Using Neural Networks for the Domain Adaptation of Synthetic Generated Document Images

Giriraj Sukumar Pawar

March 19, 2021

Contents

List of Figures

List of Tables

List of Abbreviations

GAN	Generative Adversarial Network
CNN	Convolutional Neural Network
cGAN	Conditional Adversarial Network
LSGAN	Least Squares Generative Adversarial Network
CycleGAN	Cycle-Consistent Adversarial Network
GT	Ground Truth
DIBCO	Document Image Binarization Competition
PSNR	Peak Signal-to-Noise Ratio
ResNet	Residual Network
AMT	Amazon Mechanical Turk
CUT	Contrastive Unpaired Translation
MUNIT	Multimodal Unsupervised Image-to-image Translation
DRIT	Diverse Image-to-Image Translation
GCGAN	Geometry-Consistent Generative Adversarial Networks
FastCUT	Fast Contrastive Unpaired Translation
FID	Fréchet inception distance

Chapter 1

Introduction

In recent years, deep learning methods have shown great effectiveness in many fields, including natural language processing and computer vision. But such kind of methods are still limited by poor generalization and over-fitting due to the insufficient quantity and quality of training data. Annotated data are scarce when it comes to developing deep learning models for computer vision applications. The performance of such deep learning models can be improved with the introduction of a large amount of annotated data. However, it is hard to obtain due to the high cost of human resources for data annotation, specifically in the case of numerous variants of data. To overcome the obstacle of the scarcity of annotated data, it is obvious that synthetic data is required to be generated. But deep learning models trained using synthetic data will not generalise well on real data. Hence, in the last two decades, numerous domain adaptation methodologies have been introduced. Such methods are used to transform synthetic data into realistic data by reducing the divergence between the distribution of real data and the distribution of synthetic data.

One of the most popular examples of domain adaptation technique is GANs. GAN was proposed by Ian J. Goodfellow et al. [11] as an approach to generative modelling. The generative models are estimated via an adversarial process, in which simultaneously two models are trained, the generator model is trained to generate new samples, and the discriminator model tries to classify those samples as either real or fake. The two models are trained together in a minimax two-player game, adversarial, until the discriminator model is fooled about half the time, meaning the generator model is generating plausible samples. As the research in the field of generative modelling has increased numerous extended versions of GANs are invented like Conditional Adversarial Networks (cGANs) [14], Least Squares Generative Adversarial Networks (LSGANs) [28], and Cycle-Consistent Adversarial Networks (CycleGANs) [50]. These are modified versions of GANs are extensively used in the image-to-image translation techniques. For example, translation from horse image to zebra image and from apple image to orange image. The image-to-image translation is a class of computer vision and computer graphics problems where the goal is to learn the mapping between an input image and an output image using a training set of aligned image pairs. However, for many tasks, paired training data will not be available. We need to use an approach for training a deep learning model to translate an image from a source domain to a target domain in the absence of paired examples.

As mentioned above the cGANs are a general-purpose solution for mapping from the input image to the output image. This approach is effective at synthesizing photos from label maps [3], reconstructing objects from edge maps [6], and colorizing images. The LSGANs proposes to adopt the least squares loss function for the discriminator which resolves the vanishing gradient problem during the learning process. In this work, CycleGAN is used to transform synthetic document images into realistic document images. The CycleGANs are state of the art for unpaired image-to-image translation.This work deals with synthetically generated document images, these document images are created using empty template images and handwritten crops retrieved from data sets like MNIST or any other data sets. In this work, we are using a copyrighted dataset, which can not be exposed to public use. Handwritten crops are pasted over the empty template images to generate numerous synthetic document images. But as explained above deep learning models trained using synthetic document images will not generalise well on real document images. Hence, we have used CycleGAN to perform the unpaired image-to-image translation and transform synthetic document images into realistic document images.

The architecture of CycleGAN is adapted from Johnson et al. [15]. For the generator Network, we use a sequence of downsampling convolutional blocks to encode the $256 \times 256 \times 1$ grayscale input image, 9 Residual Network (ResNet) convolutional blocks to transform the image, and a number of upsampling convolutional blocks to generate the output image of the same dimension as the input image. The reason behind using residual blocks is it resolves the vanishing gradient problem in deep neural networks. For discriminator networks, we use PatchGAN [14]. In PatchGAN, after feeding one input image to the network, it gives you the probabilities of two things: either real or fake, but not in scalar output indeed, it used the $N \times N$ output vector. Here $N \times N$ can be different depending on the dimension of an input image. We will discuss the architecture of both generator and discriminator networks in Chapter 5. The images generated by the CycleGAN were evaluated by using simplified version of GAN Quality Index [45]. The idea is to train a classifier from the GAN generated data and use its accuracy on a real test set as a metric to measure how well the GAN model distribution matches the real data distribution. The classification capability of the trained classifier is used as an objective measure to evaluate GAN. The GAN Quality Index is the ratio of the accuracy of a classifier trained using CycleGAN generated data and the accuracy of a classifier trained using synthetic data, given ground truth real data to calculate the accuracy of both classifiers.

The rest of this report is organized as follows. Chapter 2 briefly reviews related works of numerous GANs variants. Chapter 3 discusses the basis of Adversarial Nets, Convolution Neural Networks (CNNs), and other important topics. The decided approach is described in Chapter 4, the implementation details are explained in Chapter 5, and experimental results are presented in Chapter 6. Finally, this work has been concluded in Chapter 7 along with the future work.

~~Chapter 2~~

Related Works

There have been numerous amounts of research in the field of domain adaptation. Several variants of GANs are evolved over the years to resolve various problems. Especially the image-to-image translation methods are improved significantly. Let us have a look into some them in the below paragraphs.

Ian J. Goodfellow et al. [11] proposed a framework of GANs in which two models are simultaneously trained. A generative model that captures the data distribution, and a discriminative model that estimates the probability that a sample came from the training data rather than a generative model. The training procedure for a generative model is to maximize the probability of a discriminative model making a mistake. Basically, the generator learns to generate plausible data. The discriminator learns to distinguish the generator's fake data from real data. The discriminator penalizes the generator for producing implausible results. When training begins, the generator produces fake data, and the discriminator quickly learns to tell that it's fake and the generator penalized to produce plausible results. As training progresses, the generator gets closer to producing output that can fool the discriminator. Finally, if generator training goes well, the discriminator gets worse at telling the difference between real and fake. At the end of the training, eventually, we have a generator model which produces plausible results which are similar to real data. Authors have trained GAN an a range of datasets including MNIST [19], the Toronto Face Database (TFD) [39], and CIFAR-10 [16]. Also compared against already existing methods like DBN [1], Stacked CAE [1], and Deep GSN [2] . Authors do not claim that the samples generated by GANs are better than samples generated by already existed methods, authors believe that these samples are at least competitive with the better generative models in the literature and highlight the potential of the generative adversarial framework. The advantage of using GANs are primarily computational. Adversarial models may also gain some statistical advantage from the generator network not being updated directly with data examples, but only with gradients flowing through the discriminator using backpropagation. Special care should be taken during training the GANs, the generator must not be trained too much without updating the discriminator, to avoid the Helvetica Scenario [27] in which the generator collapses to produce the same output (or a small set of outputs) over and over again. Usually, GANs should produce a wide variety of outputs. The Helvetica Scenario is also called Mode Collapse [42].

Xudong Mao et al. [28] proposed another variant of GANs called Least Squares Generative Adversarial Networks (LSGANs). The Regular GANs hypothesize the discriminator as a classifier with the sigmoid cross-entropy loss function. However, they found that the cross-entropy loss function may lead to the vanishing gradients problem during the learning process. To overcome such a problem, the authors proposed the LSGANs which adopts the least-squares loss function for the discriminator. The least-squares loss function penalizes the fake samples and forces the generator to generate samples toward the decision boundary. The authors evaluated LSGANs using two datasets LSUN [47] and HWDB1.0 [24] (Handwritten Chinese Character Dataset). When trained on the LSUN dataset [47] they observed, the images generated by LSGANs are of better quality than the ones generated by the two baseline methods, DCGANs [32] and EBGANs [48] . Also, when trained on HWDB1.0 [24] (Handwritten Chinese Character Dataset), the generated characters were readable and clear. Another experiment was conducted to evaluate the stability of LSGAN on a Gaussian

mixture distribution dataset, which is designed in literature [29]. They train LSGANs and regular GAN on a 2D mixture of 8 Gaussian datasets using a simple architecture, where both the generator and the discriminator contain three fully-connected layers. It is observed that regular GANs suffer from mode collapse. GANs generate samples around a single valid mode of the data distribution. But LSGANs learn the Gaussian mixture distribution successfully. In this paper, numerous comparison experiments for evaluating the stability are conducted and the results demonstrate that LSGANs can generate higher quality images than regular GANs, DCGANs [32], and EBGANs [48] and perform more stable than regular GANs during the learning process.

Phillip Isola et al. [14] proposed cGANs as a generic solution to numerous image-to-image translation problems. The authors wanted to provide a single solution for multiple types of image-to-image translation problems. For example, synthesizing photos from label maps [3], reconstructing objects from edge maps [49] [46] , and colorizing images. cGANs not only learn the mapping from input image to output image, but also learn a loss function to train this mapping. This makes it possible to apply the same generic approach to problems that traditionally would require very different loss formulations. Authors implemented cGAN and released it as the pix2pix software to solve distinct image-to-image translation problems. This software is popular among a large number of internet users, many of them are artists, because of its wide applicability and ease of adoption without the need for parameter tweaking. In cGAN the generator uses U-Net-based architecture [34], the U-Net is an encoder-decoder with skip connections between mirrored layers in the encoder and decoder stacks. The discriminator uses a convolutional PatchGAN classifier [22], which only penalizes structure at the scale of image patches. Unlike unconditional GANs, both the generator and discriminator observe the input images. Authors performed multiple experiments during ablation studies using evaluation metrics like, Amazon Mechanical Turk (AMT) perceptual study, FCN-Score, and semantic segmentation metrics. They found that L1 distance loss encourages less blurring compared to the L2 distance loss. Also combining L1 Loss and cGAN (L1 + cGAN) generates better results compared to combining Unconditional GAN and L1 Loss ((L1 + GAN)). The cGAN appears to be more effective on the problem where the output is highly detailed or photographic. In this work, we are adopting the generator and discriminator architecture from cGANs. The pix2pix software code is available at GitHub.

Taesung Park et al. [31] proposed a framework for encouraging content preservation in unpaired image translation problems by maximizing the mutual information between input and output with patchwise contrastive learning [43]. In the patchwise contrastive learning for image-to-image translation, a generated output patch should appear closer to its corresponding input patch in comparison to other random patches present in the same input. To achieve patchwise contrastive learning, drawing patches internally from within the input image, rather than externally from other images in the dataset, forces the patches to better preserve the content of the input. This method requires neither a memory bank nor specialized architecture. The authors demonstrated that the framework enables one-sided translation in the unpaired image-to-image translation setting while improving quality, consuming less memory, and reducing training time. They call this approach Contrastive Unpaired Translation (CUT). Since contrastive representation is formulated within the same image, this method can even be trained on single images, where each domain is having only a single image. The several prior methods like CycleGAN [50], Multimodal Unsupervised Image-to-image Translation (MUNIT) [25], Diverse Image-to-Image Translation (DRIT) [21], and Geometry-Consistent Generative Adversarial Networks (GCGAN) [7] were unable to achieve significant results compared to the CUT method, on other hand, it often produced higher quality images and more accurate generations with light footprint in terms of training speed and GPU memory usage. Since CUT method is one-sided, it is memory efficient and faster compared to prior baselines. The evaluation metrics like Fréchet inception distance (FID) [12] and semantic segmentation scores are used to compare the quality of generated images using CUT method. Furthermore, the authors also introduced faster and lighter variant Fast Contrastive Unpaired Translation (FastCUT). FastCUT also produces competitive results with even lighter computation cost of training. The code and models for CUT are available at GitHub.

Monika Sharma et al. [37] is addressing a problem in the scanning process that often results in the introduction of salt and pepper noise, blur due to camera motion, or shake, water markings, coffee stains, wrinkles, or faded data. These artifacts pose many readability challenges to current text recognition algorithms and significantly degrade their performance. So, the existing denoising techniques require a dataset comprising of noisy documents paired with cleaned versions of the

same document. However, very often in the real world, such a paired dataset is not available to train a model to generate clean documents from noisy versions. Hence, the authors proposed to use CycleGAN because it is known to learn a mapping between the distributions in the absence of paired training dataset. Using CycleGAN, noisy document images transformed into denoised and clean document images to achieve image-to-image translation. They have compared the performance of CycleGAN for document cleaning tasks with a cGAN by training them over the same dataset. The only difference was CycleGAN trained using unpaired images and cGAN trained using the paired images. They have used Peak Signal-to-Noise Ratio (PSNR)[1] as a evaluation metric to evaluate the quality of transformed denoised images. Several experiments were performed on 4 separate document public document datasets, one each for background noise removal, deblurring, watermark removal, and defading. Finally, they illustrate that CycleGAN learns a more robust mapping from the space of noisy to clean documents compared to cGAN.

Chris Tensmeyer et al. [41] realized solving binarization tasks using deep learning models is very challenging. It is due to the lack of large quantities of labeled data available to train such models. They also mention there have been efforts to create synthetic data for binarization using image processing techniques but, they generally lack realism. In this paper, the authors proposed a method to produce realistic synthetic data using an adversarially trained image translation model. They extended the popular CycleGAN model to be conditioned on the ground truth binarization mask as it translates images from the domain of synthetic images to the domain of real images. They have found that modifying the discriminator to condition on the binarization Ground Truth (GT) leads to increased realism and better agreement between the GT and the produced image. They called the proposed model DGT-CycleGAN. Also shown DGT-CycleGAN model produces more realistic synthetic data. They validated their approach by pretraining deep networks on realistic synthetic datasets generated by DGT-CycleGAN, CycleGAN, and image compositing. They evaluate both pretrained only and finetuned models on each of the Document Image Binarization Competition (DIBCO) datasets. They have concluded that pretraining deep neural networks on the more realistic synthetic data generated using DGT-CycleGAN lead to better predictive performance both before and after finetuning on real data.

Jun-Yan Zhu et al. [50] proposed a modified version of GAN which is the state-of-the-art method for the image-to-image translation that can transform the images from source domain X to target domain Y in the absence of paired training dataset. This method can learn to capture special characteristics of one image collection and figuring out how these characteristics could be translated into the other image collection, all in the absence of any paired training examples. This modified version of GAN is called CycleGAN. In this method, the goal is to learn a mapping $G : X \rightarrow Y$ such that the distribution of images $G(X)$ is indistinguishable from the distribution Y using an adversarial loss. Because this mapping is highly under-constrained and coupled with an inverse mapping $F : Y \rightarrow X$ to introduce a cycle consistency loss to enforce $F(G(X)) \approx X$ and vice versa. Along with an adversarial loss and cycle consistency loss, this work also introduces identity mapping loss which helps to preserve the colour of input images. Authors considered evaluation metrics like AMT perceptual studies, FCN-score [14], and semantic segmentation metrics to compare the quality of generated images against other baseline. The several prior methods like Bi-GAN/ALI [[4], [5]], CoGAN [26], SimGAN [38] were unable to achieve compelling results. The CycleGAN method, on other hand, can produce images that are often of similar quality to the fully supervised pix2pix. [14]. Authors provide both PyTorch and Torch implementations.

As per the above literature survey, CycleGAN is a significant approach to transform synthetic document images into realistic document images as we know finding paired document images are very difficult and expensive. In this work, CycleGAN is used to solve image-to-image problem translation.

[1]Peak Signal-to-Noise Ratio: `http://www.ni.com/white-paper/13306/en/`

~~Chapter 3~~

Fundamentals

This chapter discusses the working principle of GANs and CNNs. The basic architecture of the generator and discriminator is briefly explained in section 3.1. Also, the multiple layers like convolution layer, activation layer, pooling layer, and fully connected layer have briefly explained in section 3.2.

3.1 Working principle of GANs

The GAN has two parts one is a generator and the other is a discriminator. The generator learns to generate plausible data. The generated instances become negative training examples for the discriminator. The discriminator learns to distinguish the generator's fake data from real data. The discriminator penalizes the generator for producing implausible results. When training begins, the generator produces fake data, and the discriminator quickly learns to tell that it's fake. As training progresses, the generator gets closer to producing output that can fool the discriminator. Finally, if generator training goes well, the discriminator gets worse at telling the difference between real and fake. It starts to classify fake data as real, and its accuracy decreases. Eventually generator will start producing plausible data. Both the generator and the discriminator are neural networks. The generator output is connected directly to the discriminator input. Through backpropagation, the discriminator's classification provides a signal that the generator uses to update its weights.

3.1.1 The Discriminator

The discriminator in a GAN is simply a classifier. It tries to distinguish real data from the data created by the generator. It could use any network architecture appropriate to the type of data it's classifying. The discriminator's training data comes from two sources:

1. Real data instances. The discriminator uses these instances as positive examples during training.
2. Fake data instances created by the generator. The discriminator uses these instances as negative examples during training.

In figure 3.2, the two "Sample" boxes represent these two data sources feeding into the discriminator. During discriminator training the generator does not train. Its weights remain constant while it produces examples for the discriminator to train on. The discriminator connects to two loss functions. During discriminator training, the discriminator ignores the generator loss and just uses the discriminator loss. The generator loss is used during generator training. The next section 3.1.2 describes why the generator loss connects to the discriminator. During discriminator training:

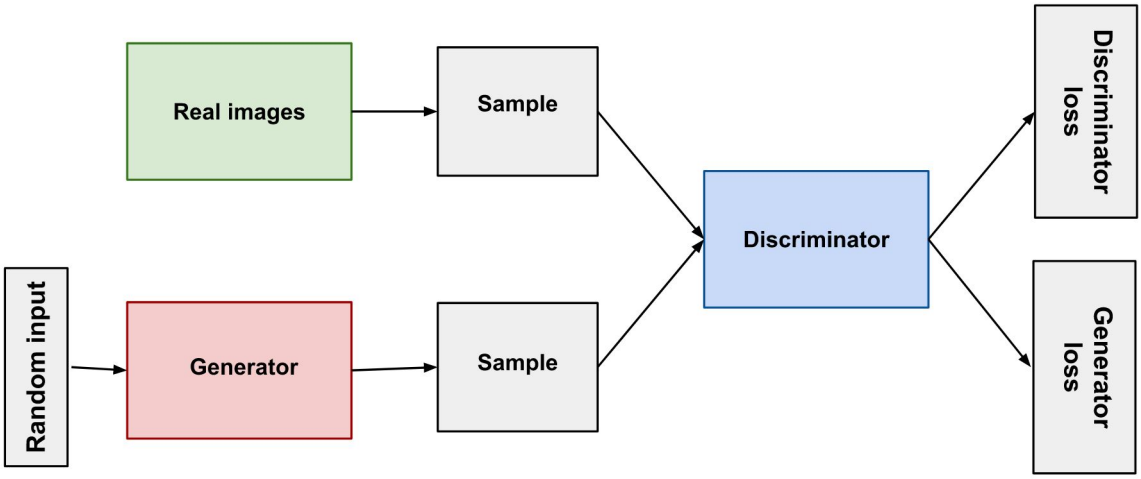

Figure 3.1: Overview of GAN Structure.

1. The discriminator classifies both real data and fake data from the generator.
2. The discriminator loss penalizes the discriminator for misclassifying a real instance as fake or a fake instance as real.
3. The discriminator updates its weights through backpropagation from the discriminator loss through the discriminator network.

3.1.2 The Generator

The generator part of a GAN learns to create fake data by incorporating feedback from the discriminator. It learns to make the discriminator classify its output as real. Generator training requires tighter integration between the generator and the discriminator than discriminator training requires. The portion of the GAN that trains the generator includes:

1. Random input
2. Generator network, which transforms the random input into a data instance.
3. Discriminator network, which classifies the generated data.
4. Discriminator output.
5. Generator loss, which penalizes the generator for failing to fool the discriminator

Neural networks need some form of input data, like an instance for classification or to predict about. But what to use as an input for a network that outputs entirely new data instances? In its most basic form, a GAN takes random noise as its input. The generator then transforms this noise into a meaningful output. By introducing noise, GAN can produce a wide variety of data, sampling from different places in the target distribution. Experiments suggest that the distribution of the noise doesn't matter much, so it is possible choose something easy to sample from, as a uniform distribution. For convenience, space from which the noise is sampled is usually of a smaller dimension than the dimensionality of the output space.

To train a neural net, the net's weights altered to reduce the error or loss of its output. In GAN, however, the generator is not directly connected to the loss that we're trying to affect. The generator feeds into the discriminator net, and the discriminator produces the output we're trying to affect. The generator loss penalizes the generator for producing a sample that the discriminator network classifies

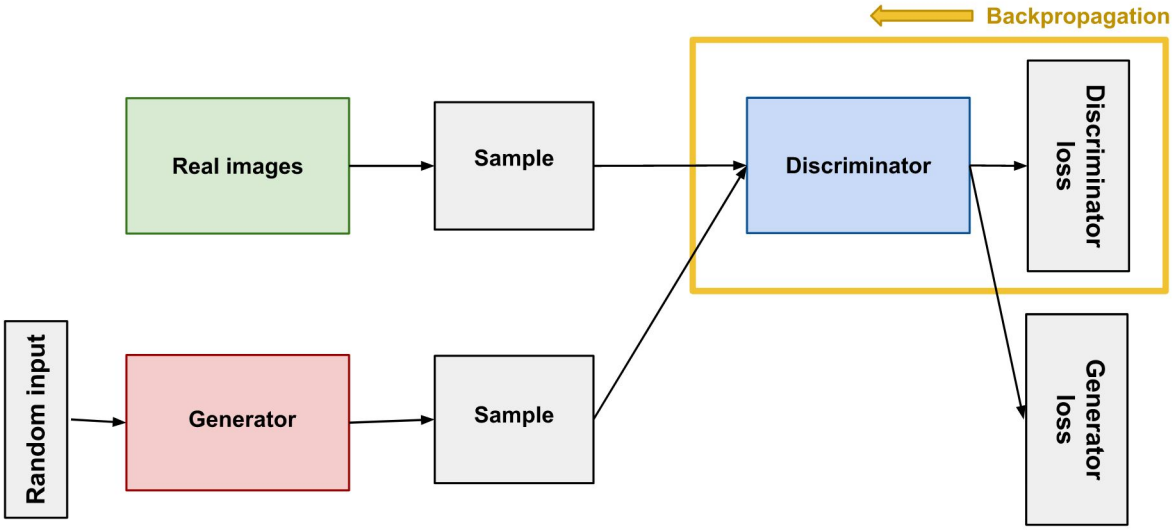

Figure 3.2: Backpropagation in Discriminator Training.

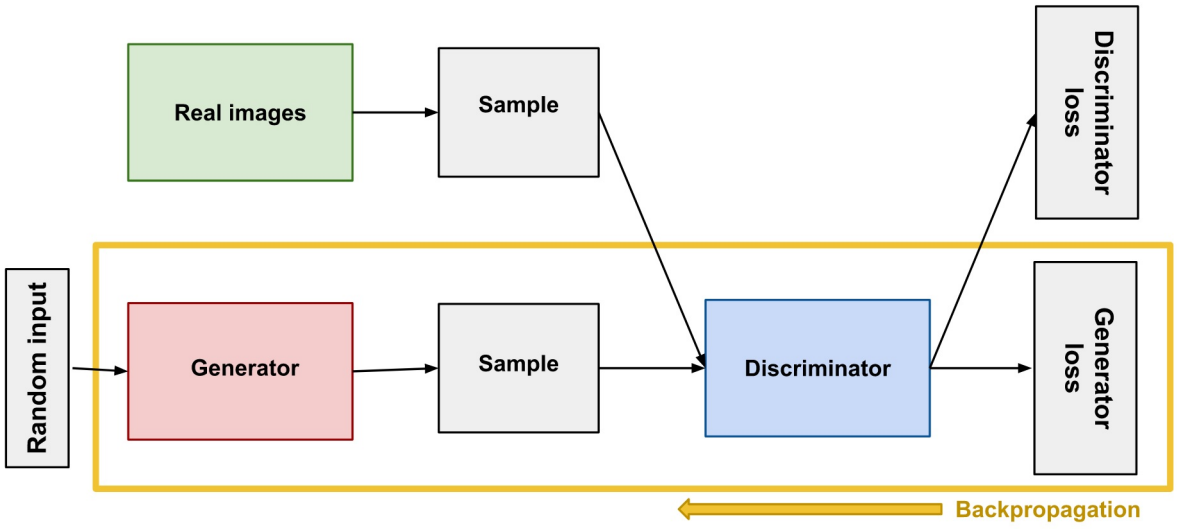

Figure 3.3: Backpropagation in Generator Training.

as fake. This extra chunk of network must be included in backpropagation. Backpropagation adjusts each weight in the right direction by calculating the weight's impact on the output — how the output would change if you changed the weight. But the impact of a generator weight depends on the impact of the discriminator weights it feeds into. So backpropagation starts at the output and flows back through the discriminator into the generator. At the same time, we don't want the discriminator to change during generator training. Trying to hit a moving target would make a hard problem even harder for the generator. The generator is trained with the following procedure:

1. Sample random noise.
2. Produce generator output from sampled random noise.
3. Get discriminator "Real" or "Fake" classification for generator output.
4. Calculate loss from discriminator classification.
5. Backpropagate through both the discriminator and generator to obtain gradients.
6. Use gradients to change only the generator weights.

3.2 Convolution Neural Networks

A tremendous interest in deep learning has emerged in recent years [20]. The most established algorithm among various deep learning models is CNNs, a class of artificial neural networks that has been a dominant method in computer vision tasks since the astonishing results were shared on the object recognition competition known as the ImageNet Large Scale Visual Recognition Competition (ILSVRC) in 2012 [[36], [17]]. CNN is a type of deep learning model for processing data that has a grid pattern, such as images, which is inspired by the organization of animal visual cortex [[13], [8]] and designed to automatically and adaptively learn spatial hierarchies of features, from low-level to high-level patterns. It is composed of multiple building blocks, such as convolution layers[1], pooling layers, and fully connected layers. The first two, convolution and pooling layers, perform feature extraction, whereas the third, a fully connected layer, maps the extracted features into final output, such as classification. Let us have a look into each layer in coming sections.

3.2.1 Convolution Layer

The convolution layer is a fundamental component of the CNN architecture that performs feature extraction, which typically consists of a combination of linear and nonlinear operations, i.e., convolution operation and activation function. It is the core building block of a CNNs that does most of the computational heavy lifting. A convolution layer plays a key role in CNN, which is composed of a stack of mathematical operations, such as convolution, a specialized type of linear operation. In digital images, pixel values are stored in a two-dimensional (2D) grid, i.e., an array of numbers as shown in figure 3.5, and a small grid of parameters called kernel, an optimizable feature extractor, is applied at each image position, which makes CNNs highly efficient for image processing, since a feature may occur anywhere in the image. As one layer feeds its output into the next layer, extracted features can hierarchically and progressively become more complex. The process of optimizing parameters such as kernels is called training, which is performed so as to minimize the difference between outputs and ground truth labels through an optimization algorithm called backpropagation [10] and gradient descent [35], among others.

[1] A convolution is a mathematical operation that slides one function over another and measures the integral of their pointwise multiplication. It has deep connections with the Fourier transform and the Laplace transform, and is heavily used in signal processing. Convolutional layers actually use cross-correlations, which are very similar to convolutions (see `https://homl.info/76` for more details).

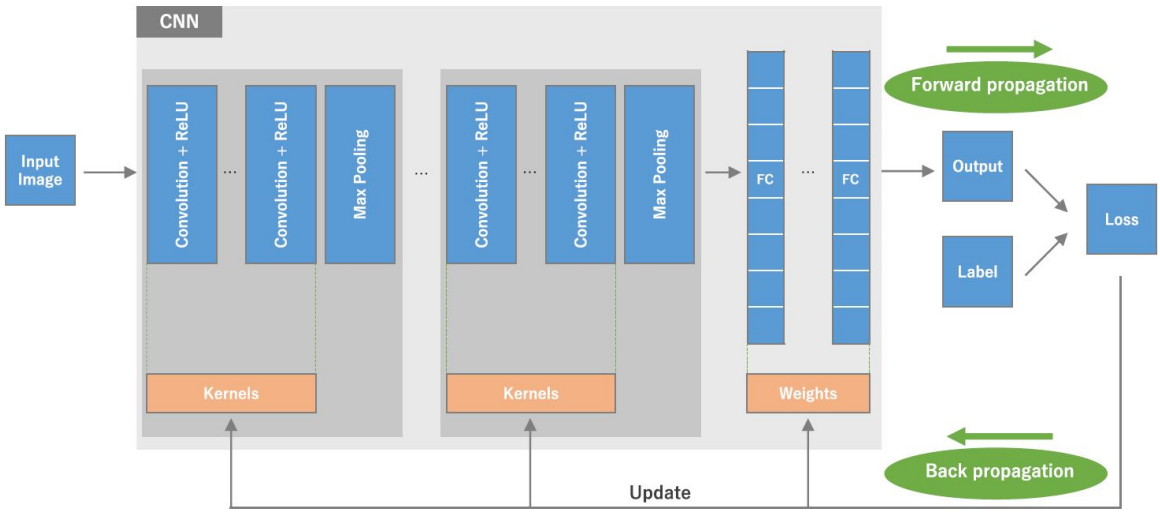

Figure 3.4: An overview of a CNN architecture and the training process. A CNNs is composed of a stacking of several building blocks: convolution layers, pooling layers (e.g., max pooling), and fully connected (FC) layers. A model's performance under particular kernels and weights is calculated with a loss function through forward propagation on a training dataset, and learnable parameters, i.e., kernels and weights, are updated according to the loss value through backpropagation with gradient descent optimization algorithm. The ReLU notation stands for rectified linear unit. It is an activation function [44].

Convolution

Convolution is a specialized type of linear operation used for feature extraction, where a small array of numbers, called a kernel, is applied across the input, which is an array of numbers, called a tensor. An element-wise product between each element of the kernel and the input tensor is calculated at each location of the tensor and summed to obtain the output value in the corresponding position of the output tensor, called a feature map (figure 3.5). This procedure is repeated applying multiple kernels to form an arbitrary number of feature maps, which represent different characteristics of the input tensors; different kernels can, thus, be considered as different feature extractors. Two key hyperparameters that define the convolution operation are size and number of kernels. The former is typically 3×3, but sometimes 5×5 or 7×7. The latter is arbitrary, and determines the depth of output feature maps. The convolution operation described above does not allow the center of each kernel to overlap the outermost element of the input tensor, and reduces the height and width of the output feature map compared to the input tensor. Padding, typically zero padding, is a technique to address this issue, where rows and columns of zeros are added on each side of the input tensor, so as to fit the center of a kernel on the outermost element and keep the same in-plane dimension through the convolution operation (figure 3.6). Modern CNN architectures usually employ zero padding to retain in-plane dimensions in order to apply more layers. Without zero padding, each successive feature map would get smaller after the convolution operation. The distance between two successive kernel positions is called a stride, which also defines the convolution operation. The common choice of a stride is 1; however, a stride larger than 1 is sometimes used in order to achieve downsampling of the feature maps. An alternative technique to perform downsampling is a pooling operation, as described below in section 3.2.2.

Nonlinear Activation Function

The outputs of a linear operation such as convolution are then passed through a nonlinear activation function. Although smooth nonlinear functions, such as sigmoid or hyperbolic tangent (tanh) function, were used previously because they are mathematical representations of a biological neuron behavior, the most common nonlinear activation function used presently is the rectified linear unit (ReLU), which simply computes the function: $f(x) = \max(0, x)$ (figure **??**) [[20], [18], [[30]], [[33]], [9]].

3.2.2 Pooling Layer

A pooling layer provides a typical downsampling operation which reduces the in-plane dimensionality of the feature maps in order to introduce a translation invariance to small shifts and distortions, and decrease the number of subsequent learnable parameters. It is of note that there is no learnable parameter in any of the pooling layers, whereas filter size, stride, and padding are hyperparameters in pooling operations, similar to convolution operations.

Max Pooling

The most popular form of pooling operation is max pooling, which extracts patches from the input feature maps, outputs the maximum value in each patch, and discards all the other values (figure 3.7). A max pooling with a filter of size 2×2 with a stride of 2 is commonly used in practice. This downsamples the in-plane dimension of feature maps by a factor of 2. Unlike height and width, the depth dimension of feature maps remains unchanged.

Global Average Pooling

Another pooling operation worth noting is a global average pooling [23]. A global average pooling performs an extreme type of downsampling, where a feature map with size of height $\times$ width is downsampled into a 1×1 array by simply taking the average of all the elements in each feature map, whereas the depth of feature maps is retained. This operation is typically applied only once before the fully connected layers. The advantages of applying global average pooling are as follows: (1) reduces the number of learnable parameters and (2) enables the CNN to accept inputs of variable size.

3.2.3 Fully connected layer

The output feature maps of the final convolution or pooling layer is typically flattened, i.e., transformed into a one-dimensional (1D) array of numbers (or vector), and connected to one or more fully connected layers, also known as dense layers, in which every input is connected to every output by a learnable weight. Once the features extracted by the convolution layers and downsampled by the pooling layers are created, they are mapped by a subset of fully connected layers to the final outputs of the network, such as the probabilities for each class in classification tasks. The final fully connected layer typically has the same number of output nodes as the number of classes. Each fully connected layer is followed by a nonlinear function, such as ReLU, as described above. The activation function applied to the last fully connected layer is usually different from the others. An appropriate activation function needs to be selected according to each task. An activation function applied to the multiclass classification task is a softmax function which normalizes output real values from the last fully connected layer to target class probabilities, where each value ranges between 0 and 1 and all values sum to 1.

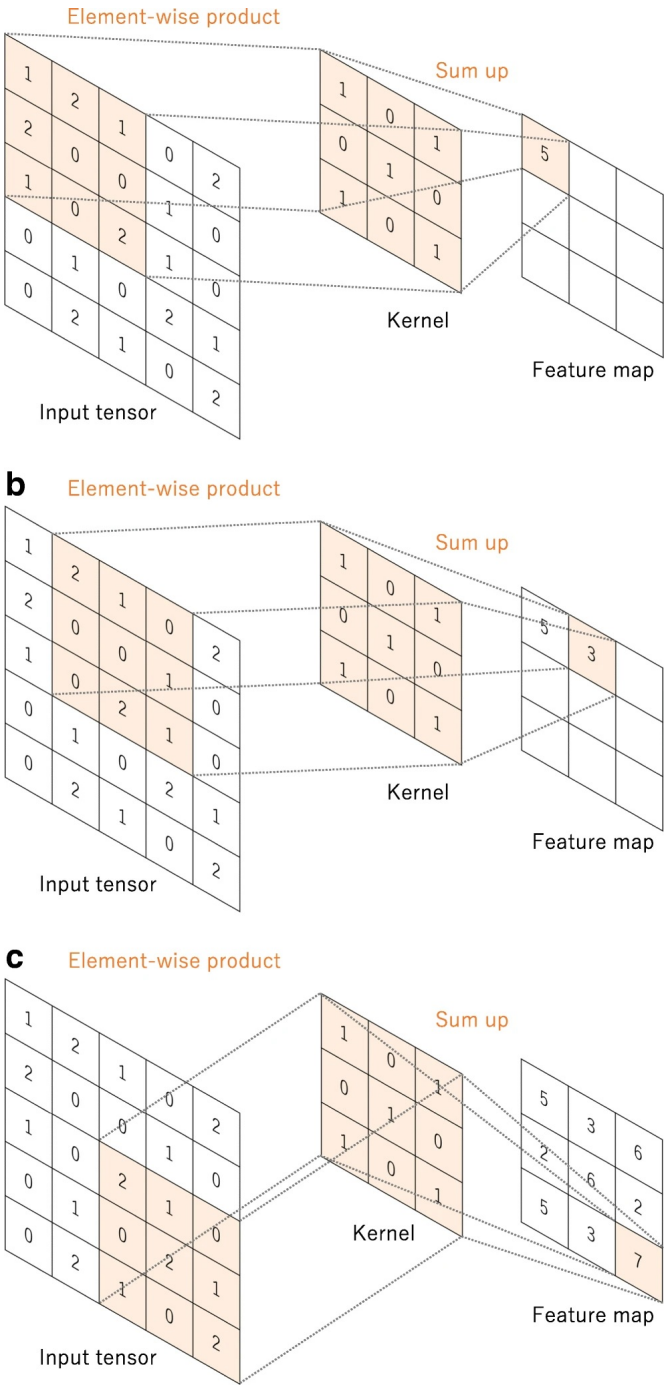

Figure 3.5: **a**–**c** An example of convolution operation with a kernel size of 3×3, no padding, and a stride of 1. A kernel is applied across the input tensor, and an element-wise product between each element of the kernel and the input tensor is calculated at each location and summed to obtain the output value in the corresponding position of the output tensor, called a feature map. [44].

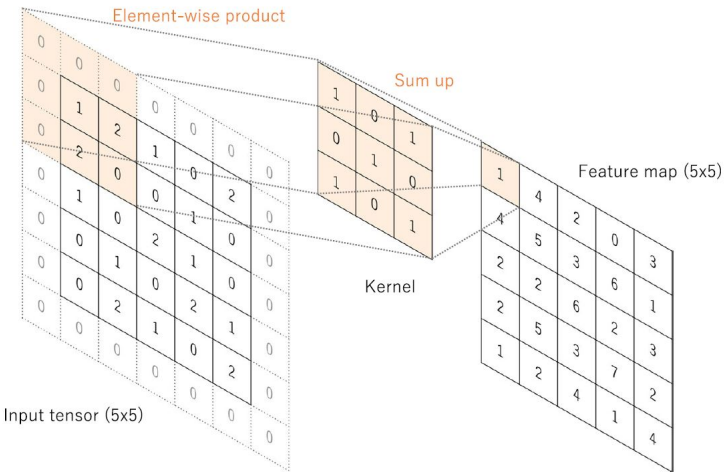

Figure 3.6: A convolution operation with zero padding so as to retain in-plane dimensions. Note that an input dimension of 5×5 is kept in the output feature map. In this example, a kernel size and a stride are set as 3×3 and 1, respectively [44].

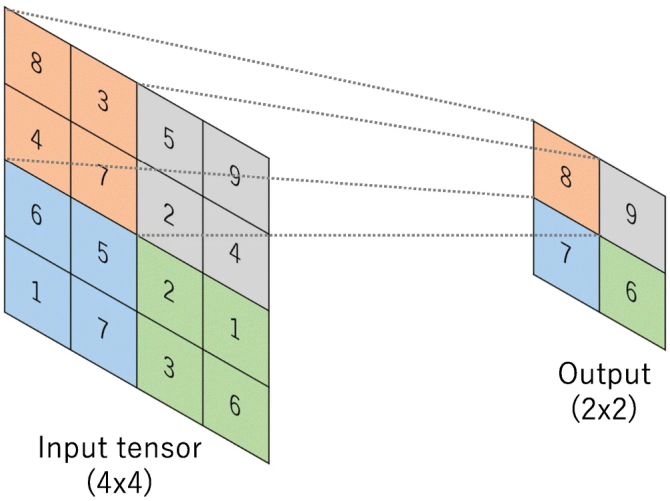

Figure 3.7: An example of max pooling operation with a filter size of 2×2, no padding, and a stride of 2, which extracts 2×2 patches from the input tensors, outputs the maximum value in each patch, and discards all the other values, resulting in downsampling the in-plane dimension of an input tensor by a factor of 2. [44].

Chapter 4

Methodology

4.1 Adversarial Nets

Ian J. Goodfellow et al. [11] described the adversarial modeling framework as most simple to apply when the models are both artificial neural networks. To learn the generator's distribution p_g over data x, a prior on input noise variables defined $p_z(z)$, then represent a mapping to data space as $G(z, \theta_g)$, where G is a differentiable function represented by a neural network with parameters θ_g. There also a second neural network $D(x, \theta_d)$ that outputs a single scalar. $D(x)$ represents the probability that x came from the data rather than p_g. D is trained to maximize the probability of assigning the correct label to both training examples and samples from G. Simultaneously G is trained to minimize $log(1 - D(G(z)))$. In other words, D and G play the following two-player minimax game with objective function $V(G, D)$:

$$\min_G \max_D V(D, G) = E_{x \sim p_{data}(x)}[\ logD(x)\ ] + E_{z \sim p(z)}[log(1 - D(G(z)))]. \tag{4.1}$$

In practice, equation 4.1 may not provide sufficient gradient for G to learn well. Early in learning, when G is poor, D can reject samples with high confidence because they are clearly different from the training data. In this case, $log(1 - D(G(z)))$ saturates. Rather than training G to minimize $log(1 - D(G(z)))$ we can train G to maximize $logD(G(z))$. This objective function results in the same fixed point of the dynamics of G and D but provides much stronger gradients early in learning.

4.2 Cycle-Consistent Adversarial Networks

4.2.1 Formulation

Our goal is to learn mapping functions between two domains X and Y. The Domain X represents synthetic document images distribution and domain Y represents real document images distribution. Given training samples $\{x_i\}_{i=1}^N$ where $x_i \in X$ and $\{y_j\}_{j=1}^M$ where $y_j \in Y$. We denote the data distribution $x \sim p_{data}(x)$ and $y \sim p_{data}(y)$. As illustrated in Figure 4.2, our model includes two mappings $G : X \rightarrow Y$ and $F : Y \rightarrow X$.In addition, we introduce two adversarial discriminators D_X and D_Y , where D_X aims to distinguish between images $\{x\}$ and translated images $\{F(y)\}$; in the same way, D_Y aims to discriminate between $\{y\}$ and $\{G(x)\}$. Our objective contains two types of terms: adversarial losses [11] for matching the distribution of generated images to the data distribution in the target domain; and cycle consistency losses to prevent the learned mappings G and F from contradicting each other.

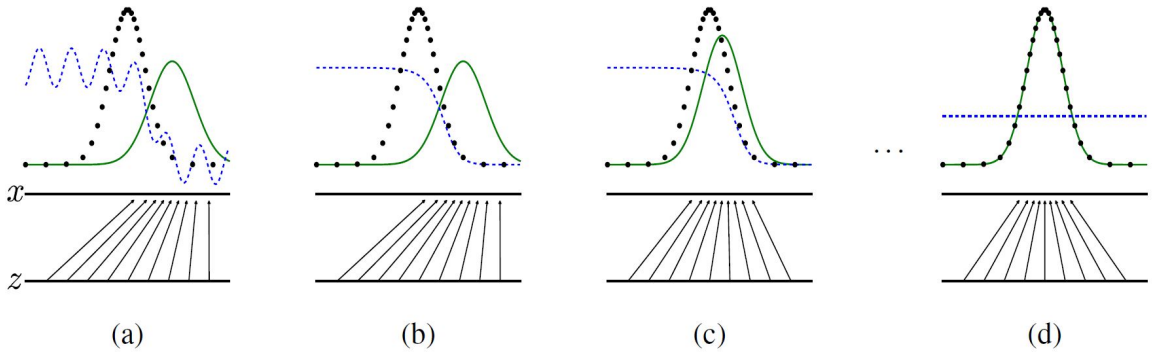

Figure 4.1: Generative adversarial networks are trained by simultaneously updating the discriminative distribution (D, blue, dashed line) so that it discriminates between samples from the data generating distribution (black, dotted line) p_x from those of the generative distribution $p_g(G)$ (green, solid line). The lower horizontal line is the domain from which z is sampled, in this case uniformly. The horizontal line above is part of the domain of x. The upward arrows show how the mapping $x = G(z)$ imposes the non-uniform distribution p_g on transformed samples. G contracts in regions of high density and expands in regions of low density of p_g. (a) Consider an adversarial pair near convergence: p_g is similar to p_{data} and D is a partially accurate classifier. (b) In the inner loop of the algorithm D is trained to discriminate samples from data, converging to $D^*(x) = \frac{p_{data}(x)}{p_{data}(x)+p_g(x)}$. (c) After an update to G, gradient of D has guided $G(z)$ to flow to regions that are more likely to be classified as data. (d) After several steps of training, if G and D have enough capacity, they will reach a point at which both cannot improve because $p_g = p_{data}$. The discriminator is unable to differentiate between the two distributions, i.e. $D(x) = \frac{1}{2}$ [11].

4.2.2 Adversarial Loss

We apply adversarial losses [11] to both mapping functions. For the mapping function $G : X \to Y$ and its discriminator D_Y , we express the objective as:

$$\mathcal{L}_{GAN}(G, D_Y, X, Y) = \mathbb{E}_{y \sim p_{data}(y)}[\log D_Y(y)] + \mathbb{E}_{x \sim p_{data}(x)}[\log(1 - D_Y(G(x)))], \tag{4.2}$$

where G tries to generate images $G(x)$ that look similar to images from domain Y , while D_Y aims to distinguish between translated samples $G(x)$ and real samples y. G aims to minimize this objective against an adversary D that tries to maximize it, i.e., $\min_G \max_{D_Y} \mathcal{L}_{GAN}(G, D_Y, X, Y)$. We introduce a similar adversarial loss for the mapping function $F : Y \to X$ and its discriminator D_X as well: i.e., $\min_F \max_{D_X} \mathcal{L}_{GAN}(F, D_X, Y, X)$.

4.2.3 Cycle Consistency Loss

Adversarial training can, in theory, learn mappings G and F that produce outputs identically distributed as target domains Y and X respectively. However, with large enough capacity, a network can map the same set of input images to any random permutation of images in the target domain, where any of the learned mappings can induce an output distribution that matches the target distribution. Thus, adversarial losses alone cannot guarantee that the learned function can map an individual input x_i to a desired output y_i. To further reduce the space of possible mapping functions, we argue that the learned mapping functions should be cycle-consistent: as shown in Figure 4.2 (b), for each image x from domain X, the image translation cycle should be able to bring x back to the original image, i.e., $x \to G(x) \to F(G(x)) \approx x$. We call this forward cycle consistency. Similarly, as illustrated in Figure 4.2 (c), for each image y from domain Y, G and F should also satisfy backward cycle consistency: $y \to F(y) \to G(F(y)) \approx y$. We incentivize this behavior using a cycle consistency

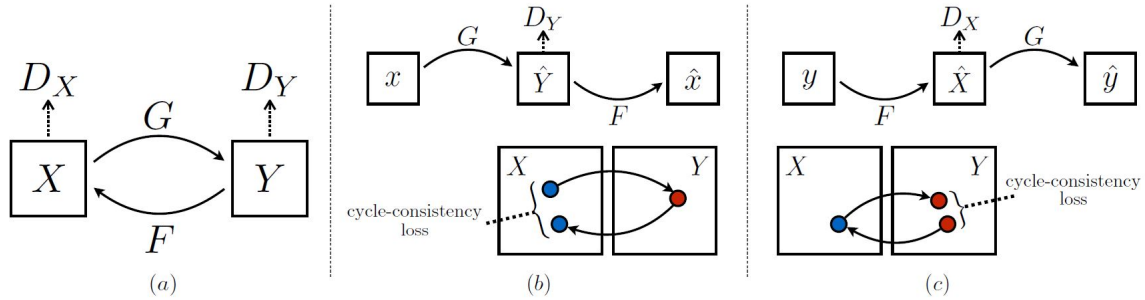

Figure 4.2: (a) CycleGAN model contains two mapping functions $G : X \rightarrow Y$ and $F : Y \rightarrow X$, and associated adversarial discriminators D_Y and D_X. D_Y encourages G to translate X into outputs indistinguishable from domain Y, and vice versa for D_X and F. To further regularize the mappings, we introduce two cycle consistency losses that capture the intuition that if we translate from one domain to the other and back again we should arrive at where we started: (b) forward cycle-consistency loss: $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$, and (c) backward cycle-consistency loss: $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$ [50].

loss:

$$\mathcal{L}_{cyc}(G, F) = \mathbb{E}_{x \sim p_{data}(x)}[\|F(G(x)) - x\|_1] + \mathbb{E}_{y \sim p_{data}(y)}[\|G(F(y)) - y\|_1]. \tag{4.3}$$

4.2.4 Identity Mapping Loss

It is helpful to introduce an additional loss to encourage the mapping to preserve color composition between the input and output. In particular, we adopt the technique of [40] and regularize the generator to be near an identity mapping when real samples of the target domain are provided as the input to the generator:

$$\mathcal{L}_{identity}(G, F) = \mathbb{E}_{y \sim p_{data}(y)}[\|(F(y) - y\|_1] + \mathbb{E}_{x \sim p_{data}(x)}[\|G(x) - x\|_1]. \tag{4.4}$$

4.2.5 Full Objective

Our full objective is:

$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{GAN}(G, D_Y, X, Y) + \mathcal{L}_{GAN}(F, D_X, Y, X) + \lambda \mathcal{L}_{cyc}(G, F), \tag{4.5}$$

where λ controls the relative importance of the two objectives. We aim to solve:

$$G^*, F^* = \arg \min_{G,F} \max_{D_X, D_Y} \mathcal{L}(G, F, D_X, D_Y) \tag{4.6}$$

Chapter 5

Implementation

It is very much important to provide proper nomenclature for our classifiers and type GANs that are being involved in our work. As already described that we are using CycleGANs for image-to-image translation, So simply going to call it CycleGAN whenever we need to. The classifier that we are training over Synthetic Document Images, we call it Synthetic Document Images Classifier. Finally. The classifier that we are training over CycleGAN Generated Document Images, we call it CycleGAN Generated Document Images Classifier. The implementation of our work is divided into 5 steps. To understand how the development pipeline works, you can have a look into section 5.3. Those steps are given below and we look into each of them one by one. Before we proceed further we will have a look into data used to train the neural networks in our work in section 5.1. All of the implementation related to the coding has been done into Python Programming Language. The neural networks are trained using GPUs (Graphics Processing Unit) like Nvidia Tesla T4 and Tesla V100-SXM2.

5.1 Dataset Preparation

The dataset preparation is one of the important aspects of training any neural network. Poor quality data leads to a poor generalization of neural networks. Our work uses 4 sets of the dataset. To train CycleGAN it is required to have a stash of Source domain images and Target domain images. We have around 100,000 Synthetic Document images in the Source Domain and the same number of Real Document Images in the Target Domain. Our work uses 10 classes of templates. We just need to provide the path of those datasets during the training of CycleGAN. To evaluate the quality of Generated Document Images by CycleGAN we have to train the classifier on Synthetic Document images at first and later on CycleGAN Generated Document Images. For Synthetic Document Images Classifier we have 10000 Synthetic Document images for each class of template, which comes in a total of 100,000 images. The Synthetic Document Images dataset is created by inserting handwritten crops over empty templates. For testing, we have around in total 1200 annotated real images. The testing dataset is unbalanced. But this can not cause a problem to compute the accuracy score of each class. The datasets used in our work can not be cited because they are not open for public use.

5.2 Python Libraries

5.3 Training Details

One of the major challenges of the implementation part was designing efficient input pipeline.

You can learn about how to load large dataset efficiently using TensorFlow in this Tutorial.

5.3.1 Creating Synthetic Document Images

5.3.2 Training of the Synthetic Document Images Classifier

5.3.3 Training of the CycleGAN

5.3.4 Training of the CycleGAN Generated Document Images Classifier

5.3.5 Classifiers Performance Analysis

5.4 Network Architecture

We adopt the architecture for our generative networks from Johnson et al. [15] who have shown impressive results for neural style transfer and super resolution.

5.4.1 Classifier Architecture

5.4.2 Generator Architecture

5.4.3 Discriminator Architecture

In our work, we use the PatchGAN discriminator, which is also called as Markovian discriminator [14].

Chapter 6

Evaluation

6.1 Evaluation Metrics

6.2 Results

Chapter 7

Conclusion

Appendix A

Architecture Diagrams

A.1 Classifier Architecture Diagram

A.2 Generator Architecture Diagram

A.3 Discriminator Architecture Diagram

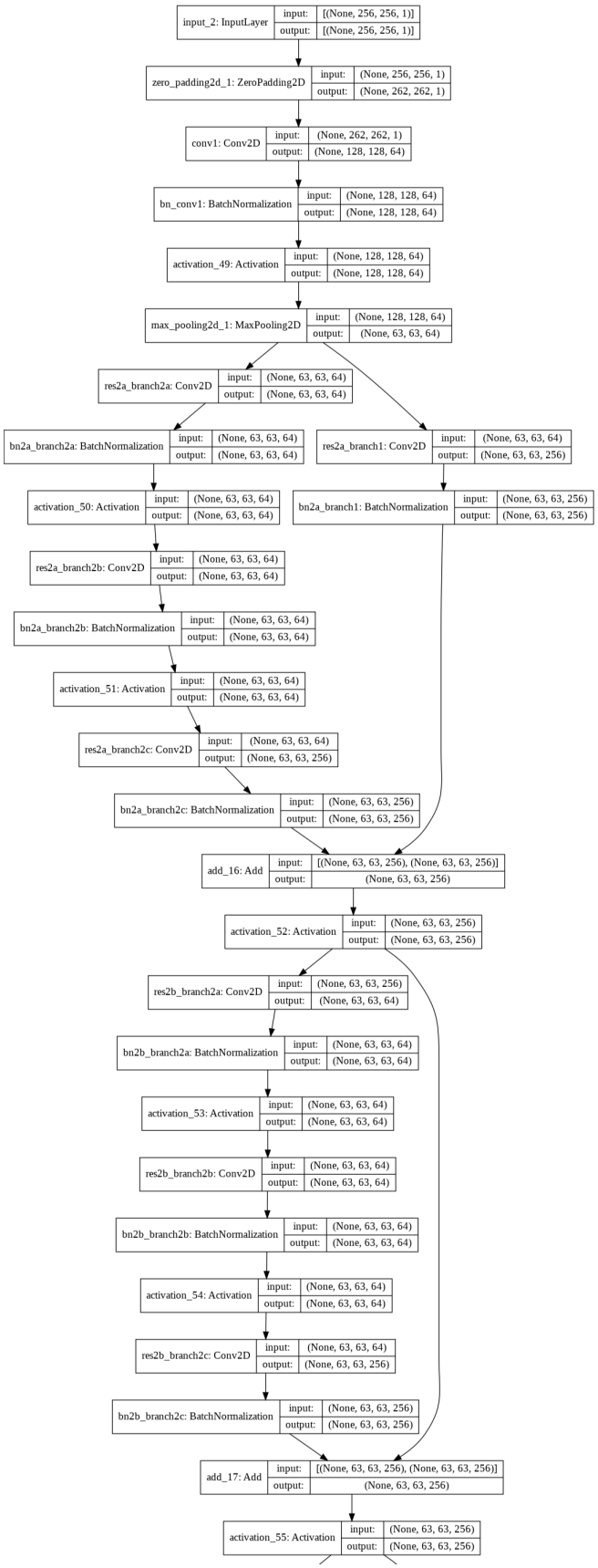

Figure A.1: Classifier Architecture Diagram. Continues in Next Pages.

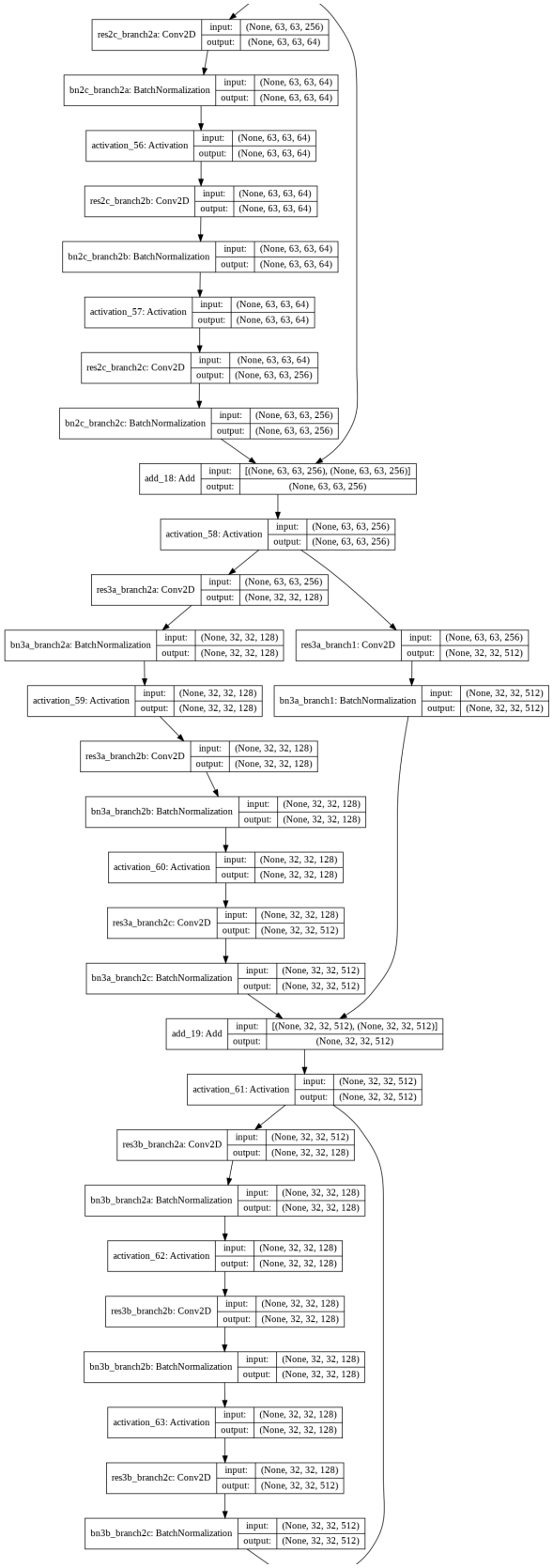

Figure A.2: Classifier Architecture Diagram. Continues in Next Pages.

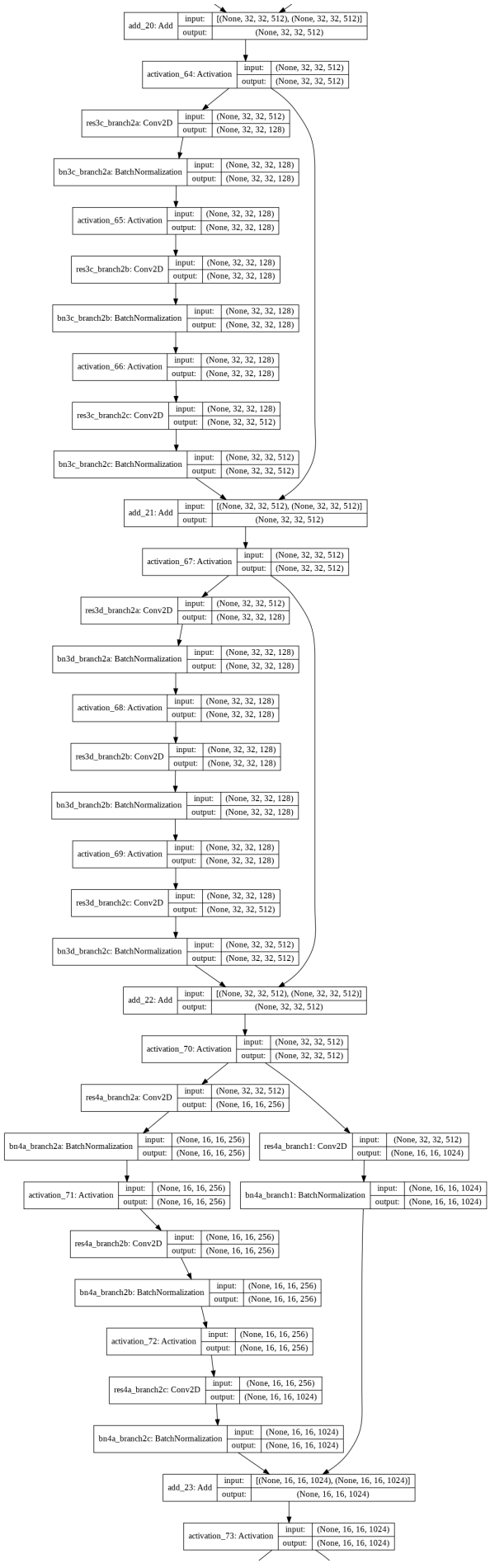

Figure A.3: Classifier Architecture Diagram. Continues in Next Pages.

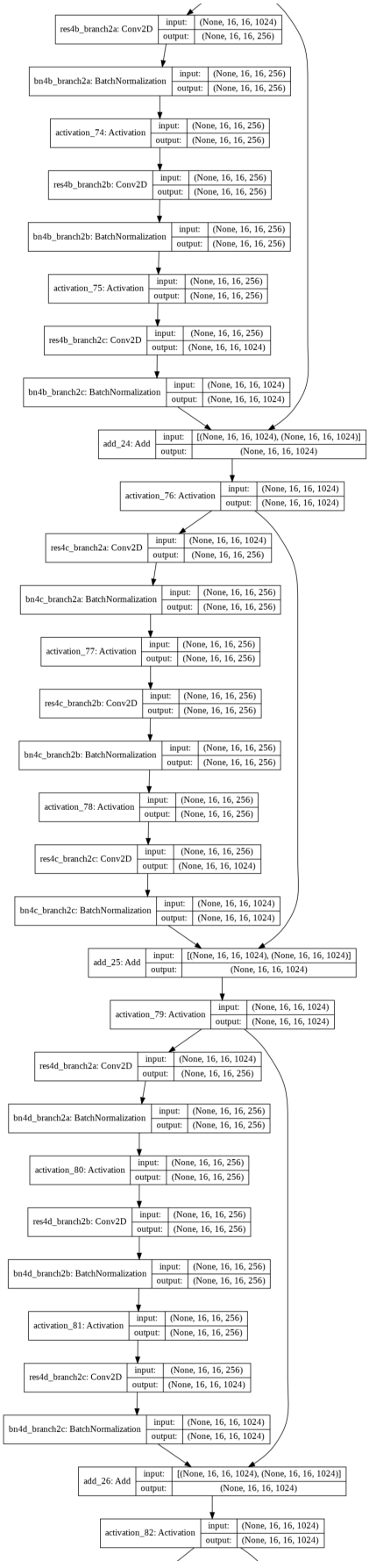

Figure A.4: Classifier Architecture Diagram. Continues in Next Pages.

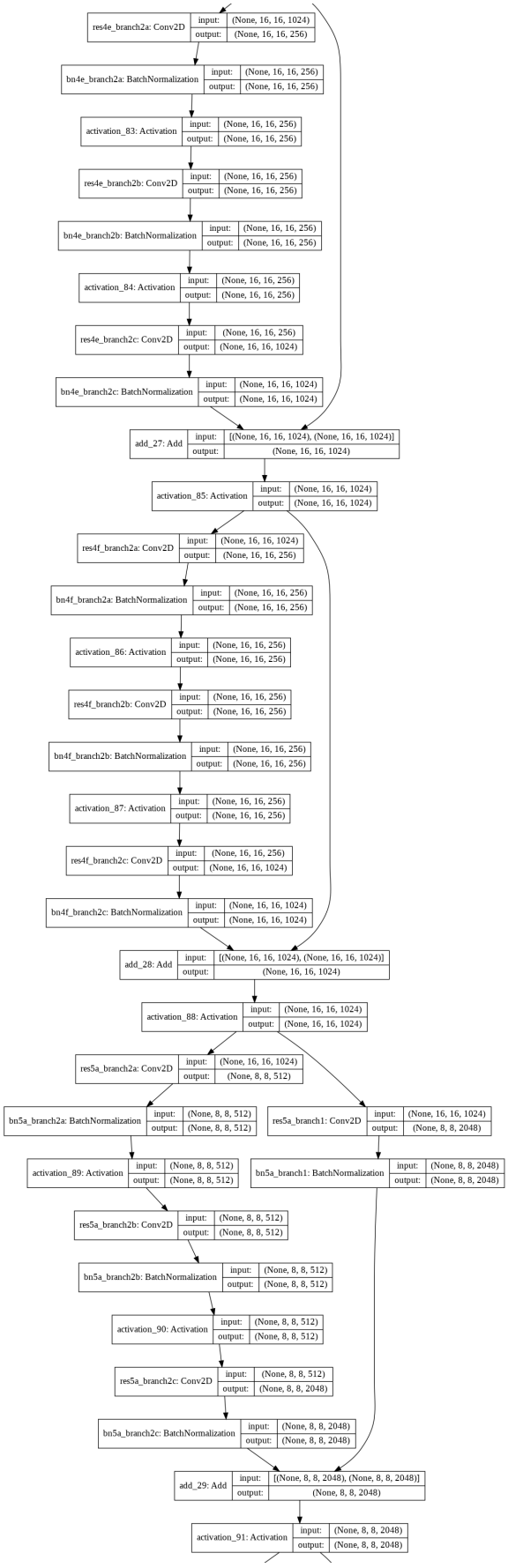

Figure A.5: Classifier Architecture Diagram. Continues in Next Pages.

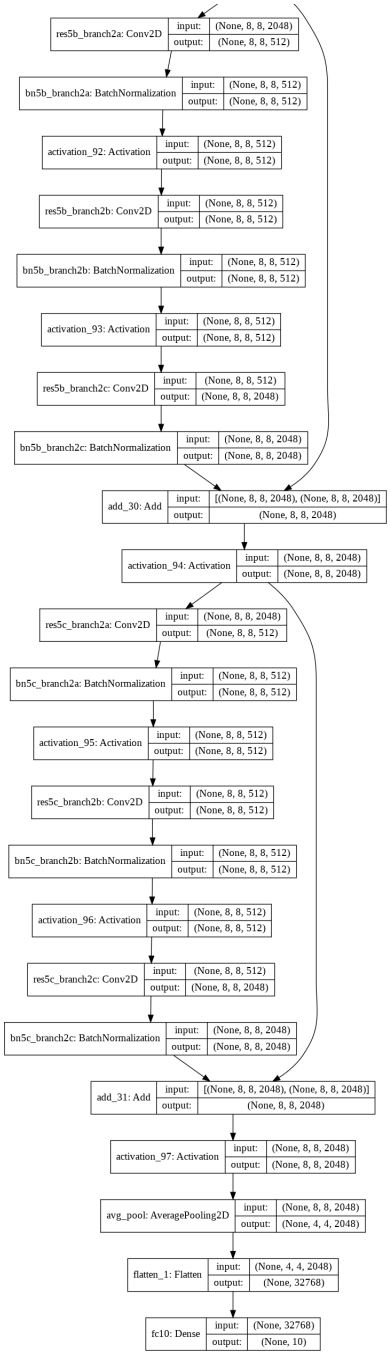

Figure A.6: Classifier Architecture Diagram. Ends Here.

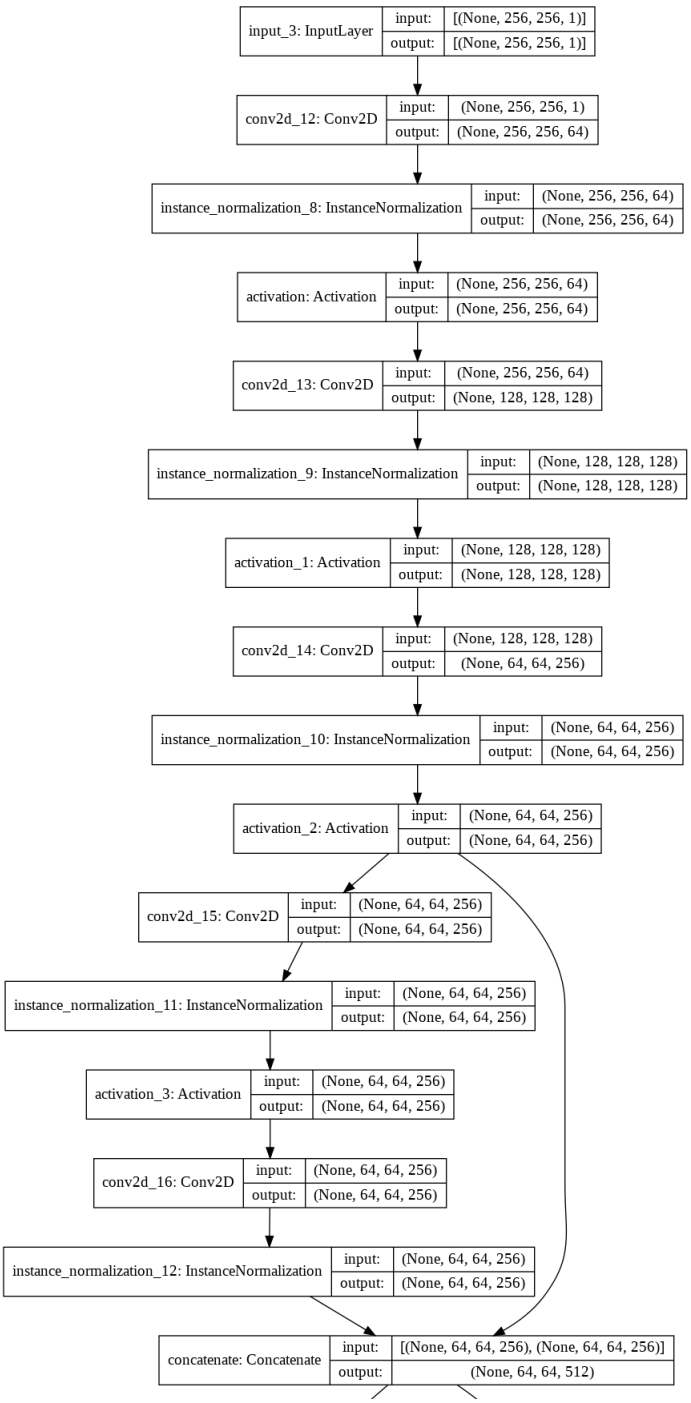

Figure A.7: Generator Architecture Diagram. Continues in Next Pages.

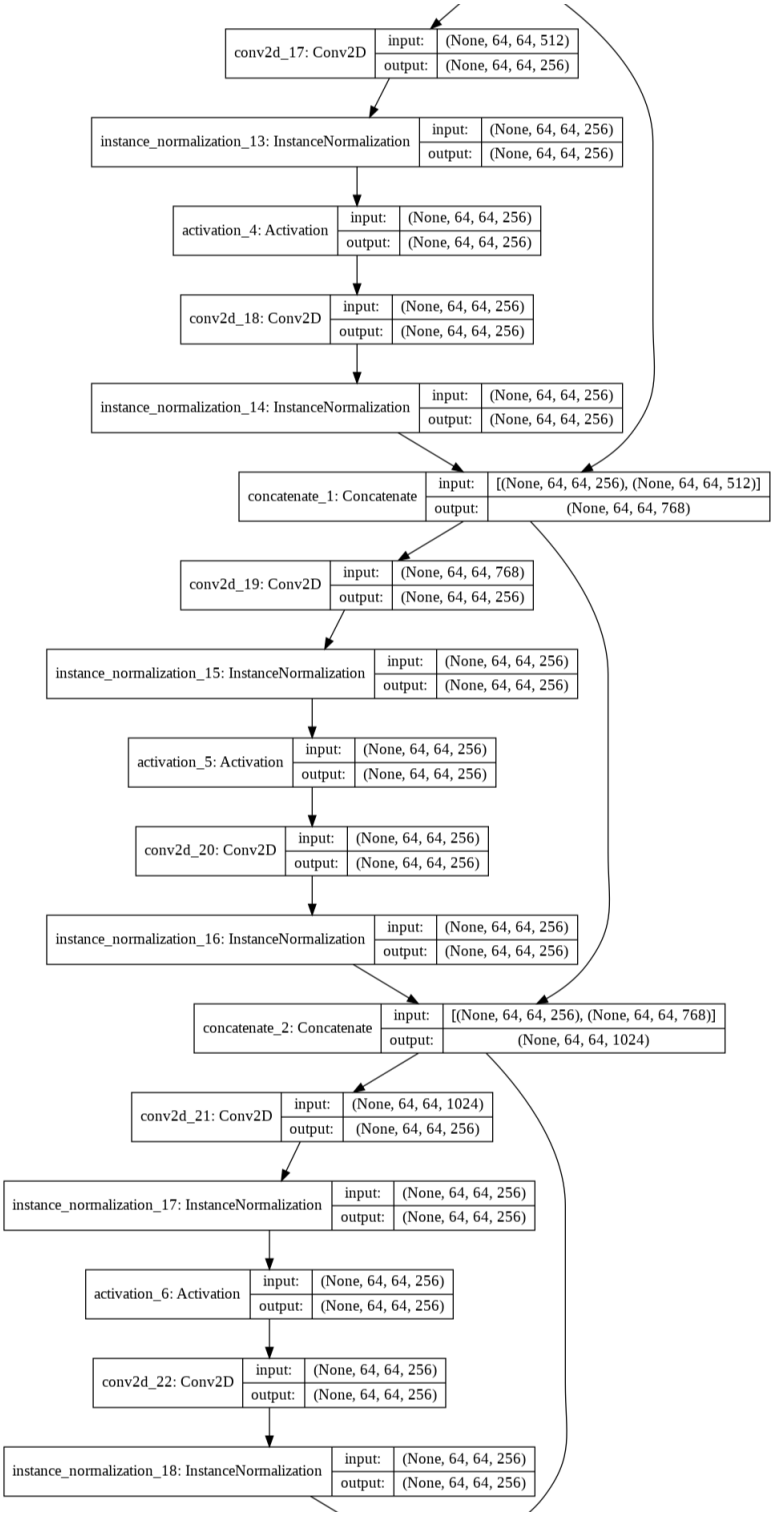

Figure A.8: Generator Architecture Diagram. Continues in Next Pages.

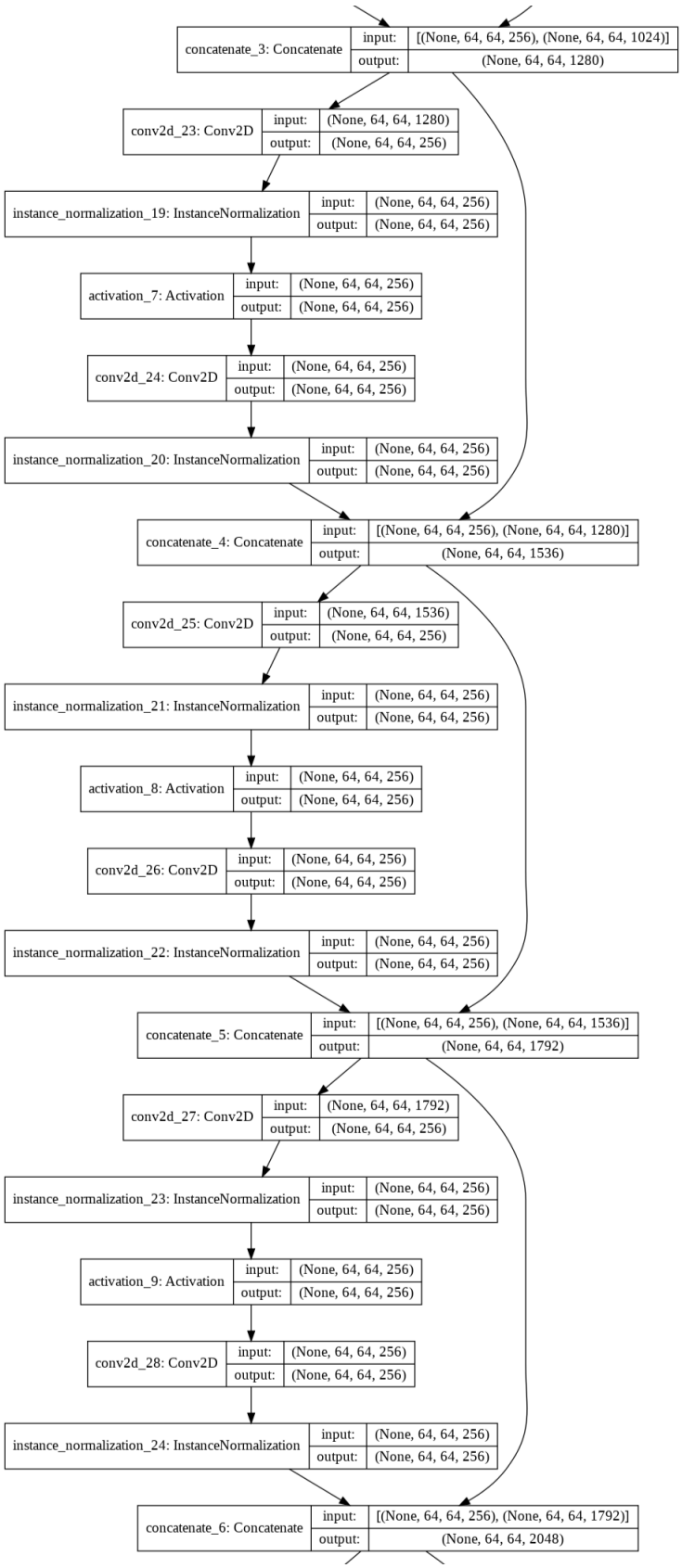

Figure A.9: Generator Architecture Diagram. Continues in Next Pages.

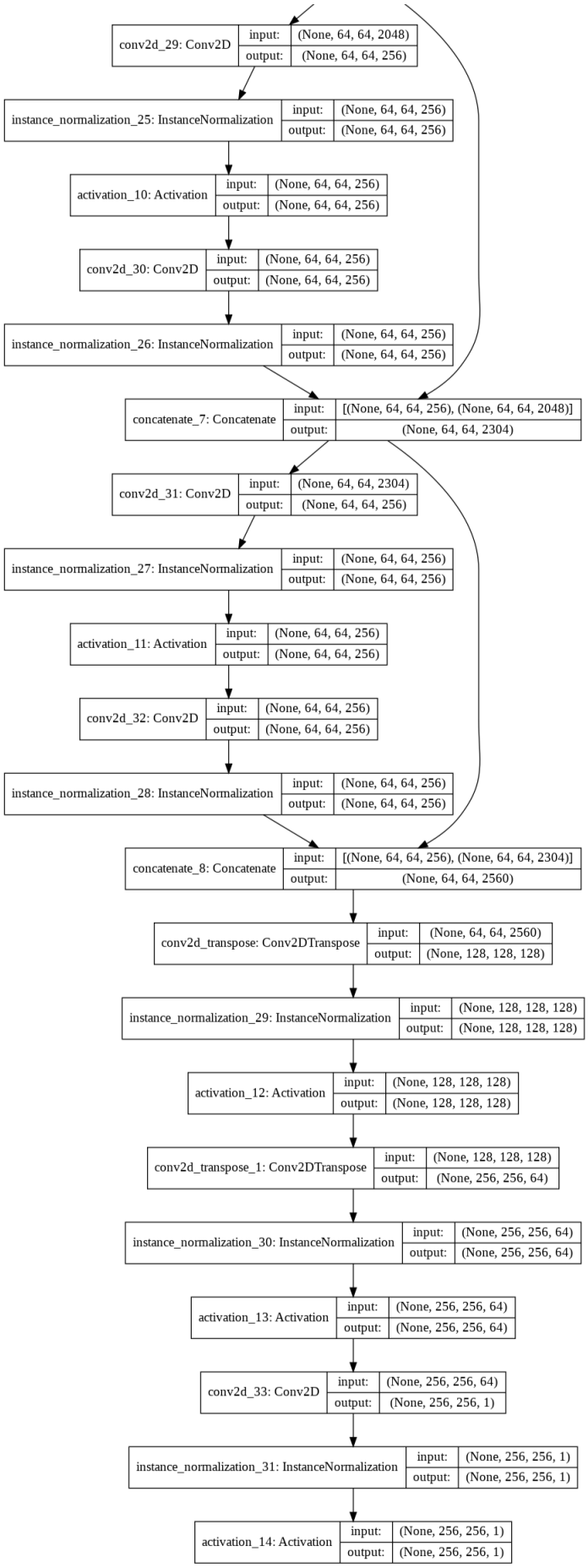

Figure A.10: Generator Architecture Diagram. Ends Here.

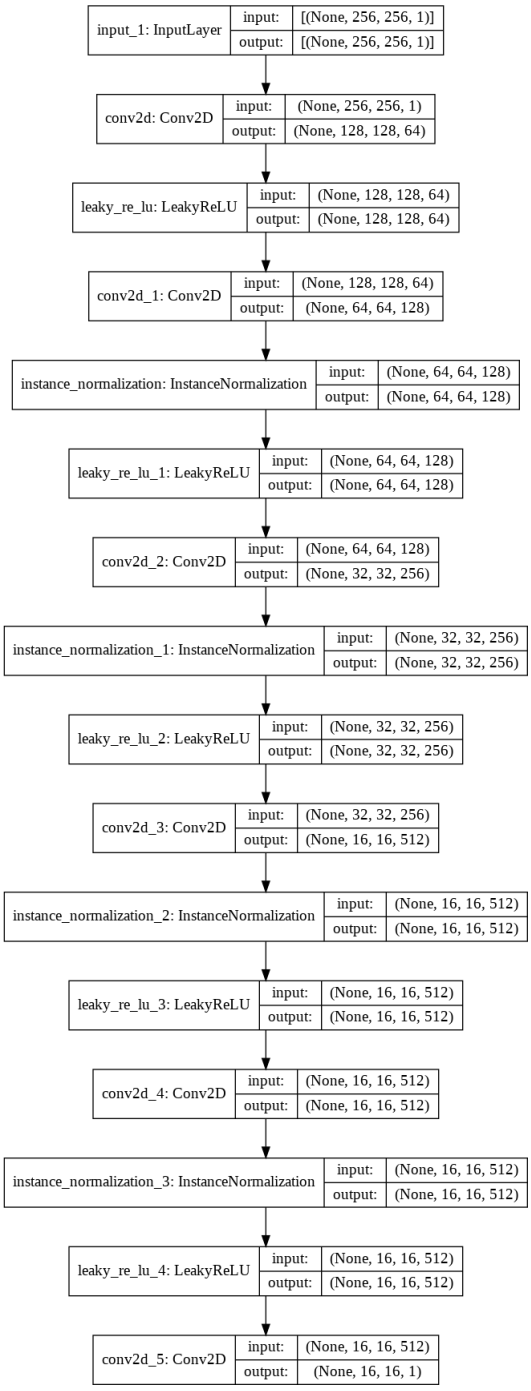

Figure A.11: Discriminator Architecture Diagram

Bibliography

[1] Yoshua Bengio, Grégoire Mesnil, Yann Dauphin, and Salah Rifai. Better mixing via deep representations, 2012.

[2] Yoshua Bengio, Eric Thibodeau-Laufer, Guillaume Alain, and Jason Yosinski. Deep generative stochastic networks trainable by backprop, 2014.

[3] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding, 2016.

[4] Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. Adversarial feature learning, 2017.

[5] Vincent Dumoulin, Ishmael Belghazi, Ben Poole, Olivier Mastropietro, Alex Lamb, Martin Arjovsky, and Aaron Courville. Adversarially learned inference, 2017.

[6] Mathias Eitz, James Hays, and Marc Alexa. How do humans sketch objects? *ACM Transactions on Graphics - TOG*, 31, 07 2012.

[7] Huan Fu, Mingming Gong, Chaohui Wang, Kayhan Batmanghelich, Kun Zhang, and Dacheng Tao. Geometry-consistent generative adversarial networks for one-sided unsupervised domain mapping, 2018.

[8] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, 1980.

[9] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. JMLR Workshop and Conference Proceedings.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[11] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[12] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium, 2018.

[13] D. H. Hubel and T. N. Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of physiology*, 195(1):215–243, 1968.

[14] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks, 2018.

[15] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution, 2016.

[16] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[19] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[20] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[21] Hsin-Ying Lee, Hung-Yu Tseng, Qi Mao, Jia-Bin Huang, Yu-Ding Lu, Maneesh Singh, and Ming-Hsuan Yang. Drit++: Diverse image-to-image translation via disentangled representations, 2019.

[22] Chuan Li and Michael Wand. Precomputed real-time texture synthesis with markovian generative adversarial networks, 2016.

[23] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network, 2014.

[24] C. Liu, F. Yin, Q. Wang, and D. Wang. Icdar 2011 chinese handwriting recognition competition. In *2011 International Conference on Document Analysis and Recognition*, pages 1464–1469, 2011.

[25] Ming-Yu Liu, Thomas Breuel, and Jan Kautz. Unsupervised image-to-image translation networks, 2018.

[26] Ming-Yu Liu and Oncel Tuzel. Coupled generative adversarial networks, 2016.

[27] P Manisha and Sujit Gujar. Generative adversarial networks (gans): What it can generate and what it cannot?, 2019.

[28] Xudong Mao, Qing Li, Haoran Xie, Raymond Y. K. Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks, 2017.

[29] Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. Unrolled generative adversarial networks, 2017.

[30] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 807–814, Madison, WI, USA, 2010. Omnipress.

[31] Taesung Park, Alexei A. Efros, Richard Zhang, and Jun-Yan Zhu. Contrastive learning for unpaired image-to-image translation, 2020.

[32] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2016.

[33] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions, 2017.

[34] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[35] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2017.

[36] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge, 2015.

[37] Monika Sharma, Abhishek Verma, and Lovekesh Vig. Learning to clean: A gan perspective, 2019.

[38] Ashish Shrivastava, Tomas Pfister, Oncel Tuzel, Josh Susskind, Wenda Wang, and Russ Webb. Learning from simulated and unsupervised images through adversarial training, 2017.

[39] Joshua Susskind, Adam Anderson, and Geoffrey E Hinton. The toronto face dataset. Technical report, Technical Report UTML TR 2010-001, U. Toronto, 2010.

[40] Yaniv Taigman, Adam Polyak, and Lior Wolf. Unsupervised cross-domain image generation, 2016.

[41] C. Tensmeyer, M. Brodie, D. Saunders, and T. Martinez. Generating realistic binarization data with generative adversarial networks. In *2019 International Conference on Document Analysis and Recognition (ICDAR)*, pages 172–177, 2019.

[42] Hoang Thanh-Tung and Truyen Tran. On catastrophic forgetting and mode collapse in generative adversarial networks, 2020.

[43] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding, 2019.

[44] Rikiya Yamashita, Mizuho Nishio, Richard Do, and Kaori Togashi. Convolutional neural networks: an overview and application in radiology. *Insights into Imaging*, 9, 06 2018.

[45] Yuancheng Ye, Lijuan Wang, Yue Wu, Yinpeng Chen, Yingli Tian, Zicheng Liu, and Zhengyou Zhang. Gan quality index (gqi) by gan-induced classifier, 2018.

[46] A. Yu and K. Grauman. Fine-grained visual comparisons with local learning. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 192–199, 2014.

[47] Fisher Yu, Ari Seff, Yinda Zhang, Shuran Song, Thomas Funkhouser, and Jianxiong Xiao. Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop, 2016.

[48] Junbo Zhao, Michael Mathieu, and Yann LeCun. Energy-based generative adversarial network, 2017.

[49] Jun-Yan Zhu, Philipp Krähenbühl, Eli Shechtman, and Alexei A. Efros. Generative visual manipulation on the natural image manifold, 2018.

[50] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks, 2020.