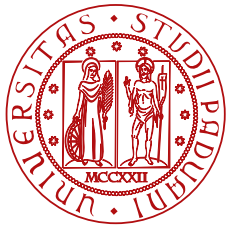


Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Master Thesis in Computer Engineering

Data Augmentation through Generative Models

Master Candidate

Mattia Toffanin

Student ID 2096045

Supervisor

Prof. Emanuele Menegatti

University of Padova

Co-supervisor

Dott. Alberto Bacchin

University of Padova

Academic Year
2023/2024

To my family, for their unconditional support

Abstract

In recent years, semantic segmentation has emerged as a critical task in computer vision, vital for applications ranging from autonomous driving to medical image analysis. This task involves labeling each pixel in an image with semantic categories such as vehicles, pedestrians, and road signs. However, training deep learning models for semantic segmentation requires substantial amounts of labeled data, which is expensive and time-consuming to acquire. To address this challenge, data augmentation techniques are applied to artificially expand the training set and enhance model performance.

Recent studies have highlighted the efficacy of augmenting data in feature space, where extracted features better capture semantic variations compared to raw input data. This approach leads to more realistic augmented data, thereby improving model robustness and generalization. Moreover, GAN-based methods have emerged as promising tools for data augmentation by generating synthetic data that closely resemble real-world data.

This thesis investigates GAN-based methods for augmenting data in the feature space for semantic segmentation tasks. The proposed methodology involves extracting features using the encoder of a segmentation model, generating augmented features similar to real features using a GAN or similar generative model, and subsequently training the segmentation model decoder using both real and generated features. Experimental evaluations focus on assessing the effectiveness of this approach in enhancing the accuracy and generalization capabilities of DeepLabV3 segmentation model across Pascal VOC dataset.

Keywords: semantic segmentation, deep learning, data augmentation, GAN, generative models

Sommario

Negli ultimi anni, la segmentazione semantica è emersa come un compito fondamentale nella visione artificiale, essenziale per applicazioni che spaziano dalla guida autonoma all'analisi di immagini mediche. Questo compito implica l'etichettatura di ogni pixel in un'immagine con categorie semantiche come veicoli, pedoni e segnali stradali. Tuttavia, l'allenamento di modelli di deep learning per la segmentazione semantica richiede una notevole quantità di dati etichettati, che sono costosi e difficili da acquisire. Per affrontare questa sfida, vengono applicate tecniche di data augmentation per espandere artificialmente il set di allenamento e migliorare le performance del modello.
Studi recenti hanno evidenziato l'efficacia della data augmentation nello spazio delle features, dove le features estratte catturano meglio le variazioni semantiche rispetto ai dati di input grezzi. Inoltre, i metodi basati su GAN sono emersi come strumenti promettenti per la data augmentation, generando dati sintetici che somigliano molto a quelli reali.
Questa tesi esplora metodi basati su GAN per la data augmentation nello spazio delle features per compiti di segmentazione semantica. La metodologia proposta prevede l'estrazione delle features utilizzando l'encoder di un modello di segmentazione, la generazione di features sintetiche simili a quelle reali mediante una GAN o un modello generativo simile, e successivamente l'allenamento del decoder del modello di segmentazione utilizzando sia le features reali che quelle generate. Le valutazioni sperimentali si concentrano sulla valutazione dell'efficacia di questo approccio nel migliorare la precisione e le capacità di generalizzazione del modello di segmentazione DeepLabV3 sul dataset Pascal VOC.

Parole chiave: segmentazione semantica, deep learning, data augmentation, GAN, modelli generativi.

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

The proposal for this thesis emerged from the need to improve the vision performance of a robotic waste sorting system, tasked with identifying and separating different materials (such as paper, plastic, etc.) on a chaotic waste stream using an industrial robot. This technology extends its outcomes beyond simple automation, influencing various aspects of operations. By alleviating humans from monotonous and labor-intensive tasks, it plays a crucial role in fostering the development of a more circular economy.
In particular, the aim is to enhance the performance of the semantic segmentation vision system by employing data augmentation techniques in order to create the basis for increasing the picking performances of the system as a whole.

In recent years, semantic segmentation has become a very critical task in computer vision, including application ranging from autonomous driving to medical image analysis. In particular, semantic segmentation is the task whose goal is to associate a label to each pixel of an image. For example, in the context of autonomous driving, semantic segmentation aims to identify traffic signs, vehicles, pedestrian, etc. etc.
Training deep learning models for the semantic segmentation task requires large amount of labeled data which is costly and time-consuming. In the field of semantic segmentation of waste, this issue becomes even more pronounced due to, for example, intrinsic variations within object classes. In order to tackle this issue, data augmentation techniques are applied, generating existing data variations to artificially increase training set and to improve performances.
In the context of waste sorting systems, recent studies like [1] have examined

the effectiveness of data augmentation techniques, where a generative model is trained to create new images to be added to the training set, thereby enhancing picking performance. These studies show that data augmentation can effectively improve performance even when the generated images are not photorealistic.
This finding suggests that models can learn from higher-level features, capturing essential characteristics without requiring perfect visual fidelity. Additionally, it might be simpler and more efficient to generate features directly, as they are abstract representations that isolate various aspects of the image and re-project them in a different space.
Starting from this, the idea arises to investigate data augmentation at the feature level (latent space) rather than solely at the image level (input space). While this approach is initially aimed at enhancing waste sorting systems with limited annotated data and challenging classification tasks, its applications could extend to various computer vision fields, potentially achieving a broader impact.

The main paradigm of the proposed method involves extracting features from the segmentation model's encoder, augmenting these features through a generative model (such as a GAN), and training the segmentation model's decoder with both real and generated features.
Building on this approach, the proposed FeatureGAN model integrates both a segmentation model and a generative model. Specifically, it combines the feature extraction process of the segmentation model's encoder with the feature augmentation capabilities of the generative model, ultimately using both real and generated features to train the decoder.

As discussed in Chapter 2, to achieve the set objectives, we began with a review of the scientific literature to understand the context in which this project is situated. This is followed by the implementation phase of the main paradigm, which is analyzed in detail in Chapter 3. The final phase, outlined in Chapter 4, involves analyzing the results obtained using the previously described method across public datasets, with the aim of investigating whether the implemented approach is effective and worthy of further development.

2

Related Works

This chapter will analyze the leading models and state-of-the-art methods for semantic segmentation, data augmentation, and generative models. For each, the main characteristics and techniques will be examined, providing a general overview of the current state of the art and the context in which this thesis project is situated.

2.1 Semantic Segmentation

2.1.1 Definition and Applications

Semantic segmentation is a fundamental task in computer vision, focusing on assigning a class label to each pixel in an image. Unlike object detection, which identifies bounding boxes around objects, semantic segmentation provides a pixel-level understanding of the scene, crucial for applications like:

- **Autonomous driving**, to recognize roads, traffic signs and obstacles.
- **Medical science**, to segment organs or lesions in medical images like CAT or MRI.
- **Robotics**, to navigate and interact with complex environments.

Its importance lies in its ability to parse complex scenes, enabling systems to interact with the environment in a detailed, structured manner. By providing dense predictions, semantic segmentation plays a critical role in improving machine perception and decision-making capabilities across various domains. [14]

An example of the semantic segmentation task for autonomous driving is represented in Figure 2.1.

Figure 2.1: Example of semantic segmentation for autonomous driving. Adapted from [2].

2.1.2 Models and Approaches

Before the advent of deep learning, semantic segmentation methods relied on a variety of techniques, such as partial differential equation-based methods, random forests. These methods often depended on manually designed features and traditional machine learning algorithms. However, with the increasing complexity of visual data, these methods began to show their limitations in terms of accuracy and scalability. The advent of deep learning has led to significant improvements in semantic segmentation accuracy, as deep learning models are capable of learning hierarchical representations of visual data from raw inputs, eliminating the need for manual feature engineering. [15]
Previous works [15] categorize modern deep learning-based semantic segmentation methods into three categories based on their working principles:

1. **Context-based methods**, these methods aim to improve segmentation accuracy by considering the broader context in which a pixel or region appears. They operate on the premise that the semantic interpretation of a pixel can be enhanced by taking into account information from its surroundings, such as adjacent pixels or regions.

2. **Feature-enhancement-based methods**, these methods aim to create better representations of visual features extracted from images to improve segmentation performance. These methods seek to overcome the limitations of traditional CNNs, which tend to lose spatial details during the extraction of high-level semantic features. Feature enhancement methods are based on the premise that features extracted from the deeper layers of a CNN are more semantically aware but lack spatial details, while features from the shallow layers are more detailed but lack semantic information.

3. **Deconvolution-based methods**, these methods are based on the fundamental principle of the **encoder-decoder** model, first encode the input image into a compact representation of high-level features, and then decode this representation to produce the desired segmentation.

   (a) The encoder part is typically a convolutional neural network (CNN) that gradually reduces the spatial resolution of the input image by applying a series of convolutions, pooling operations, and activation functions. The encoder captures the contextual and semantic information of the image, generating a low-resolution feature map that is rich in semantic information but has reduced spatial details.

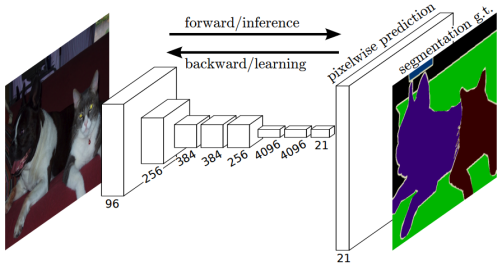
   (b) The decoder part takes the low-resolution feature map from the encoder and attempts to recover the original spatial resolution, generating a dense, pixel-level segmentation. The decoder typically uses upsampling operations, such as deconvolution or unpooling, to increase the resolution of the feature map. To improve the upsampling process and produce more accurate segmentations, the decoder can incorporate skip connections from the encoder, combining high-level features with low-level, spatially detailed features.

The most well-known and influential segmentation models, whose architectures are depicted in Figure 2.2, include:

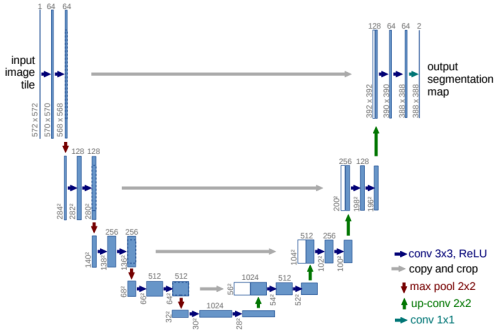
- **FCN** (Fully Convolutional Network) [23], it was the first model to use fully convolutional layers for semantic segmentation. It introduced the use of skip connections to combine high-level semantic features with low-level spatial information, paving the way for future encoder-decoder models.
- **U-Net** [19], it is widely used, especially in biomedical image segmentation. Its symmetric encoder-decoder architecture and dense skip connections make it effective at preserving spatial details and producing accurate segmentations, even with limited datasets.
- **DeepLab** models [4], they are significant for their use of Atrous Spatial Pyramid Pooling (ASPP) to capture multi-scale contextual information. The evolution from DeepLabV2 to V3+ demonstrates incremental improvements in context utilization and model architecture, leading to enhanced accuracy.

2.1.3 Datasets

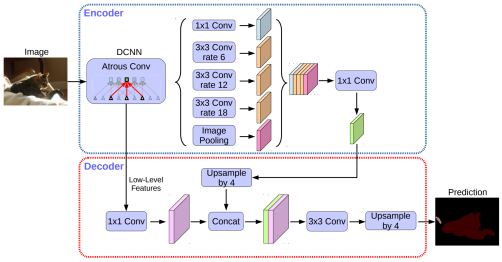
In this section, we provide an overview of the key datasets commonly used for semantic segmentation tasks: Pascal VOC, COCO, ADE20K, and Cityscapes. Each dataset has distinct characteristics that make it suitable for various segmentation challenges.



(a) FCN architecture.
Adapted from [23].



(b) U-Net architecture.
Adapted from [19].



(c) DeepLabV3+ architecture.
Adapted from [5].

Figure 2.2: Most common segmentation model architectures.

Pascal VOC

The Pascal Visual Object Classes (VOC) [11] dataset is widely recognized in the computer vision community. It consists of 20 object classes, including animals, vehicles, and household items, along with a background class. The most common version (2012) contains around 10,000 images, with pixel-level annotations for segmentation tasks.

COCO

The Common Objects in Context (COCO) [22] dataset is one of the largest and most comprehensive datasets for object detection, segmentation, and captioning. It includes over 330,000 images with more than 2.5 million object instances labeled across 80 object categories. The images are often complex, featuring multiple objects in diverse settings. COCO provides high-quality pixel-level annotations and has a resolution that typically varies from 640x480 to larger sizes. Its emphasis on contextual information makes it particularly beneficial for

training models that require understanding object relationships within scenes.

ADE20K

The ADE20K [33] dataset is designed for semantic segmentation tasks, containing over 20,000 images with comprehensive annotations. It includes 150 object categories, ranging from common objects to more abstract entities. The dataset is notable for its diversity in scenes, including indoor and outdoor environments. ADE20K is particularly valuable for applications requiring detailed understanding and segmentation of complex scenes.

Cityscapes

The Cityscapes [8] dataset focuses on urban street scenes and is specifically designed for semantic segmentation in autonomous driving applications. It consists of 5,000 high-resolution images (1024x2048 pixels) captured in various cities across Europe. The dataset provides annotations for 30 classes, including roads, pedestrians, vehicles, and buildings, making it highly relevant for understanding urban environments. The fine-grained annotations and high-resolution images make Cityscapes an essential resource for developing and evaluating segmentation algorithms in real-world scenarios.

2.1.4 Metrics

Various evaluation criteria have been proposed and are commonly used to assess the accuracy of techniques for semantic segmentation. These metrics typically involve variations of pixel accuracy and Intersection over Union (IoU). In this section, the most popular metrics currently employed to measure the performance of per-pixel labeling methods are presented. Only notation, TP_c are true positives, TN_c are true negatives, FP_c are the false positives and FN_c are false negatives. [12]

Pixel Accuracy

This is the simplest metric, calculated by taking the ratio of the number of correctly classified pixels to the total number of pixels in the image. The formula is the following:

$$PA = \frac{\sum_c TP_c}{\sum_c TP_c + FP_c} \tag{2.1}$$

Mean Intersection over Union

This is the standard metric used for segmentation tasks. It calculates the ratio between the intersection and union of two sets, specifically the ground truth and the predicted segmentation. This ratio can be expressed as the number of true positives (the intersection) divided by the sum of true positives, false negatives, and false positives (the union). The Intersection over Union (IoU) is computed for each class and then averaged across all classes.

$$\text{mIoU} = \frac{1}{C} \frac{\sum_c \text{TP}_c}{\sum_c \text{TP}_c + \text{FP}_c + \text{FN}_c} \tag{2.2}$$

2.2 Data Augmentation

Data augmentation is a crucial technique in machine learning that enhances both the quantity and diversity of training data. By creating modified versions of existing data, it helps models generalize better to unseen examples. This technique is especially vital when working with small datasets, where models are prone to overfitting, essentially memorizing the training data rather than learning to generalize.
Input space data augmentation involves directly transforming the input data, such as images or text. Common techniques include flipping, rotating, and cropping images, as well as replacing words or phrases in text. Nonetheless, the focus of this section is on feature space data augmentation and GAN-based data augmentation.

2.2.1 Feature Space Data Augmentation

Feature space data augmentation is becoming increasingly popular, particularly in the field of deep learning, for several reasons. Instead of directly manipulating input data such as images or text, this approach focuses on transforming data in the feature space, a representation of the data learned by a machine learning model.

As stated in [24], the key reasons why feature space data augmentation is beneficial are:

- Generation of more realistic data: feature space data augmentation offers the advantage of capturing semantic variations more effectively than raw input data. By manipulating features, it becomes easier to generate realistic

augmented data, leading to more robust and generalizable models. Additionally, deep features can better capture subtle image variations, making feature-level augmentation more effective than input-level methods.

- Greater generalizability: the same feature space data augmentation techniques can be applied across different types of data, regardless of their specific modality. This contrasts with input space augmentation techniques, which are often modality-specific; for example, rotations and flips of images cannot be applied to text.

In essence, feature space data augmentation allows for data augmentation in a more efficient, effective, and generalizable manner compared to input space augmentation. As in [24] there are three families of techniques for latent space data augmentation:

1. **Feature Transformation** apply transformations like affine transformations or adding noise to the feature vectors. The goal is to introduce variations in the feature representations without changing the input image's label.

2. **Feature Addition and Mixing** enrich the feature set by combining features from different sources. These sources can include mixing features from different training samples, (averaging, concatenation, or interpolation of feature vectors) or combining features from multiple layers of the same network (leveraging information from different levels of representation within the network itself).

3. **Feature Elimination**, inspired by dropout regularization, involves randomly dropping out certain features during training. This encourages the network to learn more robust and less redundant representations.

As explained in [20, 10], the most effective and useful operations in latent space are:

- **Upsample** duplicates existing training data in the latent space. This technique addresses class imbalance by increasing the representation of underrepresented categories.

- **Perturb** introduces random noise (both additive and multiplicative) to existing training data in the feature space.

- **Conditional Variational Auto-Encoder** uses a generative model used to create new examples for a given category. It learns a latent distribution of the data and generates new examples by sampling from this latent distribution.

- **Linear Deltas** generates new data points by calculating the difference between two existing examples from the same class and adding it to a third example from the same class.

- **Hard example extrapolation** targets instances that lie close to class boundaries, which are often more challenging to classify. Instead of extrapolating from random examples, it uses the class center $\mu_c = \frac{1}{m}\sum_{j=1}^{m} z_{c_j}$ as a reference point. The extrapolated representation is defined as $\hat{z}_{c_i} = z_{c_i} + \lambda(z_{c_i} - \mu_c)$, where λ is the scaling factor.
- **Hard example interpolation** involves creating a more challenging example from a seed latent representation $z_{c,i}$, which is the latent representation of the i-th instance in class c. To do this, the nearest hard example from a set of q candidates is selected based on its classification loss. The interpolation is mathematically represented as $\hat{z}_{c_i} = z_{c_i} + \lambda(s - z_{c_i})$ where s is the closest hard example and λ is a scaling factor.

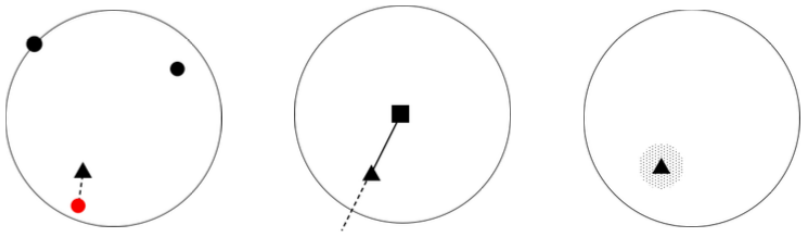


Figure 2.3: Respectively hard example interpolation, hard example extrapolation and perturb. Triangle: seed latent representation; circle outline: class boundary; black circle: sampled feature; red circle: nearest sampled hard example; square: class mean; dotted line: range of augmented example. Adapted from [7].

The results show that extrapolation in the feature space can improve classification accuracy compared to the absence of data augmentation and, in some cases, even outperform traditional (manual) input space augmentation, making feature space extrapolation the most effective data augmentation technique among those tested by introducing greater variability in the dataset and encouraging the model to learn more complex class boundaries.

One interesting study which explores the challenge of few-shot learning, where models must learn from a limited number of examples, is [31]. The paper introduces an innovative distribution calibration strategy that enhances few-shot classification accuracy by transferring statistics from classes with abundant examples (base classes) to those with limited examples (novel classes). The method assumes that similar classes exhibit similar feature distributions. Unlike other augmentation techniques that generate new features directly, this approach calibrates the mean and covariance of novel class feature distributions using statistics from the most similar base classes. This approach results in

a more accurate and generalizable model, which can be applied with various feature extractors and classification models. Experiments on few-shot classification benchmarks such as miniImageNet, tieredImageNet, and CUB demonstrate that this method surpasses state-of-the-art approaches. Furthermore, the paper provides visualizations of the generated features, illustrating that they provide an accurate estimate of the target class distribution.

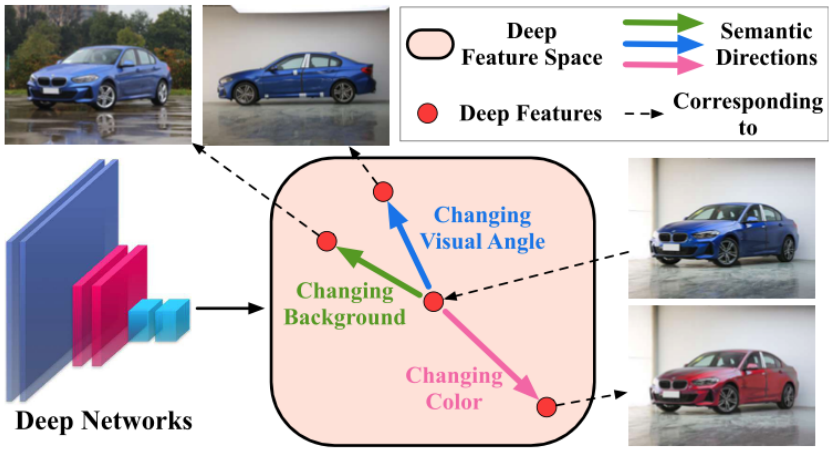


Figure 2.4: Visualization of features and semantic directions in the deep feature space. Traversing along semantic directions within the deep feature space results in significant semantic changes in the input image space, for example, including variations in car color, background elements, and viewpoint angles. Adapted from [30].

Another relevant technique in feature space is **Implicit Semantic Data Augmentation** (ISDA) [29, 30]. The principle behind ISDA arises from the observation that deep neural networks tend to linearize features, meaning that certain directions in the feature space correspond to significant semantic transformations in the input data. For example, a specific direction might represent "adding glasses" or "changing the background" of an image. ISDA leverages this property (shown in Figure 2.4) to augment training data without explicitly generating new images. Instead, vectors are added to existing features, "shifting" the image in semantic space without the need to create new images. Heres how ISDA works in detail:

1. **Covariance Matrix Estimation**, for each class in the training data, ISDA computes an online covariance matrix. This matrix captures the semantic variations within the class, reflecting how features of images belonging to the same class vary along different semantic directions.

2. **Sampling Semantic Directions**, to augment an images features, ISDA samples a random vector from a multivariate normal distribution with zero mean, using the estimated covariance matrix for that class. This vector represents a random semantic direction in the feature space.
3. **Applying the Semantic Transformation**, the sampled random vector is then added to the images features, resulting in augmented features. This is equivalent to translating the image along the sampled semantic direction in the feature space.

Interesting is the fact that, instead of explicitly augmenting the training data, ISDA optimizes a loss function that accounts for all possible implicit semantic augmentations without needing to explicitly generate them. This approach results in a more computationally efficient algorithm compared to explicit data augmentation, as it avoids the overhead of creating and processing additional augmented data while still capturing the diversity introduced by semantic transformations. The authors applied ISDA for semantic segmentation purposes to the Cityscapes dataset, reproducing PSPNet and DeepLabV3 with standard settings. ISDA improved mIoU by nearly 1% for both models, with a 6% increase in training time.

2.2.2 GAN-based Data Augmentation

GAN-based data augmentation is a technique that uses Generative Adversarial Networks (GANs) to increase the size and diversity of training datasets. Essentially, a GAN is trained on an existing dataset and then used to generate new synthetic data samples that resemble the original data. These synthetic data can then be used to train deep learning models, enhancing their performance, especially in situations with limited data.

The first study to take into account is **Smart Augmentation** [21]. Smart Augmentation aims to learn the best data augmentation strategies during the training of a neural network. The goal of Smart Augmentation is to improve the performance of a 'target' neural network (network B) that is learning a specific task, such as gender or scene classification. The method involves using a second neural network (network A) to generate new training data for network B. Network A takes as input multiple samples from the same class (e.g., images of male faces) and learns to combine them to create a new sample that still belongs to that class. The new sample is then used to train network B. The loss of network B is used to guide the training of network A, so that network A learns to generate augmentations that help network B achieve better accuracy.

In [18] it is described a GAN-based data augmentation method called **VeGAN**, designed to improve the accuracy of a semantic segmentation network in detecting cars in aerial images. VeGAN generates synthetic car images from a latent space, focusing on underrepresented areas. These images are used to augment the training dataset, helping the segmentation network generalize better. Results show that using VeGAN for data augmentation significantly improved the networks Intersection-over-Union (IoU) from 93.4% to 94.9%.

GANs and generative models, more generally, are discussed in the next section.

2.3 Generative Models

Generative methods in deep learning have emerged as a transformative approach in the field of artificial intelligence, enabling the creation of new data that mimics the distribution of existing datasets. These methods involve training models to generate samples that are indistinguishable from real-world data, allowing for applications across a broad spectrum of domains, from image synthesis to natural language processing.

At the core of generative methods are models like **Generative Adversarial Networks** (GANs), **Variational Auto-Encoders** (VAEs), and more recently, **diffusion models**. These techniques are designed not just to predict outputs based on inputs, but to understand and replicate the underlying data distribution. This capability makes them powerful tools for tasks such as data augmentation or for generating high-quality images, audio, and even 3D models.

In this section, GAN and its various versions are presented, along with a brief introduction to Diffusion Models.

2.3.1 Generative Adversarial Networks (GAN)

A **Generative Adversarial Network** (GAN) is a type of unsupervised model capable of learning the structure of input data and synthesizing new examples that are statistically indistinguishable from the training data. GANs achieve this by employing two neural networks: a **generator** and a **discriminator**, locked in an adversarial training process. The generator learns to create new data examples, while the discriminator learns to distinguish between real and generated examples. During training, the generator creates samples and provides them

to the discriminator. The discriminator then assesses the likelihood that the provided samples are real or synthetic and updates its parameters to improve its discriminative ability. Simultaneously, the generator updates its parameters based on the discriminator's feedback, striving to generate samples that can fool the discriminator into classifying them as real. This iterative process continues until the generator produces highly plausible samples that the discriminator can no longer distinguish from real data.

More formally, the discriminator, essentially a binary classifier, aims to maximize its ability to correctly identify both real and generated data. It typically uses a binary cross-entropy loss function for this purpose. The objective is to minimize the cross-entropy, which implies maximizing the probability of assigning correct labels (real or generated) to the input data. The generator, on the other hand, aims to deceive the discriminator by generating data that is incorrectly classified as real. Its loss function is typically designed to maximize the probability that the discriminator mistakenly classifies the generated samples as real. The two networks are locked in a **min-max game**, where the discriminator seeks to minimize its loss function (while simultaneously maximizing the generator's loss function), whereas the generator aims to minimize its own loss function (while maximizing the discriminator's loss function). The simultaneous optimization of these loss functions leads to a training process in which the generator continuously improves its ability to produce realistic data, while the discriminator becomes more adept at detecting fakes. The architectural scheme of the original GAN is depicted in Figure 2.5
In particular the original standard GAN loss function, also known as the **min-max loss**, uses the **Binary Cross Entropy loss** and is computed following the formula:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{\text{data}}(x)}\left[\log D(x)\right] + \mathbb{E}_{z\sim p_z(z)}\left[\log(1-D(G(z)))\right]. \quad (2.3)$$

where $\log D(x)$ refers to the probability that the discriminator is rightly classifying the real image and $\log(1 - D(G(z)))$ refers to the probability that the discriminator is rightly classifying the fake image generated by the generator. It is common for GANs to experience performance issues. The most prevalent problems include:

- **Mode Collapse**, it occurs when the generator repeatedly produces a small subset of similar samples or, in extreme cases, the same sample for different latent variable inputs. In other words, the generator learns to "trick"

the discriminator by finding certain examples that deceive it, rather than learning to represent the complex distribution of real data. If the discriminator is not skilled enough to distinguish between real and generated data, the generator may become 'satisfied' with producing a small number of samples that successfully deceive it.

- **Vanishing Gradients**, it occurs when the gradients that should propagate from the discriminator to the generator become extremely small, hindering the generator's ability to learn and effectively update its parameters. If the discriminator becomes too accurate in distinguishing between real and generated data, the gradients that the generator receives may become extremely small. This occurs because the discriminator provides a learning signal to the generator based on its ability to be deceived. However, with weak gradients, the generator struggles to learn from its mistakes and improve the quality of the synthetic data.

In short, there is a very fine balance between the quality of the discriminator and the generator. [26, 13]

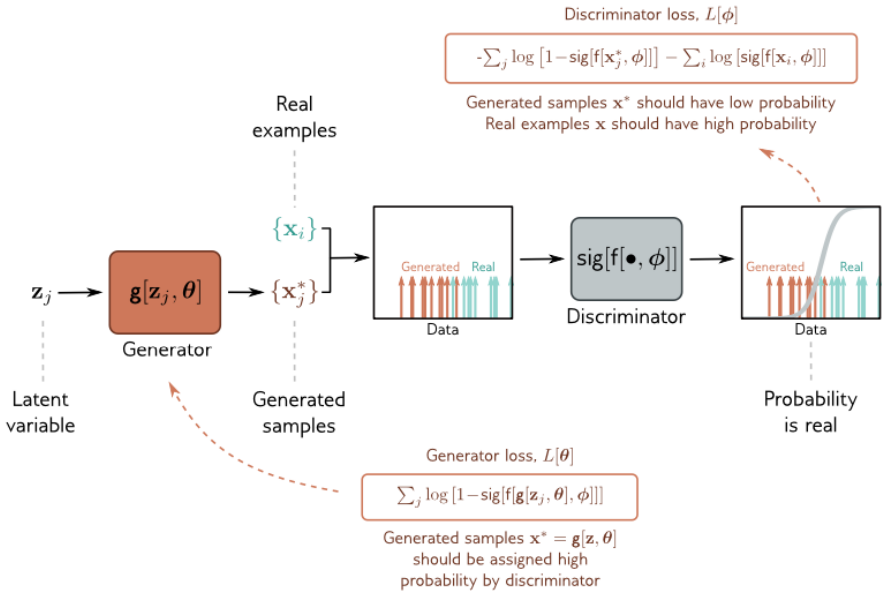


Figure 2.5: Original GAN architecture.
Adapted from [26].

Deep Convolutional GAN (DCGAN)

The **Deep Convolutional Generative Adversarial Network**, or DCGAN, is a type of GAN designed to improve the realism of generated images by employing convolutional layers. Unlike traditional GANs, which relied on fully connected layers, DCGAN introduced a convolutional architecture, allowing the model to

better capture spatial hierarchies and produce high-quality images.
The DCGAN architecture consists of a generator and a discriminator.
The generator transforms a random noise vector into an image through a series of transposed convolutions, using techniques like batch normalization and ReLU activations to stabilize training.
The discriminator, on the other hand, is a convolutional neural network that distinguishes real images from generated ones, with LeakyReLU activations to enhance gradient flow and prevent the "dying ReLU" problem. Together, the generator learns to create increasingly realistic images while the discriminator improves its ability to identify generated images. The DCGAN architecture, demonstrated in the original paper [27] through various experiments, showed the potential of using convolutional networks for stable and high-quality image generation, setting a precedent for future work in GAN-based image synthesis.
The architecture of DCGAN is represented in Figure 2.6.

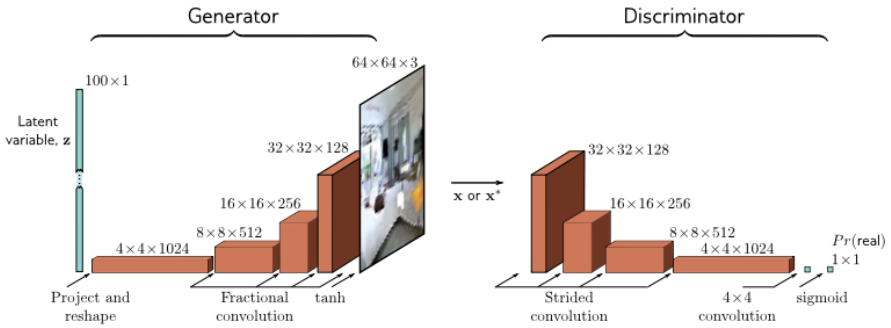


Figure 2.6: Deep Convolutional GAN architecture.
Adapted from [26].

Wasserstein GAN (WGAN)

Wasserstein Generative Adversarial Networks (WGANs) are a variant of GANs that aim to address the issues of stability and convergence during training. As previously discussed, training GANs can be challenging due to the inherently unstable nature of the adversarial loss function. WGANs tackle this problem by utilizing the **Wasserstein distance**[1], a more stable metric for mea-

[1]The Wasserstein distance, also known as the Earth Mover's Distance, is a metric used to measure the dissimilarity between two probability distributions. It represents the minimum cost of transforming one distribution into another, considering both the amount of change and the distance it must be moved.

suring the difference between the real data distribution and the generated data distribution. Instead of using a discriminator to directly classify images as real or fake, WGANs employ a **critic** that estimates the Wasserstein distance between the two distributions. The critic is trained to approximate the Wasserstein distance, while the generator is trained to minimize it. The use of the Wasserstein distance provides more meaningful gradients to the generator, improving training stability and reducing the likelihood of mode collapse. [26]

Conditional GAN (cGAN)

Conditional Generative Adversarial Networks (cGANs), as the name suggests, introduce the concept of conditioning into the generation process. As represented in Figure 2.7, this means that the generator, rather than producing random samples, is guided by additional input information such as class labels, specific attributes, or even other images. For example, a cGAN could be trained to generate images of faces conditioned on desired characteristics such as age, gender, or expression. This conditioning is implemented by providing the additional information to both the generator and the discriminator during training. As a result, the discriminator learns to evaluate not only the realism of the generated image but also its consistency with the conditioning information. cGANs have significantly expanded the flexibility and control over image generation, paving the way for a wide range of applications, including image generation from textual descriptions, image-to-image translation, and attribute-based image manipulation. [26]

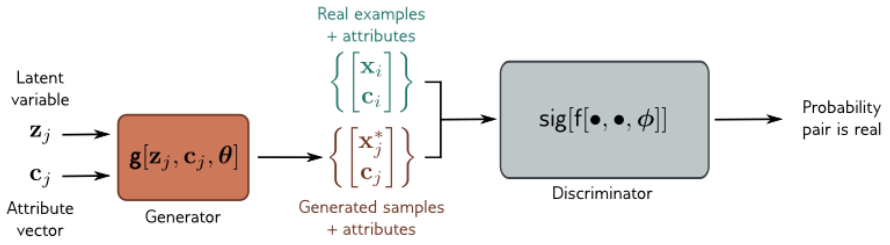


Figure 2.7: Conditional GAN architecture.
Adapted from [26].

Auxiliary Classifier GAN (ACGAN)

Auxiliary Classifier Generative Adversarial Networks (ACGANs) are an extension of cGANs that incorporate an **auxiliary classifier** into the discrimina-

tor, as shown in Figure 2.8. This classifier is responsible for predicting the class label of the input image, whether it is real or generated. The inclusion of this auxiliary classification objective offers several advantages. First, it provides an additional learning signal to the discriminator, enabling it to better distinguish between real and generated images. Second, it encourages the generator to produce images that not only appear realistic but are also clearly recognizable as belonging to the desired class.
For instance, an ACGAN trained on an animal dataset generates realistic images of animals from different classes (e.g., dog, cat, bird), with the auxiliary classifier ensuring that each image is correctly classified into its corresponding class.[26]

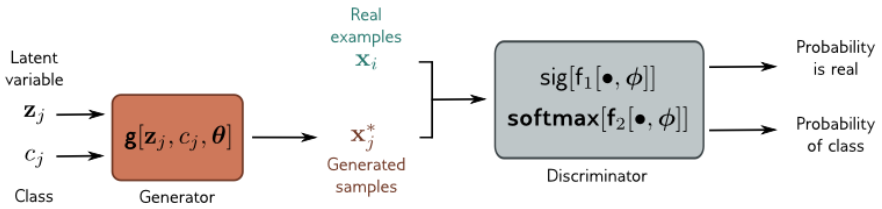


Figure 2.8: Auxiliary Classifier GAN architecture.
Adapted from [26].

InfoGAN

The **Information Maximizing Generative Adversarial Network** (InfoGAN) [6] distinguishes itself from previous architectures by aiming to automatically discover meaningful features in the data without the need for explicit labels or attributes. As shown in Figure 2.9, InfoGAN achieves this by introducing a vector of **attribute variables** c into the generator, in addition to the random latent variable z. The discriminator, beyond its usual role of distinguishing between real and generated images, is also trained to predict the values of these attribute variables. The key insight behind InfoGAN is that attribute variables that capture significant and interpretable features of the data will be easier for the discriminator to predict.
For instance, in a dataset of handwritten digits, InfoGAN might automatically uncover attributes such as the digit's orientation, stroke thickness, and shape. This unsupervised approach to representation learning makes InfoGAN particularly valuable for analyzing and generating unlabeled data. [26]

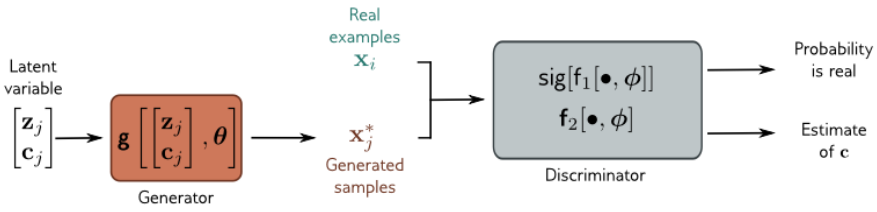


Figure 2.9: InfoGAN architecture.
Adapted from [26].

Pix2Pix

Pix2Pix, introduced by [16], is a neural network designed to translate an input image into an output image with a different style, leveraging the U-Net architecture. The goal is to generate an output image that not only maintains structural consistency with the input but also appears realistic in the desired style. For example, Pix2Pix can be used for tasks such as coloring gray-scale images, translating maps into satellite images, converting sketches into photo-realistic images, and generating building facades from label maps. The architectural scheme is represented in Figure 2.10.

During training, Pix2Pix employs a combination of two loss functions:

- **Content Loss**, which measures the structural difference between the input and output images, often quantified using the l_1 norm.
- **Adversarial Loss**, similar to traditional GANs, a discriminator is used to assess the realism of the generated image. The discriminator attempts to distinguish between real (input-output) image pairs and synthetic pairs generated by the model. This loss function encourages the generator (the U-Net) to produce images that are indistinguishable from real ones, both globally and at a local level. [26, 28]

SPADE

SPADE (SPatially-Adaptive DEnormalization) is a sophisticated technique used in semantic image synthesis, specifically designed to address the limitations of traditional normalization methods that often fail to preserve semantic information during the image generation process. SPADE operates by adaptively normalizing the features of a neural network based on the semantic content of an input segmentation map, ensuring that critical details are maintained and effectively utilized throughout the generation process.

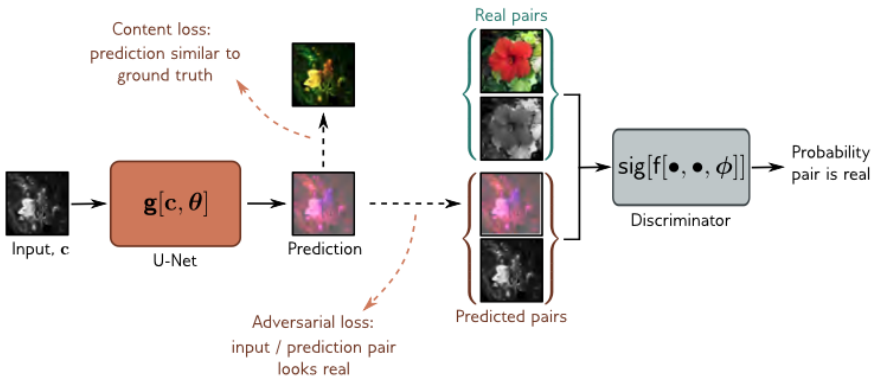


Figure 2.10: Pix2Pix architecture.
Adapted from [26].

The primary purpose of SPADE is to generate photo-realistic images from semantic segmentation maps. The ability to produce images that are not only visually convincing but also semantically accurate makes SPADE a powerful tool in these domains.

The fundamental component of SPADE (whose layer scheme is shown in Figure 2.11) is its ability to perform spatially adaptive denormalization. For each activation layer within the neural network, SPADE modulates the normalized activations using learned parameters γ and β, which are specific to each spatial location in the segmentation map. This modulation ensures that the original semantic information is preserved and effectively incorporated into the image generation process, avoiding the loss of critical details that occurs with conventional normalization methods.

The SPADE-based model is composed of three main components: the **generator**, the **discriminator**, and the **encoder**. The entire model scheme architecture is depicted in Figure 2.13.

The generator is responsible for creating the final photo-realistic image from a given segmentation map. As represented in Figure 2.12, it is structured around a series of ResNet blocks[2], each incorporating a SPADE layer that adjusts the activations based on the input segmentation map. The generator begins with either a random input vector or a down-sampled version of the segmentation map and progressively up-samples the features to generate a high-resolution

[2]A ResNet block is a component of residual neural networks that uses skip connections to bypass one or more layers, helping to prevent the vanishing gradient problem and enabling the training of deeper networks.

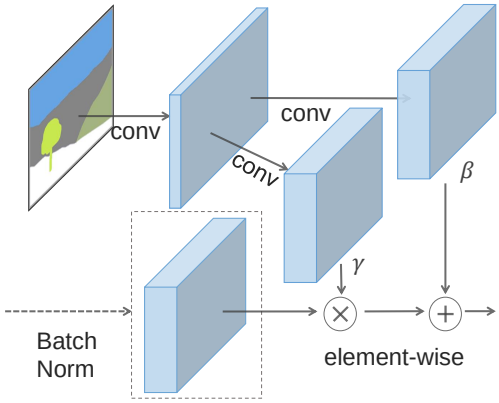


Figure 2.11: SPADE layer. The mask is projected into an embedding space and convolved to generate modulation parameters γ and β as tensors with spatial dimensions, which are then applied element-wise to the normalized activation. Adapted from [25].

image. Unlike traditional models, SPADE eliminates the need for a separate encoder to process the segmentation map, as the necessary semantic information is directly integrated into the network through the SPADE layers.

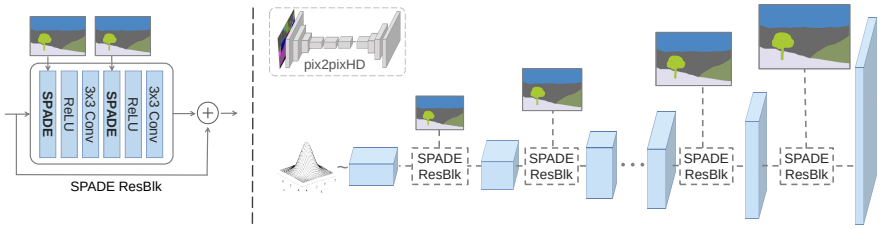


Figure 2.12: SPADE ResBlk and generator architecture. Adapted from [25].

The discriminator plays a crucial role in the adversarial training process, distinguishing between real images and those generated by the generator. It typically follows a **multi-scale design**, as seen in architectures like Pix2Pix, and is enhanced with **spectral normalization**[3] to stabilize training. The discriminator receives the concatenation of the segmentation map and the generated image as input, and its objective is to accurately classify the generated images as fake.

[3]Spectral normalization works by scaling the weight matrix of each layer in a neural network based on its largest singular value, which is called the spectral norm. The process involves computing the singular value decomposition (SVD) of the weight matrix to obtain its largest singular value, and then dividing the matrix by this value. This ensures that the layer's operation is constrained, limiting how much it can amplify the input, thereby stabilizing the training process and preventing gradient explosions.

This adversarial feedback helps the generator improve its output, making the synthesized images increasingly realistic over time.

The encoder is utilized when the model is configured for multi-modal image synthesis, where diverse outputs can be generated from the same segmentation map. The encoder processes a real image to produce a latent representation, which is then combined with the segmentation map in the generator to produce an image that reflects the style captured by the encoder. This setup allows the model to synthesize multiple variations of an image that all share the same semantic structure but differ in style, enabling applications such as style transfer and guided image synthesis.

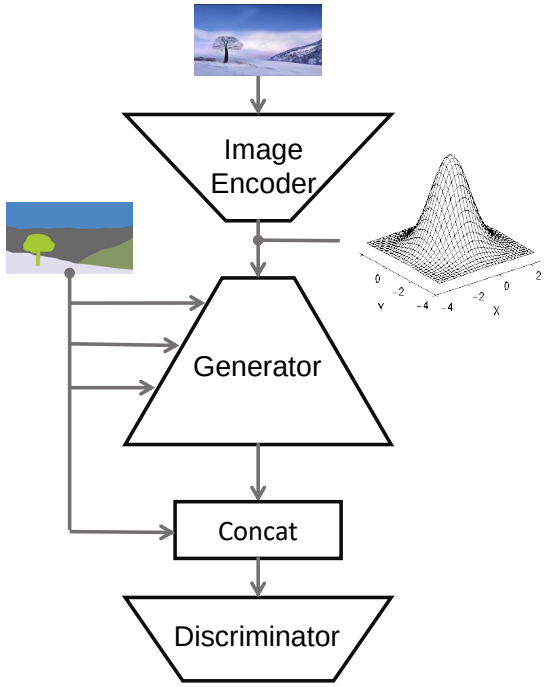


Figure 2.13: SPADE model architecture.
Adapted from [25].

In the SPADE framework, **hinge loss** is used as the adversarial loss to guide the training of both the generator and discriminator. This loss improves the stability of the training process, especially in GANs, and helps achieve better image quality.
For the discriminator hinge loss creates a margin between real and fake images. It works by penalizing the discriminator if the score for a real image is less than

1, or if the score for a fake image is greater than -1. The loss is formulated as:

$$L_D = -\mathbb{E}_{x_{\text{real}}}[\min(0, -1 + D(x_{\text{real}}))] - \mathbb{E}_{x_{\text{fake}}}[\min(0, -1 - D(x_{\text{fake}}))] \tag{2.4}$$

where $D(x_{\text{real}})$ and $D(x_{\text{fake}})$ are the discriminators outputs for real and fake images, respectively.
For the generator, hinge loss aims to make the fake images appear real by maximizing the discriminators output for them. The loss for the generator is:

$$L_G = -\mathbb{E}_{x_{\text{fake}}}[D(x_{\text{fake}})] \tag{2.5}$$

This encourages the generator to improve its performance by increasing the discriminators score for generated images, making them closer to real images. This formulation ensures smoother gradient updates and more stable adversarial training, resulting in high-quality, photo-realistic outputs.

SPADE introduces significant improvements over earlier models like Pix2Pix, which often struggled to maintain semantic coherence during normalization. By preserving and effectively utilizing detailed semantic information, SPADE-based models consistently produce superior results in both visual quality and alignment with the input segmentation map. Extensive experimental evaluations have demonstrated that SPADE-based models outperform previous methods on benchmarks like COCO-Stuff, ADE20K, and Cityscapes, making them a cutting-edge solution in the field of semantic image synthesis. [25]

2.3.2 Diffusion Models

Diffusion models are a class of generative models that have gained attention for their ability to generate high-quality synthetic data, especially images. The core idea behind diffusion models revolves around two key processes: a **forward process** and a **reverse process**, working together to transform noise into meaningful data. Unlike traditional generative models like GANs, diffusion models gradually deconstruct data into noise and then learn how to reverse this process to regenerate coherent data from pure noise.

At the heart of diffusion models is the principle of **progressive noise perturbation**. In the forward process, noise is incrementally added to a data sample until the data is entirely transformed into random noise. This step-wise degradation is governed by stochastic transitions, each adding a small amount of

Gaussian noise. The forward process continues over a predefined number of steps, denoted as T, where x_0 represents the original data and x_T corresponds to the final noisy state.
The reverse process is where the magic happens. The model learns to gradually reverse the noise injection by predicting each step backwardsfrom x_T (pure noise) to x_0 (a reconstructed data sample). This reverse operation is learned through training a neural network that aims to estimate how to remove the noise at each stage. Essentially, the model reconstructs the data in small increments, one noise-removal step at a time, ultimately producing realistic data from random noise.
In terms of architecture, diffusion models often follow an **encoder-decoder structure**. The encoder corresponds to the forward process, progressively adding noise through a series of iterations. As noise is applied, the structure of the data becomes less and less recognizable, with each step encoding a noisier version of the original data. On the other side, the decoder takes on the role of reversing this process. Starting from the fully noisy data, it progressively de-noises the input, step by step, attempting to recover the original data distribution. A representation of the progressive noise perturbation is shown in Figure 2.14.

One of the central challenges in this process is to learn an effective mapping from noisy data back to clean data. The model must understand how to incrementally remove noise while retaining the underlying structure of the data. This task is typically framed as a minimization problem, where the model is trained to reduce the difference between the reconstructed data and the original data at each stage of the reverse process. A common loss function used for this is the **variational lower bound** (ELBO), which quantifies how well the model can reconstruct the data while considering the stochastic nature of the transitions.

In practice, diffusion models operate in three stages. First, in the forward process, noise is progressively added to real data, transforming it into random noise. Next, during training, the model learns to reverse this process, predicting how to de-noise the input at each step. Finally, once trained, the model can generate new data by starting from random noise and following the learned reverse process to create a plausible, high-quality sample.

In summary, diffusion models are a powerful generative approach, relying on the manipulation of noise through an encoder-decoder pipeline to generate realistic data from random inputs. Their unique capability to learn both the for-

ward and reverse processes makes them highly effective for producing complex, structured data distributions.

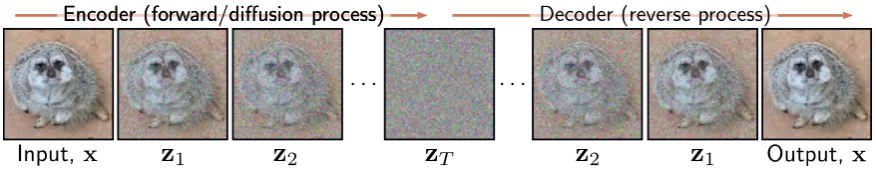


Figure 2.14: Diffusion models forward and reverse process. Adapted from [26].

2.4 Method Positioning in Literature

The proposed method combines the two major techniques just discussed: feature space data augmentation and the use of a generative model. This approach is chosen because data augmentation operations at the feature level have shown to provide a notable performance increase, but the goal is to develop a less "handcrafted" and more generalizable method by employing a generative model. At the time of writing this thesis, this approach is highly innovative, as no other method uses a GAN to generate features for enhancing the performance of a semantic segmentation model.

3 Methods

This chapter presents the proposed method for augmenting features using generative models.

The main steps behind the proposed method are:

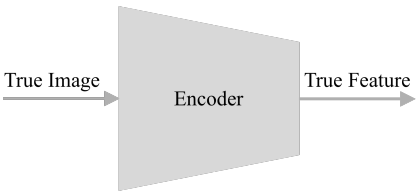
1. **Feature extraction** from the segmentation model encoder.
2. **GAN training** for feature augmentation.
3. **Feature generation** using the generative model.
4. **Segmentation model decoder training** with both real and generated features.

3.1 Model Architecture

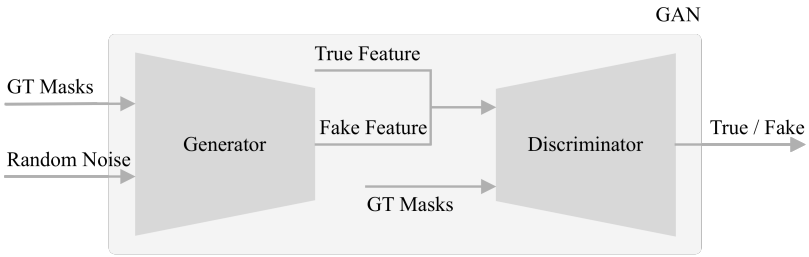
The proposed model consists of a segmentation model and a generative model. The data flow is represented in Figure 3.2.

The chosen segmentation model is DeepLabV3 [3], one of the most advanced and high-performing models currently available. In particular, it is a DeepLabV3 model with a pretrained ResNet101 [32] backbone on ImageNet [9], providing a solid starting point for feature extraction.

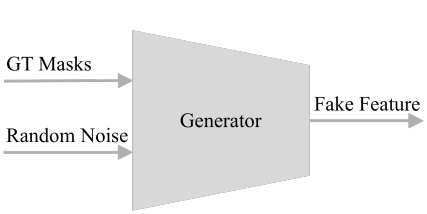
Thus, the segmentation model consists of an encoder that extracts features from input images and a decoder that classifies these features into the output mask. Specifically, starting with an image of shape $3 \times H \times W$, the pretrained ResNet101 extracts features of shape $2048 \times \frac{H}{8} \times \frac{W}{8}$. In contrast, the decoder classifies the features into a segmentation mask of shape num_classes $\times$ H $\times$ W.



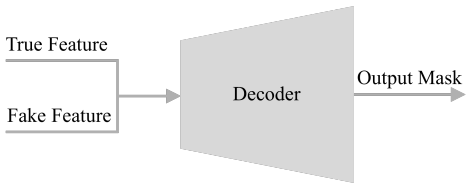
(a) Feature extraction.



(b) GAN training.



(c) Feature generation.



(d) Segmentation model decoder training.

Figure 3.1: Main operations of the proposed method.

As the generative model, a modified version of SPADE [25], referred to as FeatureGAN, has been selected.

The generative model consists of a generator that aims to produce realistic features conditioned on a segmentation mask, and a discriminator that seeks to differentiate between real and synthetic pairs of features and segmentation masks, as a standard cGAN.

Specifically, the generator takes a random vector and a segmentation mask as input to produce synthetic features that match the shape of those extracted from the ResNet101 backbone. Meanwhile, the discriminator classifies the pairs of real or synthetic features and segmentation masks as either real or fake.

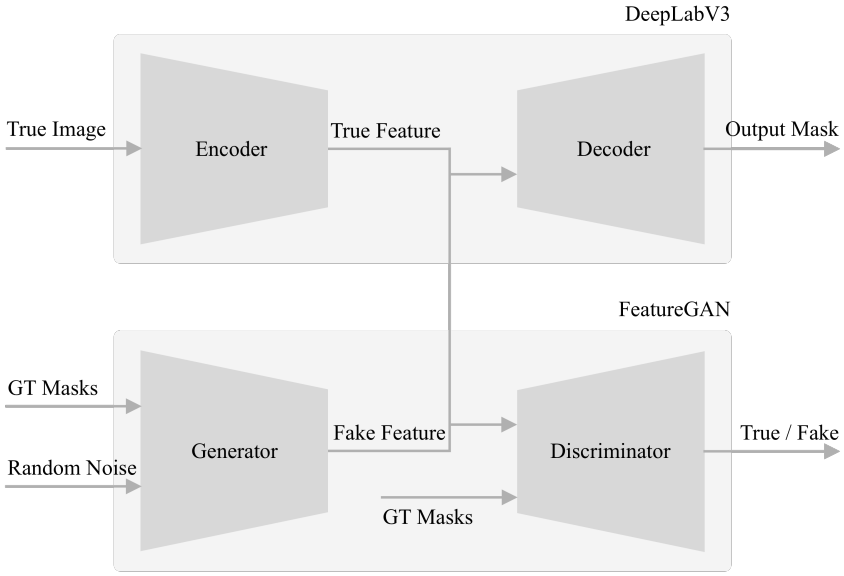


Figure 3.2: Main model architecture. The SPADE image encoder is omitted for sake of clarity.

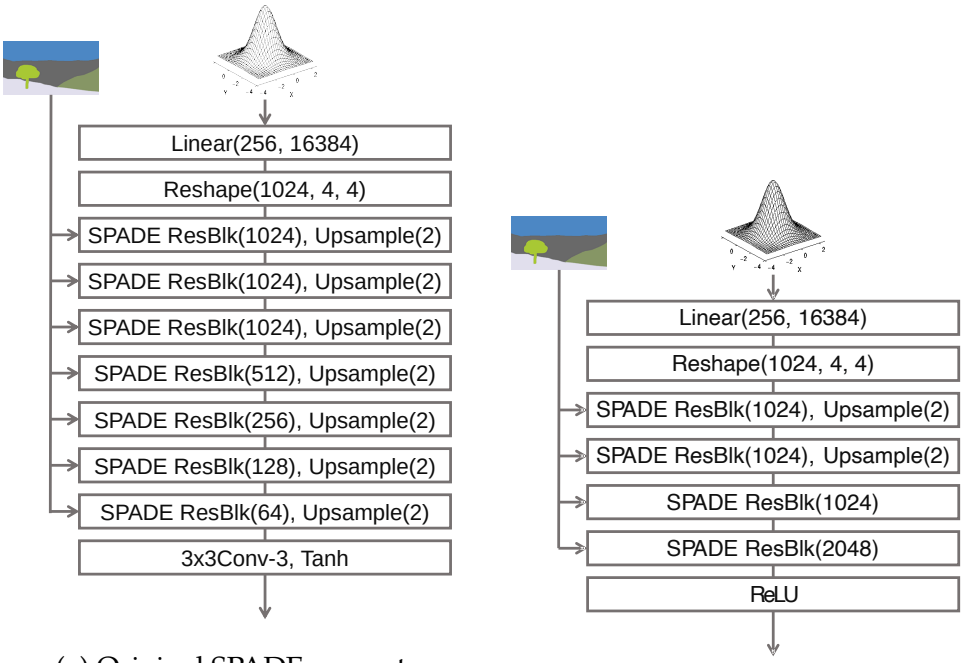
3.1.1 FeatureGAN

The proposed model for synthetic feature generation builds upon the SPADE [25] architecture, originally designed for image-to-image translation based on segmentation masks. SPADE was chosen as the starting point due to its proven effectiveness in generating high-quality images conditioned on segmentation masks, which is crucial to ensuring adequate conditioning for the generator within the GAN.

FeatureGAN Generator

For the new version of the generator, the SPADE generator has been modified to produce features of the same shape as those extracted through the segmentation model encoder, specifically $2048 \times \left(\frac{\text{input\_size}}{8}\right) \times \left(\frac{\text{input\_size}}{8}\right)$, instead of generating images of size $3 \times \text{input\_size} \times \text{input\_size}$. As in the original SPADE model, the architecture of the generator consists of a series of SPADE ResBlks with nearest neighbor up-sampling and the comparison is shown in Figure 3.3.

For the vast majority of the experiments conducted, the same original SPADE encoder was used in conjunction with the generator to create a Variational Auto-Encoder (VAE) that captures the style of the input image and applies it to the generated features. The encoder maps the input image to a mean vector μ and



(a) Original SPADE generator architecture. Adapted from [25].

(b) FeatureGAN generator architecture.

Figure 3.3: Comparison between original SPADE and FeatureGAN generators.

a variance vector σ, from which a random vector is sampled and subsequently passed as input to the generator. As in the original SPADE project [25], shown in Figure 3.4, the encoder is composed of 6 convolutional layers with a stride of 2, followed by two fully connected layers that generate the mean and variance of the output distribution.

FeatureGAN Discriminator

As a conditional GAN, the FeatureGAN discriminator takes as input both the features (real or fake) and the corresponding conditioning segmentation mask. To concatenate these two inputs, which have different spatial dimensions, a series of convolutional layers is first applied to the mask to match the spatial dimensions of the features. Following this, another series of convolutional layers processes the concatenated output to produce the final predictions. The final architecture of the discriminator implemented is shown in Figure 3.5.

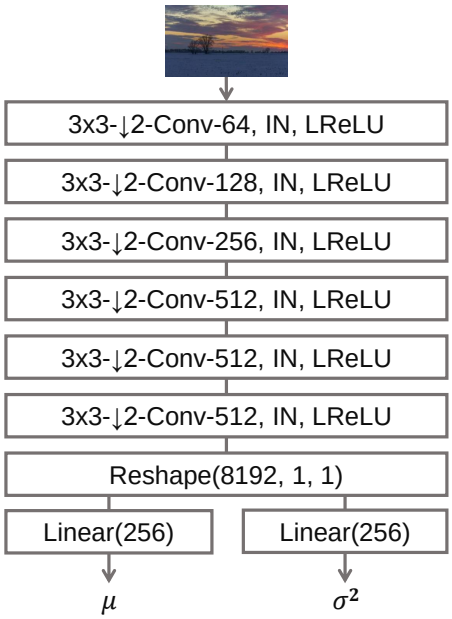


Figure 3.4: Additional image encoder architecture.
Adapted from [25].

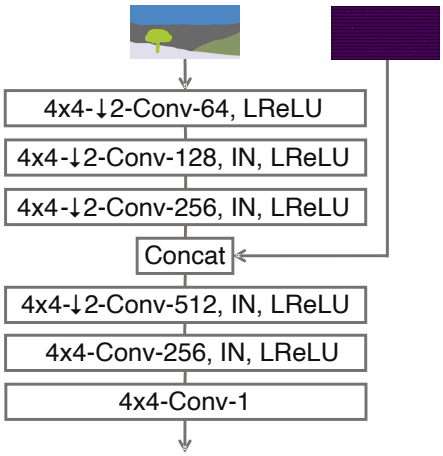


Figure 3.5: FeatureGAN discriminator architecture. In the top right, the features are shown and are concatenated to the first convolutional block output.

3.2 Loss Functions

Throughout the training process, several distinct loss functions are employed. This section provides a brief overview of these loss functions, while their impact on model parameters is discussed in detail in Section 3.3.

3.2.1 KullbackLeibler Divergence Loss

To train the additional image encoder for the generator, the KL divergence loss is employed.
The loss function is defined as:

$$L_{KLD} = \mathcal{D}_{KL}\left(q(z \mid x) \parallel p(z)\right) = \int q(z \mid x) \log \frac{q(z \mid x)}{p(z)} \, dz \tag{3.1}$$

where the prior distribution $p(z)$ is a standard Gaussian, and the variational distribution $q(z \mid x)$ is determined by a mean vector and a variance vector, both computed by the image encoder where x represents the input image.
When the model is set to use the image encoder for for multi-modal and style-guided image synthesis, the image encoder and the generator compose a Variational Auto-Encoder (VAE). In training a Variational Auto-Encoder (VAE), the KL divergence loss encourages the variational distribution $q(z \mid x)$ to approximate the Gaussian prior $p(z)$. This regularization shapes a well-structured latent space, promoting smooth sampling and interpolation. [17]

3.2.2 Hinge Loss

The generator and discriminator are trained using hinge loss as the GAN loss, following the implementation in the original SPADE model.

For the discriminator the loss function is defined as:

$$\begin{aligned} L_D = &- \mathbb{E}_{x \sim p_{\text{data}}} \left[\min\left(0, -1 + D(x)\right)\right] \\ &- \mathbb{E}_{z \sim p_z} \left[\min\left(0, -1 - D(G(z))\right)\right] \qquad (3.2) \\ = &+ \mathbb{E}_{x \sim p_{\text{data}}} \left[\max\left(0, 1 - D(x)\right)\right] \\ &+ \mathbb{E}_{z \sim p_z} \left[\max\left(0, 1 + D(G(z))\right)\right] \qquad (3.3) \end{aligned}$$

The first term $-\mathbb{E}_{x \sim p_{\text{data}}} \left[\min\left(0, -1 + D(x)\right)\right] = \mathbb{E}_{x \sim p_{\text{data}}} \left[\max\left(0, 1 - D(x)\right)\right]$ penalizes the discriminator when it incorrectly classifies real data (i.e., when $D(x) < 1$). In other words, it aims to push the discriminator to assign a high value (close to 1) for real samples.
The second term $-\mathbb{E}_{z \sim p_z} \left[\min\left(0, -1 - D(G(z))\right)\right] = \mathbb{E}_{z \sim p_z} \left[\max\left(0, 1 + D(G(z))\right)\right]]$ penalizes the discriminator when it classifies generated data as real (i.e., when $D(G(z)) > -1$). Here, the goal is to keep the discriminator assigning low values

(close to -1) for generated samples.

For the generator the loss function is defined as:

$$L_G = -\mathbb{E}_{z\sim p_z} D\left(G\left(z\right)\right) \tag{3.4}$$

It means that the generator aims to minimize Equation 3.4, which implies maximizing $D\left(G\left(z\right)\right)$ in order to generate samples that the discriminator classifies as real.

3.2.3 Feature Matching Loss

Similar to SPADE and the earlier Pix2Pix model, a feature matching loss based on the discriminator is utilized to enhance the GAN loss. In this approach, features are extracted from various layers of the discriminator, allowing the model to learn to align these intermediate representations between real and synthesized images. [28]
The feature matching loss is calculated as:

$$L_{FM} = \mathbb{E}_{x\sim p_{\text{data}}, z\sim p_z} \sum_{i=1}^{T} \left[\left\|D^{(i)}(x) - D^{(i)}(G(z))\right\|_1\right] \tag{3.5}$$

where T is the total number of layers and $D^{(i)}$ is the ith-layer feature extractor of the discriminator.

3.2.4 Cross-Entropy Loss

The loss used for the semantic segmentation task is the cross-entropy loss. The formula is:

$$L_{CE} = -\frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{C} y_{i,c} \log(\hat{y}_{i,c}) \tag{3.6}$$

In this form, N is the number of pixels of a single image, C is the number of classes, $y_{i,c}$ is the ground truth label for class c of i-th pixel, and $\hat{y}_{i,c}$ is the predicted probability for that class and pixel which is the segmentation model decoder output over both real and synthetic features.
Alternatively, when the labels are represented in one-hot encoding, the cross-

entropy formula is expressed as:

$$L_{CE} = -\frac{1}{N}\sum_{i=1}^{N}\log(\hat{y}_{i,c_i}) \tag{3.7}$$

where c_i is the correct class for pixel i.

3.3 Training Procedure

The training procedure is described in Algorithm 1.

For each epoch, we iterate over the training set using a batch b composed of images i and segmentation masks m. In each iteration, we extract real features f_{real} from images through the segmentation model's encoder, generate fake features f_{fake} with the generator, and then use the discriminator to distinguish between real p_{real} and fake p_{fake} features.

For the first `freezed_epochs` epochs we pretrain exclusively the GAN without fine-tune the segmentation model. During this pretraining phase, only the generator and discriminator are trained. In the generator step, we compute the KLD loss, generator loss, and feature matching loss, following Equations 3.1, 3.4, and 3.5, respectively. These losses are summed, back-propagated, and used to update the generators parameters. In the discriminator step, we calculate the discriminator loss for both real and fake features according to Equation 3.3, sum them, back-propagate, and update the discriminators parameters.
This pretraining phase is essential for the GAN to learn to generate features that are statistically similar to those extracted from the pretrained backbone.

After the specified number of `freezed_epochs`, we begin training the segmentation model as well. In this phase, the generator step is modified to also compute the segmentation loss for both real and fake features, following Equation 3.6. Additionally, both the generator and segmentation model decoder parameters are updated. In this way, the generator is guided to produce features that are not only statistically similar to real ones but are also semantically valuable for the segmentation models decoder.

Algorithm 1 Training procedure

for each epoch e **do**
  for each batch $b = (i, m)$ in train dataset **do**
    $f_{real} \leftarrow seg\_encoder(i)$ {extract real features}
    $\mu, \sigma \leftarrow image\_encoder(i)$ {get noise parameters}
    $z \sim \mathcal{N}(\mu, \sigma)$ {sample random vector}
    $f_{fake} \leftarrow G(z, m)$ {generate fake features}
    $p_{real} \leftarrow D(f_{real}, m)$ {discriminate real features}
    $p_{fake} \leftarrow D(f_{fake}, m)$ {discriminate fake features}
    if $e < freezed\_epochs$ **then**
      {train generator}
      $L_{KLD} = \mathcal{D}_{KL}(z \parallel \mathcal{N}(0, 1))$ {KLD loss for image encoder}
      $L_G \leftarrow -p_{fake}$ {hinge loss for generator}
      $L_{FM} = \sum_{i=1}^{T} \left[\left\| D^{(i)}(f_{real}) - D^{(i)}(f_{fake}) \right\|_1 \right]$ {FM loss for generator}
      $L_{TOT} \leftarrow L_{KLD} + L_G + L_{FM}$
      back-propagate L_{TOT}
      update generator parameters
    else
      {train generator and segmentation model}
      $L_{KLD} = \mathcal{D}_{KL}(z \parallel \mathcal{N}(0, 1))$ {KLD loss for image encoder}
      $L_G \leftarrow -mean(p_{fake})$ {hinge loss for generator}
      $L_{FM} = \sum_{i=1}^{T} \left[\left\| D^{(i)}(f_{real}) - D^{(i)}(f_{fake}) \right\|_1 \right]$ {FM loss for generator}
      $m_{real} \leftarrow seg\_decoder(f_{real})$ {output mask for real features}
      $m_{fake} \leftarrow seg\_decoder(f_{fake})$ {output mask for fake features}
      $L_{CE,real} = -\frac{1}{HW} \sum_{h=1}^{H} \sum_{w=1}^{W} \sum_{c=1}^{C} m_{h,w,c} \log(m_{real_{h,w,c}})$
      $L_{CE,fake} = -\frac{1}{HW} \sum_{h=1}^{H} \sum_{w=1}^{W} \sum_{c=1}^{C} m_{h,w,c} \log(m_{fake_{h,w,c}})$
      $L_{TOT} \leftarrow L_{KLD} + L_G + L_{FM} + L_{CE,real} + L_{CE,fake}$
      back-propagate L_{TOT}
      update generator and segmentation decoder parameters
    end if
    {train discriminator}
    $L_{D,real} = -\left[\min\left(0, -1 + p_{real}\right)\right]$ {hinge loss for real features}
    $L_{D,fake} = -\left[\min\left(0, -1 - p_{fake}\right)\right]$ {hinge loss for fake features}
    $L_{TOT} \leftarrow L_{D,real} + L_{D,fake}$
    back-propagate L_{TOT}
    update discriminator parameters
  end for
end for

4

Experiments and Results

This chapter presents the conducted experiments and the obtained results.

4.1 Implementation Details

The project was implemented using PyTorch, with additional libraries such as Torchvision and TensorBoard to facilitate result visualization and performance tracking. Experiments were conducted on a high-performance NVIDIA TITAN RTX GPU, providing 24 GB of memory, which enabled efficient processing of large batch sizes and complex model architectures.

For this study, the Pascal VOC 2012 dataset was used, comprising 2913 images and corresponding segmentation masks, which was split into training, validation, and test sets with ratios of 0.7, 0.15, and 0.15, respectively. To initialize the segmentation model, we used the pretrained weights provided by Torchvision, specifically for the model's backbone trained on ImageNet. These initial weights provided a stable starting point for feature extraction, essential for effective fine-tuning in semantic segmentation tasks and consequently for training the FeatureGAN. Additionally, all other model components, including the generator and discriminator, were initialized using the Xavier initialization method to ensure balanced weight distribution at the start of training.

In training the segmentation model and GAN, several key hyper-parameters were set based on preliminary experiments and prior research. The generator in the GAN was optimized using a learning rate of 2×10^{-4}, while the discriminator was set at a higher learning rate of 8×10^{-4}. This setup helped maintain stable

adversarial training by preventing the generator from overpowering the discriminator. For the segmentation model, a learning rate of 5×10^{-4} was applied. A batch size of 32 was used to ensure computational efficiency and generalization. The FeatureGAN is pretrained for a number of 75 epochs before training also the segmentation model.

A peculiarity of the training strategy involved freezing the segmentation models backbone for the entire duration of the training. This approach ensured stability in the feature space, which was critical to allow the GAN to adapt effectively to the features extracted. Without freezing, ongoing adjustments in the feature space would make it challenging for the GAN to keep up as described later in Subsection 5.2.4.

Loss functions were weighted to balance the training dynamics.
The Kullback-Leibler Divergence (KLD) loss was weighted at 0.05, following the original SPADE paper setup. However, the Feature Matching (FM) loss was adjusted to 7.0, slightly lower than in the original implementation, to avoid the generator overpowering the discriminator.

An initial image encoder was sometimes employed in early experiments, though it was omitted in most of the final setups to streamline the model architecture.

The study focused on comparing segmentation model performance with and without feature-space data augmentation through GANs. This allowed for insights into how GAN-driven feature augmentation impacts model performance, particularly in scenarios where the segmentation models backbone is frozen.

4.2 Qualitative Results

In this section, we present the qualitative results. We begin by comparing the distribution of real extracted features with that of the generated fake features in the latent space, followed by a visual comparison between the two.

4.2.1 UMAP Visualization of Real and Fake Features

In this subsection, we utilize Uniform Manifold Approximation and Projection (UMAP) to visualize the distributions of extracted real features and generated fake ones. UMAP is a powerful dimension reduction technique that allows

us to project high-dimensional data into a lower-dimensional space while preserving the local and global structure of the data.

By applying UMAP, we can effectively analyze how the real and synthetic features are distributed in the latent space. This visualization helps us identify patterns and relationships between the features, providing insights into the performance of the generator in producing effective representations. Furthermore, this approach facilitates a comparative analysis, allowing us to assess the similarities and differences between real extracted features and the generated ones, ultimately guiding us in refining our model.

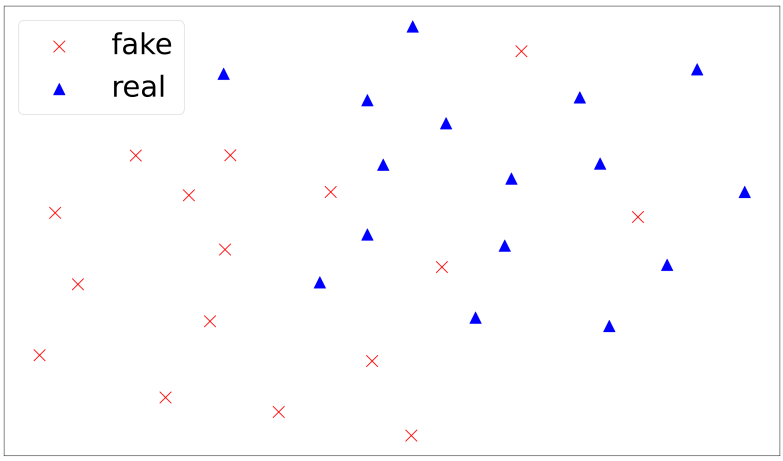
From the figure 4.1, it is clear that at the beginning of GAN training (figure 4.1a), the generated features have a very different distribution compared to the extracted features. However, after GAN training (figure 4.1b), these two distributions begin to blend together and become more difficult to distinguish.

4.2.2 Visual Representation of Features

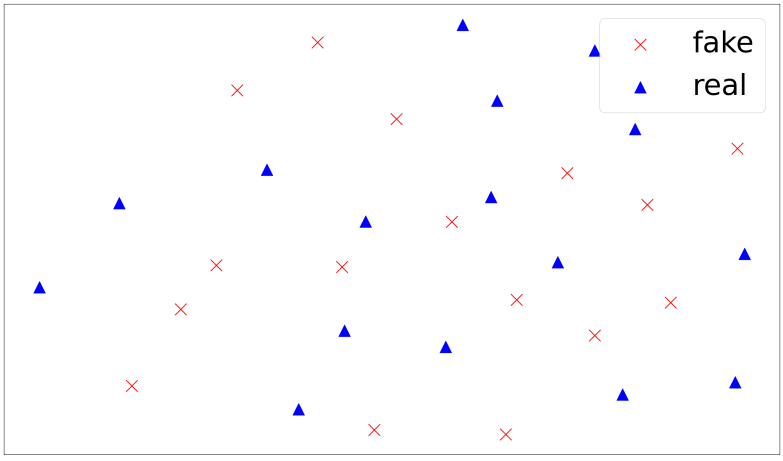
In this subsection, we provide a visual representation of the generated and real features, enabling a direct comparison between the characteristics of features produced by the GAN and those extracted from the actual data.

To produce this visualization, we use a function that manipulates the shape of the feature data. Given an original feature shape of $2048 \times 32 \times 32$, we start by sub-sampling along the channel dimension, reducing the feature size to $\frac{2048}{n\_subsampling} \times 32 \times 32$. We then reshape this output by stacking the channels along the vertical dimension, resulting in a shape of $\left(\frac{2048}{n\_subsampling} * 32\right) \times 32$. Finally, we create a grid by arranging the reshaped features from an entire batch, making it easier to observe their spatial distribution.

In figure 4.2, it is shown the visual representations of the real extracted features from the original images (figure 4.2a) and the synthetic features generated with the corresponding masks as condition factor (figure 4.2b). Notably, the generated features closely resemble the real ones, yet they retain enough variation to provide effective data augmentation.

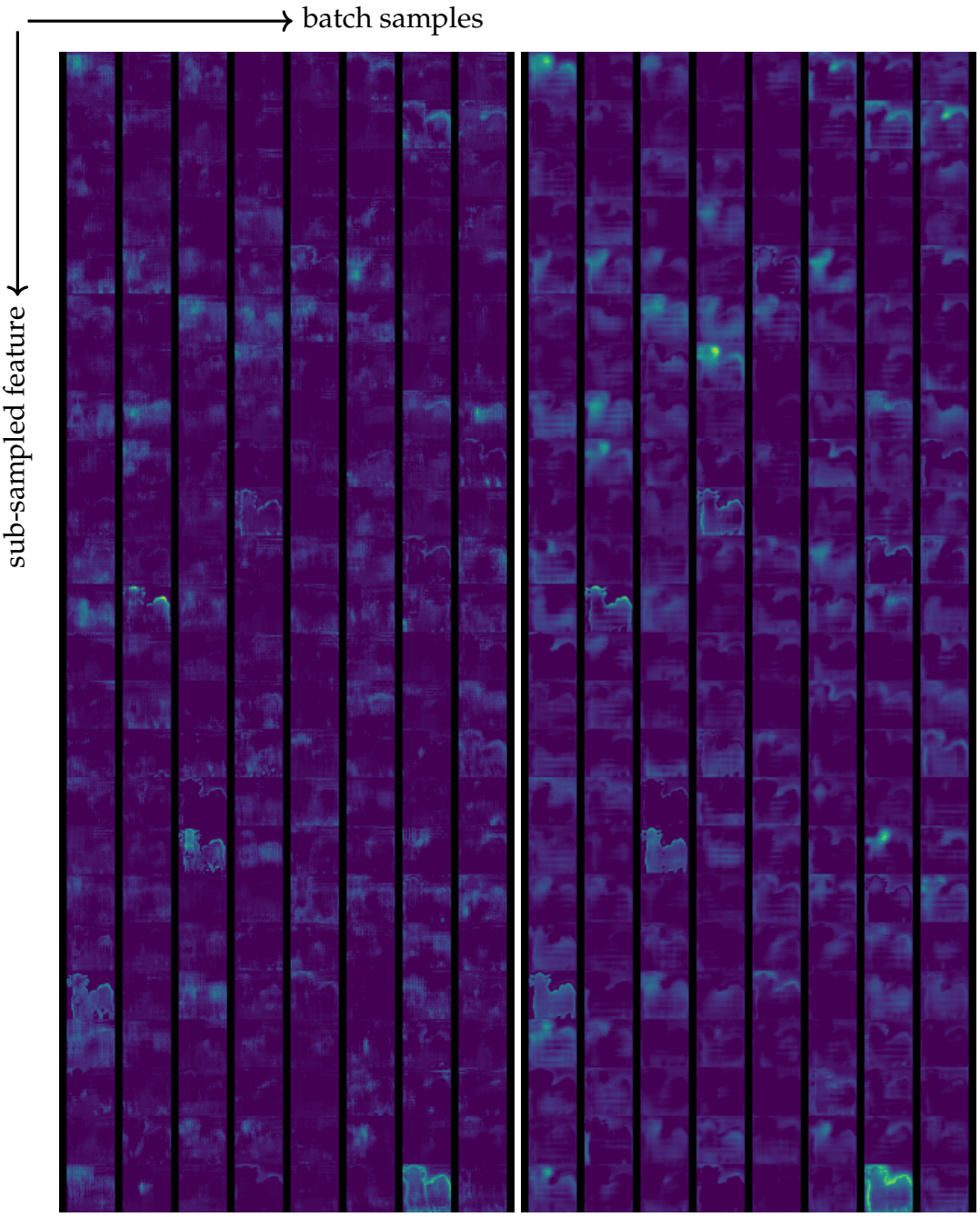


(a) Visualization of UMAP distributions of features before the GAN Training.



(b) Visualization of UMAP distributions of features after the GAN training.

Figure 4.1: Comparison between UMAP distributions of real extracted and generated features before and after the GAN training. Red cross marks represent synthetic feature samples, whereas blue triangles indicate real extracted feature samples. It is easy to observe that, with training, the distributions of real and synthetic features blend together.



(a) Visual representation for real extracted features.

(b) Visual representation for synthetic features.

Figure 4.2: Comparison between the visual representations of real extracted and synthetic features. Each individual sub-sampled feature is displayed along the vertical axis, while the batch samples are arranged horizontally.

4.3 Quantitative Results

After various experiments, the final proposed method achieve the results shown in the table 4.1. The performance gains are evident in both mIoU and pixel accuracy, with an absolute increase of 1.4% in mIoU using a 1:1 ratio of synthetic to real features, and a further improvement to 1.48% in mIoU with a 2:1 ratio. This 2:1 ratio emphasizes that the generated features are suitable for further enhancing the generalization of the segmentation models decoder.
These improvements are similar to those obtained by ISDA, which reported a 1.03% increase in mIoU on the Cityscapes dataset using DeepLabV3. However, our method demonstrates a slightly higher mIoU increase, suggesting a more effective utilization of synthetic features for improving segmentation performance.

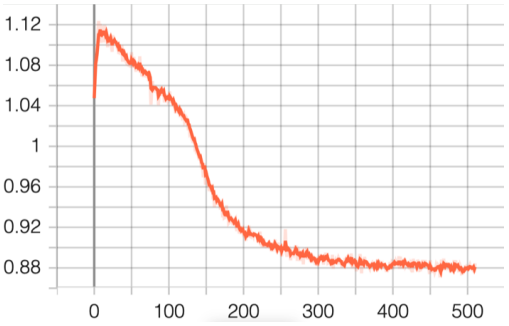
Experiment	PA	mIoU
Training DeepLabV3 w/o FeatureGAN	84.55%	58.64%
Training DeepLabV3 w/ FeatureGAN	84.88%	60.04%
Training DeepLabV3 w/ 2:1 FeatureGAN	**84.99**%	**60.12**%

Table 4.1: Results of final model. "2:1 FeatureGAN" denotes that DeepLabV3 training utilizes a ratio of two synthetic feature examples for every one real feature example. Bold numbers indicate best results.

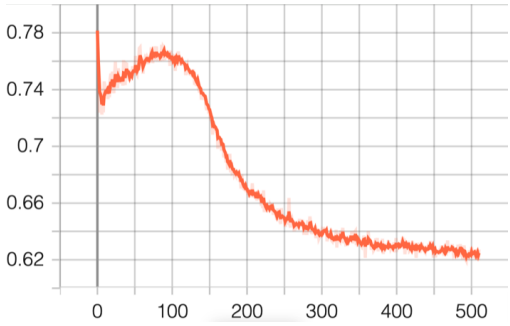
The progression of the GAN losses is shown in Figure 4.3. It's evident that, as expected from the hinge loss, the discriminator loss for fake instances and the generator loss converge to a similar value of 0.88. The contribution of the segmentation loss to the GAN training is also evident, as after 75 epochs the losses start to converge more effectively, highlighting the added benefit of the segmentation loss in the training process.

4.4 Hyper-parameter Tuning

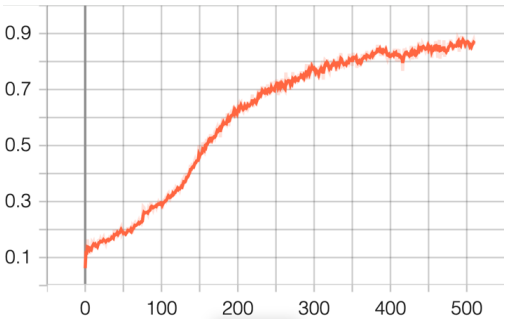
Throughout the development of this thesis project, numerous experiments were conducted. This section presents the most significant ones, starting from the earliest trials to the most recent, to illustrate the progression toward the final result. Additional studies are presented in chapter 5.



(a) Hinge training loss for discriminator on synthetic features.



(b) Hinge training loss for discriminator on synthetic features.



(c) Hinge training loss for generator.

Figure 4.3: GAN loss progression of the final method.

The first relevant experiment used a generator loss function that did not yet include the segmentation loss, and no hyper-parameter tuning had been performed for parameters like the weight for the feature matching loss. As a result, the GAN struggled to converge effectively. However, despite these limitations, a slight improvement was observed, with mIoU and pixel accuracy increasing by less than half a percentage point like shown in table 4.2. For this reason, efforts continued to identify a loss function and hyper-parameter settings that would enable the GAN to train more effectively.

Experiment	PA	mIoU
Training DeepLabV3 w/o FeatureGAN	84.55%	58.64%
Training DeepLabV3 w/ FeatureGAN	84.62%	58.98%

Table 4.2: Results of first experiment.

Therefore, the search continued to improve the generator's loss by creating the loss function described in sections 3.2 and 3.3, which also includes the seg-

mentation loss. This led to a second significant experiment (visible in the orange graph of figure 5.2), where an actual improvement in the GANs convergence was observed, raising optimism for the newly implemented loss function.

It could be argued that, to improve the GAN's convergence by limiting the generator's power, the discriminator could have been trained for more iterations than the generator. However, this approach was avoided because, after the initial pretraining phase of the GAN, both the generator and the segmentation model decoder are trained simultaneously in each step (for efficiency reasons). Thus, to maintain consistency, it was decided to avoid alternating between training the generator every n iterations and training it every iteration afterward.
For this reason, the same convergence goal for the GAN was achieved by tuning the λ_{feat} weight of the feature matching loss. The effects of λ_{feat} are shown in figure 5.2. In particular, a λ_{feat} value of 7.0 is finally used, which is decreased with respect to the original SPADE implementation.

Another important fact that emerged from the numerous experiments conducted is that increasing the number of pretraining epochs for the GAN is counterproductive for the final performance of the segmentation model as shown in table 4.3. This finding further highlights the importance of the newly implemented GAN loss. In the end, the number of pretraining epochs is set to 75.

Experiment	PA	mIoU
Pretraining of 200 epochs	84.54%	58.52%
Pretraining of 120 epochs	84.76%	59.55%
Pretraining of 75 epochs	84.88%	60.04%

Table 4.3: Results of experiment with different number of pretraining epochs.

5

Conclusion

The aim of this thesis was to enhance segmentation performance despite a limited amount of data. Given the constraints of working with limited labeled data, this research sought to explore innovative augmentation strategies within the feature space using a generative model, focusing specifically on improving semantic segmentation accuracy. By adopting a feature generation approach rather than traditional image-based augmentation, the model aimed to leverage the internal structure of the segmentation model more effectively. The model used a DeepLabV3 architecture as the segmentation model and a modified version of SPADE to generate features conditioned on segmentation masks. The results obtained on the PASCAL VOC dataset, using a relatively small set composed by 2913 examples, are promising and highlight the fundamental contribution of the custom loss implemented to enable effective GAN training. Unlike conventional augmentation methods, this approach directly enhances feature-level representations, which can be particularly useful in applications where gathering large, annotated datasets is costly or impractical.

During this research, several challenges were encountered, including training the GAN properly and designing both the generator and discriminator models. One of the most significant challenges was achieving stable GAN training. The modified SPADE architecture required careful tuning to prevent mode collapse and to maintain diversity in the generated features. Several attempts with different hyper-parameter settings were necessary before achieving satisfactory stability. Additionally, the limited scope of this research (testing exclusively on the PASCAL VOC dataset) may not fully capture the variety of real-world scenar-

ios. Despite these challenges, and limitations such as freezing the segmentation models backbone, the results indicate that this approach has potential. Future studies could evaluate this approach on larger and more diverse datasets and on other kind of segmentation models to verify its applicability.

This work opens several avenues for future research.
One promising direction would be to implement this method unfreezing the segmentation model backbone. This certainly involves a change in the training strategy, making the training process even more computationally complex, but it is likely to lead to better results in terms of semantic segmentation performance.
Another potential improvement could involve extending feature space augmentation to other parts of the network, such as multi-layer augmentation, which may further diversify the representations learned by the model.
Future studies could also evaluate the performance of the proposed method in contexts with very limited labeled data, such as in few-shot learning.
It might also be interesting to investigate the use of a different generative model to generate the features, such as another type of GAN or even a diffusion model.

Overall, this thesis underscores the value of innovative data augmentation strategies and their potential in advancing machine learning models, especially for tasks like semantic segmentation where traditional data collection and labeling processes are both costly and time-intensive. This research provides a framework that could inspire similar methodologies across various machine learning applications, especially those constrained by limited data. By leveraging a GAN to augment in the feature space, this work could serve as a foundation for future experiments that aim to maximize data efficiency.

References

[1] Alberto Bacchin et al. "WasteGAN: Data Augmentation for Robotic Waste Sorting through Generative Adversarial Networks". In: *arXiv preprint arXiv:2409.16999* (2024).

[2] A Chen and C Asawa. "Going beyond the bounding box with semantic segmentation". In: *The Gradient* (2018).

[3] Liang-Chieh Chen. "Rethinking atrous convolution for semantic image segmentation". In: *arXiv preprint arXiv:1706.05587* (2017).

[4] Liang-Chieh Chen et al. "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs". In: *IEEE transactions on pattern analysis and machine intelligence* 40.4 (2017), pp. 834–848.

[5] Liang-Chieh Chen et al. "Encoder-decoder with atrous separable convolution for semantic image segmentation". In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 801–818.

[6] Xi Chen et al. "Infogan: Interpretable representation learning by information maximizing generative adversarial nets". In: *Advances in neural information processing systems* 29 (2016).

[7] Tsz-Him Cheung and Dit-Yan Yeung. "Modals: Modality-agnostic automated data augmentation in the latent space". In: *International Conference on Learning Representations*. 2020.

[8] Marius Cordts et al. "The cityscapes dataset". In: *CVPR Workshop on the Future of Datasets in Vision*. Vol. 2. 2015, p. 1.

[9] Jia Deng et al. "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[10] Terrance DeVries and Graham W Taylor. "Dataset augmentation in feature space". In: *arXiv preprint arXiv:1702.05538* (2017).

[11] Mark Everingham et al. "The pascal visual object classes (voc) challenge". In: *International journal of computer vision* 88 (2010), pp. 303–338.

[12] Alberto Garcia-Garcia et al. "A review on deep learning techniques applied to semantic segmentation". In: *arXiv preprint arXiv:1704.06857* (2017).

[13] Ian Goodfellow et al. "Generative adversarial nets". In: *Advances in neural information processing systems* 27 (2014).

[14] Yanming Guo et al. "A review of semantic segmentation using deep neural networks". In: *International journal of multimedia information retrieval* 7 (2018), pp. 87–93.

[15] Shijie Hao, Yuan Zhou, and Yanrong Guo. "A brief survey on semantic segmentation with deep learning". In: *Neurocomputing* 406 (2020), pp. 302–321.

[16] Phillip Isola et al. "Image-to-image translation with conditional adversarial networks. arXiv e-prints". In: *arXiv preprint arXiv:1611.07004* 1611 (2016).

[17] Diederik P Kingma. "Auto-encoding variational bayes". In: *arXiv preprint arXiv:1312.6114* (2013).

[18] Robert Krajewski, Tobias Moers, and Lutz Eckstein. "VeGAN: Using GANs for augmentation in latent space to improve the semantic segmentation of vehicles in images from an aerial perspective". In: *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2019, pp. 1440–1448.

[19] M Krithika Alias AnbuDevi and K Suganthi. "Review of semantic segmentation of medical images using modified architectures of UNET". In: *Diagnostics* 12.12 (2022), p. 3064.

[20] Varun Kumar et al. "A closer look at feature space data augmentation for few-shot intent classification". In: *arXiv preprint arXiv:1910.04176* (2019).

[21] Joseph Lemley, Shabab Bazrafkan, and Peter Corcoran. "Smart augmentation learning an optimal data augmentation strategy". In: *Ieee Access* 5 (2017), pp. 5858–5869.

[22] Tsung-Yi Lin et al. "Microsoft coco: Common objects in context". In: *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*. Springer. 2014, pp. 740–755.

[23] Jonathan Long, Evan Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 3431–3440.

[24] Alhassan Mumuni and Fuseini Mumuni. "Data augmentation: A comprehensive survey of modern approaches". In: *Array* 16 (2022), p. 100258.

[25] Taesung Park et al. "Semantic image synthesis with spatially-adaptive normalization". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 2337–2346.

[26] Simon J.D. Prince. *Understanding Deep Learning*. The MIT Press, 2023. URL: `http://udlbook.com`.

[27] Alec Radford. "Unsupervised representation learning with deep convolutional generative adversarial networks". In: *arXiv preprint arXiv:1511.06434* (2015).

[28] Ting-Chun Wang et al. "High-resolution image synthesis and semantic manipulation with conditional gans". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 8798–8807.

[29] Yulin Wang et al. "Implicit semantic data augmentation for deep networks". In: *Advances in Neural Information Processing Systems* 32 (2019).

[30] Yulin Wang et al. "Regularizing deep networks with semantic data augmentation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44.7 (2021), pp. 3733–3748.

[31] Shuo Yang, Lu Liu, and Min Xu. "Free lunch for few-shot learning: Distribution calibration". In: *arXiv preprint arXiv:2101.06395* (2021).

[32] Qi Zhang. "A novel ResNet101 model based on dense dilated convolution for image classification". In: *SN Applied Sciences* 4 (2022), pp. 1–13.

[33] Bolei Zhou et al. "Scene parsing through ade20k dataset". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 633–641.

Additional Studies

In this chapter, we present alternative approaches that illustrate the path to the final implementation, along with ablation studies to assess the importance of each component of the final model.

5.1 Alternative Approaches

Throughout the development of this thesis project, three distinct approaches were explored to arrive at the final method described in Chapter 3.

The first approach involves offline feature extraction using the segmentation model's encoder, followed by training a cGAN conditioned on segmentation masks from scratch. The goal is to generate synthetic features resembling the extracted ones, which are then used to train the segmentation model's decoder. However, this method proves challenging due to the high likelihood of mode collapse in the synthetic features and difficulties in conditioning on segmentation masks.

The second approach focuses on pretraining a full SPADE model to generate synthetic images from segmentation masks. Once trained, the SPADE generator is placed at the head of the segmentation model's encoder to generate synthetic images, which are then passed through the encoder to obtain synthetic features. These features are used to train the segmentation model's decoder, while a discriminator works in parallel to differentiate between real and synthetic features. However, this approach has shown some limitations, as pretraining the entire SPADE model requires a large dataset (which is not always available), and the quality of the generated images is not consistently high. Additionally, while discrimination occurs at the feature level, generation is still done at the image level.
The model architecture of this approach is represented in figure 5.1

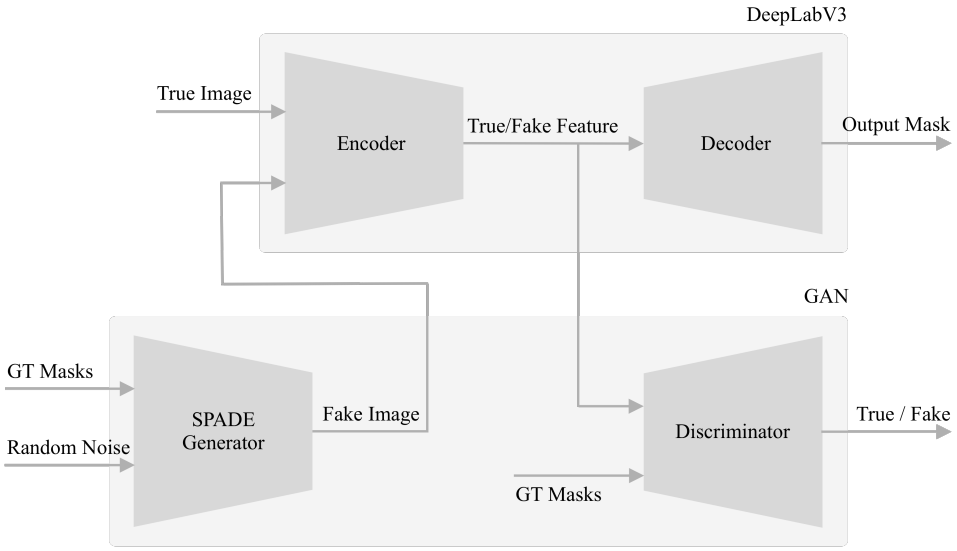


Figure 5.1: Second approach model architecture. The SPADE image encoder is omitted for sake of clarity.

5.2 Ablation Study

During the development of the project, numerous experiments were conducted with different parameter configurations and settings. Some of the results obtained from these adjustments are presented below to validate the proposed final method of chapter 3.

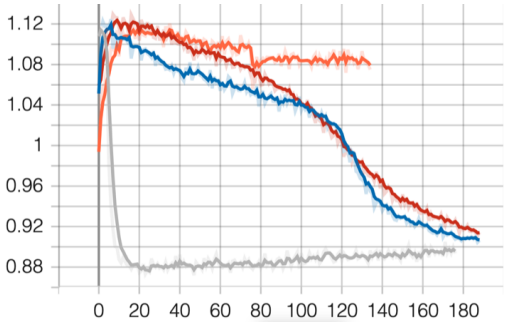
5.2.1 Image Encoder

The use of the image encoder was found to be non-essential, as there was a decrease of 0.97% in mIoU and 0.01% in pixel accuracy compared to the final model. This may be due to having a more streamlined generator, which is therefore easier to train.

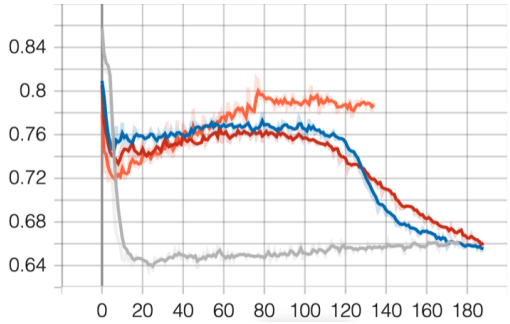
5.2.2 Feature Matching Loss Weight

In Figure 5.2, the GAN loss curves are shown for different values of the λ_{feat} parameter, which weights the feature matching loss. The gray, blue, red, and orange lines represent the losses with λ_{feat} values of 2.0, 5.0, 7.0, and 10.0, respectively. It can be observed that with λ_{feat} set to 10.0 (as in the original SPADE work), the GAN fails to converge because the generator quickly becomes

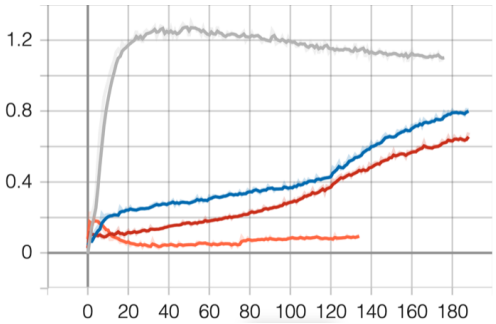
too strong, preventing the discriminator from improving. Conversely, reducing λ_{feat} decreases the generators initial ability to deceive the discriminator, leading to more effective GAN training. On the other hand, if λ_{feat} is set too low (for example 2.0, gray line), the generator struggles to learn the distribution of real features.



(a) Hinge training loss for discriminator on synthetic features.



(b) Hinge training loss for discriminator on synthetic features.



(c) Hinge training loss for generator.

Figure 5.2: Feature matching loss weight effects on GAN training. The gray, blue, red, and orange lines represent the losses with λ_{feat} values of 2.0, 5.0, 7.0, and 10.0, respectively.

5.2.3 Hinge Loss

Before implementing the hinge loss, the Wasserstein loss was used, both with weight clipping and with gradient penalty. However, this led to challenging convergence during GAN training, resulting in data augmentation in the feature space that was entirely ineffective and even counterproductive.

5.2.4 Freezed Encoder

Two additional experiments were conducted with the backbone of the segmentation model unfrozen: the first without data augmentation and the second with data augmentation. The results obtained are reported in table 5.1. It is evident that with the unfrozen backbone, the data augmentation applied is not effective at all, as the latent space of the features to be generated changes significantly with each update of the backbone, making it difficult for the GAN to keep up with these changes.
For this reason, the decision was made to freeze the backbone and train only the decoder of the segmentation model. This approach keeps the latent feature space stable, allowing for a simplified GAN training process.

Experiment	PA	mIoU
Training DeepLabV3 w/o FeatureGAN	87.92%	67.65%
Training DeepLabV3 w/ FeatureGAN	86.20%	60.90%

Table 5.1: Results of experiments with unfreezed backbone with and without FeatureGAN.

Acknowledgments

As I reach the conclusion of this work, I would like to express my gratitude to all the professors I had the pleasure of meeting during this masters degree journey. Their passion for their fields instilled in me a curiosity that became the driving force, motivating me to complete my studies with the same interest and commitment.
First and foremost, I would like to thank Professor Emanuele Menegatti for giving me the opportunity to work in a scientific laboratory and apply my studies in a research area that challenged and inspired me.
I would also like to thank Dottor Alberto Bacchin for guiding me through this thesis journey with great dedication, offering not only invaluable advice but also continuous support and encouragement.
I am deeply grateful to my family for giving me the privilege of pursuing my studies without pressure.
Lastly, I would like to thank Caterina, my life partner, for her patience and support, especially during the most challenging times.