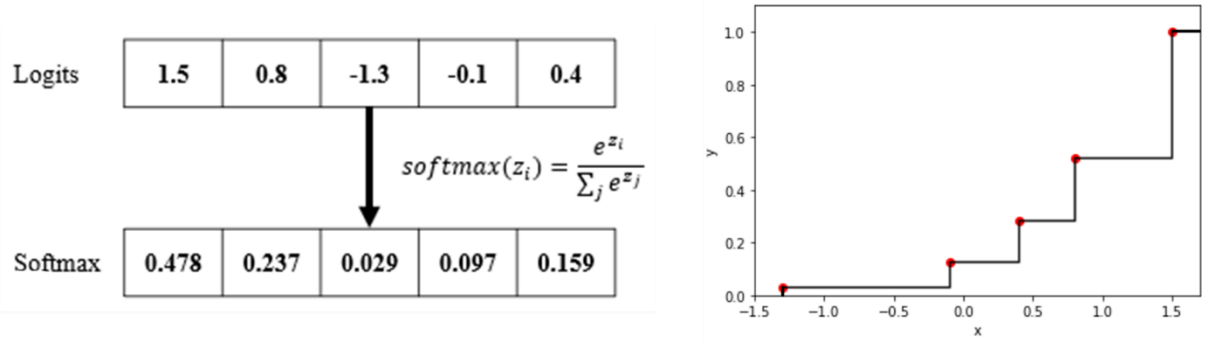

Figure 2-2. Softmax calculation example and cumulative graph. In given logits, higher values have higher softmax probability values with range [0, 1]. In the cumulative graph, the biggest probability (close to the right end, $x = 1.5$ in this case) has the highest step.

Softmax [27] calculates the probability scores among classes from output logits. **Figure 2-2** illustrates an example of softmax function. It normalizes real input vectors in [0, 1] range and

converts them into a probability space where the summation of elements is 1. Therefore, softmax layers are mainly used in classification tasks and segmentation tasks that have a certain number of classes.

Recently, there are many variations such as Exponential Linear Unit (ELU) [28], Scaled Exponential Linear Unit (SELU) [29], Swish [30], and ReLU6 [17]. Various activation functions widen the choices in designing a neural network architecture and optimize the performance.

iii. Normalization

In training deep learning models, normalization methods adjust values for processing data such as input of models and those after convolutional layers. In data augmentation, one of common normalization methods is modifying the scale of RGB images from [0, 255] integers to [0, 1] as floating values. Moreover, given data can be standardized through mean and standard deviation calculations, resulting in the stability of training.

There also have been normalization methods implemented by adding layers with certain operations. Batch Normalization (BN) [31] normalizes the layer input by calculating the mean and the variance. It improves mini-batch network learning by allowing higher learning rates and reducing internal covariate shift. In the training procedure, BN layers fix the means and variances of layer inputs and learn the transform coefficients γ and β. In the inference, those parameters are frozen and used in calculating the layer outputs. BN layers are usually located after convolutional layers and before activation functions. Weight Normalization [32], Layer Normalization [33], Instance Normalization [34], and Group Normalization [35] are also considered as normalization methods for stable learning.

iv. Loss Functions

During training a machine learning model, we need to set the object function for the model performance evaluation and optimization. In the evaluation, the difference between the predictions and the reality can be the criteria which measures how well the model generates the desired results. An optimizer aims to decrease the error in training networks, meaning that the predicted and the real distribution approach to be similar through backpropagation.

$$MAE = \frac{\sum_{i=1}^{N} |y_i - \hat{y}_i|}{N} \tag{2-1}$$

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

$$\mathcal{H} = -\frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{C} \left[y_{i,c} log \hat{y}_{i,c} + \left(1 - y_{i,c}\right) log\left(1 - \hat{y}_{i,c}\right) \right] \tag{2-3}$$

Mean Absolute Error (MAE) Loss (or L1 Loss, Equation **2-1**) and Mean Square Error (MSE) Loss (or L2 Loss, Equation **2-2**) are commonly used as regression loss functions. Besides, the cross-entropy (Equation **2-3**) is widely used in bi-class or multi-class computer vision tasks such as image classification and segmentation.

Equation **2-1** and Equation **2-2** use y_i and $\hat{y}_i$ as a data point and a target point in given N points. Both equations calculate errors from the discrepancy between the given data and the target data. Equation **2-3** compares between the real data probability distributions and the predicted probability distributions. In the equation, $y_{i,c}$ is the predicted probability with a range $[0, 1]$ and $\hat{y}_{i,c}$ denotes the actual probability like a binary. That is, $\hat{y}_{i,c}$ is 1 only when when ith pixel has a

label c. Based on the binary cross-entropy among the whole classes, we can measure whether the predicted labels are closer to the groundtruth ones.

Decreasing a loss function is the main objective in network training, but regularization methods should be considered to prevent overfitting that a model works well only for a training data.

v. Optimizers

Optimizers update weights of the neural network by calculating the gradient descent used for finding the minimum point through the backpropagation from a loss function. The optimization relies on training data information such as a batch size, the kind of optimizers, the learning rate, and the number of iterations. An optimizer has its own optimization algorithm to be applied with the objective function and given parameters, drawing the model to the right direction forward the optimal minimum.

Stochastic Gradient Descent (SGD) [36] simplifies the gradient computation by estimating the gradient based on a mini-batch of dataset, instead of calculating the entire gradient. SGD with momentum [37] and Nesterov's Accelerated Gradient (NAG) [38] accelerates SGD in terms of training speed with additional terms in updating the model weights. RMSProp [39] keeps a moving average of the squared gradient and adjusts the weight updates by the square magnitude. Beyond these optimizers, Adam [40] is also one of the most popular optimizers that combines SGD with momentum and RMSProp. Also, there are many variations such as AdaGrad [41], AdaDelta [42], AdaMax [40], and NADAM [43].

2.1.2 Regularization Methods

There are various regularization methods to restrain the overfitting problem. The regularization term is mainly applied on top of the objective function. In a wider view, modifications on factors such as input data, losses, model structures can be also considered as regularization methods.

In input level, we can give variations for preventing repetitive inputs, resulting in mitigating an overfitting phenomenon. For example, in image-related tasks, we create training data manifolds from data augmentation methods on both images and groundtruth labels. Also, in loss level, we can also provide additional losses such as L2 regularization loss which hinder a fast convergence of a given network. These losses come from model weights or noises on output features. Finally, in architecture level, we can consider manipulating the given model through functional layers or more convolutional layers. These layers give the flexibility or more capacity to the given model which can result in better generalization results.

Our work considers methods mentioned above except additional convolutional layers due to limited computation resources. We apply data augmentation to given input images, calculate other auxiliary losses in loss level, and use Dropout layers from backbone networks in architecture level.

i. Data Augmentation as Preprocessing

We can manipulate input data for creating diverse versions in each epoch. Data augmentation generates slightly manipulated copies of input data for preventing the model from repeatedly training only with the original input data. For image semantic segmentation, there have been several simple techniques to modify input images such as

- Simple affine distortions including translations, rotations, and skewing [44], and scaling [45]
- Giving variations on image characteristics like contrast, color histogram, and so on [46]
- Cropping [11], flipping [45], applying patches like 2D gaussians [47] or random transformations [48], erasing some image areas [49] [50] [51], and mixing multiple patches from different images [52] [53].
- Moreover, there are automated augmentation systems with multiple random strategies [54] [55] [56] or increasing magnitudes of augmentation [57] [21].

ii. Additional Losses

$$R(\theta) = \|\theta\|_1 = \lambda \sum_i |\theta_i| \tag{2-4}$$

$$R(\theta) = \|\theta\|_2 = \frac{\lambda}{2} \sum_i \theta_i^2 \tag{2-5}$$

An additional loss term can be added to the main loss function and a parameter can be tuned for mitigating overfitting phenomenon. Absolute-value (L1, Equation **2-4**) and Least-Square (L2, Equation **2-5**) regularization terms [1] are popular as one of regularization methods in training machine learning models. The terms are collected through all weight vectors (θ_i) and added to the main objective loss. With the regularization parameter λ, we can determine the magnitude of weights penalization by causing the weight decay in gradient descent.

In classification problems, repetitive class probability maps can cause the overfitting problem which makes a network insensitive to new data. To solve this problem, we can consider granting variations to the output probability distributions with label smoothing [2], disturbing by intentionally generating incorrect training labels [58], and penalizing distributions [59]. Those distributions have a role of noise in calculating the main loss, occurring an additional loss which can be seen as a regularizer like L1 or L2 losses mentioned above.

iii. Dropping Network Layers and Nodes

Also, we can expect the regularization effect by layer or structure transformations. Dropout [3] and Dropconnect [4] literally drop random layers and random weights, respectively. **Figure *2-3*** shows how Dropout and Dropconnect works, resulting that those methods provide the effect of training diverse random version of a given network. Dropout ignores layers with a given probability whereas DropConnect gives 0 weights in random weight connections between layer stages. Structure manifolds can also allow a regularization effect in several layers inside a given network.

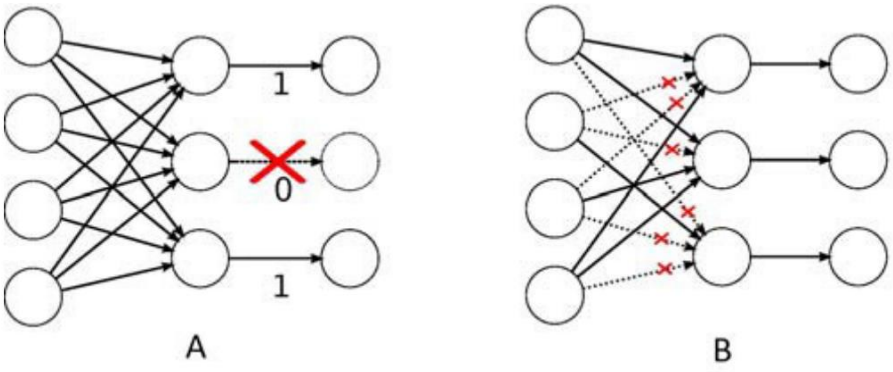

Figure 2-3. The comparison between (a) Dropout and (b) DropConnect, reprinted from *[60]*.

iv. Additional Layers

CNN models like VGG [13], ResNet [14], or MobileNet [17] [18] [19] have several backbone variations with the different number of parameters. In each model, the more parameters a backbone version has, the better it performs. Beyond the different versions given from backbone model papers, there have been trials to mitigate overfitting phenomena through additional simple modules inside a given network. In ResNet models, for example, Shake-Shake regularization [5] blends forward and backward flows with additional layers in the residual blocks (**Figure *2-4***). Compared to the Softmax method which has a fixed drop rate, the parameter α or β are trained like convolutional layers. Those parameters have fixed values at test time, like a softmax layer that it does not drop any stream in a test time.

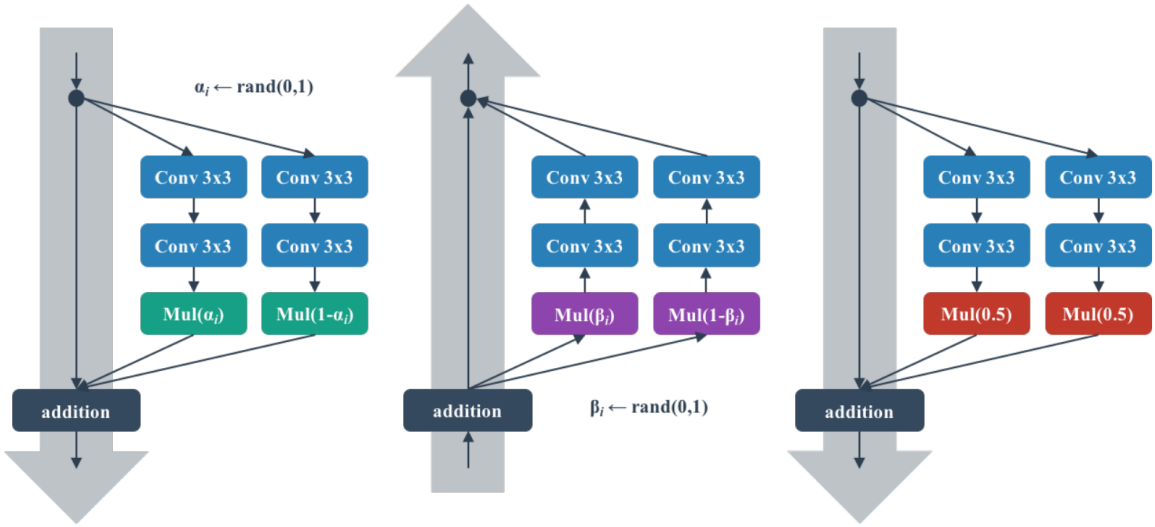

Figure 2-4. The description of Shake-Shake regularization, reprinted from [54]. Left: Forward training pass, Center: Backward training pass, Right: At test time. α and β are trainable and used in the forward training and the backpropagation, respectively.

2.2 Semantic Segmentation

Semantic Segmentation is one of popular computer vision tasks which processes given grayscale or RGB images and determines whether every pixel is related among pre-defined labels. The principal difference between object detection and semantic segmentation is that the former is patch-based and provides a bounding box of each object whereas the latter is pixel-based and shows an exact contour of each segmented object in images. Since many deep learning models and datasets have contributed to the improvement of segmentation tasks, Semantic Segmentation is utilized in various fields such as autonomous driving [61] and medical image analysis [62].

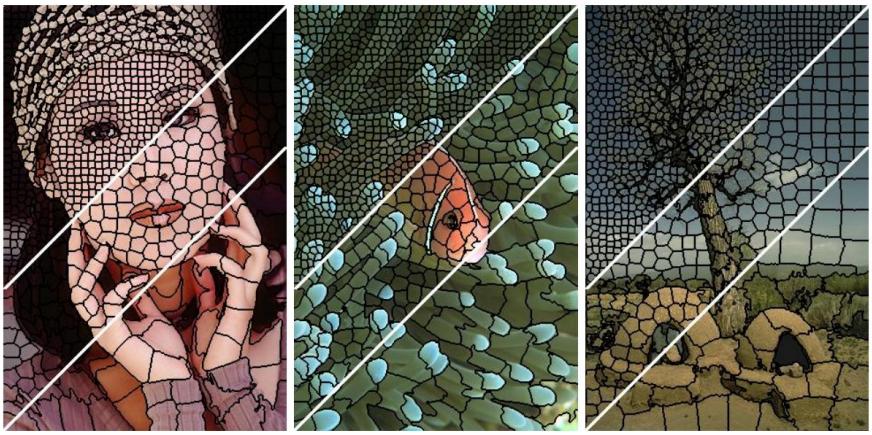

Figure 2-5. Image segmented using SLIC Superpixel into superpixels of size 64, 256, and 1024 pixels, reprinted from *[63]*.

There are several works before deep learning models become popular in semantic segmentation. Superpixel method [63] (**Figure *2-5***) segments an image into sets of pixels with the fixed pixel size via analyzing inter-pixel information. Kato et al. [64] proposed the use of Markov Random Field (MRF) in segmentation, which requires the neighborhood graphical model where

each pixel has a priori, and parameters are estimated in CIELUV colorspace rather than RGB. Simple Linear Iterative Clustering (SLIC) Superpixel method [63] generates superpixels by clustering pixels based on their color similarity in CIELAB colorspace. These methods require the computation power for calculating the relationship among pixels in the form of probability or color distances. Although there have been abundant of deep learning models for semantic segmentation, MRF [65] or Conditional Random Field (CRF) [66] [67] which is a variant of MRF are applied with those models to provide the additional information for better performance.

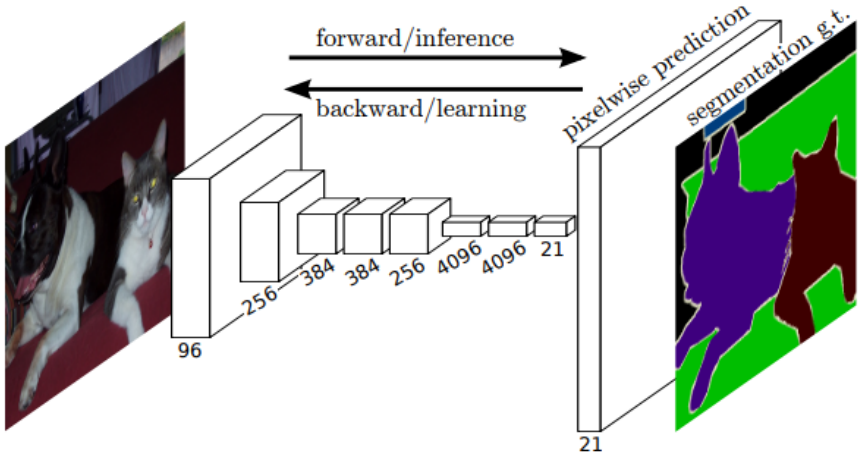

Figure 2-6. The description of training fully-convolutional networks for semantic segmentation, reprinted from *[68]*.

Beyond the methods requiring the prior information of the input data, Fully Convolutional Network (FCN) [68] became popular with adapting complex CNN models to semantic segmentation (**Figure *2-6***). Based on the structure in **Figure *2-6***, there have been numerous works with more various and complex structures such as U-Net [69], V-Net [70], SegNet [71], Pyramid Scene Parsing Network (PSPNet) [72], High-Resolution Network (HRNet) [73], and DeepLab

models [67] [74] [75]. For supervised learning, training CNN models for semantic segmentation only need the parameter tuning such as a learning rate, a weight decay rate, and a batch size with an annotated dataset.

A model can classify adjacent pixels in images after training with datasets which include images and corresponding annotations. In semantic segmentation task, there are several datasets typically used: VOC2012 [76] [77], CamVid [78], CityScapes [79], ADE20k [80], MS COCO [81], and KITTI [82]. Before training with a dataset, we can expect the faster convergence and better results with pretrained weights trained from another dataset if both datasets have similarities such as object categories. ImageNet [12] pretrained weights are frequently considered due to its amount of image data and categories which cover other datasets. Transfer Learning [83] also has an advantage if a model needs to solve a related problem which pretrained models already solved. In this case, weights from some stages in pretrained models are transferred to a destination model, like exploiting related knowledge.

2.3 Generative Adversarial Network (GAN)

2.3.1 Structure

Goodfellow et al. [6] proposed Generative Adversarial Networks (GAN), an adversarial framework which trains two models at the same time – a generator and a discriminator. A generator G aims to create fake data for mimicking a discriminator D between the real and the fake, from a given noise vector. The training procedure of the discriminator D focuses on the correct decision for given real examples and fake generated data. Therefore, GAN models enable semi-supervised and unsupervised learning. **Figure *2-7*** shows a general GAN model structure.

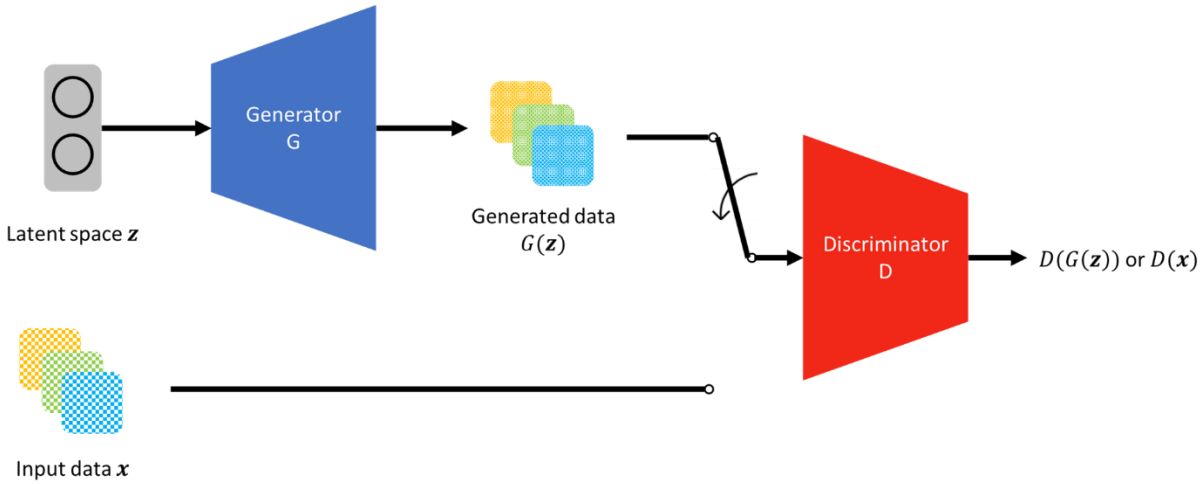

Figure 2-7. A general GAN model *[6]* structure. Both the data $G(\boldsymbol{z})$ from the given latent space $\boldsymbol{z}$ and the input data $\boldsymbol{x}$ are sequentially flowed to the discriminator for generating $D\big(G(\boldsymbol{z})\big)$ and $D(\boldsymbol{x})$, respectively. The discriminator is thereafter updated from the loss which compare between $D\big(G(\boldsymbol{z})\big)$ and $D(\boldsymbol{x})$.

From the given latent space $\boldsymbol{z}$ which can be a random noise vector, the generator network G aims to create the fake data $G(\boldsymbol{z})$ for mimicking the input data $\boldsymbol{x}$ whereas the training objective of the discriminator network D is maximizing the probability to correctly classify $G(\boldsymbol{z})$ and $\boldsymbol{x}$ as the fake and the real data, respectively. Thus, GAN network can be interpreted as 'two-player minimax game' between two networks [6].

$$\min_G \max_D V(G,D) = \mathbb{E}_{\boldsymbol{x}\sim p_{data}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}\left[\log\left(1 - D\big(G(\boldsymbol{z})\big)\right)\right] \tag{2-6}$$

Equation **2-6** shows the objective function of a general GAN framework. GAN training is unsupervised adversarial training where the generator *G* minimizes $\log(1 - D(G(\boldsymbol{z})))$ and the discriminator *D* maximizes $\log D(\boldsymbol{x})$.

For preventing overfitting on a given dataset, *D* is usually updated after $k > 1$ steps of training G. Also, we can consider substituting the optimization function of G from $\log(1 - D(G(\boldsymbol{z})))$ to $\log(D(G(\boldsymbol{z})))$ in the early training steps where G is poor enough that D can reject samples due to the discrepancy with the input data, resulting in the saturation of $\log(1 - D(G(\boldsymbol{z})))$.

$$\begin{aligned} V(G,D) = &\int_{\boldsymbol{x}} p_{data}(\boldsymbol{x}) \log\big(D(\boldsymbol{x})\big)\, dx + \int_{z} p_{\boldsymbol{z}}(\boldsymbol{z}) \log(1 - D(g(\boldsymbol{z})))\, dz \\ = &\int_{\boldsymbol{x}} [p_{data}(\boldsymbol{x}) \log\big(D(\boldsymbol{x})\big) + p_g(\boldsymbol{x}) \log\big(1 - D(\boldsymbol{x})\big)] dx \end{aligned} \tag{2-7}$$

$$\frac{d}{dD(x)}[p_{data}(\boldsymbol{x}) \log\big(D(\boldsymbol{x})\big) + p_g(\boldsymbol{x}) \log(1 - D(\boldsymbol{x}))] = \frac{p_{data}(\boldsymbol{x})}{D(\boldsymbol{x})} - \frac{p_g(\boldsymbol{x})}{1 - D(\boldsymbol{x})} = 0 \tag{2-8}$$

By Equation **2-8**, the optimal discriminator $D^*(\boldsymbol{x}) = \frac{p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x})+p_g(\boldsymbol{x})}$ can be derived from the fixed generator G in continuous space for maximizing $V(G, D)$ (Equation **2-7**). With enough capacity of G and D, the discriminator converges to $D(\boldsymbol{x}) = 0.5$ where two distributions cannot be differentiated.

2.3.2 GAN Derivatives

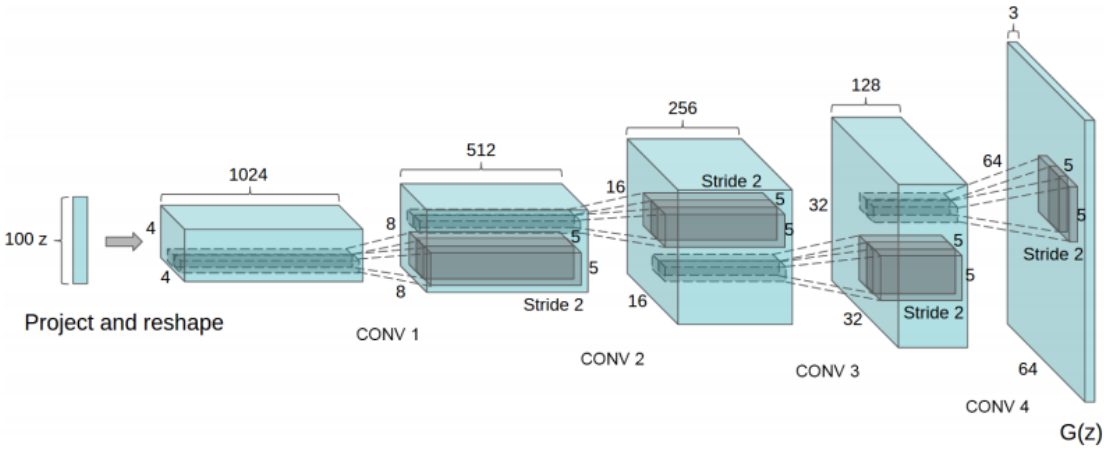

Figure 2-8. DCGAN generator structure, reprinted from *[84]*. '*DeconvolutionalLayer-BN-ReLU*' blocks are used for increasing the height and the width of each feature map before $G(z)$ whereas the '*DeconvolutionalLayer-Tanh*' block generates $G(z)$ in the figure.

Deep Convolutional GAN (DCGAN) [84] extended GAN to CNN models and proposed a set of constraints for stable DCGAN training by specifying different types of layers. The main block of the generator (**Figure *2-8***) is '*DeconvolutionalLayer-BN-ReLU*'. The deconvolutional layers [85] in the block increase the resolution of feature maps and ReLU activation layers are used at the end of each block. However, $G(z)$ is calculated from the '*DeconvolutionalLayer-Tanh*' block, not the same one as the main block mentioned above. Meanwhile in the discriminator, the feature maps

are downsampled through '*Convolution-BN-LeakyReLU*' blocks where BN layer is unused in the first block whose input is $G(z)$. The number of channels in each feature map increases from 3 to 512 with the same kernel size to that of deconvolutional layers, resulting in a larger sigmoid feature vector size. Also, DCGAN removed fully-connected hidden layers for deeper convolutional structures. DCGAN architectures have been widely utilized in later frameworks.

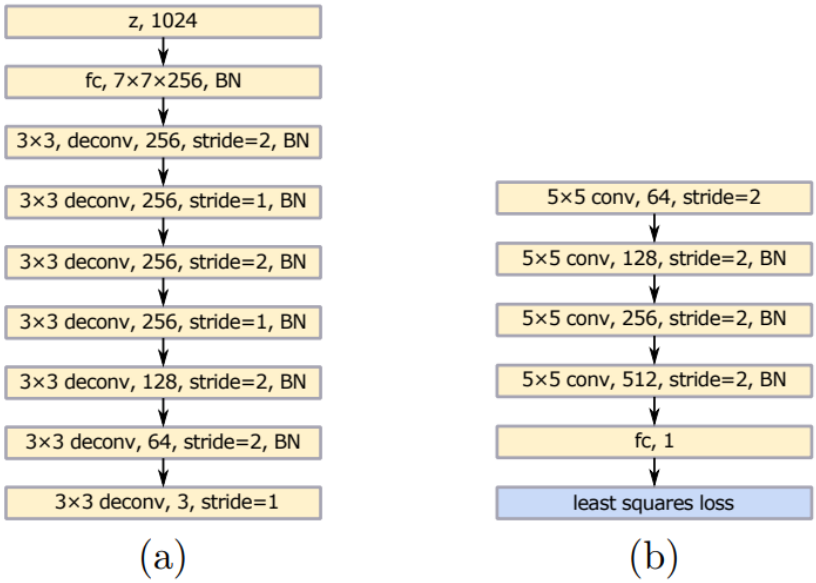

Figure 2-9. LSGAN *[86]* structure - (a) and (b) describe the generator and the discriminator, respectively, reprinted from *[86]*.

LSGAN [86] pointed the vanishing gradients problem in training a regular GAN model whose discriminator has a role of a classifier by using the sigmoid cross-entropy loss function. **Figure *2-9*** describes the LSGAN architecture, following DCGAN criteria such as where to locate BN layers, ReLU or LeakyReLU activation usages. To solve the problem of vanishing gradients from

the fake samples which are categorized correctly but far from the real data, the objective functions (Equation **2-9** and **2-10**) are redefined with using least-square terms in the sigmoid function.

$$\min_D V(D) = \frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{data}(\boldsymbol{x})}[(D(\boldsymbol{x})-1)^2] + \frac{1}{2}\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\big(D\big(G(\boldsymbol{z})\big)\big)^2] \tag{2-9}$$

$$\min_G V(G) = \frac{1}{2}\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}\left[\big(D\big(G(\boldsymbol{z})\big)-1\big)^2\right] \tag{2-10}$$

Through Equation **2-9**, LSGAN discriminator is fixed during updating LSGAN generator, resulting that generated samples are moving toward the decision boundary, which leads them to be closer to real data and prevents a GAN network from the vanishing gradient problem. This characteristic comes from the least-square terms in Equation **2-9** and **2-10** which are only flat at the specific point, whereas the sigmoid cross-entropy terms on Equation **2-6** saturate on the relatively large x values.

Wasserstein GAN (WGAN) [87] also aims to improve the training stability and remove issues such as vanishing gradients and mode collapse [88]. The authors propose Earth-Mover (EM) distance substituting other distances between probability distributions. Through EM distance, we can expect a GAN model to generate more meaningful gradients. Also, training WGAN critic with constrained weights till optimality limits the possibility of mode collapse.

According to Gulrajani et al. [89], WGAN still occurs either vanishing or exploding gradients from the clipping threshold on the critic, even though it made a great step forward the stable GAN training. Gulrajani et al. [89] proposed WGAN-GP (Gradient Penalty) for enforcing the Lipschitz constraint alternatively by the constraints on the gradient norm of the critic's output with respect

to its input. The objective of WGAN-GP is mitigating those issues by introducing the constraint with a penalty on the gradient norm of the critic's output.

There also have been structural evolutions of GAN models such as Boundary Equilibrium GAN (BEGAN) [90], Self-Attention GAN (SAGAN) [91], and BigGAN [92]. Also, numerous GAN models are utilized in computer vision tasks such as style transfer [93] [94], image super resolution [95] [96], and image-to-image translation [93] [97].

2.3.3 GAN in Semantic Segmentation

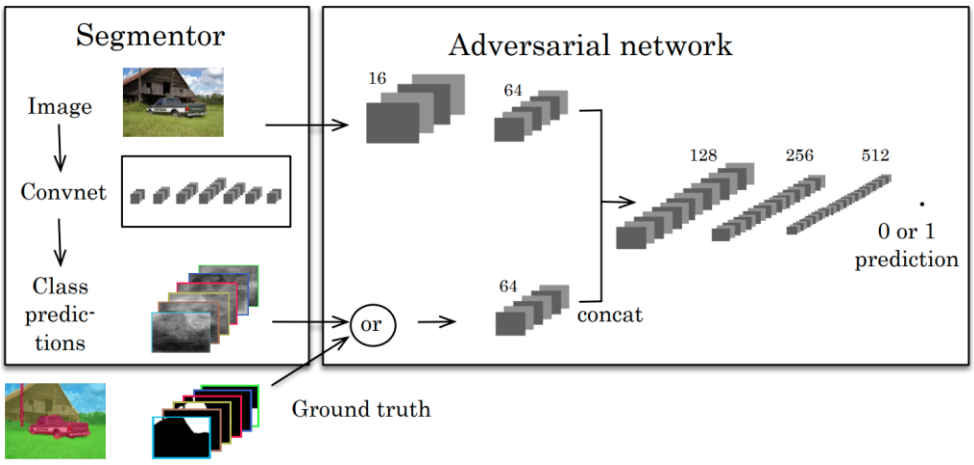

Figure 2-10. The structure for semantic segmentation using the adversarial network, reprinted from *[98]*. RGB images are processed in the segmentor, resulting in class predictions. The adversarial network gets either predictions or groundtruth labels as well as RGB images as input and produces class labels as the output.

Luc et al. [98] proposes the method utilizing the GAN structure for semantic segmentation. **Figure *2-10*** shows the overview of the approach of [98]. The segmentor creates class predictions from a given image and the adversarial network calculates a single prediction score from the concatenated data from the image and either predictions or groundtruth labels. Through this

procedure, the adversarial network contributes an additional binary-crossentropy loss to the segmentor network loss. Xue et al. [99] propose SegAN with the similar structure for medical image segmentation, but it calculates the multi-scale feature loss as the form of L1 losses.

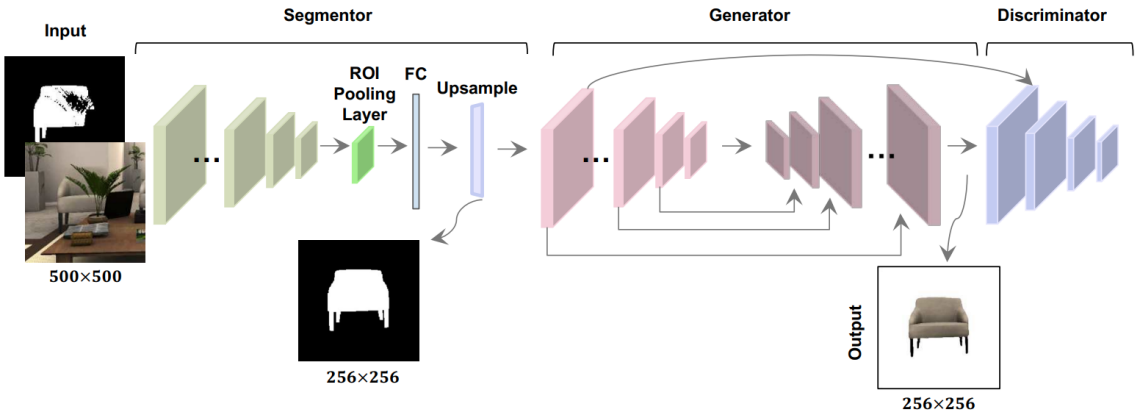

Figure 2-11. The three-stage segmentation architecture of *[100]*, reprinted from *[100]*. A RGB image and a mask for visible regions of an object are the segmentor input. From a mask which shows an intermediate mask, the generator output shows the full object segmentation result.

GAN models can also perform semi-supervised segmentation tasks. Souly et al. [101] follows the GAN structure illustrated in **Figure *2-7*** but the discriminator generates confidence maps rather than classification scores. Hung et al. [102] build own system which starts from RGB images and performs segmentation tasks whereas [101] starts from random noise vectors. Both works train networks with both labeled and unlabeled data. Moreover, **Figure *2-11*** describes the architecture of Ehsani et al. [100] with a more sophisticated structure with three parts. The framework detects occluded parts of objects by attaching the generator and the discriminator after the segmentor,

2.4 Knowledge Distillation

2.4.1 Temperature in Softmax Probabilities

Hinton et al. [7] proposed Knowledge Distillation as the method to transfer the knowledge from a deep and complex model to a smaller model which is more efficient in terms of deployment. Classic objective functions aim to generate correct answers by maximizing the average log probability of them, but the functions also take other probabilities from wrong answers into the calculation. Therefore, with the given probability information from the training data, the model should be generalized to new data with the correct direction.

Distilling the knowledge from a larger network to a smaller one enables the smaller model to be generalized like the way the larger model was, resulting that the distilled model outperforms ones without distillation. In [7], the authors introduce the temperature concept for generating "soft targets" for the knowledge from high-end models. Temperature T is a constant value which divides logits in softmax calculations. Soft and hard targets are from $T > 1$ and $T < 1$ conditions, respectively. The former weakens the dominant class whereas the latter strengthens it, resulting that the temperature gives variations in a given probability distribution. Through a high entropy from the soft targets due to higher probabilities in minor classes, less data and a higher learning rate are possible in training a simpler model due to more information than hard targets and less gradient variance.

	Class					
	0	1	2	3	4	5
Logits z_i	**-1.3**	**0.8**	**1.3**	**0.4**	**-0.6**	**0.1**
Softmax (T=1)	**0.029**	**0.239**	**0.394**	**0.160**	**0.059**	**0.119**
T=0.5	**0.003**	**0.223**	**0.605**	**0.100**	**0.014**	**0.055**
T=2	**0.075**	**0.215**	**0.276**	**0.176**	**0.107**	**0.151**

$$q_i = \frac{e^{z_i/T}}{\sum_j e^{z_j/T}}$$

Figure 2-12. Softmax with temperature calculation example.

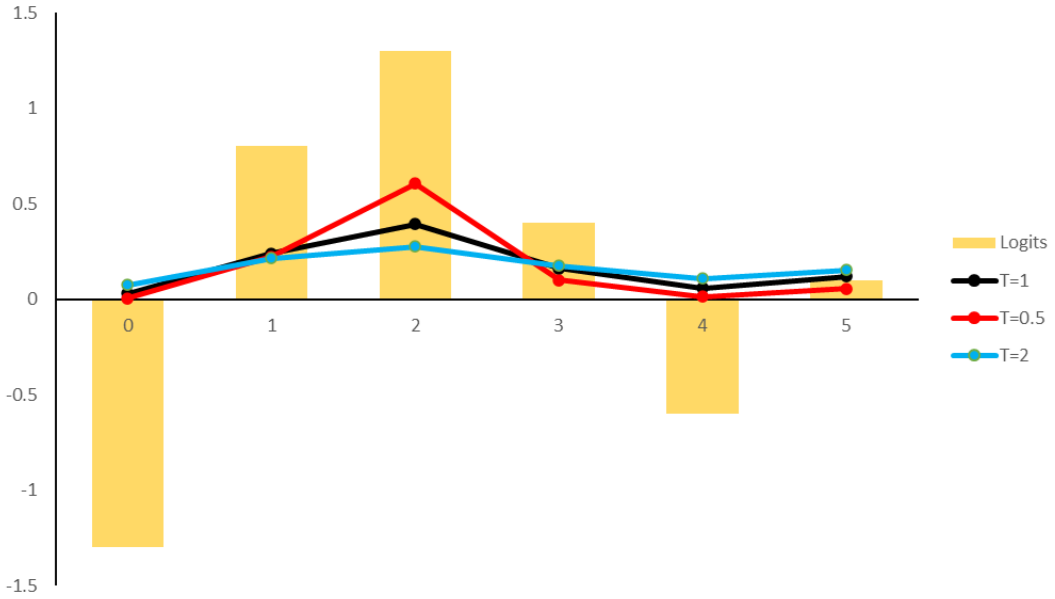

Figure 2-13. The corresponding softmax probabilities for given logits in **Figure 2-12**. A classification network should be optimized for correctly indicate that the prominent class is *Class 2*. Compared with the default temperature ($T = 1$, Black line) in softmax calculations, $T = 0.5$ (Red line) strengthens *Class 2* and helps the network converge faster. In the case of $T = 2$ (Blue line), classes other than *Class 2* have higher probabilities and the distribution among classes is closer to the uniform distribution which increases the uncertainty of the main network.

Like in **Figure 2-2**, **Figure *2-12*** illustrates an example of softmax calculation with different temperature values. The figure supposes the 6-class classification problem and the logits come from right before the softmax activation layer. The default softmax layer converts these logits into probabilities which have the total probability value is 1. Therefore, the given input data should be classified as 'class 2' because the logit of class 2 is the highest.

When we compare three temperatures as illustrated in **Figure 2-13**, a temperature lower than 1 strengthens the highest probability (*class 2* in this case) and weakens minor probabilities whereas a temperature higher than 1 mitigates the probability distributions among classes and allows higher probabilities on minor classes. As we calculate the cross-entropy loss (Equation **2-3**) for backpropagation, higher temperatures make a network converge faster whereas lower ones give more loss values in the given logits from the classification network. For instance, the cross-entropy losses for each temperature in **Figure 2-13** are $\{0.693,\ 0.405,\ 0.903\}$ for $T = \{1, 0.5, 2\}$, respectively. Our initial thought for temperature as a regularizer comes from this characteristic, by granting an additional loss to a given segmentation network.

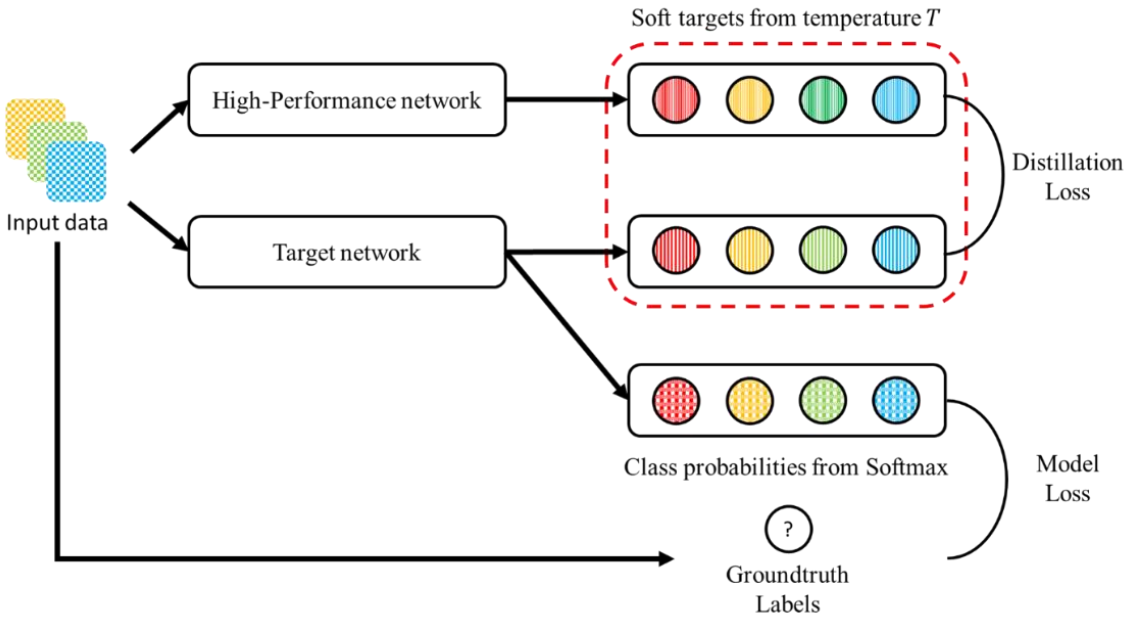

Figure 2-14. Knowledge distillation framework description *[7]*. Probability maps from the input data are used in the main loss for training the target network. For the distillation loss, the temperature T is applied to softmax calculations in both networks in the figure. The target network is trained by the backpropagation from the combination of both losses.

Through this form of distillation described in **Figure *2-14***, a temperature in the softmax is used in the model with better performance for producing a transfer set which knowledge is transferred to the smaller model through. The temperature becomes a higher value than 1 in the distillation and turns back to 1 after the lighter model is trained. Thus, soft targets are used in the cross-entropy loss with those from both the smaller and the larger models, which is added in the training loss of each model. In terms of regularization, training a model with soft targets and a part of training dataset prevents the model from overfitting and enables it to recover the information from the full dataset.

2.4.2 Applications

As a high-performance model is necessary in the distillation, one can recognize that it is deemed as a teacher and a target model can be seen as a student. We can have several options to select the student model: A simplified or quantized version of the teacher, a smaller network for efficiency, or the same network as the teacher [103]. There have been various works which improve the Teacher-Student structure. You et al. [104] proposed multiple teacher networks learning strategy. Mirzadeh et al. [105] introduced intermediate models between the teacher and the student as teacher assistants for filling their gap. Tarvianen et al. [106] proposed the Mean Teacher method which averages model weights on the teacher side.

Beyond the teacher-student relationship, there also have been methods which train two or more neural networks simultaneously. Zhang et al. [107] proposed the Deep Mutual Learning (**Figure *2-15***) method for training multiple neural networks at the same time, by adding the Kullback-Leibler Divergence to each model loss. With the average loss of $N-1$ KL divergence values in each model when there are N networks, we can expand this structure to multiple models, resulting in sharing knowledge and training multiple models mutually.

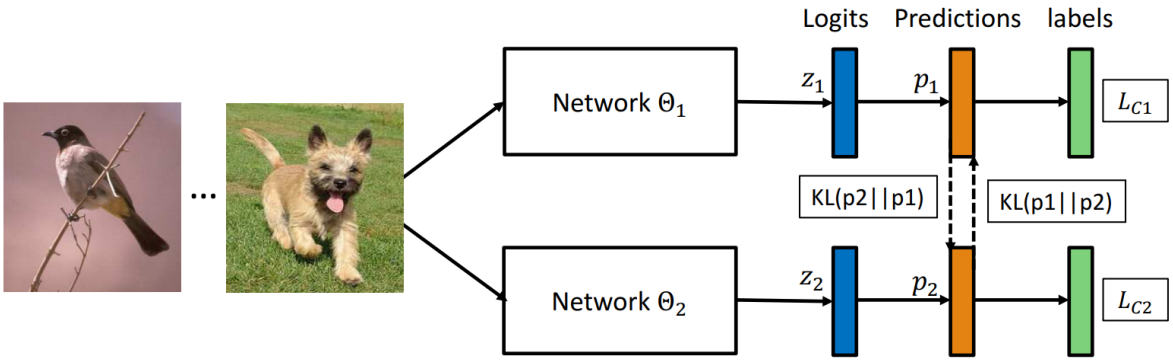

Figure 2-15. Deep mutual learning description, reprinted from *[107]*. In the case of two models, each network is trained with a supervised learning loss, and a Kullback-Leibler (KL) divergence based mimicry loss to match the probability estimates of its peers. This framework can extend to more networks in the student cohort.

Different from other works whose input data is given, the data-free framework from Chen et al. [108] starts from the random signals in the generator and uses the given teacher and the student networks in the discriminator side. In **Figure 2-16**, the generator weights are updated through the backpropagation with the loss functions from the teacher network by passing training samples to the teacher discriminator. Then, the student discriminator is updated by calculating the knowledge distillation loss between the outputs from each discriminator, resulting in better performance in the lightweight student network.

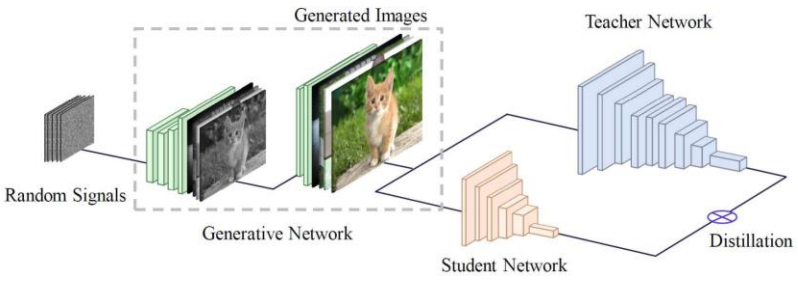

Figure 2-16. The diagram of the data-free method, reprinted from *[108]*. The generator is trained for approximating images in the original training set by extracting useful information from the given network. Then, the student network can be effectively learned by using generated images and the teacher network.

Based on the idea of utilizing the GAN structure for semantic segmentation from [98], Liu et al. [109] introduces their distillation framework (**Figure 2-17**) which passes two predicted score maps from a teacher network and a student network to the discriminator. In this framework, there are three distillation loss terms: Pair-wise, Pixel-wise, and Holistic. From similarity maps from each feature map from two networks, pair-wise loss transfers the pair-wise relations among pixels. Pixel-wise loss is calculated from KL divergence between two soft targets of class probabilities. Through Wasserstein loss proposed in [87], the discriminator can produce a holistic embedding representing to show whether the input image and the segmentation map match.

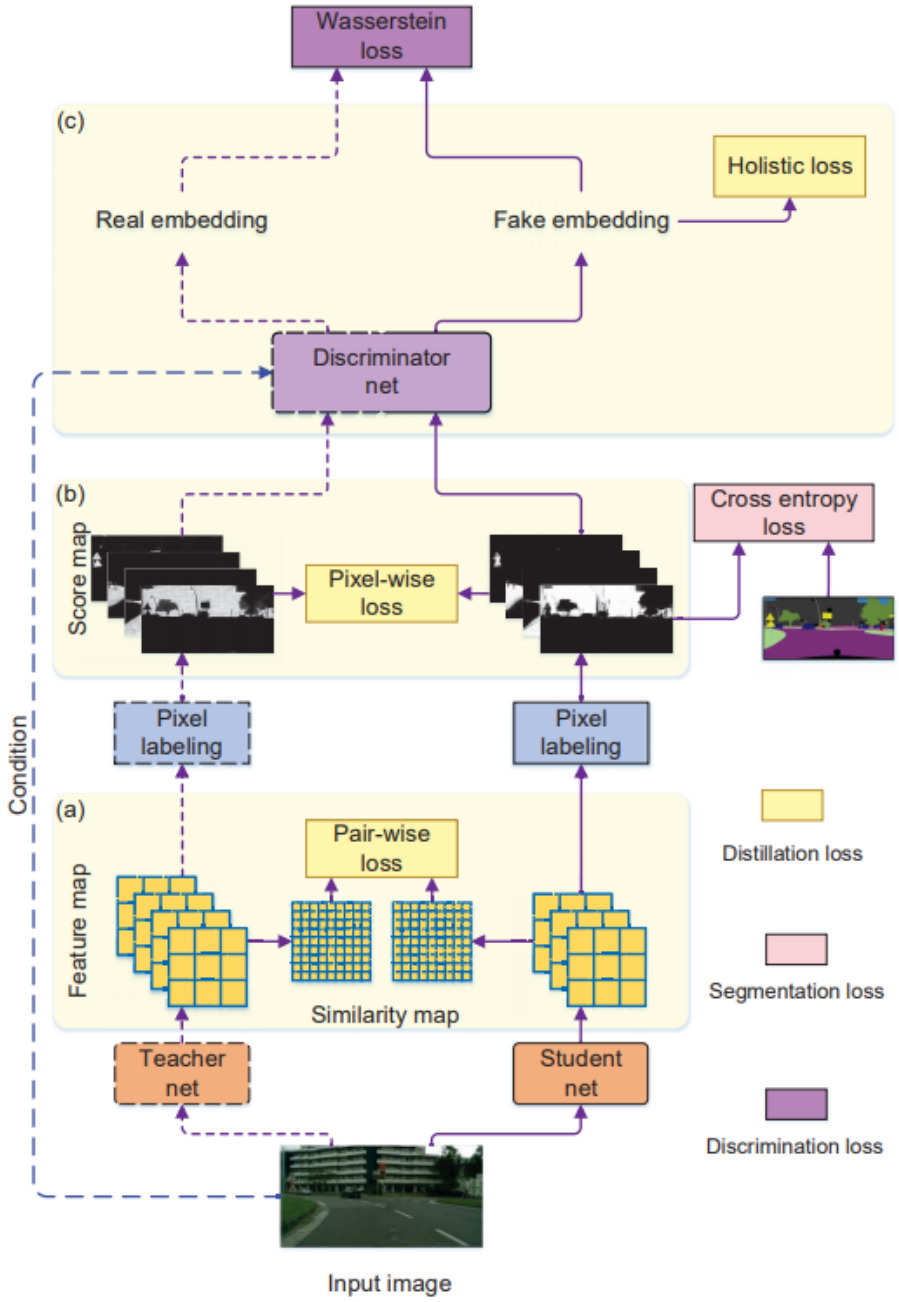

Figure 2-17. The distillation framework with three stages for semantic segmentation, reprinted from *[109]*. Different from **Figure 2-14**, the framework uses additional losses such as (a) Pair-wise loss and (c) Holistic loss whereas (b) Pixel-wise loss is similar to the distillation loss in **Figure 2-14**. On top of the teacher and the student, the holistic loss comes from the discriminator originally used in a GAN.

3

Adversarial Framework

Section 3.1 explains the structure of adversarial framework for semantic segmentation. In Section 3.1.1, We describe our backbone segmentation networks and illustrates metrics for evaluation. In Section 3.1.2, we suggest the use of LSGAN losses in the DCGAN discriminator to substitute DCGAN losses which are prone to saturate and prevent the mode collapse. Also, we consider location maps in loss calculations for ignoring meaningless pixels.

In **Section 3.2**, we introduce our algorithm for the framework. In the framework, we apply temperature T_g on the generator (segmentation network) and T_d on the discriminator, generating

two additional losses: Adversarial loss ($\mathcal{L}_{adv}$) and Discriminator loss ($\mathcal{L}_D$). Specifically, the adversarial loss $\mathcal{L}_{adv}$ affects the segmentation network as a regularizer like L2 regularization loss.

3.1 GAN-Resembled Framework

Generative Adversarial Network (GAN) consists of two networks: A generator and a discriminator. With a noise latent space, the generator aims to create fake samples and disguise the discriminator whereas the discriminator aims to label both 'real' and 'fake' samples correctly. Through this adversarial process, we can expect that the generator can produce fake ones very similar enough to behave like real ones. Although our work is inspired by the GAN structure, the main difference between our adversarial framework and GAN is that a generator in our framework does not calculate probability maps from random noise. Rather than the generative process like the generator in GAN models, a segmentation network in our framework follows a supervised learning task where RGB input images are given for output feature maps.

Although knowledge distillation shows the improvement of a student network from teacher network(s), it inevitably requires allocating two or more neural networks into the GPU memory. The training procedure of knowledge distillation is not suitable in a single GPU environment due to the memory consumption of each network. Therefore, inspired by the work of Hung et al. [102], we use both a segmentation network and a discriminator for an auxiliary loss for penalizing the segmentation network which provides the probability maps. As described in **Figure *3-1***, we concatenate two softmax layers with different temperatures T_g and T_d after the logits from the segmentation network for the calculation of the adversarial loss.

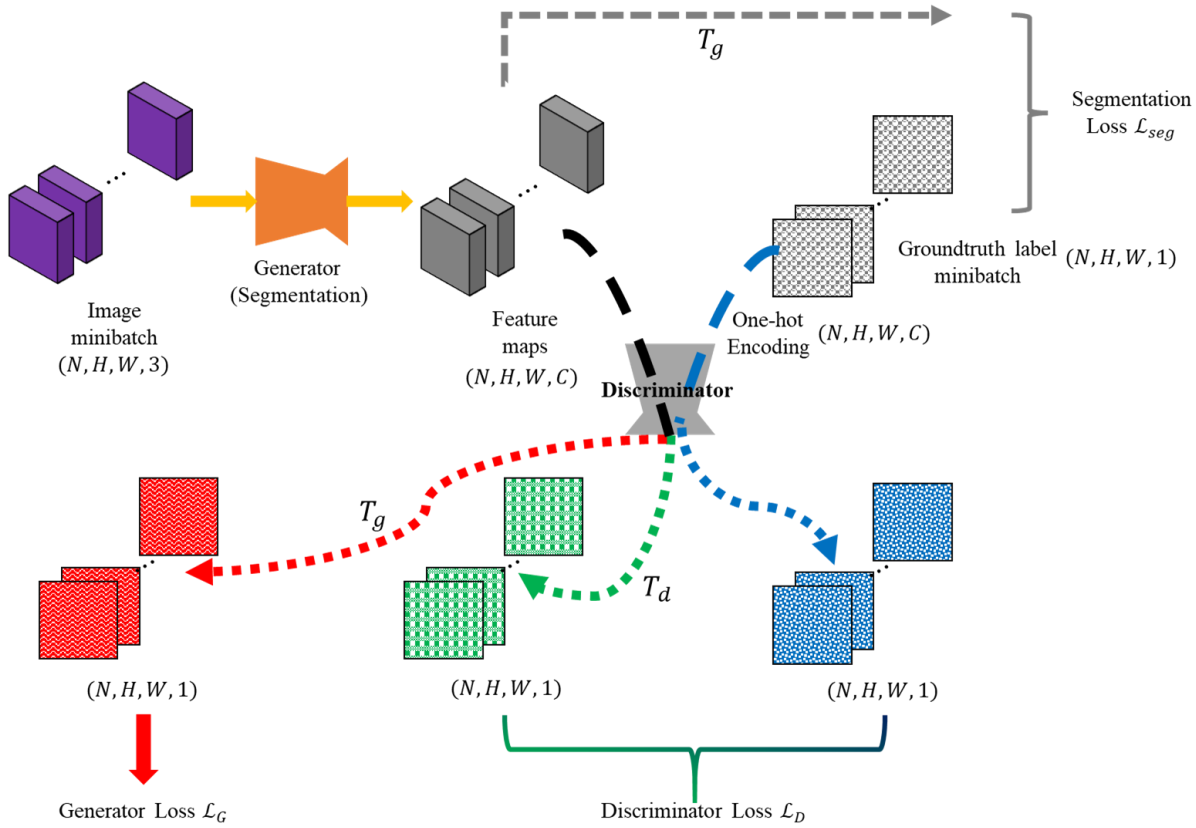

Figure 3-1. Our framework with segmentation network and discriminator. The segmentation network generates two softmax probability maps with given feature maps through dividing by T_g and T_d. We calculate both the segmentation loss $\mathcal{L}_{seg}$ and the adversarial loss $\mathcal{L}_{adv}$ with the generative temperature T_g whereas we pass the probability maps calculated with the discriminative temperature T_d to the discriminator ($\mathcal{L}_D$).

There are three losses: The segmentation loss $\mathcal{L}_{seg}$, the generator loss $\mathcal{L}_{\mathrm{G}}$, and the discriminator loss $\mathcal{L}_{\mathrm{D}}$. The segmentation loss $\mathcal{L}_{seg}$ is the cross-entropy loss from the segmentation softmax probabilities with the generative temperature T_g and the groundtruth labels. The generator loss $\mathcal{L}_{\mathrm{G}}$ is calculated from the same softmax probabilities used in $\mathcal{L}_{seg}$. Score

maps from the outputs of the discriminator whose inputs are the segmentation feature maps and the one-hot encoded labels are used in the discriminator loss $\mathcal{L}_{\mathrm{D}}$.

Figure 3-2 describes the detailed loss flows from segmentation feature maps and groundtruth labels in **Figure *3-1*** with temperatures. In the figure, the adversarial loss $\mathcal{L}_{adv}$ and the discriminator loss $\mathcal{L}_D$ are used for updating the segmentation network and the discriminator, respectively. When added to the segmentation loss, the generator loss with weight α can take a role as a regularizer like L2 regularization term.

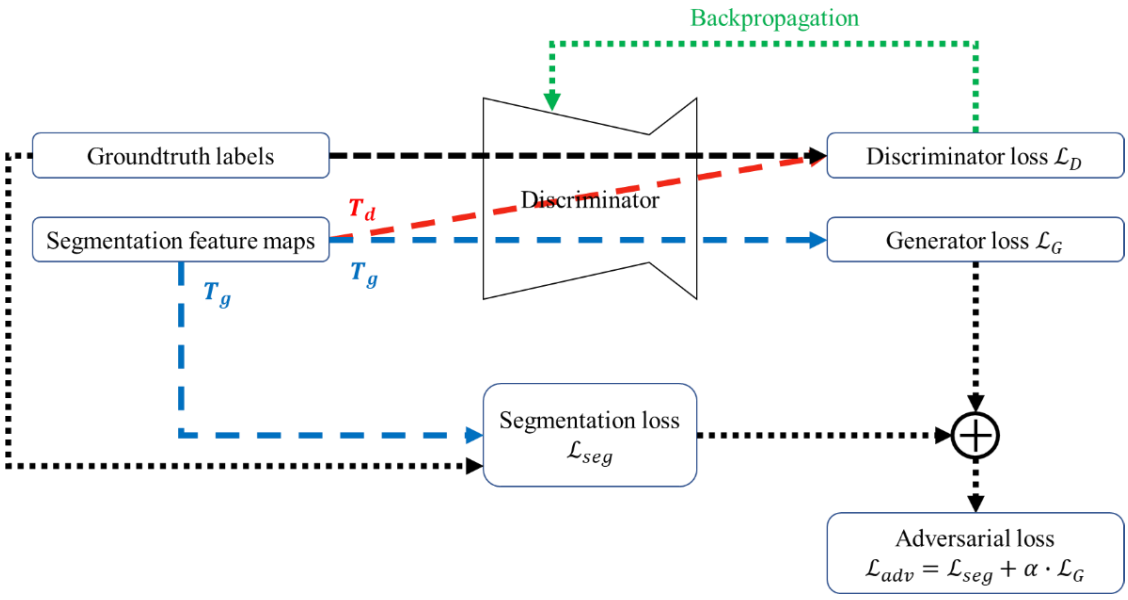

Figure 3-2. The details for explaining procedures after a given segmentation network in **Figure 3-1**. Both the segmentation feature maps and groundtruth labels are passed into the discriminator for calculating the adversarial loss $\mathcal{L}_{adv}$ from the generative temperature T_g and the discriminator loss $\mathcal{L}_D$ from the discriminative temperature T_d, respectively. We update the discriminator with $\mathcal{L}_D$ directly whereas we need to add $\mathcal{L}_{adv}$ to $\mathcal{L}_{seg}$ with weight α for updating the generator.

3.1.1 Segmentation Network

In **Figure *3-1***, the segmentation network has a role of generator, resulting that it provides the probability map to the discriminator with groundtruth labels. For the given dataset, it calculates the probability information of each categorized class. For given dimensions $(B, H, W, 3)$ where the batch size is B and H, W are the fixed height and width, respectively, the output dimension of the feature map is (B, H, W, C) where C is the number of classes.

In each training epoch, data augmentation methods such as random zooming, random crop, and resizing should be applied to convert the input image size from different size of images of a given dataset to the fixed input size. As a result, the segmentation network aims to generate the probability maps whose height and width are same as those of the input data. With the output maps, the groundtruth labels in **Figure *3-2*** are converted to maps with the same dimension, from $(B, H, W, 1)$ to (B, H, W, C) through one-hot encoding, for calculating the multi-class cross-entropy loss ($\mathcal{L}_G$, Equation **2-3**).

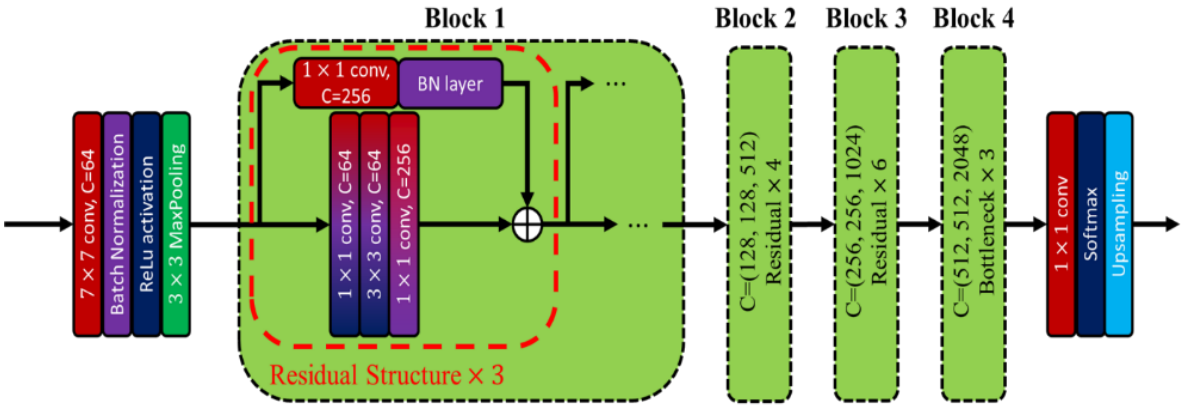

Figure 3-3. ResNet-50 architecture description *[14]*. ResNet models focuses on the residual structure for better performance. Each block has different numbers of residuals and output channels. In the case of deeper models such as ResNet-101 and ResNet-152, the number of 'Residual Structure' and the number of parameters increase, resulting in better performance than ResNet-50. Considering less GPU memory usage, we select ResNet-50 as a segmentation network in **Figure 3-1**.

Our model selection for semantic segmentation is ResNet-50 [14] and FCN [68] with VGG-16 [13] backbone. ResNet models have a residual structure which reuses the input of 3 convolutional layers and combines both the input and the output of 3 conv layers as a form of the addition layer. In the residual structure of each block, two blocks (1×1 conv and BN layer) work for matching the number of output channels, which only exist once in the beginning of each block. In other residuals, the input and the output of three convolutional layers are added elementwise. In each block, the number of channel is multiplied by 2. After Block 4, the 1×1 conv layer converts the number of channels to that of classes and the probability map is generated through softmax and upsampling layers with the dimension (B, H, W, C).

Although there are several applications of ResNet architecture for semantic segmentation, such as DeepLabV2 [74] and PSPNet [72] which have better performance than the original ResNet

networks. However, the models mentioned are not considered due to the model complexity which occurs the increase of training time greatly and the GPU memory consumption from excessive allocations of feature maps in each network.

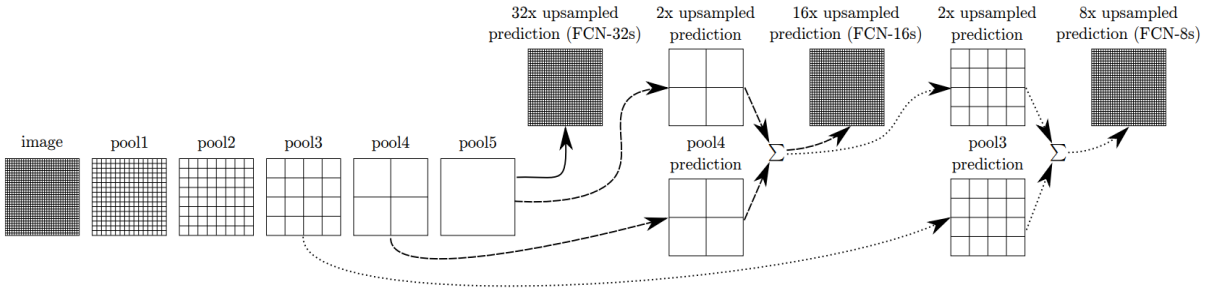

Figure 3-4. FCN architecture from convolutional networks, reprinted from *[68]*. FCN upsampling modules are built upon VGG-16 or VGG-19 model, according to *[68]*. FCN-32s upsamples stride 32 predictions back to pixels in a single step. FCN-16s combines predictions from both the final layer and the *pool4* layer and provides finer information with stride 16 upsampling. FCN-8s combines the additional predictions from *pool3* layer, at stride 8. With ResNet-50, we select FCN-8s with VGG-16 as a segmentation network in **Figure 3-1**.

FCN [13] utilizes convnets for semantic segmentation and improves the network performance by combining the coarse feature maps from higher layers with the fine maps from lower layers. **Figure *3-4*** shows how FCN extracts semantic information from convolutional layers from *pool3* to *pool5*. Apparently, FCN-32s is the output probability map which is the same as a prediction map of a typical semantic segmentation model. In FCN-16s and FCN-32s, the upsampled prediction maps from each previous stage and the prediction from *pool4* or *pool3* are added elementwise, providing more detailed feature maps. In our experiments, we selected FCN-8s with VGG-16 backbone, which is the same configuration as the original paper.

Semantic segmentation results are to be evaluated in terms of Pixel Accuracy and Intersection-on-Union (IoU) from the label information of the predicted and the groundtruth. The main difference of both terms is that the former calculates the number of correctly predicted pixels over the number of groundtruth pixels in each class whereas the latter calculates the intersection between the prediction and the groundtruth over the total number of pixels which are included in either the prediction or the groundtruth. **Figure *3-5*** presents the better comparison between two metrics.

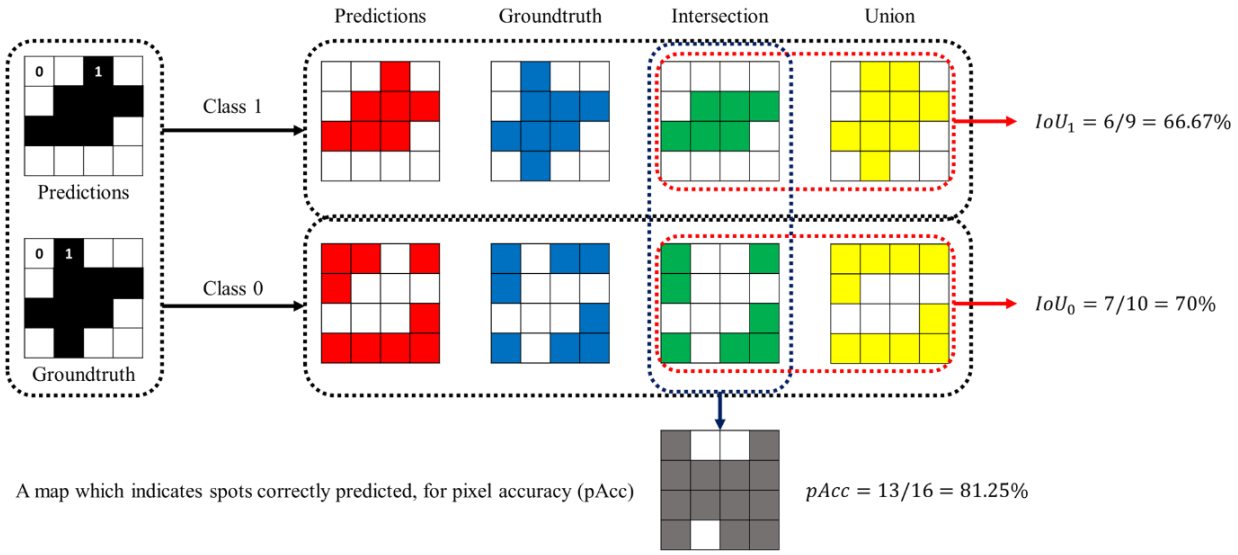

Figure 3-5. An illustration for explaining pixel Accuracy (pAcc) and Intersection-of-Union (IoU). Pixel accuracy counts the correctly predicted pixels in whole classes whereas IoU divides the number of true positives (TP) from the sum of true positives (TP), false positives (FP), and false negatives (FN) in each class.

In **Figure *3-5***, we can also see the intersection areas and the union areas of each class, based on the bi-class predictions and the groundtruth. For the pixel accuracy, we count the total pixel

number from intersection areas of all classes, like the gray map in the bottom of the figure. Therefore, the pixel accuracy of this case is $13/16 = 81.25\%$.

For calculating mean IoU (mIoU), we need to calculate $IoU = \frac{\#\ of\ pixels\ from\ Intersection}{\#\ of\ pixels\ from\ Union}$ in each class. In **Figure *3-5***, $IoU_0 = 7/10 = 70\%$ and $IoU_1 = 6/9 = 66.67\%$. mIoU is the mean value of IoUs - $mIoU = \frac{\sum_{i=1}^{C} IoU_c}{\#\ of\ classes}$ where C is the number of classes. Therefore, mIoU in **Figure *3-5*** is $\{(7/10) + (6/9)\} / 2 = 68.33\%$.

As the pixel accuracy counts only the number of intersection pixels from all classes and divide them into the number of valid total pixels except for pixels to be ignored, a dominant class which has a larger number of pixels than other classes affects the accuracy, resulting that it cannot indicate the accuracy information of minor classes. Meanwhile, mIoU can be an alternative way to overcome this problem because it calculates the average value by adding IoU probabilities from all classes. In our experiments, we indicate both metrics for comparison.

3.1.2 Discriminator Network

In **Figure *3-1***, two temperatures T_g and T_d are applied to each softmax layer and the discriminator distinguish which probability map is for real or fake. Luc et al. [98] proposed the adversarial framework for semantic segmentation which uses the single binary at the end of the discriminator, indicating whether a given label map is the groundtruth or the synthetic. Hung et al. [102] proposed to generate the confidence map from the discriminator and calculate the adversarial loss $\mathcal{L}_{adv}$ for penalizing the segmentation network as an auxiliary loss. However, both works use the same feature map in the segmentation cross-entropy loss and the losses $\mathcal{L}_G$ and $\mathcal{L}_D$ from the

discriminator. Thus, our work aims to apply temperatures for preventing overfitting by providing an additional probability map to either side and keep the original probability map from the segmentation network.

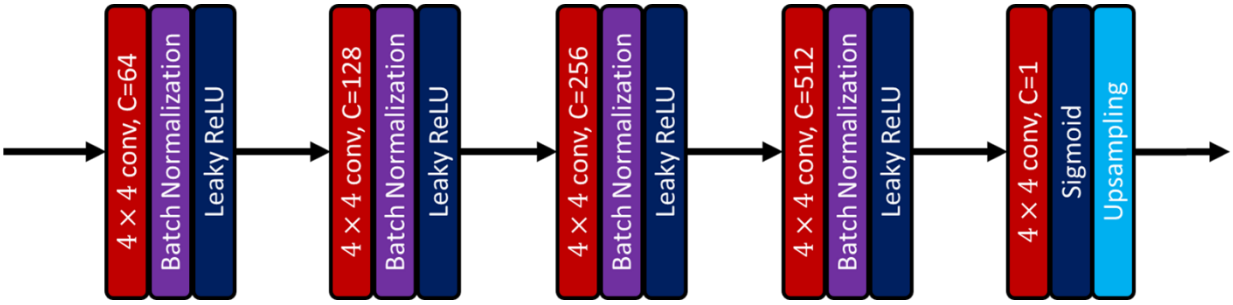

Figure 3-6. The structure of the discriminator in our framework in **Figure 3-1**. Compared with segmentation networks (FCN-8s-VGG16 and ResNet-50), the discriminator is relatively simple and lightweight.

The discriminator (**Figure *3-6***) comes from DCGAN discriminator [84], but the kernel size of convolutional layers is 4×4 whereas DCGAN has the kernel size 5×5. Also, the block after the layers with the channel size $C = 512$ converts logit maps to the probability maps through the convolutional layer with kernel size 1 and the sigmoid activation function. For given probability maps whose dimensions are (B, H, W, C), the discriminator generates the probability maps where each pixel has value $[0, 1]$ with the dimension $(B, H, W, 1)$, resulting that we can calculate the discriminator loss through all pixels except for ones with ignored classes (**Figure *3-7***).

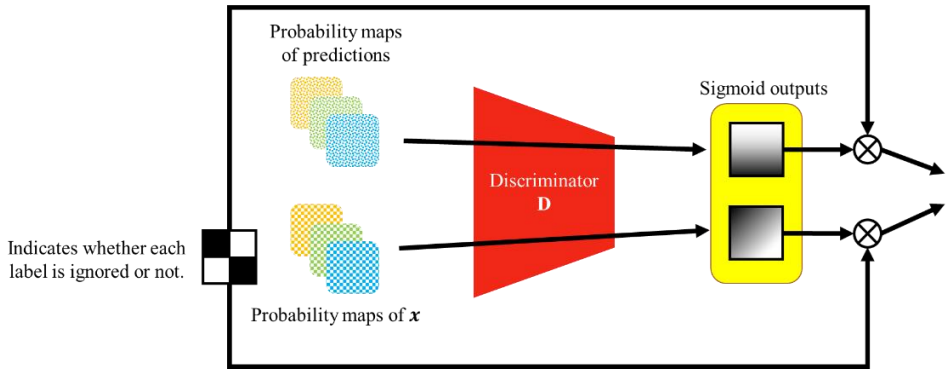

Figure 3-7. The illustrations for sigmoid calculations with the location information. Through this process, we can ignore unnecessary locations by multiplying the location map (binary values) into sigmoid outputs.

We use different losses for the generator (segmentation network) and the discriminator as shown in **Table *3-1***. As we use batch normalization layers, the discriminator converges faster for given probability maps. Moreover, DCGAN losses are prone to be saturated, causing that the discriminator cannot be trained properly and the generator loss $\mathcal{L}_G$ does not penalize the segmentation model due to the constant loss value. For resolving these issues, we train the discriminator with a very small learning rate (e.g. 10^{-7}) and use LSGAN losses without modifying the structures, like the original LSGAN [86] added additional layers for classification tasks.

Table 3-1. Loss comparison between DCGAN and LSGAN.

	$\mathcal{L}_G$	$\mathcal{L}_D$
DCGAN	$\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}\left[\log\left(D\left(G(\boldsymbol{z})\right)\right)\right]$	$\mathbb{E}_{\boldsymbol{x}\sim p_{data}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}\left[\log\left(1-D\left(G(\boldsymbol{z})\right)\right)\right]$
LSGAN	$\frac{1}{2}\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}\left[\left(D\left(G(\boldsymbol{z})\right)-1\right)^2\right]$	$\frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{data}(\boldsymbol{x})}[(D(\boldsymbol{x})-1)^2] + \frac{1}{2}\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}\left[\left(D\left(G(\boldsymbol{z})\right)^2\right)\right]$

In **Table *3-1***, $D(G(\boldsymbol{z}))$ is corresponding to the discriminator output using the segmentation network probability maps, whereas $D(\boldsymbol{x})$ denotes the discriminator output from the groundtruth probability maps. Based on this notations, LSGAN losses in **Figure 3-7** can be calculated as the form of average values which are similar to Least-Square loss calculations (Equation **2-2**), resulting in the dimension reduction of discriminator outputs.

3.2 Temperature as a Regularizer

According to [7], soft targets are generated from temperature $T = 2$ experimented with other temperature values and original softmax outputs ($T = 1$) are not discarded for the cross-entropy calculation. Inspired by this, we add an additional softmax layer for generating another probability map with different temperature value other than 1 ($T < 1$).

ALGORITHM 1: TRAINING PROCEDURE WITH TEMPERATURES

Initialize the segmentation network G and the discriminator D.

for $e = 0,1, \dots, num\_epochs - 1$:

  for $i = 0,1, \dots, num\_minibatches - 1$:

    Input: $\boldsymbol{x}_i \in \mathbb{R}^{N \times H \times W \times C}$, $\boldsymbol{y}_i = \mathbb{R}^{N \times H \times W}$

    Freeze G

      $\mathcal{L}_G \leftarrow \frac{1}{2}\mathbb{E}\left[\left(D\left(G(\boldsymbol{x}_i)_{T_g}\right) - 1\right)^2\right]$

      $\mathcal{L}_D \leftarrow \frac{1}{2}\mathbb{E}[(D(\boldsymbol{y}_i) - 1)^2] + \frac{1}{2}\,\mathbb{E}\left[\left(D\left(G(\boldsymbol{x}_i)_{T_d}\right)^2\right)\right]$

      Update D with $\mathcal{L}_D$.

    end

    Unfreeze G

      $\mathcal{L}_{seg} \leftarrow -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C}\boldsymbol{y}_{i,onehot,c}\log(G(\boldsymbol{x}_i)_{T_G})_c$

      $\mathcal{L}_{adv} \leftarrow \mathcal{L}_{seg} + \alpha \cdot \mathcal{L}_G + \frac{\lambda}{2}\sum_i w_i^2$

      Update G with $\mathcal{L}_{adv}$.

    end

  end

end

ALGORITHM 1 describes our training procedures with the given parameters and datasets. Because we mimic a general GAN structure by modifying inputs from noises $\boldsymbol{z}$ and input data $\boldsymbol{x}$

to $\boldsymbol{x}_i$ and $\boldsymbol{y}_i$, respectively. Therefore, we modify $\mathcal{L}_G$ and $\mathcal{L}_D$ equations given in **Figure *2-7*** with temperatures T_G and T_D.

We freeze the generator for training the discriminator with the discriminator loss $\mathcal{L}_D$ from outputs $D(\boldsymbol{y}_i)$ and $D(G(\boldsymbol{x}_i))$, resulting in average values as losses among probability maps. We pass $\mathcal{L}_D$ and the loss from the frozen generator ($\mathcal{L}_G$) to the discriminator and the segmentation model, respectively.

After the discriminator steps, we unfreeze the generator and train it by backpropagating the adversarial loss $\mathcal{L}_{adv}$ which is the combination of the segmentation loss $\mathcal{L}_{seg}$, the generator loss $\mathcal{L}_g$ with parameter α, and L2 weight decay with parameter λ. When we calculate multi-class cross-entropy loss $\mathcal{L}_{seg}$ for the given number of classes C, we convert $\boldsymbol{y}_i$ to one-hot encoded version $\boldsymbol{y}_{i,onehot,c}$ where c denotes the class $c = [1, C]$. Also, $G(\boldsymbol{x}_i)_{T_G}$ indicates the output probability maps from a given minibatch input $\boldsymbol{x}_i$ and the temperature T_G.

α and λ can be interpreted as additional parameters for penalizing the segmentation model (generator) on top of the segmentation loss $\mathcal{L}_{seg}$. We expect that $\mathcal{L}_G$ can also work as a regularizer like L2 weight decay. Therefore, we test with and without L2 decay parameters given from the original papers, with the same environments such as data augmentation, weight initialization, and base structures.

4

Evaluation

4.1 Experiment Details

To validate our framework, we choose the segmentation networks as FCN-8s-VGG16 [68] and ResNet-50 [14] among several backbone versions of each model. We mention that DCGAN [84] discriminator is prone to saturate in training (Section 3.1.2), however it is widely used in various GAN models. Therefore, we substitute the objective functions from those of DCGAN to LSGAN loss functions. For the faster convergence of networks, we initialize the weight of segmentation networks as the corresponding ImageNet [12] pretrained weights for each network in the training

with VOC2012 augmentation dataset [76] [77] and CityScapes dataset [79] and use BN layers in the discriminator (**Figure *3-6***).

We find the optimal segmentation learning rate $lr = 5 \times 10^{-3}$ among the values $\{10^{-3},\ 2 \times 10^{-3},\ 5 \times 10^{-3},\ 10^{-2}\}$ and apply Cosine learning rate with Warmup [110] epoch 5 and Mixed Precision in Tensorflow. Mixed Precision by training models in 16-bit Floating-Point (FP16) enables larger batch size and higher resolution than normal 32-bit Floating-Point (FP32) training [111]. We set lower learning rates $lr_d = \{10^{-7},\ 5 \times 10^{-8}\}$ from the values $\{2 \times 10^{-8},\ 5 \times 10^{-8},\ 10^{-7},\ 2 \times 10^{-7}\}$ to train the discriminator for preventing the mode collapse from BN layers, whereas learning rates of discriminators from several works [84] [98] [102] are normally in range of $[10^{-4},\ 10^{-3}]$. The same environment of Cosine Learning rate with Warmup epochs in the segmentation side is also applied in the discriminator training. For penalizing the segmentation network via the generator loss $\mathcal{L}_G$, we multiply the loss weight α into it and find the optimal value from the value pool $\{0.01,\ 0.02,\ 0.05,\ 0.1,\ 0.2\}$. With the parameters α and λ (L2 regularization parameter), we update the generator via ALGORITHM 1.

We use SGD and Adam optimizers in the segmentation network and the discriminator network, respectively. Due to the limit of 8GB VRAM from one NVIDIA RTX 2070 Super GPU, we set the image crop size 416×416 which is selected as one of multipliers of 32 for the bilinear interpolation after the softmax layer in ResNet-50. Image augmentation methods used in our experiments are RGB mean value subtraction, image zoom, rotation, random flip, and random crop, resulting in the better performance of BatchNorm layers with batch size 16 [35]. In summary, **Table *4-1*** indicates our training parameter configurations in common. The segmentation results

are evaluated through pixel accuracy (pAcc) and mean IoU (mIoU) metrics described in **Figure *3-5***.

Table 4-1. Global training parameters for segmentation and discriminator networks.

Data augmentation parameters	
Random crop size	416×416 (Batch size 16)
Image zoom ratio	$[0.5, 2.0]$
Image rotation degree	$[-10.0, 10.0]$
Hyperparameters	
Segmentation learning rate lr	5×10^{-3}
Discriminator learning rate lr_d	10^{-7} (FCN-8s-VGG16) 5×10^{-8} (ResNet-50)
SGD parameter (Momentum)	0.9
Adam parameter (β_1, β_2)	(0.5, 0.999)
BatchNorm momentum	0.9
Loss weight α	0.1 (FCN-8s-VGG16 with CityScapes dataset) 0.05 (Other experiments)

4.2 Datasets

4.2.1 CityScapes

CityScapes dataset [79] consists of high-resolution (1024×2048) images for semantic segmentation tasks with 2,975 training images and 500 validation images with 19 classes. We apply random crop which cuts images as 416×416 in arbitrary areas. With this dataset, we set the total epochs 200 for convergence. In validation, we optimize the parameters with the validation set in original resolutions of each image. Due to the limited access of annotated labels of test folds, we evaluate the test set without the annotated labels.

4.2.2 VOC2012

VOC2012 dataset [76] is widely used in training networks for computer vision tasks. For semantic segmentation, we use the segmentation image sets which include 1,464 training images and 1,449 validation images with 21 classes. Beyond the original VOC2012 dataset, VOC2012 augmented dataset [77] provides more images than the original dataset. We choose VOC2012 augmented dataset for running models with larger number of images. In the case of semantic segmentation, the dataset includes 10,582 training images with 21 classes. We set the total epochs 50 for VOC2012 augmented dataset. In validation, we optimize parameters with applying the input size (416×416) in the validation set. Because annotated labels of test folds are not public, we evaluate the test set without the annotated labels.

4.3 Results

4.3.1 FCN-8s-VGG16

As we mentioned in Section 3.1.1, FCN has 3 versions: FCN-32s, FCN-16s, and FCN-8s. In the original FCN paper [13], FCN-8s training procedure consists of three steps: Training FCN-32s like a typical semantic segmentation model, adding *pool4* maps to upsampled FCN-32s prediction maps and fine-tuning the network with additional stages, and adding pool3 maps to upsampled FCN-16s prediction maps and fine-tuning the FCN-8s network. Thus, training FCN-8s needs a longer training time for obtaining a high performance in semantic segmentation. Therefore, we follow the 'at-once' version from the journal version of FCN paper [112], which enables the one-stage training like FCN-32s by changing addition terms from *pool4* and *pool3* to 0.01**pool4* and 0.0001**pool3* when adding them to upsampled FCN-32s and FCN-16s layers, respectively. We set the discriminator learning rate $lr_d = 10^{-7}$ for both datasets.

Table 4-2. The L2 regularization parameter test in FCN-8s-VGG16. We call gray cells as the baseline because the parameter is given from the original paper *[13]*. Each case result denotes a pixel accuracy and a mean IoU (pAcc/mIoU).

	$\lambda = 0$	$\lambda = 10^{-4}$	$\lambda = 2 \times 10^{-4}$	$\lambda = 5 \times 10^{-4}$
VOC2012	**92.82/69.26**	92.72/69.02	92.78/68.94	92.51/68.14
CityScapes	**94.31/67.03**	94.30/66.94	94.31/66.88	94.10/65.52

Before applying temperatures with the discriminator, we find the optimal L2 regularization parameter λ for temperature usages on top of eligible regularization methods. In [13], $\lambda = 5 \times 10^{-4}$ is given, but we test FCN-8s with $\lambda = \{0, 10^{-4}, 2 \times 10^{-4}, 5 \times 10^{-4}\}$ in VOC2012

validation sets and CityScapes validation sets. **Table *4-2*** shows our test results on both datasets. In both datasets, we obtain the best results without L2 regularization in terms of pixel accuracies and mIoUs, increasing over 1 percentage point in mIoU.

On top of the best result from **Table *4-2***, we apply temperatures on both the segmentation network (T_g) and the discriminator (T_d) with $\alpha = 0.05$ for VOC2012 validation sets and $\alpha = 0.1$ for CityScapes validation sets in ALGORITHM 1. For providing the original probability information on either side, T_g is fixed in testing $T_d = \{0.5, 1, 2\}$ and vice versa.

Table 4-3. The performance of T_d tests with fixed $T_g = 1$ in FCN-8s-VGG16 and the discriminator. Each case result denotes a pixel accuracy and a mean IoU (pAcc/mIoU).

	Baseline	Best L2	Adversarial	$T_d = 0.5$	$T_d = 2$
VOC2012	92.51/68.14	92.82/69.26	**92.83**/69.33	92.79/69.10	92.78/**69.36**
CityScapes	94.10/65.52	94.31/67.03	**94.32**/67.30	94.29/**67.36**	94.30/67.10

Table 4-4. The performance of T_g tests with fixed $T_d = 1$ in FCN-8s-VGG16 and the discriminator. Each case result denotes a pixel accuracy and a mean IoU (pAcc/mIoU).

	Baseline	Best L2	Adversarial	$T_g = 0.5$	$T_g = 2$
VOC2012	92.51/68.14	92.82/69.26	92.83/69.33	**92.90**/**69.82**	92.72/68.96
CityScapes	94.10/65.52	94.31/67.03	94.32/67.30	**94.36**/**67.42**	94.28/67.38

Table 4-3 and **Table 4-4** denote our temperature experiment results on both datasets. In both tables, 'Best L2' denotes the best performance without the discriminator (**Table 4-2**) and 'Adversarial' denotes the pixel accuracy and mIoU results in the temperature configuration $T_g =$

$T_d = 1$. We can observe that the $T_g = 0.5$ and $T_d = 1$ configuration from **Table 4-4** gives the best results among our settings, showing that the mIoU improvements are a little bit lower than 2 percent points higher than those from the Baseline configuration. (**Figure *4-1*** and **Figure *4-2***) Also, even though we cannot quantitively evaluate our framework with test sets because the correct labels of those in each dataset are not given, our best configuration generates label predictions with more details. (**Figure *4-3*** and **Figure *4-4***)

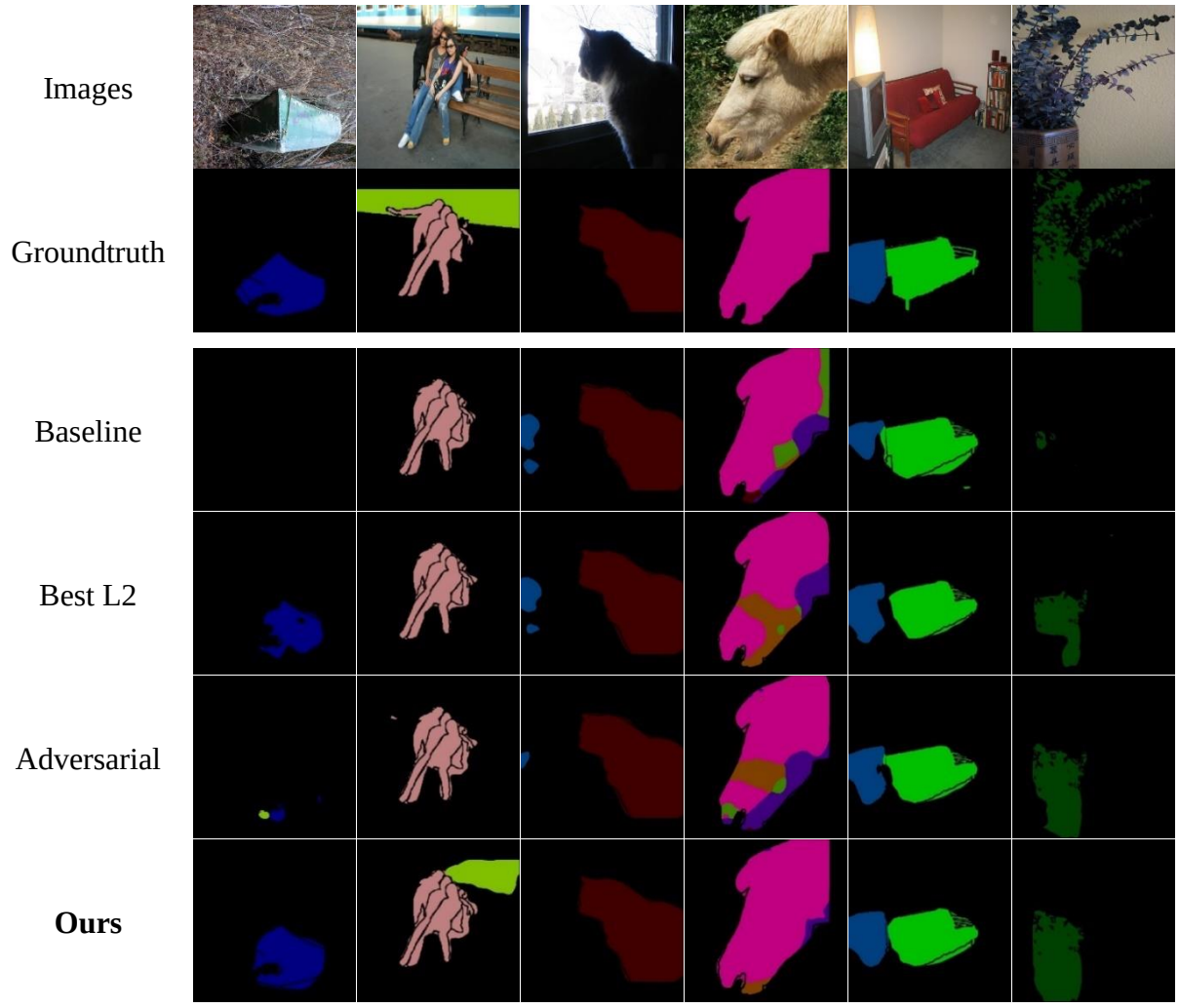

Figure 4-1. The segmentation results on FCN-VGG16 with VOC2012 validation sets. The first and the second rows indicate the images and the groundtruth annotations, respectively. From the third column, the segmentation results are from our model configurations: Baseline (Parameter $\lambda = 5 \times 10^{-4}$), Best L2 (No L2 Regularization), Adversarial, and Ours ($T_g = 0.5$ *and* $T_d = 1$).

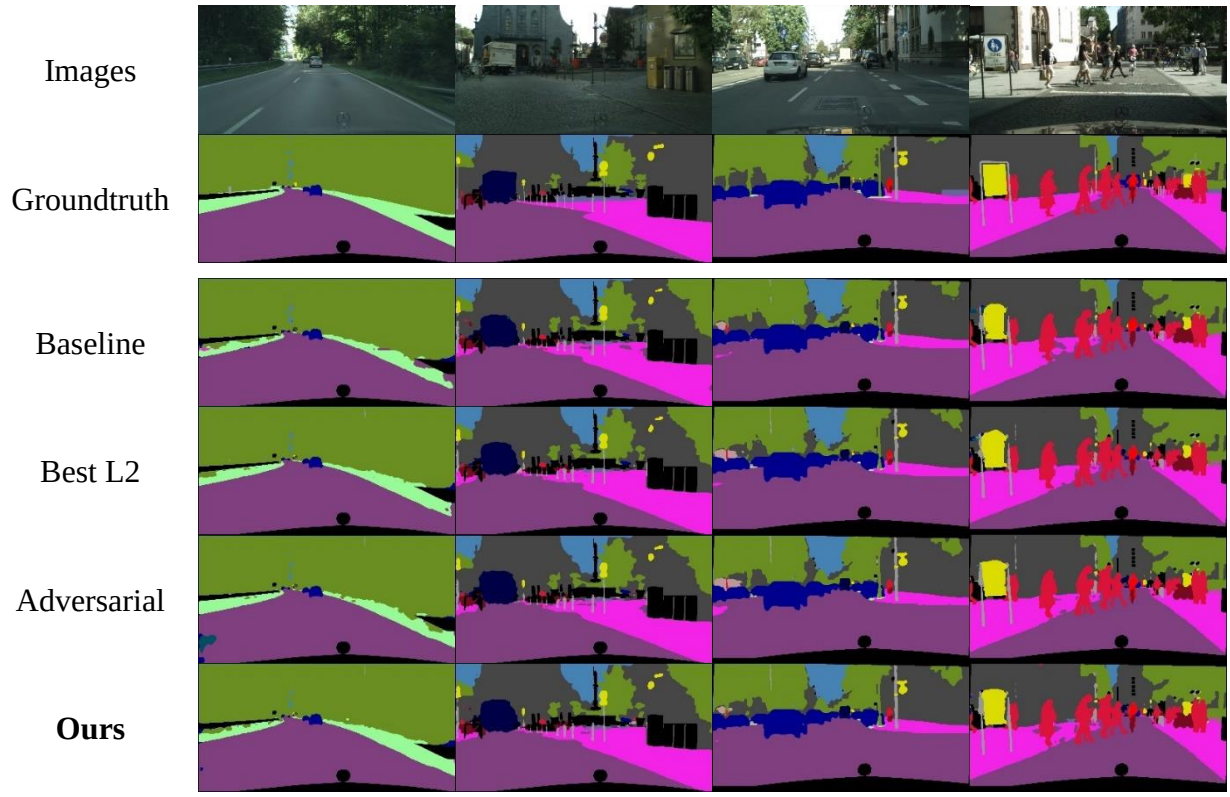

Figure 4-2. The segmentation results on FCN-VGG16 with CityScapes validation sets. The first and the second rows indicate the images and the groundtruth annotations, respectively. The segmentation results are printed from the third row to the last row. Annotations from configurations in four rows: Baseline (Parameter $\lambda = 5 \times 10^{-4}$), Best L2 (No L2 Regularization), Adversarial, and Ours ($T_g = 0.5$ and $T_d = 1$).

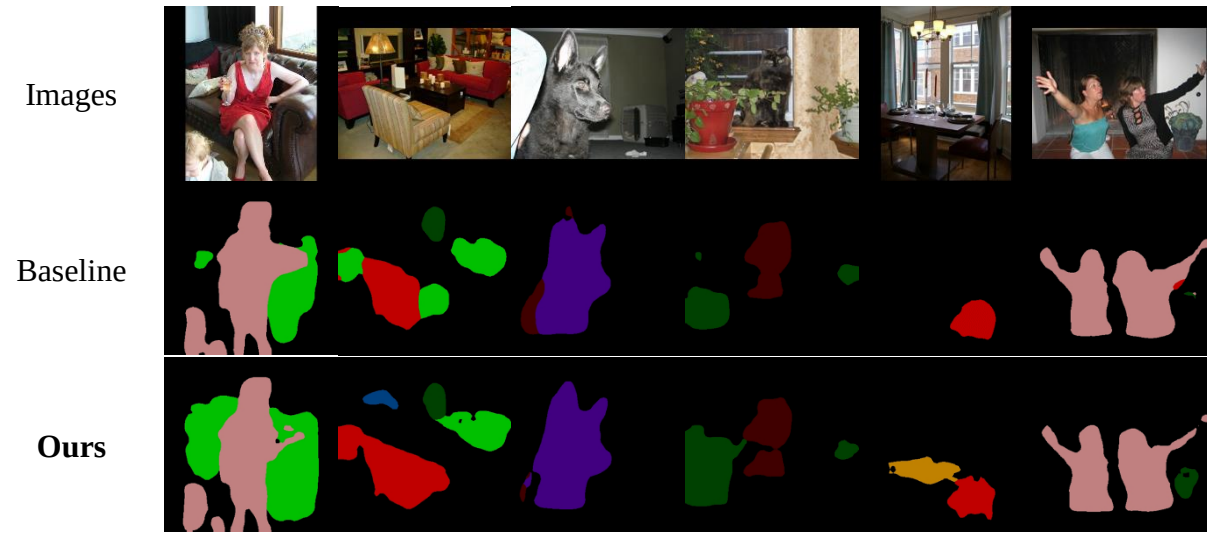

Figure 4-3. The segmentation results on FCN-VGG16 with VOC2012 test sets. From the image in the first row, we can observe that our method show better estimations in each image than those from the baseline configuration given in [13].

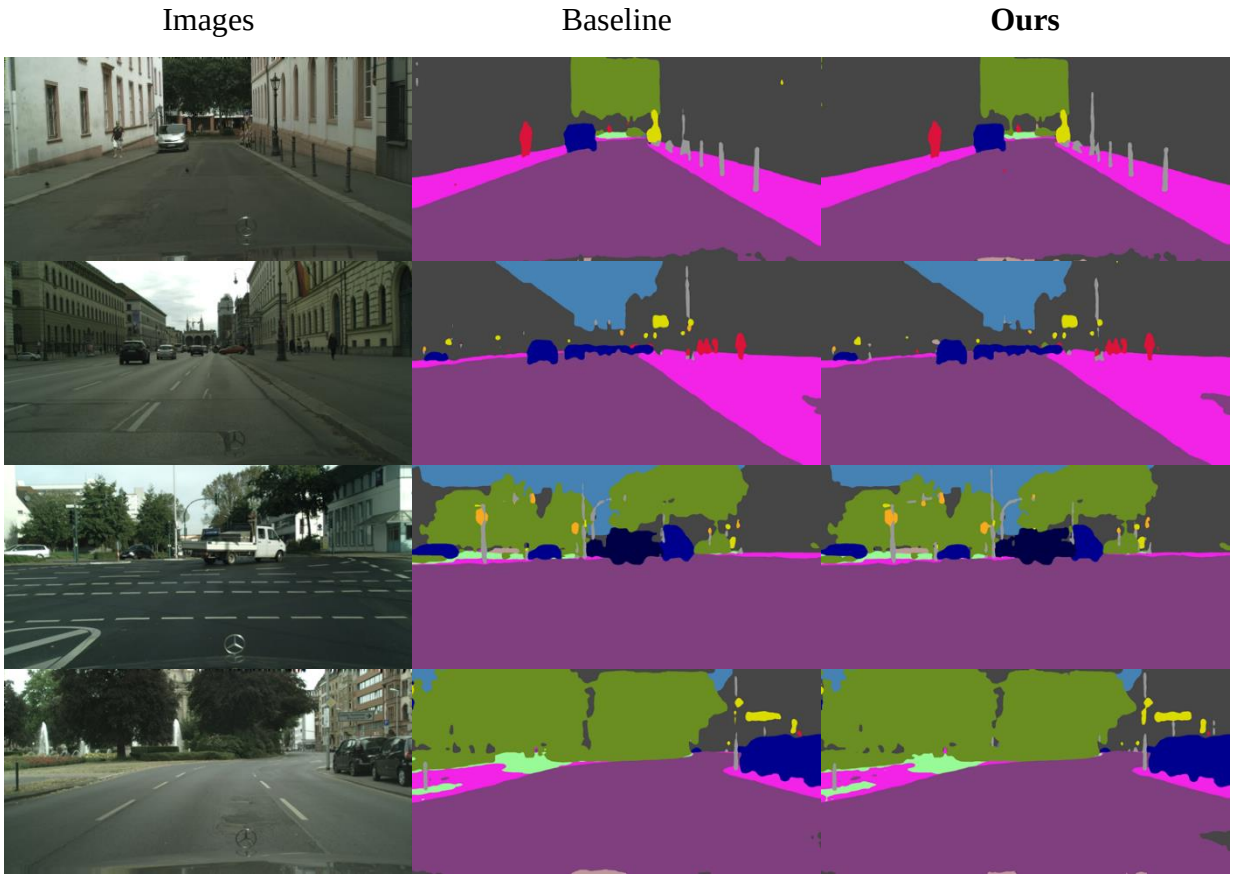

Figure 4-4. The segmentation results on FCN-VGG16 with CityScapes test sets. From the image in the first column, labels from our method in the third column are more detailed than ones (the second column) which are from the baseline configuration given in [13].

4.3.2 ResNet-50

ResNet [14] has several versions which have the number of convolutional layers $\{34, 50, 101, 152\}$. Among them, we select *ResNet-50* model as the segmentation network for the balance between the performance and the limit of GPU memory. For the better performance, Dilated Residual Networks (DRN) [113] can be considered, which converts a ResNet to a dilated network by using convolutional layers with a dilation rate, resulting that it preserves the receptive field. Thus, higher resolutions due to dilated convolutions increase the memory consumption in the output of each convolutional layer. As a result, we do not use dilated convolutional layers in ResNet-50 model due to the GPU memory limitation. In ResNet-50 experiments, we set the discriminator learning rate $lr_d = 5 \times 10^{-8}$ for both datasets.

Table 4-5. The L2 regularization parameter test in ResNet-50. We call gray cells as the baseline because the parameter is given from the original paper [14]. Each case result denotes a pixel accuracy and a mean IoU (pAcc/mIoU).

	$\lambda = 0$	$\lambda = 10^{-4}$	$\lambda = 2 \times 10^{-4}$	$\lambda = 5 \times 10^{-4}$
VOC2012	90.62/**61.97**	90.60/61.88	90.59/61.71	**90.69**/61.94
CityScapes	92.50/58.57	92.47/58.57	92.74/59.44	**92.85**/**60.35**

Given in [14], $\lambda = 10^{-4}$ case is the 'baseline' in our ResNet-50 experiments with $\lambda = \{0, 10^{-4}, 2 \times 10^{-4}, 5 \times 10^{-4}\}$ like Section 4.2.1. In **Table *4-5***, we select $\lambda = 0$ as the optimal L2 regularization parameter for VOC2012 validation sets and $\lambda = 5 \times 10^{-4}$ as the optimal value for CityScapes validation sets. The performance difference is about 1.8 percent points higher mIoU

in CityScapes validation sets between the optimal and the baseline, whereas $\lambda = 0$ and $\lambda = 5 \times 10^{-4}$ improve a little, compared with the baseline ($\lambda = 10^{-4}$) in VOC2012 validation sets.

From the optimal λ values on **Table *4-5***, we calculate the losses based on temperatures on both the segmentation network (T_g) and the discriminator (T_d) with $\alpha = 0.05$ for both datasets in ALGORITHM 1. The T_g and T_d control is the same as in Section 4.2.1.

Table 4-6. The performance of T_d tests with fixed $T_g = 1$ in ResNet-50 and the discriminator. Each case result denotes a pixel accuracy and a mean IoU (pAcc/mIoU).

	Baseline	Best L2	Adversarial	$T_d = 0.5$	$T_d = 2$
VOC2012	90.60/61.88	90.62/61.97	90.68/62.03	90.68/61.86	**90.74**/**62.37**
CityScapes	92.47/58.57	**92.85**/**60.35**	92.83/60.16	92.64/59.35	92.83/60.17

Table 4-7. The performance of T_g tests with fixed $T_d = 1$ in ResNet-50 and the discriminator. Each case result denotes a pixel accuracy and a mean IoU (pAcc/mIoU).

	Baseline	Best L2	Adversarial	$T_g = 0.5$	$T_g = 2$
VOC2012	90.60/61.88	90.62/61.97	90.68/62.03	**90.88**/**62.54**	90.52/61.72
CityScapes	92.47/58.57	92.85/60.35	92.83/60.16	**93.02**/**60.75**	92.70/60.22

Table *4-6* and **Table *4-7*** show experiment results of temperatures applied on both datasets. In both tables, 'Best L2' comes from the optimal λ (**Table *4-5***) that $\lambda = 0$ and $\lambda = 5 \times 10^{-4}$ are for VOC2012 validation sets and CityScapes validation sets, respectively. Also, 'Adversarial' results are calculated from the probability maps where $T_g = T_d = 1$.

In **Table *4-6***, the best results are from different configurations on each dataset - $T_g = 1$ and $T_d = 2$ for VOC2012 and $\lambda = 5 \times 10^{-4}$ without temperatures and the discriminator for CityScapes validation sets. However, in **Table *4-7***, $T_g = 0.5$ and $T_d = 1$ configuration results in the best results in both datasets. Although mIoU improves a little bit (about 0.7 percent point) on VOC2012 validation sets whereas over 2 percent points are increased in the mIoU of CityScapes validation sets, this configuration improves the ResNet-50 network. (**Figure *4-5*** and **Figure *4-6***) Due to the policy that annotations of test sets in each dataset are private, we cannot quantitively evaluate our framework with test sets. However, we perform segmentation tasks with trained segmentation models and observe that our best configuration generates label predictions with more details. (**Figure *4-7*** and **Figure *4-8***)

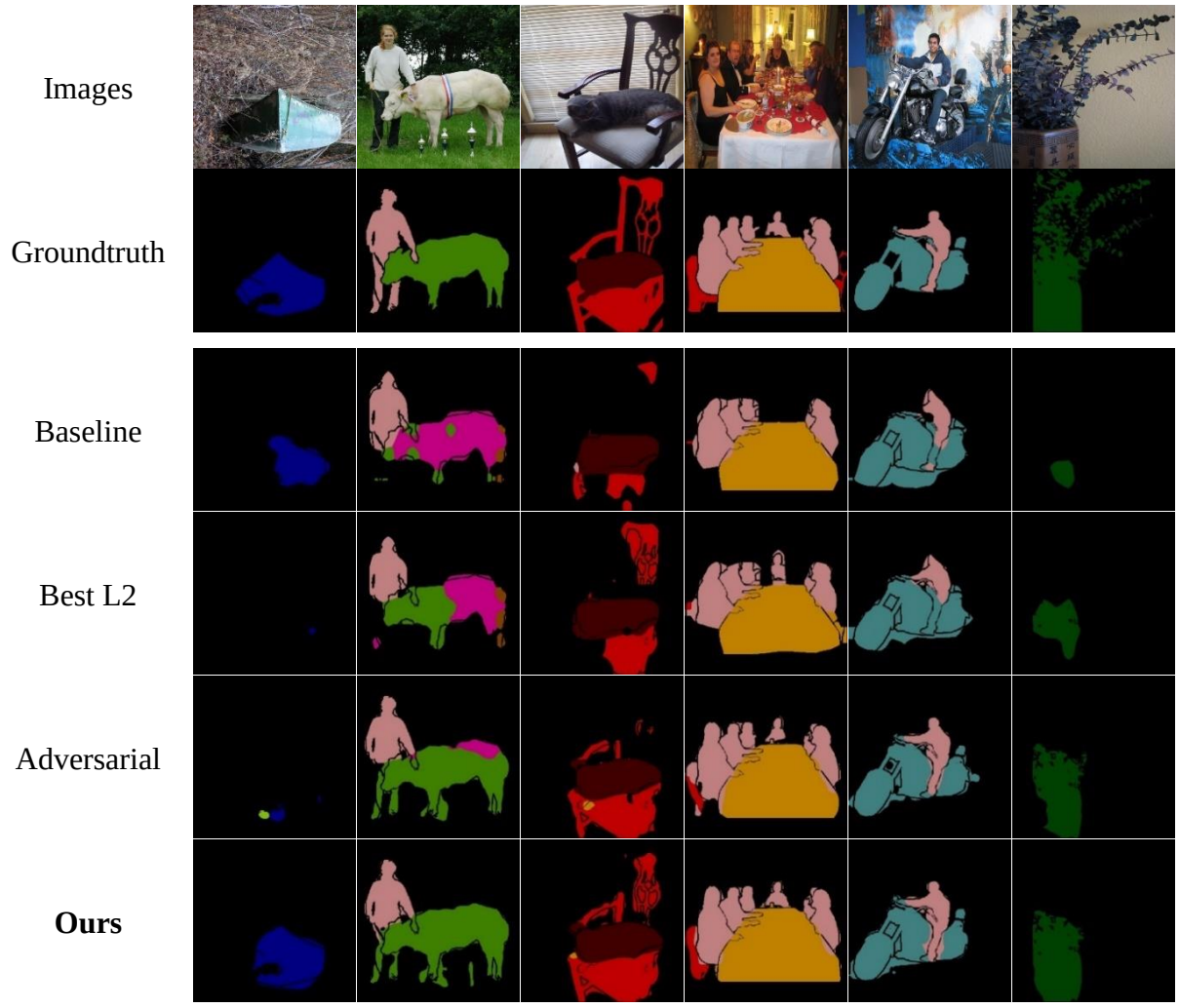

Figure 4-5. The segmentation results on ResNet-50 with VOC2012 validation sets. Row 1, 2 for images and ground annotations. Column 3-6 for results from different model configurations: Baseline (Parameter $\lambda = 10^{-4}$), Best L2 (No L2 Regularization), Adversarial, and Ours ($T_g = 0.5$ and $T_d = 1$).

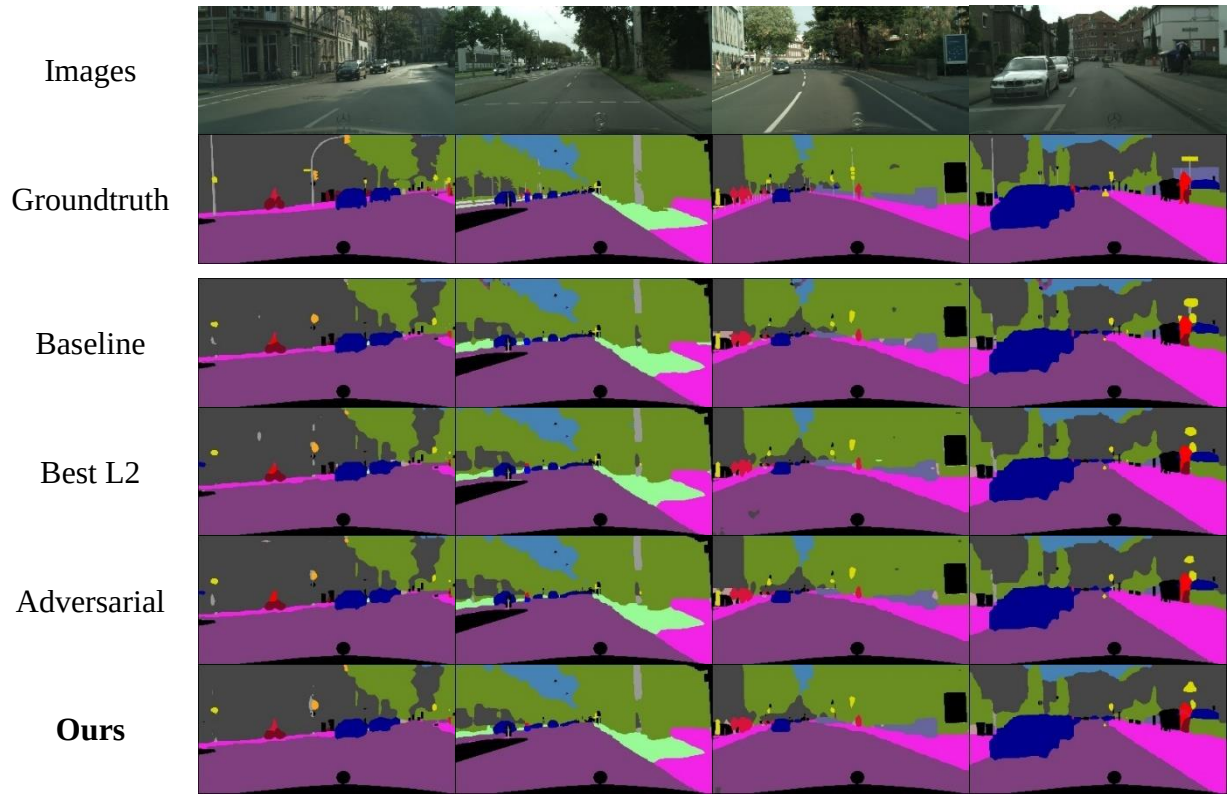

Figure 4-6. The segmentation results on ResNet-50 with CityScapes validation sets. Row 1, 2 for images and ground annotations. Row 3-6 for results from different model configurations: Baseline (Parameter $\lambda = 10^{-4}$), Best L2 ($\lambda = 5 \times 10^{-4}$), Adversarial, and Ours ($T_g = 0.5$ and $T_d = 1$).

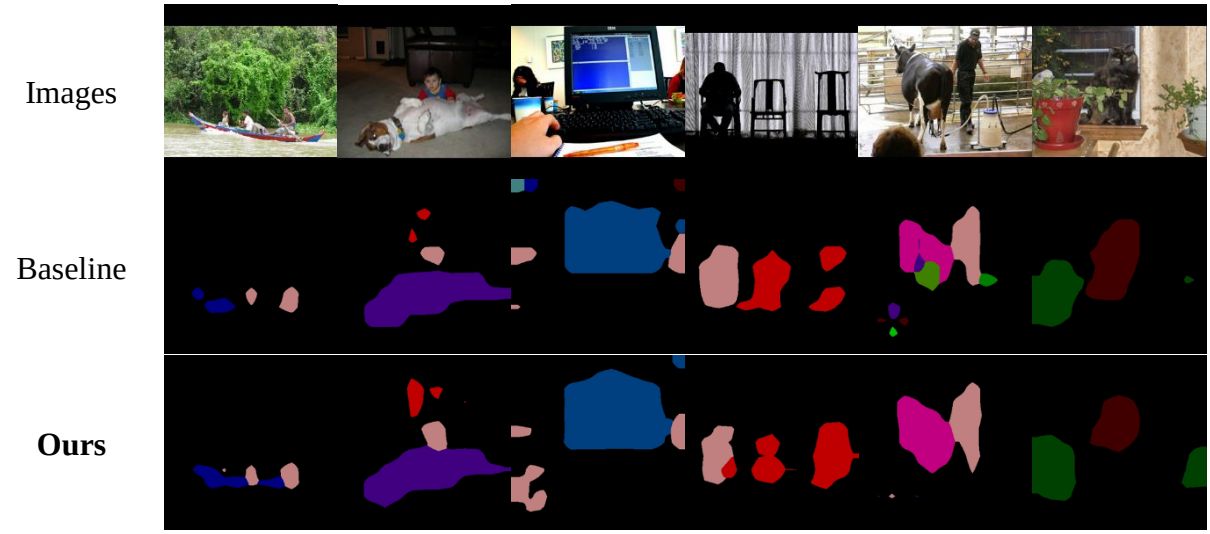

Figure 4-7. The segmentation results on ResNet-50 with VOC2012 test sets. From the image in the first row, we can observe that our method show better estimations in each image than those from the baseline configuration given in [14].

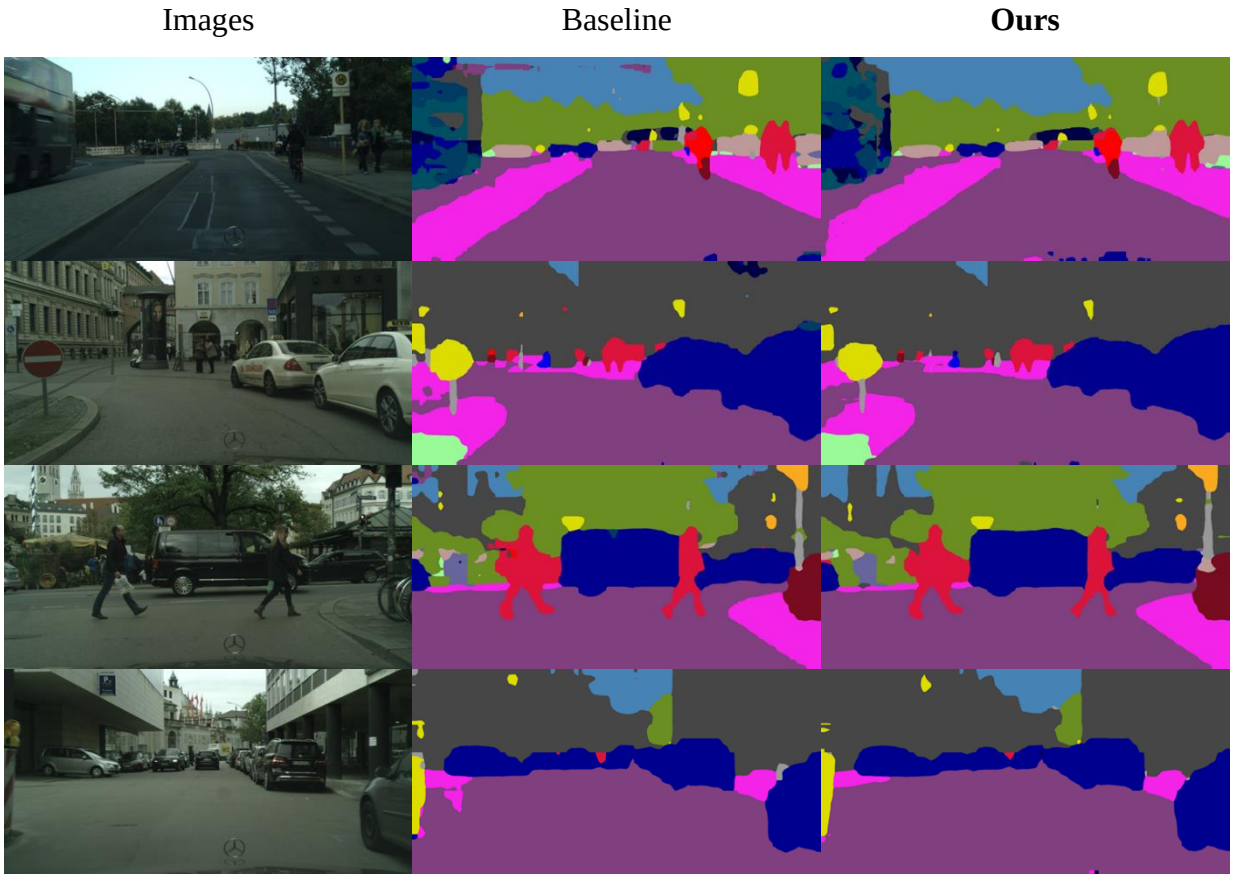

Figure 4-8. The segmentation results on ResNet-50 with CityScapes test sets. From the image in the first column, labels from our method in the third column are more detailed than ones (the second column) which are from the baseline configuration given in [14].

4.3.3 Time Consumption

Table 4-8. The comparison of the number of parameters in networks and time consumption. A segmentation model takes T_{train} in training each epoch and T_{adv} is the training time consumption per epoch on adversarial framework. Our framework needs more time due to the usage of the discriminator.

Segmentation Network	Dataset	# of seg params	# of D params	T_{train}	T_{adv}
FCN-8s-VGG16	VOC2012	134.5M	2.79M	6.8mins	9.2mins
	CityScapes	134.4M	2.78M	3.5mins	4.5mins
ResNet-50	VOC2012	23.6M	2.79M	2.6mins	4.1mins
	CityScapes	23.6M	2.78M	2.6mins	3.0mins

Table *4-8* shows the number of parameters in segmentation and discriminator networks and training time consumptions. We can understand that the penalization by an adversarial loss enables a segmentation network to be improved even though the concatenated discriminator is not big enough to be compared with the main network capacity. When we observe average time consumption per epoch, the FCN-8s-VGG16 with VOC2012 experiment shows 2.4 minutes time increase from T_{train} to T_{adv} whereas others have [0.5, 1.5] minutes increase range. This phenomenon comes from the additional memory consumption in RAM memory due to the GPU memory exhaustion, expecting this to be mitigated when using the larger GPU memory.

4.4 Summary and Discussion

In Section 4.2.1 and Section 4.2.2, we empirically show that the temperature configuration of $T_g = 0.5$ and $T_d = 1$ mitigates the overfitting phenomena by penalizing the segmentation network with the adversarial loss $\mathcal{L}_{adv}$ on top of the segmentation loss $\mathcal{L}_{seg}$ and other regularization methods such as data augmentation and L2 weight decay. **Table *4-9*** shows the summary of improvements in validation sets with using the discriminator and temperatures on both networks.

Table 4-9. Summary of validation results from our framework. Pixel accuracy and mean IoU (pAcc/mIoU) are the metrics for the performance evaluation.

Segmentation Network	Dataset	Baseline	Our Method
FCN-8s-VGG16	VOC2012	92.51/68.14	92.90/69.82
	CityScapes	94.10/65.52	94.36/67.42
ResNet-50	VOC2012	90.60/61.88	90.88/62.54
	CityScapes	92.47/58.57	93.02/60.75

Our method improves the segmentation performance in FCN-8s-VGG16 and ResNet-50 networks with validation/test sets of VOC2012 and CityScapes. In validation sets in each dataset, we gradually improve the performance via modifying L2 regularization parameters, concatenating the DCGAN discriminator, and searching the optimal temperature configuration on probability maps. Despite the absence of the groundtruth labels in test sets of each dataset, we observe that label predictions are more accurate and detailed in not only validation sets but also test sets. As we

mention in **Table *4-9***, the lightweight discriminator with the small number of parameters results in increased training time but higher accuracies in test time.

We have done experiments only on the task of semantic segmentation. However, we believe that our approach with temperature as a regularizer can be used in wider applications such as classification, pose estimation, and other vision or other deep learning tasks which use probability maps generated from machine learning models. By doing so, we can expect to diminish the overfitting phenomena in those areas and provide higher performance.

5

Conclusion

5.1 Conclusion

We proposed a novel method to decrease overfitting. First, it is by adding temperatures on the softmax layers, providing one more freedom to manipulate probability maps for loss calculation. Second, it is by concatenating an extra discriminator with two probability maps with manageable temperatures to the main network. The task of the discriminator focuses on preventing overfitting by examining a loss from the difference between the probability map with temperature T_d and the groundtruth map. An adversarial loss is calculated from the probability map with temperature T_g by passing through the discriminator to the main segmentation network for penalization.

For the framework evaluation, we selected the segmentation (generator) model as FCN-8s-VGG16 and ResNet-50 with the consideration of memory. DCGAN discriminator calculates the discriminator loss and the adversarial loss, both based on LSGAN losses, with temperature T_d and T_g, respectively. We tested with temperatures $T = \{0.5, 1, 2\}$ in either side (T_g or T_d) for searching the optimal temperature configuration.

From our experiment results in **Chapter 4**, using only the generative temperature $T_g = 0.5$ to emphasizes the dominant class in the probability distributions which are used in the segmentation loss and the adversarial loss yield the best performance in validation sets in two datasets with the default temperature $T_d = 1$ for the discriminator loss. Although the annotation information of test sets in each dataset are not given, we find that our framework mitigates overfitting properly in test sets. It is supported by label predictions from our optimal configurations are more detailed than those from baseline configurations of segmentation models. Moreover, the training time surge and the increased number of parameters are not large enough to burden the computation.

5.2 Future Works

The purpose of our work is to attenuate the overfitting in semantic segmentation by introducing temperatures on two softmax probability maps. For implementing our framework on top of regularization methods, we examined a L2 regularization parameter to find the optimal performance without changing the main model structure and image augmentation methods. For this reason, we compared cases with and without L2 regularization and observed that it does not work properly in given models. Therefore, we set the starting point of the L2 regularization

parameter $\lambda = 0$ in the most of our experimental cases. Our adversarial framework compensates the lack of the regularization effect with the better performance than the L2 regularization method. However, there are some limitations in our framework.

Even though selected models for semantic segmentation in our framework are widely used, those are not the state-of-the-art like DeepLab models [67] [74] [75] or lightweight models such as MobileNet [17] [18] [19] and EfficientNet [20] [21]. We need to evaluate with those models in the future. In the discriminator side, only DCGAN and LSGAN components are considered due to the loss stability of LSGAN, whereas there are abundant ways to overcome GAN disadvantages such as weight clipping (WGAN [87]), penalizing the gradient (WGAN-GP [89]), and modifying structures. Also, we can use additional temperature parameters with multiple softmax targets or search other pooling methods in the sigmoid outputs with temperatures. Our future work should be the improved version based on those ideas and we should investigate further for more efficient feature-based regularization methods.

References

[1] A. Y. Ng, "Feature Selection, L1 vs. L2 Regularization, and Rotational Invariance," in *Proceedings of the twenty-first International Conference on Machine Learning (ICML)*, 2004.

[2] R. Müller, S. Kornblith and G. Hinton, "When Does Label Smoothing Help?," in *33rd Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

[3] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *The Journal of Machine Learning Research*, 2014.

[4] L. Wan, M. Zeiler, S. Zhang, Y. Le Cun and R. Fergus, "Regularization of Neural Networks using DropConnect," in *International Conference on Machine Learning (ICML)*, 2013.

[5] X. Gastaldi, "Shake-Shake regularization," in *arXiv preprint arXiv:1705.07485*, 2017.

[6] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville and Y. Bengio, "Generative Adversarial Networks," in *arXiv preprint arXiv:1406.2661*, 2014.

[7] G. Hinton, O. Vinyals and J. Dean, "Distilling the Knowledge in a Neural Network," in *arXiv preprint arXiv:1503.02531*, 2015.

[8] D. E. Rumelhart, G. E. Hinton and R. J. Williams, "Learning Internal Representations by Error Propagation," Institute of Cognitive Science, University of California, San Diego, 1985.

[9] D. E. Rumelhart, G. E. Hinton and R. J. Williams, "Learning Representations by Back-propagating Errors," *Nature*, 1986.

[10] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based Learning Applied to Document Recognition," *Proceedings of the IEEE*, 1998.

[11] A. Krizhevsky, I. Sutskever and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2012.

[12] J. Deng, W. Dong, R. Socher, L. J. Li, K. Li and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *2009 IEEE conference on computer vision and pattern recognition*, 2009.

[13] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-scale Image Recognition," *arXiv preprint arXiv*, 2014.

[14] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[15] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, "Going Deeper with Convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[16] G. Huang, Z. Liu, L. Van Der Maaten and K. Q. Weinberger, "Densely Connected Convolutional Networks," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[17] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto and H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," in *arXiv preprint arXiv:1704.04861*, 2017.

[18] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov and L. C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[19] A. Howard, M. Sandler, G. Chu, L. C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le and H. Adam, "Searching for MobileNetV3," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

[20] M. Tan and Q. V. Le, "Efficientnet: Rethinking Model Scaling for Convolutional Neural Networks," in *International Conference on Machine Learning (ICML)*, 2019.

[21] M. Tan and Q. V. Le, "EfficientNetV2: Smaller Models and Faster Training," in *arXiv preprint arXiv:2104.00298*, 2021.

[22] S. Hochreiter, Y. Bengio, P. Frasconi and J. Schmidhuber, "Gradient Flow in Recurrent Nets: The Difficulty of Learning Long-Term Dependencies," 2001.

[23] V. Nair and G. E. Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines," in *Proceedings of the 27th International Conference on International Conference on Machine Learning (ICML)*, 2010.

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

[25] A. L. Maas, A. Y. Hannun and A. Y. Ng, "Rectifier Nonlinearities Improve Neural Network Acoustic Models," in *The 30th International Conference on Machine Learning (ICML)*, 2013.

[26] K. He, X. Zhang, S. Ren and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2015.

[27] J. S. Bridle, "Training Stochastic Model Recognition Algorithms as Networks can lead to Maximum Mutual Information Estimation of Parameters," in *Advances in Neural Information Processing Systems*, 1990.

[28] D. A. Clevert, T. Unterthiner and S. Hochreiter, "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)," in *arXiv preprint arXiv:1511.07289*, 2015.

[29] G. Klambauer, T. Unterthiner, A. Mayr and S. Hochreiter, "Self-Normalizing Neural Networks," in *arXiv preprint arXiv:1706.02515*, 2017.

[30] P. Ramachandran, B. Zoph and Q. V. Le, "Swish: A Self-Gated Activation Function," in *arXiv preprint arXiv:1710.05941*, 2017.

[31] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariance Shift," in *International Conference on Machine Learning (ICML)*, 2015.

[32] T. Salimans and D. P. Kingma, "Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks," in *arXiv preprint arXiv:1602.07868*, 2016.

[33] J. L. Ba, J. R. Kiros and G. E. Hinton, "Layer Normalization," in *arXiv preprint arXiv:1607.06450*, 2016.

[34] D. Ulyanov, A. Vedaldi and V. Lempitsky, "Instance Normalization: The Missing Ingredient for Fast Stylization," in *arXiv preprint arXiv:1607.08022*, 2016.

[35] Y. Wu and K. He, "Group Normalization," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

[36] L. Bottou, "Large-Scale Machine Learning with Stochastic Gradient Descent," in *Proceedings of COMPSTAT'2010*, 2010.

[37] N. Qian, "On the Momentum Term in Gradient Descent Learning Algorithms," in *Neural Networks*, 1999.

[38] I. Sutskever, J. Martens, G. Dahl and G. Hinton, "On the importance of initialization and momentum in deep learning," in *International Conference on Machine Learning (ICML)*, 2013.

[39] G. Hinton, N. Srivastava and K. Swersky, "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent," 2012.

[40] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *arXiv preprint arXiv:1412.6980*, 2014.

[41] J. Duchi, E. Hazan and Y. Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization," *Journal of Machine Learning Research*, 2011.

[42] M. D. Zeiler, "ADADELTA: AN ADAPTIVE LEARNING RATE METHOD," in *arXiv preprint arXiv:1212.5701*, 2012.

[43] T. Dozat, "Incorporating Nesterov Momentum into Adam," in *International Conference on Learning Representations (ICLR)*, 2016.

[44] P. Y. S. D. &. P. J. C. Simard, "Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis," in *International Conference on Document Analysis and Recognition (ICDAR)*, 2003.

[45] I. N. H. &. Y. K. Sato, "APAC: Augmented PAttern Classification with Neural Networks," in *arXiv preprint arXiv:1505.03229*, 2015.

[46] D. M. U. &. S. J. Ciregan, "Multi-column Deep Neural Networks for Image Classification," in *Computer Vision and Pattern Recognition (CVPR)*, 2012.

[47] R. G. Lopes, D. Yin, B. Poole, J. Gilmer and E. D. Cubuk, "Improving Robustness Without Sacrificing Accuracy with Patch Gaussian Augmentation," in *arXiv preprint arXiv:1906.02611*, 2019.

[48] G. Kang, X. Dong, L. Zheng and Y. Yang, "PatchShuffle Regularization," in *arXiv preprint arXiv:1707.07103*, 2017.

[49] H. Inoue, "Data Augmentation by Pairing Samples for Images Classification," in *arXiv preprint arXiv:1801.02929*, 2018.

[50] T. DeVries and G. W. Taylor, "Improved Regularization of Convolutional Neural Networks with Cutout," in *arXiv preprint arXiv:1708.04552*, 2017.

[51] Z. Zhong, L. Zheng, G. Kang, S. Li and Y. Yang, "Random Erasing Data Augmentation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

[52] R. Takahashi, T. Matsubara and K. Uehara, "Data Augmentation using Random Image Cropping and Patching for Deep CNNs," *IEEE Transactions on Circuits and Systems for Video Technology*, 2019.

[53] S. Yun, D. Han, S. J. Oh, S. Chun, J. Choe and Y. Yoo, "CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

[54] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan and Q. V. Le, "AutoAugment: Learning Augmentation Strategies from Data," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[55] S. Lim, I. Kim, T. Kim, C. Kim and S. Kim, "Fast AutoAugment," in *arXiv preprint arXiv:1905.00397*, 2019.

[56] X. Zhang, Q. Wang, J. Zhang and Z. Zhong, "Adversarial Autoaugment," in *International Conference on Learning Representations (ICLR)*, 2019.

[57] E. D. Cubuk, B. Zoph, J. Shlens and Q. V. Le, "Randaugment: Practical automated data augmentation with a reduced search space," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2020.

[58] L. Xie, J. Wang, Z. Wei, M. Wang and Q. Tian, "DisturbLabel: Regularizing CNN on the Loss Layer," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[59] G. Pereyra, G. Tucker, J. Chorowski, Ł. Kaiser and G. Hinton, "Regularizing neural networks by penalizing confident output distributions," in *5th International Conference on Learning Representations (ICLR)*, 2017.

[60] E. A. Smirnov, D. M. Timoshenko and S. N. Andrianov, "Comparison of Regularization Methods for ImageNet Classification with Deep Convolutional Neural Networks," in *AASRI Conference on Computational Intelligence and Bioinformatics*, 2014.

[61] M. Siam, S. Elkerdawy, M. Jagersand and S. Yogamani, "Deep Semantic Segmentation for Automated Driving: Taxonomy, Roadmap and Challenges," in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, 2017.

[62] B. Kayalibay, G. Jensen and P. van der Smagt, "CNN-based Segmentation of Medical Imaging Data," in *arXiv preprint arXiv:1701.03056*, 2017.

[63] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua and S. Süsstrunk, "SLIC Superpixels," 2010.

[64] Z. &. P. T. C. Kato, "A Markov random field image segmentation model for color textured images," *Image and Vision Computing*, 2006.

[65] Z. Liu, X. Li, P. Luo, C. C. Loy and X. Tang, "Deep Learning Markov Random Field for Semantic Segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 2017.

[66] S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, ... and P. H. Torr, "Conditional Random Fields as Recurrent Neural Networks," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2015.

[67] L. C. Chen, G. Papandreou, I. Kokkinos, K. Murphy and A. L. Yuille, "DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2017.

[68] J. Long, E. Shelhamer and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

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

[70] F. Milletari, N. Navab and S. A. Ahmadi, "V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation," in *2016 Fourth International Conference on 3D Vision (3DV)*, 2016.

[71] V. Badrinarayanan, A. Kendall and R. Cipolla, "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

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

[73] J. Wang, K. Sun, T. Cheng, B. Jiang, C. Deng, Y. Zhao, D. Liu, Y. Mu, M. Tan, X. Wang, X. Wang, W. Liu and B. Xiao, "Deep High-Resolution Representation Learning for Visual Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2020.

[74] L. C. Chen, G. Papandreou, F. Schroff and H. Adam, "Rethinking Atrous Convolution for Semantic Image Segmentation," in *arXiv preprint arXiv:1706.05587*, 2017.

[75] L. C. Chen, Y. Zhu, G. Papandreou, F. Schroff and H. Adam, "Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

[76] M. Everingham and J. Winn, "The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Development Kit," in *Pattern Analysis, Statistical Modelling and Computational Learning, Tech.*, 2012.

[77] B. Hariharan, P. Arbeláez, L. Bourdev, S. Maji and J. Malik, "Semantic Contours from Inverse Detectors," in *In 2011 International Conference on Computer Vision (ICCV)*, 2011.

[78] G. J. Brostow, J. Fauqueur and R. Cipolla, "Semantic object classes in video: A high-definition ground truth database," *Pattern Recognition Letters*, 2009.

[79] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth and B. Schiele, "The Cityscapes Dataset for Semantic Urban Scene Understanding," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[80] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso and A. Torralba, "Scene Parsing through ADE20K Dataset," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[81] T. Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, ... and C. L. Zitnick, "Microsoft COCO: Common Objects in Context," in *European Conference on Computer Vision (ECCV)*, 2014.

[82] A. Geiger, P. Lenz, C. Stiller and R. Urtasun, "Vision meets Robotics: The KITTI Dataset," *The International Journal of Robotics Research*, 2013.

[83] J. Yosinski, J. Clune, Y. Bengio and H. Lipson, "How transferable are features in deep neural networks?," arXiv preprint arXiv:1411.1792, 2014.

[84] A. Radford, L. Metz and S. Chintala, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks," in *arXiv preprint arXiv:1511.06434*, 2015.

[85] M. D. Zeiler, G. W. Taylor and R. Fergus, "Adaptive Deconvolutional Networks for Mid and High Level Feature Learning," in *2011 International Conference on Computer Vision (ICCV)*, 2011.

[86] X. Mao, Q. Li, H. Xie, R. Y. Lau, Z. Wang and S. Paul Smolley, "Least Squares Generative Adversarial Networks," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017.

[87] M. Arjovsky, S. Chintala and L. Bottou, "Wasserstein Generative Adversarial Networks," in *International Conference on Machine Learning (ICML)*, 2017.

[88] L. P. B. P. D. &. S.-D. J. Metz, "Unrolled Generative Adversarial Networks," in *International Conference on Learning Representations*, 2017.

[89] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin and A. Courville, "Improved Training of Wasserstein GANs," in *arXiv preprint arXiv:1704.00028*, 2017.

[90] D. Berthelot, T. Schumm and L. Metz, "BEGAN: Boundary Equilibrium Generative Adversarial Networks," in *arXiv preprint arXiv:1703.10717*, 2017.

[91] H. Zhang, I. Goodfellow, D. Metaxas and A. Odena, "Self-Attention Generative Adversarial Networks," in *International Conference on Machine Learning (ICML)*, 2019.

[92] A. Brock, J. Donahue and K. Simonyan, "Large Scale GAN Training for High Fidelity Natural Image Synthesis," in *arXiv preprint arXiv:1809.11096*, 2018.

[93] J. Y. Zhu, T. Park, P. Isola and A. A. Efros, "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017.

[94] T. Karras, S. Laine and T. Aila, "A Style-Based Generator Architecture for Generative Adversarial Networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[95] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, ... and W. Shi, "Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[96] X. Wang, K. Yu, S. Wu, J. Gu, Y. Liu, C. Dong, ... and C. Change Loy, "ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

[97] T. Kim, M. Cha, H. Kim, J. K. Lee and J. Kim, "Learning to Discover Cross-Domain Relations with Generative Adversarial Networks," in *International Conference on Machine Learning (ICML)*, 2017.

[98] P. Luc, C. Couprie, S. Chintala and J. Verbeek, "Semantic Segmentation using Adversarial Networks," in *arXiv preprint arXiv:1611.08408*, 2016.

[99] Y. Xue, T. Xu, H. Zhang, L. R. Long and X. Huang, "Segan: Adversarial Network with Multi-scale L1 Loss for Medical Image Segmentation," in *Neuroinformatics*, 2018.

[100] K. Ehsani, R. Mottaghi and A. Farhadi, "SeGAN: Segmenting and Generating the Invisible," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[101] N. Souly, C. Spampinato and M. Shah, "Semi Supervised Semantic Segmentation Using Generative Adversarial Network," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017.

[102] W. C. Hung, Y. H. Tsai, Y. T. Liou, Y. Y. Lin and M. H. Yang, "Adversarial Learning for Semi-Supervised Semantic Segmentation," in *arXiv preprint arXiv:1802.07934*, 2018.

[103] J. Gou, B. Yu, S. J. Maybank and D. Tao, "Knowledge Distillation: A Survey," *International Journal of Computer Vision*, 2021.

[104] S. You, C. Xu, C. Xu and D. Tao, "Learning from Multiple Teacher Networks," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017.

[105] S. I. Mirzadeh, M. Farajtabar, A. Li, N. Levine, A. Matsukawa and H. Ghasemzadeh, "Improved Knowledge Distillation via Teacher Assistant," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

[106] A. Tarvainen and H. Valpola, "Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results," in *arXiv preprint arXiv:1703.01780*, 2017.

[107] Y. Zhang, T. Xiang, T. M. Hospedales and H. Lu, "Deep Mutual Learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[108] H. Chen, Y. Wang, C. Xu, Z. Yang, C. Liu, B. Shi and Q. Tian, "Data-Free Learning of Student Networks," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

[109] Y. Liu, K. Chen, C. Liu, Z. Qin, Z. Luo and J. Wang, "Structured Knowledge Distillation for Semantic Segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[110] I. Loshchilov and F. Hutter, "SGDR: Stochastic Gradient Descent with Warm Restarts," in *International Conference on Learning Representations (ICLR)*, 2017.

[111] T. He, Z. Zhang, H. Zhang, Z. Zhang, J. Xie and M. Li, "Bag of Tricks for Image Classification with Convolutional Neural Networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[112] J. Long, E. Shelhamer and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 2016.

[113] F. Yu, V. Koltun and T. Funkhouser, "Dilated Residual Networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

Publication by the Author

[1] Chanho Kim and Won-Sook Lee. “Temperature as a Regularizer for Semantic Segmentation”, *European Symposium on Artificial Neural Networks (ESANN)*, 2021.