

Data Augmentation with Artistic Style

Xu Zheng

A Dissertation

Presented to the University of Dublin, Trinity College
in partial fulfilment of the requirements for the degree of

Master of Science in Computer Science (Intelligent Systems)

Supervisor: Aljosa Smolic

August 2018

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Xu Zheng

August 29, 2018

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Xu Zheng

August 29, 2018

Acknowledgments

I would like to thank my supervisor, Prof. Aljosa Smolic, for his support and feedback throughout the project.

I would also like to thank Mr. Sebastian Lutz, Mr. Tejo Chalasani and Mr. Koustav Ghosal, for their advice on research directions during the whole year. Their guidance leads to the final success of this thesis.

Lastly, I would like to thank my girlfriend Ruiqi Xiao whose proofreading played an indispensable part in the completion of my thesis.

XU ZHENG

University of Dublin, Trinity College

August 2018

Summary

Deep Convolutional Neural Networks (CNNs) has gained a great success in computer vision tasks, and the success of training deep CNNs heavily depends on a significant amount of labelled data. The recent research has found that neural style transfer algorithm can apply the artistic style of an image to another image without changing its high-level semantic content, so we think it is possible to employ the neural style transfer algorithm as a data augmentation strategy. The contribution of this dissertation is a thorough evaluation of the effectiveness of the neural style transfer as a data augmentation method in the image classification task. We explore the state-of-the-art neural style transfer algorithms and apply them as a data augmentation method on the Caltech101 and Caltech256 image dataset and VGG deep neural networks, where we found a 2% improvement from 83% to 85% of the image classification accuracy, compared with traditional data augmentation strategies. We also combine this new method with the conventional data augmentation approaches to further improve the performance of image classification. This work shows the potential of neural style transfer in computer vision field, such as helping us to reduce the difficulty of collecting sufficient labelled data and improve the performance of generic image-based deep learning algorithms.

Contents

| | |
|---|-------------|
| Acknowledgments | iii |
| Summary | iv |
| List of Tables | viii |
| List of Figures | ix |
| Chapter 1 Introduction | 1 |
| 1.1 Overview | 1 |
| 1.2 Motivation | 2 |
| 1.3 Dissertation Structure | 5 |
| Chapter 2 Background Research | 6 |
| 2.1 Neural Style Transfer | 6 |
| 2.1.1 Image Semantic Content | 7 |
| 2.1.2 Image Style | 9 |
| 2.1.3 Descriptive Neural Style Transfer | 12 |
| 2.1.4 Generative Neural Style Transfer | 13 |
| 2.1.5 N-Styles FeedForward Network | 14 |

| | | |
|---------------------------------|---|-----------|
| 2.2 | The Art of Data Augmentation | 16 |
| 2.2.1 | Traditional Data Augmentation | 16 |
| 2.2.2 | Advanced Data Augmentation | 17 |
| 2.3 | TensorFlow and Keras | 18 |
| Chapter 3 Design | | 21 |
| 3.1 | Style Transfer Architecture | 21 |
| 3.1.1 | Transformation Network | 22 |
| 3.1.2 | Loss Network | 23 |
| 3.2 | Image Classification Network | 25 |
| 3.3 | Experiments | 28 |
| Chapter 4 Implementation | | 30 |
| 4.1 | Augmentation Module | 30 |
| 4.1.1 | Transformation Network | 31 |
| 4.1.2 | Loss Network | 32 |
| 4.1.3 | Dataset | 33 |
| 4.1.4 | Training | 33 |
| 4.2 | Image Classification Network | 34 |
| 4.2.1 | Configurations | 34 |
| 4.2.2 | Dataset | 35 |
| 4.3 | Experiments | 36 |
| 4.3.1 | Stylized Image Overview | 36 |
| 4.3.2 | Content Weights | 36 |
| Chapter 5 Evaluation | | 39 |

| | | |
|--------------------------------|--|-----------|
| 5.1 | Traditional Image Augmentation | 39 |
| 5.2 | Single Style | 40 |
| 5.3 | Combined Methods | 41 |
| 5.4 | Content Weights Change | 42 |
| 5.5 | VGG19 | 43 |
| Chapter 6 Conclusion | | 46 |
| 6.1 | Limitations | 46 |
| 6.2 | Future Work | 47 |
| Bibliography | | 50 |
| Appendix A Image Styles | | 55 |
| Appendix B GitHub Link | | 60 |

List of Tables

| | | |
|-----|--|----|
| 4.1 | Image Classification Dataset | 35 |
| 5.1 | Traditional Image Augmentation | 40 |
| 5.2 | Single Style Transfer Augmentation | 41 |
| 5.3 | Combined Style Transfer Augmentation | 42 |
| 5.4 | Different Content Weights | 43 |
| 5.5 | Experiments on VGG19 with both Caltech101 and Caltech256 | 44 |

List of Figures

| | | |
|-----|---|---|
| 1.1 | Style Transfer: The style of Sunflower (Van Gogh) is applied on the photo and generate a new stylized image (b), which can keep the main content of the phone but as well as contain the texture style from image Sunflower. | 4 |
| 2.1 | Feature Map: Typical-looking of the image information encoded in the first layer (a), and the 5th layer (b). Source: Trained AlexNet looking at a picture of a cat. Every single box shows a feature map of a specific filter in that layer [15]. | 8 |

| | | |
|-----|--|----|
| 2.2 | Style Generation. The original image \vec{x} is passed through the convolutional neural network and compute the feature maps for each layer l in the network. G^l is used to represent the Gram matrices responses of a set of layers. The white noise image $\hat{\vec{x}}$ is passed through the same network, and we compute the \hat{G}^l . The loss function E^l is computed on every layer based on the difference between two Gram matrices. The total loss function L is a weighted sum of the contributions E^l from each layer. Finally, gradient descent is used to update the pixel values on the noise image, and a new image is found which produces the same Gram matrices G^l as the original texture does [19]. | 11 |
| 2.3 | Style representations and content representations in a Convolutional Neural Network (CNN) in different layers [21]. | 13 |
| 2.4 | The feature maps x is normalized across both spatial dimensions and subsequently scaled and shifted using style-dependent parameter vectors γ_s, β_s . s marks the style we want to apply [11]. | 15 |
| 2.5 | Traditional Data Augmentation: applying transformations on the original image doesn't change the fact that this is still a cat image [24]. . . | 17 |
| 2.6 | TensorFlow dataflow: reading data, preprocessing, training, and checkpointing state [30]. | 19 |
| 2.7 | The hardware stack for Keras: Model level and cross-platform [32]. . . | 20 |
| 3.1 | System overview: image transformation network is trained based on the loss computed by a pretrained loss network [5]. | 23 |
| 3.2 | Receptive Field Overview [24]. | 26 |

| | | |
|-----|--|----|
| 3.3 | VGGNet Overview: Strictly use 3×3 filters with 2×2 max-pooling layers [36]. | 27 |
| 3.4 | The pipeline of the overall architecture. | 29 |
| 4.1 | Stylized images created by the transformation network of 8 different styles. | 37 |
| 4.2 | Stylized images with the same style and different content weights. Wave 2 has more content weight than Wave 1 | 38 |
| 5.1 | Comparison between the original image(airplane) and two different stylized images. Your Name add too many colours and shapes on the original image, which may lead to a bad performance in terms of the image classification accuracy. | 41 |
| 5.2 | Comparison between the original image(airplane) and stylized images with different content weights. The images look similar to each other. | 43 |
| 6.1 | The process of using classification loss to optimize the transformation network | 48 |
| A.1 | The Great Wave off Kanagawa, Hokusai, 1829 - 1832 | 55 |
| A.2 | A Snow picture from the Internet, simple but beautiful | 56 |
| A.3 | The Muse, Pablo Picasso, 1935 | 56 |
| A.4 | Rain Princess, a painting by Leonid Afremov | 57 |
| A.5 | Sunflower from one of Van Gogh's series of paintings | 57 |
| A.6 | A typical udnie painting | 58 |
| A.7 | A screenshot from the Japaness Animation: Your Name (2016) | 58 |
| A.8 | The Scream, Edvard Munch, 1893 - 1910 | 59 |

Chapter 1

Introduction

1.1 Overview

Many image data augmentation techniques have been widely used in the state-of-the-art Convolutional Neural Network (CNN) architectures [1][2]. These traditional data augmentation techniques, such as flipping, rotation, and cropping, are found effective to generate additional labelled images and avoid overfitting for a complex and deep CNN model. But these techniques can only add limited noise to the original images. With the fast development of deep CNN architectures, more effective data augmentation techniques maybe needed for the future research.

In this dissertation, we research a data augmentation technique based on neural image style transfer algorithm [3] and apply it on image classification tasks [4] to verify its effectiveness. The recent research [5] has found that neural style transfer algorithm can apply the artistic style of an image to another image, without changing the high-level semantic content. In our data augmentation method, we employ the neural style transfer algorithm to create new stylized images from the original image dataset to

enlarge the training dataset and thus improve the performance and avoid overfitting of the trained models. Since there are many different image styles, we also research if different image styles have different effectiveness for image classification problems.

The datasets used in this project are the Caltech101 dataset [6] and Caltech256 dataset [7], which contain images from 102 categories and 257 categories respectively. We split both datasets with 70% of images used for training, and 30% of images used for validation. Pretrained VGG16 model and VGG19 model [2] are applied to perform the image classification task. As a result, our style transfer augmentation technique improves classification accuracy for both datasets on VGG16 and VGG19 models, where we found around a 2% improvement of the image classification accuracy. We also combine this method with the traditional data augmentation approaches to further improve the performance of a CNN model. This dissertation shows the potential of style transfer as a useful technique in computer vision field, helping us to reduce the difficulty of collecting sufficient labelled data and improve the performance of generic image-based algorithms.

1.2 Motivation

Deep Convolutional Neural Networks has gained a great success in computer vision tasks since AlexNet won the ILSVRC competition in 2012 [1]. With the fast development of the computing resources, such as GPU and CPU, the CNN models have become more and more deep and complex [8]. Hence, successfully training a model needs more and more labelled data, which is not easily accessible.

For many images and video machine learning projects, gathering raw data can be very time-consuming and challenging. For some critical image-based tasks like

cancer detection [9] and cancer classifying [10], researchers are often blocked by the lack of reliable data. What's more, for many machine learning algorithms, if we only use the raw data the model is easily to get overfitting and does not have enough generalisation on unseen data. We realise that it is of great importance for people to access a significant amount of reliable and labelled image data, hence exploring more effective data augmentation techniques would be a valuable research and can help us to find potential solutions to address the problems mentioned above.

The neural artistic style transfer has been found an effective way to apply the artistic style to an image without changing the high-level content of the original images. The basic idea of this algorithm is to jointly minimise the distance of the content representation and the style representation learned on the layers of the convolutional neural networks. Since it allows us to keep the main content of the original images and add enough noise to the images, we think it can be an effective way to generate new images and augment the training dataset. The figure 1.1 shows a style transfer example where we apply the style of Sunflower (Van Gogh) on a photo.

In our design, we make use of neural style transfer to augment the dataset by transferring styles of the images in the dataset to another style. We then explore and propose another augmentation method where we combine style transfer and traditional methods to further augment the dataset. Finally, we evaluate the performance on validation dataset and use classification accuracy as the metrics to compare these augmentation strategies to figure out how well the style transfer performs as a data augmentation strategy.



(a) Raw Images



(b) Stylized Image

Figure 1.1: Style Transfer: The style of Sunflower (Van Gogh) is applied on the photo and generate a new stylized image (b), which can keep the main content of the photo but as well as contain the texture style from image Sunflower.

1.3 Dissertation Structure

The thesis is organised into six chapters, starting with an introduction to this research and its motivation in Chapter 1. Then it is followed by a literature review on the background of the deep neural network, development of style transfer and the art of data argumentation strategies. We also touch upon the techniques and software used in this dissertation in Chapter 2, such as Tensorflow and Keras. In Chapter 3, we present our networks design for style transfer and image classification, and the mechanism behind the design. Chapter 4 provide the implementation details and different models we trained in this project. Chapter 5 gives a summary of the evaluation results from multiple perspectives. We finally discuss the limitation and future works in Chapter 6.

Chapter 2

Background Research

This chapter presents a literature review on the development of neural style transfer algorithms, including the descriptive model and the generative model. We then introduce the usage and effectiveness of the state-of-the-art data augmentation methods used in image classification tasks during the past few years. Finally, we look into the development of the techniques and software used in this project.

2.1 Neural Style Transfer

The main goal of style transfer is to apply the texture synthesis from one image onto another without changing the high-level content of the original image. During this process, we need to find a stylized image that contains the semantic content of a content image and texture style of an arbitrary image [11]. Recent works of image style transfer with Convolutional Neural Networks (CNNs) allow us to split and combine the representations of high-level semantic content and the style in an image [3], which leads to a new approach for style transfer with CNNs. In this section, we will first understand

the image content representations and reconstruction. Then we look into how to find the texture style of images. Finally, we focus on the state-of-the-art style transfer techniques based on image content reconstruction and style reconstruction.

2.1.1 Image Semantic Content

It is a tough task for computers to identify the semantic content of an image, which therefore makes it difficult to render an image in different styles with high-level features. The main limitation is that the computer can not understand the content and explicitly identify the high-level semantic information from an image like human beings.

Many non-parametric algorithms can perform style transfer while preserving the structure of the content image, but the major fundamental limitation for them is that they only use low-level image features [3].

However, the recent research has shown that the high-level information of an image can be learned by CNNs optimised for computer visions tasks [1][12]. A CNN model trained with sufficient data on one task, such as image recognition, can generalise across datasets and even tasks, such as texture classification and style classification [13].

CNNs usually have several to a hundred hidden layers, depending on the task and the size of the dataset. Each layer in the network contains a set of filters learned by the model. These filters become increasingly complicated when the network becomes deeper. Given an input image, it will be encoded by the filters in each layer in a Convolutional Neural Network. By visualising the image information encoded at different layers in a network, it is found that higher layers are more complicated than lower layers and contain more complex features [12]. Please refer to 2.1 to have a look at how feature map looks like in different layers. In the later work, deconvolutional network

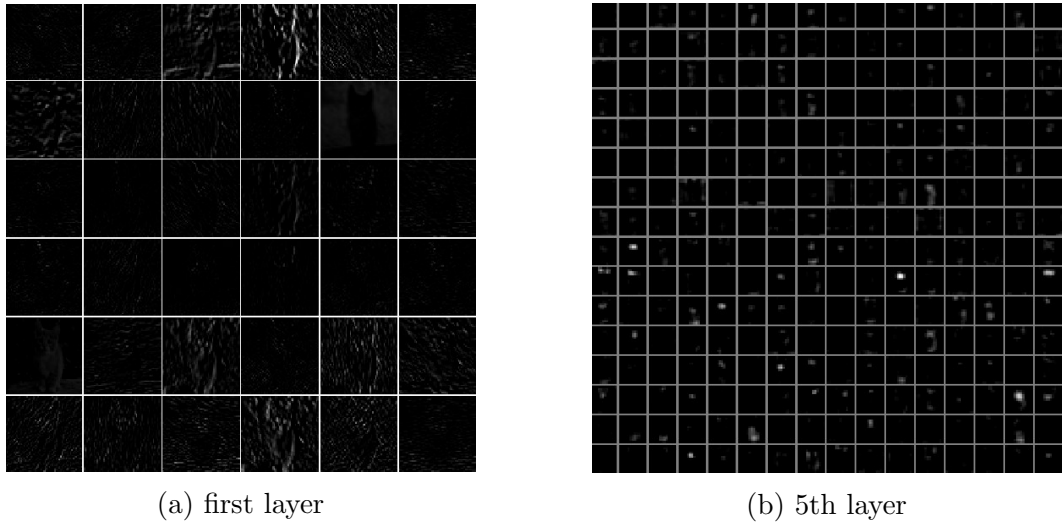


Figure 2.1: Feature Map: Typical-looking of the image information encoded in the first layer (a), and the 5th layer (b). Source: Trained AlexNet looking at a picture of a cat. Every single box shows a feature map of a specific filter in that layer [15].

is applied to visualise the features learned by a model [14]. The author found that the feature maps can present only low-level features at low-level layers, such as object edges or colours, but high-level layers can detect more circular features like faces and eyes.

These findings show that images content can be represented and compared by the feature maps stored in different layers in CNNs. In another work the author wants to characterise the information in the model directly to analysis the representations of the image [16]. To achieve that, an image needs to be reconstructed from the image information encoded in the network, i.e., computing an approximate inverse from the feature map. Given an image \vec{x} , the image will go through the CNN model and be encoded in each layer by the filter responses to it. We use F^l to store the feature maps in a layer l where F_{ij}^l is the feature map of the i^{th} filter at position j in layer l . Let P^l be the feature maps for the content image in layer l , and we can update the pixels of

image x to minimise the loss \mathcal{L} to make sure these two images have the similar feature maps in the network and thus have the similar semantic content:

$$\mathcal{L}_{content} = \frac{1}{2} \sum_{i,j} (F_{ij}^l - P_{ij}^l)^2$$

The loss is defined as squared-error in the original work. To minimise the loss function, we can perform gradient descent on a blank image, which matches the feature responses of the original image, to reconstruct the image from the image representations and visualise the information left in different layers. The gradient of the above loss for the feature maps in layer l equals:

$$\frac{\Delta \mathcal{L}_{content}}{\Delta F_{ij}^l} = \begin{cases} (F^l - P^l)_{ij} & F_{ij}^l > 0 \\ 0 & F_{ij}^l < 0 \end{cases}$$

2.1.2 Image Style

There are two basic texture-generating approaches. The first one is called non-parametric resampling, which is to create new images by resampling pixels or whole patches of the original image [17]. This method can not help us extract the style and build a textures model from a given image, so we need to consider defining a parametric texture model. Javier's work presents such a parametric model for texture synthesis, which is based on a set of pairwise joint statistics and it is probably the best parametric model [18]. In the later work [3], the author found that images that produce the same texture representation in the neural network can have the same style, which enables us to create new images of one style with the texture representation. Based on this finding, a new approach to modelling textures with regard to CNNs is proposed. The feature maps

in a few layers of a trained network are used for representing the texture by the correlations between them. Instead of using these feature maps directly, the correlations between the different channels of the feature maps are given by Gram matrix G^l , where the $G_{i,j}^l$ is the inner product between the vectorised i^{th} feature map and j^{th} feature map in layer l :

$$G_{i,j}^l = \sum_k F_{ik}^l F_{jk}^l$$

The original texture is passed through the CNNs, and the Gram matrices G^l on the feature responses of some layers are computed. If we want to create a new image based on the texture information encoded in the networks, we can pass a white noise image through the CNNs and compute the Gram matrices difference on every layer included in the texture model as the loss. If we use the loss to perform gradient on the white noise image and try to minimise the Gram matrices difference, we can find a new image that has the same style as the original image texture. Figure 2.2 shows how texture generation works. The loss is computed by the mean-squared distance between the Gram matrix of two images. So let A^l and G^l be the Gram matrix of two images in layer l , the loss of that layer equal:

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (G_{ij}^l - A_{ij}^l)^2$$

and the loss for all chosen layers:

$$\mathcal{L}_{style} = \sum_{l=0}^L w_l E_l$$

Where L is the collection of chosen style representations layers, w_l are the weighting

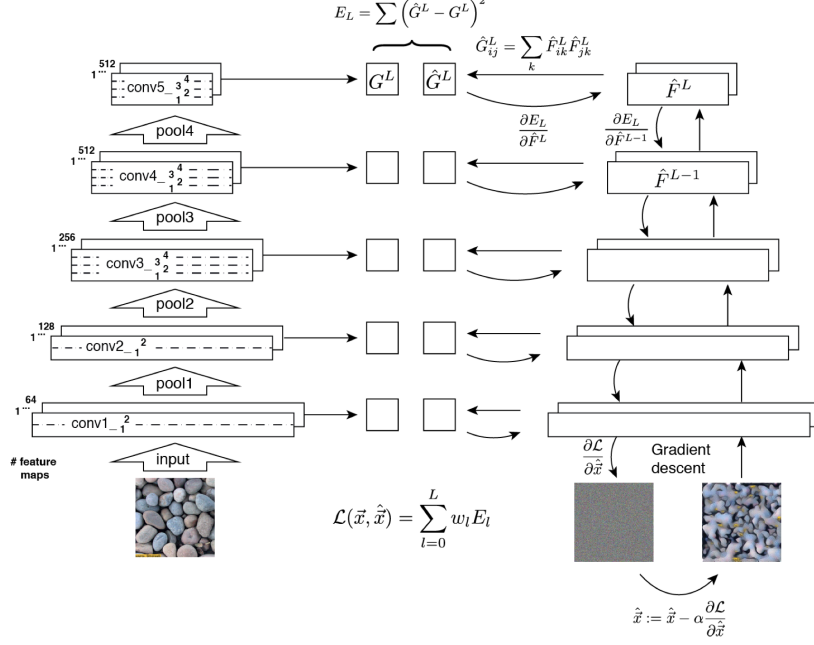


Figure 2.2: Style Generation. The original image \vec{x} is passed through the convolutional neural network and compute the feature maps for each layer l in the network. G^l is used to represent the Gram matrices responses of a set of layers. The white noise image $\hat{\vec{x}}$ is passed through the same network, and we compute the \hat{G}^l . The loss function E^l is computed on every layer based on the difference between two Gram matrices. The total loss function L is a weighted sum of the contributions E^l from each layer. Finally, gradient descent is used to update the pixel values on the noise image, and a new image is found which produces the same Gram matrices G^l as the original texture does [19].

factors for each layer in the collection. When performing gradient descent on the input images, the derivative can be computed:

$$\frac{\Delta E_l}{\Delta F_{ij}^l} = \begin{cases} \frac{1}{N_i^2 M_i^2} ((F^l)^T (G^l - A^l))_{ji} & F_{ij}^l > 0 \\ 0 & F_{ij}^l < 0 \end{cases}$$

2.1.3 Descriptive Neural Style Transfer

Current neural style transfer algorithms can be classified into two categories, descriptive neural methods and generative neural methods [20]. In this section, we review both of them and compare their performance in the different situations.

The first descriptive neural model for style transfer was proposed in 2016 [3]. The author proposed to use Gram matrices to represent the style information and use feature maps to store the content information. During the experiments, the VGG network is used to reconstruct the stylized image by minimising the loss by gradient descent with backpropagation. The input image is passed through the CNN, and some filtered images are computed at each layer. The filtered images become smaller as along the processing hierarchy since there is some downsampling process such as max-pooling layer and average-pooling(fig 2.3). The basic idea of this work is to jointly minimise the distance of the feature representations of a white noise image from the image content representation and the painting style representation defined on some layers of the Convolutional Neural Networks. The author considers style transfer to be successful if the generated image looks like the style image as well as presents the principal objects of the content image.

This work firstly demonstrated the capacity of CNNs in creating artistic images by reconstructing the image content and style separately. Some other interesting findings are also proposed in this work. The author found that replacing the max-pooling operation by average-pooling can create slightly more appealing results. Besides, this work makes it possible for us to work out the trade-off between style and content to create the images we like. What is more, the author also performed some experiments on the initialisations, and he found that different initialisation methods do not impact

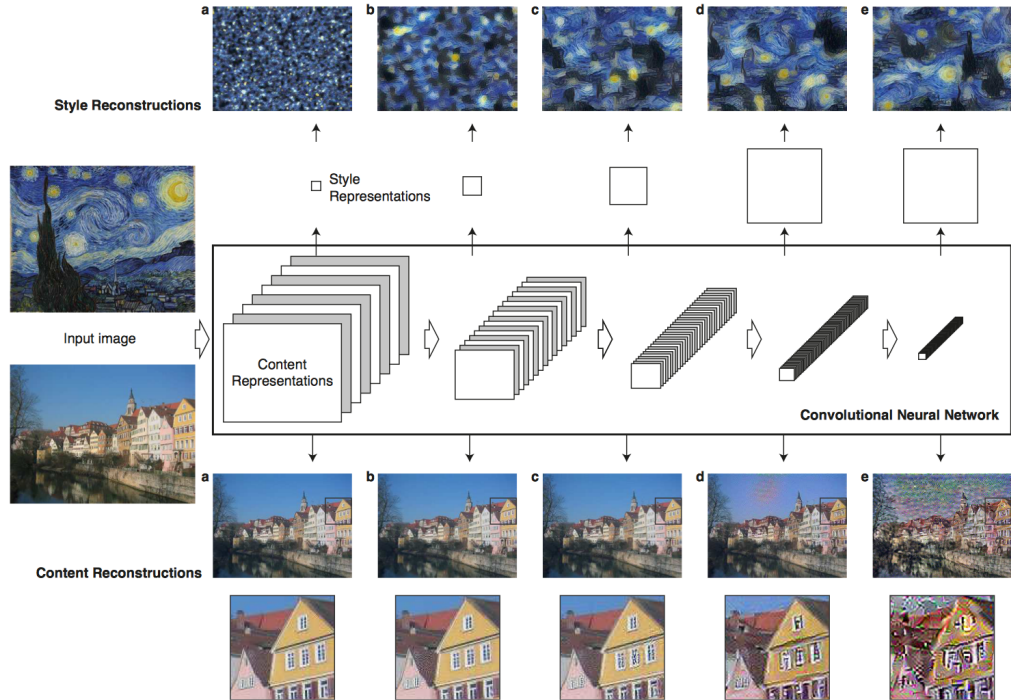


Figure 2.3: Style representations and content representations in a Convolutional Neural Network (CNN) in different layers [21].

the outcome of the synthesis produced in this work. Since then, this creative method has received increasing attention, and many improvements have been made upon it.

2.1.4 Generative Neural Style Transfer

Although the descriptive neural style transfer model can produce impressive stylized images, there are still some efficiency issues since the work is based on image iteration. Every time we want to generate a new image we need a forward and backward pass through the trained network. As we are going to employ the style transfer as a data argumentation strategy, the performance is of great importance during the training process.

Based on the descriptive model, fast neural style transfer methods have been pro-

posed recently. Instead of using image iteration method, these algorithms with higher efficiency are based on model iteration. In these works, the authors trained models that can generate stylized images directly so that we can get stylized images quickly. This method is called Generative Neural Style Transfer [5].

In a Generative Neural Style Transfer system, there are mainly two components: an image transformation network and a loss network. The image transformation network is usually a deep residual convolutional neural network parameterised by weights. It is used to transform input images into output images, which will be fed to the loss network. The loss function in the loss network computes a scalar value to measure the difference between the output image and a target image, including content reconstruction loss and style reconstruction loss.

2.1.5 N-Styles FeedForward Network

Johnson's work [5] opens the door to generative models in style transfer field. This new method is much faster than descriptive ones, but the limitations are also apparent: each generative network is trained for a single style, which means that we need to train multiple networks for different styles, hence it is still time-consuming and inflexible when different styles are involved. Based on this observation, an algorithm [22] that can train a conditional style transfer network for multiple styles is proposed. Their work comes from the observation that many styles can share the computation to some degree. If we train different networks for different styles, the shared computation is thrown away.

Based on this point, the author trains the generative model $T(c, s)$ with abundant paintings of set artistic styles. This model can produce new stylized images when given

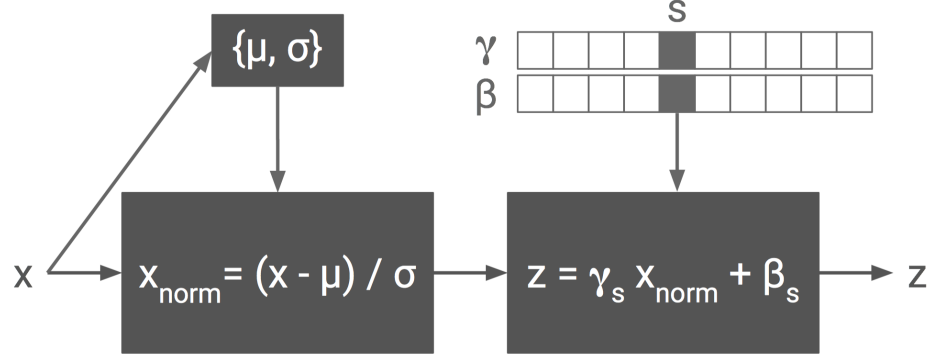


Figure 2.4: The feature maps x is normalized across both spatial dimensions and subsequently scaled and shifted using style-dependent parameter vectors γ_s , β_s . s marks the style we want to apply [11].

an input image and the identity of the style. To share the convolutional weights of the model across many styles, a method named conditional instance normalisation is proposed. The purpose of this method is to transfer the feature maps x of a layer into z specific to a style s . If there are N styles and C feature maps, we will get a $N \times C$ matrices. We use γ and β as hyperparameters so following formula can achieve a transformation on a specific style:

$$z = \gamma_s \left(\frac{x - \mu}{\sigma} \right) + \beta_s$$

In this formula, μ and σ are the mean value and standard deviation of x from spatial axes, while γ and β are selected from the matrices according to the style s (Figure 2.4). With this normalisation, we can use only one feed forward pass through the network to stylize the input image into N different styles.

2.2 The Art of Data Augmentation

For many computer vision tasks, data augmentation is an important approach to enlarge the training dataset and avoid overfitting. In this section, we will introduce the most common data augmentation methods used in computer vision tasks in recent years.

2.2.1 Traditional Data Augmentation

The label-preserving transformations can be used to generate additional images so that CNNs can infer the transformation invariance from the dataset [23]. These transformation strategies are considered the most natural and universal method to reduce overfitting for training a neural network on images [1], including flipping, rescaling, and cropping, etc. Many deep learning frameworks also provide a set of APIs, allowing us to easily augment the dataset on the fly. Figure 2.5 shows that traditional data augmentation can generate new training instances but does not change the main content in the original images, hence the CNNs can learn these images as well.

In the research back to 2012[1], the author applied two kinds of argumentation methods to improve the performance of their model. The first one is horizontal reflections, without which their network would get substantial overfitting with only five layers. The second method is to perform PCA on the RGB values of the image pixels in the training set, and a few found principal components are used for training, which can reduce over 1% on the top-1 error rate. Similarly, The ZFNet [14] and VGGNet [2] also apply multiple different crops and flips for each training image to boost training set size and improve the performance of their models. These well-known CNNs architectures achieved outstanding results in the ILSVRC competitions, and their success

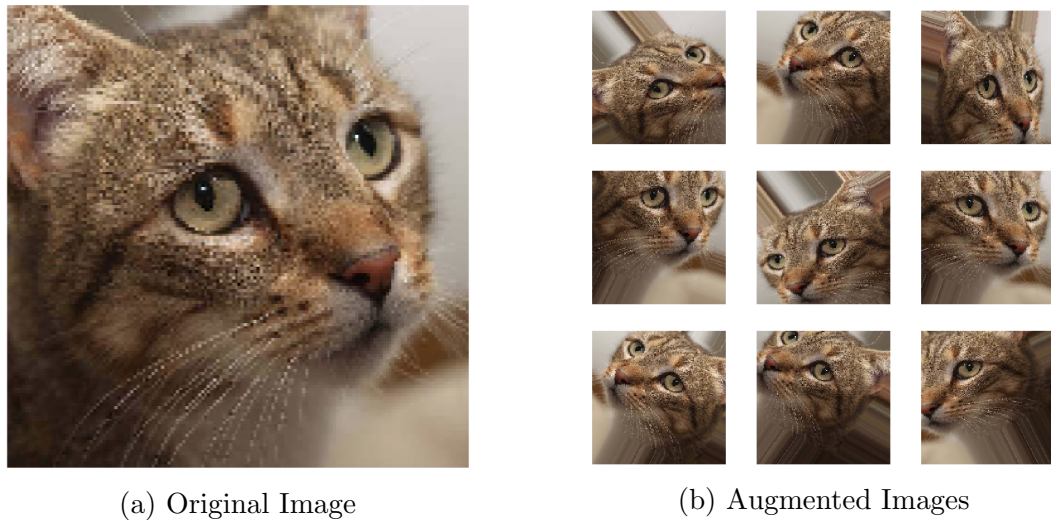


Figure 2.5: Traditional Data Augmentation: applying transformations on the original image doesn't change the fact that this is still a cat image [24].

demonstrated the effectiveness and importance of data augmentation when training a deep neural network.

2.2.2 Advanced Data Augmentation

Since data augmentation has been shown an effective way to increase the performance of training a CNN model, there is also much research on more advanced data augmentation techniques.

In the work [25], the author explores to use Generative Adversarial Nets [26] to generate stylized images to augment the dataset. Their work is called neural augmentation, which makes use of CycleGAN [27] to create new stylized images for each of the examples in the dataset. This method is finally tested on a 5-layer network with the MNIST dataset and performs better than most of the traditional approaches. They also explored to combine neural nets with the images classification network to build a new system, where the transformation neural nets can be trained with the classification

loss so that they can identify the best augmentations for a given dataset.

Another method proposed recently [28] also shows an improvement in the accuracy for image classification task, compared with traditional data augmentation strategies. Their work is named SamplePairing. The basic idea of this technique is to synthesize a new image from two images randomly picked from the training set. The synthesizing method is simple: a new image is created based on the average value of each pixel of two images. This sample data augmentation technique is quite easy to implement, but it is very effective for image classification tasks. Since this method can generate lots of new images quickly, it is especially useful when the number of image examples available for training is limited, such as medical computer vision tasks.

2.3 TensorFlow and Keras

This section briefly describes the software and tools involved in the experiments. With the development of technologies, there are many platforms and tool available for training CNNs, but we finally choose Tensorflow, Python, and Keras as the tools, since there are many advantages.

We choose Tensorflow as the essential framework. TensorFlow is a high-performance numerical computation and machine learning library. There is a set of Python APIs which enables us to build deep learning models and algorithms easily and quickly. Besides, as the architecture of Tensorflow is very flexible, its portability allows us to run and deploy our models across CPU, GPU and TPU. Not just for PC or a cluster of servers, we can also deploy the models on mobile and edge devices [29].

In Tensorflow, the model is represented by data flow and the mathematical operators are represented by a node, and the edges between two nodes can carry tensors (multi-

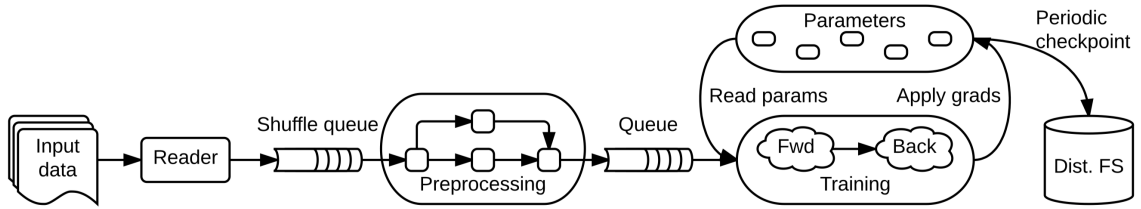


Figure 2.6: TensorFlow dataflow: reading data, preprocessing, training, and checkpointing state [30].

dimensional arrays). With the data flow graph built by nodes and edges, the user can compose the layers of a model with high-level scripting interface. Tensorflow hold the execution until the entire program is available, so it can optimise the program using global information. With Keras, high GPU utilisation is achieved by using the graph's dependency structure to issue a sequence of kernels to the GPU without waiting for intermediate results [30]. The training pipeline of Tensorflow can be found in figure 2.6.

Keras is the one of the most popular model-level neural network libraries based on Tensorflow. It does not handle low-level operations such as tensor manipulation and differentiation. Many primary platforms for CNNs like CNTK and Theano are also supported by Keras. It is of great importance that we can go from idea to result with the least possible delay to perform proper research, so the goal of Keras is to enable the researcher to do fast experimentation [31].

Keras has a simple and consistent interface optimised for common use cases. The error logs are also very clear which makes it easy for its users to debug. Besides, any piece of code that you write with Keras can be run with any of these backends without having to change anything in the system [32]. Figure 2.7 shows the hardware stack of Keras.

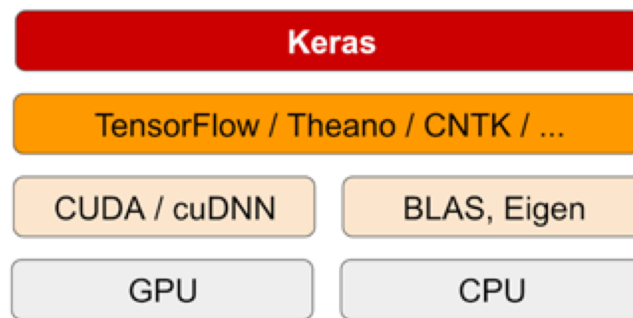


Figure 2.7: The hardware stack for Keras: Model level and cross-platform [32].

Chapter 3

Design

The chapter introduces the high-level design of the style transfer model and the image classification models (VGG16 and VGG19). The first section explains the architecture and the mechanism of the fast style transfer model used in this project. Then we describe the architectures of image classification models. The last section gives an overview of how to employ the style transfer as an image augmentation strategy to improve the performance of two VGG models.

3.1 Style Transfer Architecture

In the previous work for texture synthesis, the author produces high-quality stylized images with optimising the input image based on a forward and backward pass through the trained network [3]. This method is time-consuming and needs massive computation for each single input image. The goal of this dissertation is to verify if style transfer can be used as a data augmentation strategy, so we need to style transfer model that can generate new images efficiently.

In our design, we choose a generative neural style transfer networks, where we train a feed-forward network named as transformation network for each style which can generate stylized images quickly without backward pass through the network.

3.1.1 Transformation Network

To successfully train a generative style transfer model, we need two neural networks in the architecture. The first network is named as the transformation network, which is used to generate new images based on input images. And the second network is a loss network used to evaluate the generated images and compute the loss. We then use the loss function to optimise the transformation network to generate better images. The overview of the whole architecture can be found in figure 3.1.

We can treat the transformation network as $f_W(x)$. We use deep residual convolutional neural network [33] to build the transformation network, and more implementation detail will be introduced in Chapter 4. The network can be fed an input image x and transfer it to an output image \hat{y} through the network $\hat{y} = f_W(x)$. This network is trained with stochastic gradient descent [34] to minimise a weighted combination of loss functions. We can treat the overall loss as a linear combination of the two the content reconstruction loss and style reconstruction loss, and the weights of two losses can be modified and fine-tuned. By minimising the overall loss, we can get the model well trained with weights W^* :

$$W^* = \operatorname{argmin}_W \mathbf{E}_{x, \{y_i\}} \left[\sum_{i=1} \lambda_i \ell_i(f_W(x), y_i) \right]$$

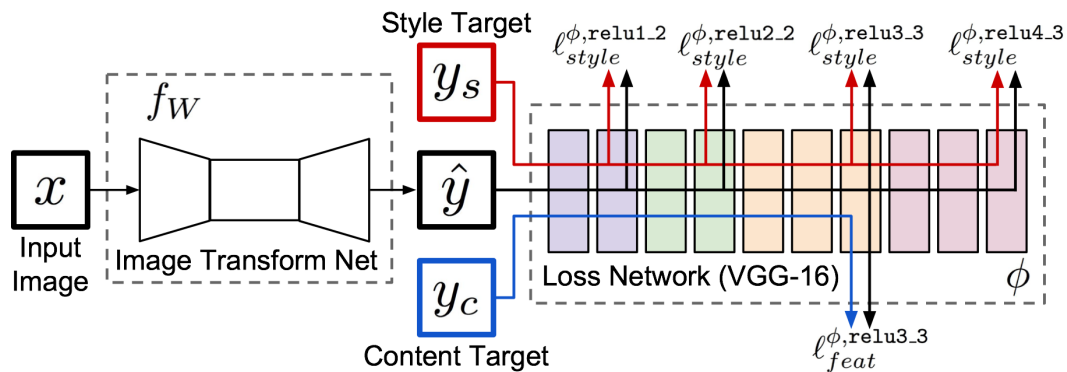


Figure 3.1: System overview: image transformation network is trained based on the loss computed by a pretrained loss network [5].

3.1.2 Loss Network

Since we already define the transformation network that can generate stylized images, we also need to create a loss network that is used to represent several loss functions to evaluate the generated images and use the loss to optimise the transformation network based on stochastic gradient descent.

We use a deep convolutional neural network θ pretrained for image classification on imageNet dataset [35] to measure the texture and content differences between the generated image and the target image. The recent work has shown that deep convolutional neural networks can encode the texture information and high-level content information in the feature maps [19][16]. Based on this finding, we define a content reconstruction loss ℓ_c^θ and a style reconstruction loss ℓ_{style}^θ in the loss network and use their weighted sum to measure the total difference between the stylized image and the image we want to get. For every style, we train the transformation network with the same pretrained loss network, which means the loss network will stay the same throughout the training process.

Content Reconstruction Loss: The images with similar features should have similar feature representations computed by the deep CNN loss network [16]. We use $\theta_j(x)$ to represent the j th layer of the loss network θ for processing image x and the content image, which should be the shape of $C_j * H_j * W_j$. Then we use the Euclidean distance to represent the *content reconstruction loss*. Our goal is to find a generated image from the transformation network that can minimise the content reconstruction loss so that we can finally get the $\theta_j(\hat{y})$ that has the similar high-level semantic features but does not have to be the same with the content target image:

$$\ell_{contet}^{\theta,j}(\hat{y}, y) = \frac{1}{C_j H_j W_j} \|\theta_j(\hat{y}) - \theta_j(y)\|_2^2$$

Style Reconstruction Loss: We also want the generated images to have the similar texture synthesis with the style target image, so we want to penalise the style differences with the style reconstruction loss. More specifically, the work [19] found that we can use the Gram Matrix from the feature map to calculate the texture difference between the two images.

Same as above, with a feature map $\theta_j(x)$ in j th layer of shape $C_j * H_j * W_j$, the *Gram matrix* are defined as $G_j^\theta(x)$ to be a $C_j \times C_j$ matrix:

$$G_j^\theta(x)_{c,c'} = \frac{1}{C_j H_j W_j} \sum_{h=1}^{H_j} \sum_{w=1}^{W_j} \theta_j(x)_{h,w,c} \theta_j(x)_{h,w,c'}$$

The Style Reconstruction Loss can be represented as the squared Frobenius norm between the Gram matrices from two different images:

$$\ell_{style}^{\theta,j}(\hat{y}, y) = \frac{1}{C_j H_j W_j} \|\theta_j(\hat{y}) - \theta_j(y)\|_F^2$$

We do not require the \hat{y} and y to have the same size since their activations at $\theta_j(x)$ are of the same shape in the deep convolutional neural network.

Terms Balancing: As we have above definitions, generating an image \hat{y} can be seen as solving the optimising problem in figure 3.1. We initialise the image with white noise, and the work [3] found that the initialisation has a minimal impact on the final results. λ_c , λ_s , and λ_{TV} are hyperparameters that we can tune according to the monitoring of the results. To get the stylized image, we need to minimise a weighted combination of two loss functions:

$$\hat{y} = \underset{y}{\operatorname{argmin}} \lambda_c \ell_{content}^{\theta,j}(y, y_c) + \lambda_s \ell_{style}^{\theta,J}(y, y_s) + \lambda_{TV} \ell_{TV}(y)$$

3.2 Image Classification Network

To evaluate the effectiveness of this design, we also need to perform image classification tasks with the stylized images. When performing image classification, for each given image, the programs or algorithms need to produce the most reasonable object categories [4]. The performance of the algorithm will be evaluated based on if the predicated category matches the ground truth label for the image. Since we will provide input images from multiple categories, the overall performance of an algorithm is the average score of overall test images. The score usually is Top-1 accuracy or Top-5 accuracy.

Once we train the transformation network that can generate the stylized images, we will apply it on the training dataset to create a larger dataset. The stylized images are saved on the disk with the ground truth categories. We then use them with their original images to train the neural networks to solve the image classification problems.

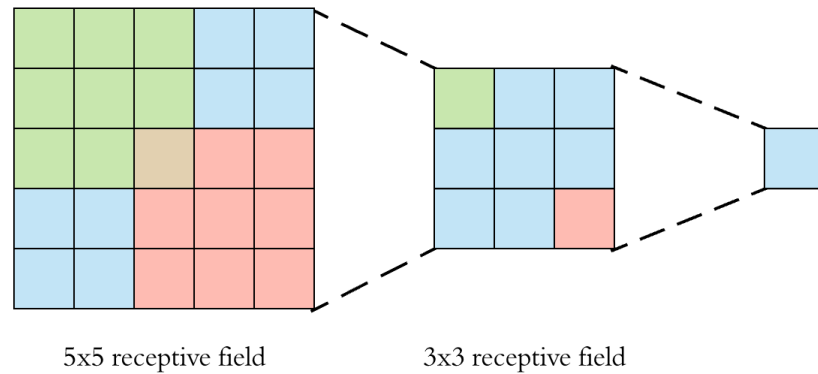


Figure 3.2: Receptive Field Overview [24].

In this research, the model we chose is VGGNet, which is a simple but effective model [2]. Their team got the first place in the localisation and the second place in the classification task in ImageNet Challenge 2014. This model strictly used 3×3 filters with stride and pad of 1, along with 2×2 max-pooling layers with stride 2. $3 \times 3 \times 3$ convolutional layers back to back have an effective receptive field of 7×7 . Compared with one 7×7 filter, 3×3 filter size can have the same effective receptive field with fewer parameters. Figure 3.2 shows the overview of the receptive field.

VGGNet also uses scale jittering as one of the data augmentation techniques. In the original work, this model took a long time to train on 4 Nvidia Titan Black GPUs for two to three weeks. But with the development of GPU and smaller dataset, we can train the model within hours based on Caltech dataset, which is acceptable for this research. There are two different architectures of VGGNet. The first one is VGG16 that has 16 convolutional layers, while the other one, VGG19, has 19 convolutional layers and slightly better performance than VGG16. The high-level architecture of VGG16 and VGG19 can be found in figure 3.3



Figure 3.3: VGGNet Overview: Strictly use 3×3 filters with 2×2 max-pooling layers [36].

3.3 Experiments

Our goal of this research is to explore how useful style transfer can be compared and combined with other traditional approaches for data augmentation.

To fully understand the effectiveness of the style transfer on VGGNet for image classification problem, we need to do experiments from multiple perspectives. The first experiments are to use traditional transformations alone. For each input image, we generate a set of duplicate images that are shifted, rotated, or flipped from the original image. Both the original image and duplicates are fed into the neural net to train the model. The classification performance will be measured on the validation dataset as the baseline to compare these augmentation strategies.

The second experiments are to apply the well-trained transformation network to augment the training dataset. For each input image, we select a style image from a subset of different styles from famous artists and use the transformation network to generate new images from the original image. We store the newly generated images on the disk, and both original and stylized images are fed to the image classification network. For the chosen styles, we try to select the images that look very different. The pipeline can be found in figure 3.4

Furthermore, to explore if we can get better results, we go further to combine two approaches and generate more duplicated images. More details about the combination will be described in Chapter 4.

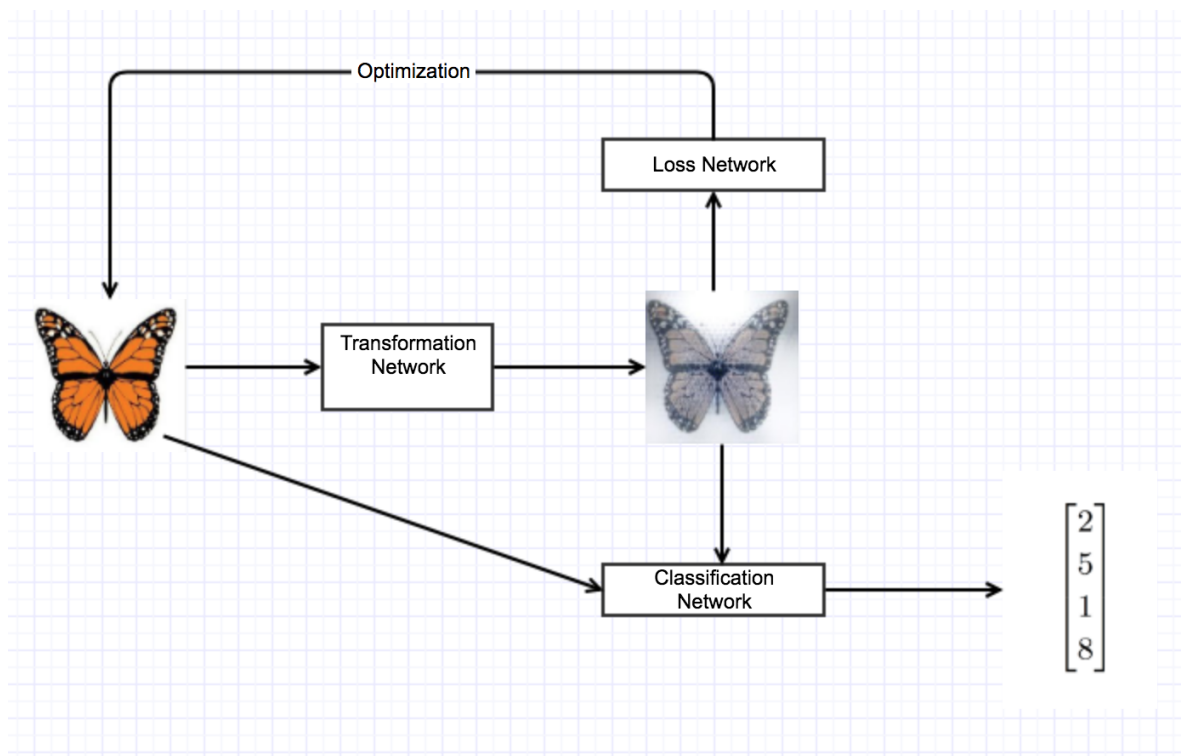


Figure 3.4: The pipeline of the overall architecture.

Chapter 4

Implementation

In this chapter, we introduce the experiments and the specific implementation details of neural style transfer networks and image classification networks, following the design laid out in Chapter 3. We discuss the implementation of Augmentation Module firstly. Next, we talk about the image classification network (VGGNet). Finally, we combine the transformation network and VGGNet and show the overall pipeline of the system. The entire implementation code and datasets can be found on the attached CD.

4.1 Augmentation Module

Before any stylized images are created, we need to train a transformation network for each given style. As designed in Chapter 3, to successfully train a transformation network we need two different deep neural networks (see figure 3.1). We describe the implementation of these two different networks one by one in this section.

4.1.1 Transformation Network

The first network is named transformation network, which is trained to be a generative network that can generate stylized images based on the given style image. We follow the state of the art implementation from [37] and [38], with some enhancement based on our experiments.

Five residual blocks are used in the transform net to avoid optimisation difficulty when the network gets deep [33]. For each residual block, we give an input tensor x . After a series of convolution layer, batch normalisation layer and ReLU layer, we get the output $F(x)$ from the residual block. Then we sum the output tensor and the input tensor into $H(x)$ and feed it to following up layers. Other none residual convolutional layers are followed by Ulyanov's instance normalisation (we do not use batch normalisation) and ReLU layers. At the output layer, we use a scaled tanh to get an output image has pixels in the range from 0 to 255. The exact outlook of the transform net architectures can be found below:

```
conv1 = _conv_layer(image, 32, 9, 1)
conv2 = _conv_layer(conv1, 64, 3, 2)
conv3 = _conv_layer(conv2, 128, 3, 2)
resid1 = _residual_block(conv3, 3)
resid2 = _residual_block(resid1, 3)
resid3 = _residual_block(resid2, 3)
resid4 = _residual_block(resid3, 3)
resid5 = _residual_block(resid4, 3)
conv_t1 = _conv_tranpose_layer(resid5, 64, 3, 2)
conv_t2 = _conv_tranpose_layer(conv_t1, 32, 3, 2)
```

```
conv_t3 = _conv_layer(conv_t2, 3, 9, 1, relu=False)
preds = tf.nn.tanh(conv_t3) * 150 + 255./2
```

We use 9×9 filter size at first and last layers, and all other convolutional layers use 3×3 filters. With small filters, we can make a deep neural network and in the original VGGNet work the author has found that a deep neural network with small filters outperforms a shallow net with larger filters [2].

4.1.2 Loss Network

The output images from the transformation network are fed to the second network named as Loss Net. We use the pretrained VGG19 on ImageNet as the loss network. As described in Chapter 3, we sum the content reconstruction loss and style reconstruction loss into the final loss of the evaluation:

$$total_loss = content_loss + style_loss + tv_loss$$

The network generally follows the architecture presented in figure 3.3. Here we give each of the layers a name for the convenience of loss analysis. The names for each layer from bottom to above are:

```
conv1_1 conv1_2 pool1
conv2_1 conv2_2 pool2
conv3_1 conv3_2 conv3_3 conv3_4 pool3
conv4_1 conv4_2 conv4_3 conv4_4 pool4
conv5_1 conv5_2 conv5_3 conv5_4 pool5
```


We do not name the last three fully connected layers since we do not use them in the loss network.

Content Reconstruction Loss: Content reconstruction loss is calculated at layer *relu4_2* of the VGG19 loss network. In Johnson’s implementation [5], *relu2_2* is employed. Since the previous work shows that higher layers are more complex than lower layers and present more complex features, we use higher layer *relu4_2* in our implementation.

Style Reconstruction Loss: Style reconstruction loss is calculated at layer *relu1_1*, *relu2_1*, *relu3_1*, *relu4_1* and *relu5_1* of the VGG19 loss network, which are shallower than in the original implementation, because it leads to larger scale style features in transformations during the implementation [37].

4.1.3 Dataset

The transformation network is trained on the COCO 2014 train collection [39], which contains more than 83k training images and is enough to get the transformation network well trained. Since these images are used to feed the transformation network, we do not need the label information. Before going through the transformation network, all the images are resized to 256×256 .

4.1.4 Training

After we define the networks, we need to start the training process. All the models are coded with Tensorflow and Keras in Python. The platform is Ubuntu 16.04 with one single GTX 1080 GPU and Intel i7 7700k CPU.

For the code part, the batch size is 4 with two epochs over the training images.

Adam is applied as the optimizer [40]. The learning rate is set at 1×10^{-3} . We use these default values following the original work and found that they work well for generating new stylized images [5], so we keep these values the same during all the training process.

We select eight different style images that look different from each other to get eight different transformation network for later usage (augment the image classification dataset). The style images are presented in Appendix A. It takes 4 to 6 hours on the given machine to get a transformation network well trained. More detailed documentation has been put in the code in the attached CD.

4.2 Image Classification Network

For image classification tasks we use two different datasets and pretrained VGG16 and VGG19 networks on ImageNet [4]. The architecture can be found in figure 3.3. For both networks, the original dataset has 1000 classes. Since we only have 102 categories and 257 categories in the dataset we chose, we add one more dense layer on the top of the last Fully Connected layer to make sure the output of our network is located in the correct categories. These two networks are also trained in the same platform mentioned above: Ubuntu 16.04 with one single GTX 1080 GPU and Intel i7 7700k CPU. All the information about the training process as well as the results are stored in the Tensorboard and can be found in the attached CD.

4.2.1 Configurations

Stochastic gradient descent is used as an optimiser with learning rate at 1×10^{-4} . We choose accuracy as the metrics. The default number of epochs is 70, but we also use

EarlyStopping provided by Keras to stop the training when we find no improvement of the validation loss after ten epochs. The highest test accuracy at all the epochs is reported as the best score. The batch size is 32 for both training and validation process. In the original VGGNet implementation, the image size for training is 224×224 , so we also resize the input image to the same size.

4.2.2 Dataset

We're using 2 different datasets in our experiments, caltech101 [6] and caltech256 [7]. We keep training and testing on a fixed number of images and repeating the experiment with different augmented data and compare the results with others. The images are divided by a 70:30 split between training and validation for both datasets. Please refer to table 4.1 for the information summary of the two datasets.

Caltech101: This dataset was collected by choosing 101 object categories from Google Images. There it contains 102 classes as there is one more background category. The total image number is 9144. For each class, there are 40 to 800 images, and images are roughly 300x200 pixels in size.

Caltech256: This dataset has more than 30 thousands images. Compared with Caltech101, there are several improvements. Firstly, the category number more than doubled. And then the minimum amount for each class is increased to 80. Since this dataset has much more images, it takes more time to train the classification network on this dataset.

| Name | Published Date | Total Categories | Images |
|------------|----------------|------------------|--------|
| Caltech101 | 2003 | 102 | 9144 |
| Caltech256 | 2006 | 257 | 30607 |

Table 4.1: Image Classification Dataset

4.3 Experiments

For the chosen styles, we try to select the images that look very different. Finally, eight different images are chosen as the style input to train the transformation network. Please find appendix for the information of 8 style images.

Once we obtain the augmented images, we feed them into a neural net that does classification. The output of the VGGNet is a score matrix for the weights for each class. We can also replace VGGNet with more complex network such as ResNet that can predict image classes or other states of the art network and do the similar experiment in the future work. Figure 4.1 shows the stylized images created by the transformation network.

4.3.1 Stylized Image Overview

To test the effectiveness of various augmentation techniques, we run multiple experiments on the Caltech dataset with a single style, combined styles and we even combine our approach with the traditional augmentation strategies.

4.3.2 Content Weights

We also want to explore if the content weights and style weights impact the performance of our approach, we change the portion of the content weights and style weights to train a new transformation network for Wave style. The original portion is $Content_Weight : Style_Weight = 9 : 100$. We increased the content weight to $Content_Weight : Style_Weight = 9 : 80$. The output is compared in figure 4.2. Then we use the new stylized images to train the classification network and compare the performance with the original setting.
















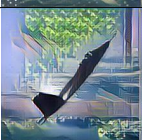
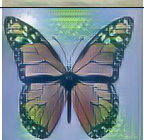
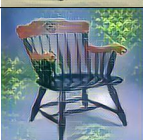





















| Style Name | Airplane | Butterfly | Chair | Cup | Watch |
|---------------|---|---|--|---|---|
| None |  |  |  |  |  |
| Snow |  |  |  |  |  |
| Wave |  |  |  |  |  |
| Your Name |  |  |  |  |  |
| La Muse |  |  |  |  |  |
| Rain Princess |  |  |  |  |  |
| Scream |  |  |  |  |  |
| Sunflower |  |  |  |  |  |

Figure 4.1: Stylized images created by the transformation network of 8 different styles.







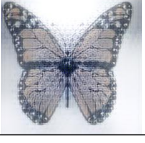



| Style Name | Airplane | Butterfly | Chair | Cup | Watch |
|------------|---|---|--|---|---|
| Wave 1 |  |  |  |  |  |
| Wave 2 |  |  |  |  |  |

Figure 4.2: Stylized images with the same style and different content weights. Wave 2 has more content weight than Wave 1

Chapter 5

Evaluation

This chapter presents the evaluation results of this project. We evaluate the results from multiple perspectives based on the image classification Top-1 accuracy of the VGGNet. The results of experiments on traditional augmentation are collected as the baseline. We then do experiments on every single style and further combined styles. We also combine the traditional methods with style transfer method together to verify the effectiveness and see if we can get better results. The Tensorboard that presents all the results can be found on the attached CD.

5.1 Traditional Image Augmentation

We first train the VGGNet without any data augmentation. Since the VGGNet applied in this research is pretrained based on ImageNet, we did not get any overfitting issue and reached a classification accuracy at 83.34% in one hour training time. We also apply two different traditional image augmentation strategies, Flipping, and Rotation, to train the VGGNet. Finally, we combine Flipping and Rotation. Detailed results

can be found in table 5.1. We found the model itself works the best without any augmentation. And Rotation with Flipping can only reach the 77% accuracy. Besides, the time required for training is quite different. For None and Flipping, training only takes less than one hour. But for FlippingRotation it takes around four hours. During the following experiments, we can still find that Rotation works poorly and the model converges slowly.

| Traditional Image Augmentation | | |
|--------------------------------|---------------|---------------|
| Style Name | Best Accuracy | Training Time |
| None | 0.8334 | 1h |
| Flipping | 0.8305 | 41mins |
| FlippingRotation | 0.77 | 4h |

Table 5.1: Traditional Image Augmentation

5.2 Single Style

We select eight different styles that look different from each other to train the transformation network(all styles can be found in Appendix A). For each image in the training set, it is resized to $224 * 224$, and then we feed it to the well-trained transformation network to generate one stylized image for each style. Both the original images and the stylized images are fed to VGGNet to compute the classification loss. Then we collect the validation accuracy of each training. Only the best accuracy value is collected, and the results can be found in table 5.2. Compared with the traditional strategies, we can see that 7 out of 8 styles work better than the traditional strategies. It can also be seen that the Snow style works the best, reaching an accuracy of 85.26%. YourName style only reaches 82.61%. I think the reason is that the snow style adds to much noise and colour to the original images. The figure 5.1 give a comparison between the



Figure 5.1: Comparison between the original image(airplane) and two different stylized images. Your Name add too many colours and shapes on the original image, which may lead to a bad performance in terms of the image classification accuracy.

original image(airplane) and two different stylized images.

| Single Style with VGG16 and Caltech101 | | |
|--|---------------|---------------|
| Style Name | Best Accuracy | Training Time |
| Scream | 0.849 | 54mins |
| Wave | 0.8468 | 1h 17 mins |
| Udnie | 0.8404 | 54mins |
| Snow | 0.8526 | 1h24mins |
| Your Name | 0.8261 | 51mins |
| Sunflower | 0.8461 | 1h2mins |
| RainPrincess | 0.8494 | 1h19mins |
| LAMuse | 0.8436 | 1h8mins |

Table 5.2: Single Style Transfer Augmentation

5.3 Combined Methods

Firstly, We combine two different styles to verify if we can get better results. In this experiment, we take the original images and feed them to two different transformation networks and generate two stylized images for each input image. We then merge the stylized images and original images to compose the final training dataset, so we finally

have three times the number of the original dataset. We use this augmented data set to train the VGGNet. Meanwhile, we also try to combine the traditional augmentation methods with style transfer together to see if we can get better performance. The results can be found in table 5.3.

| Combined Method With VGG16 and Caltech101 | | | |
|---|------------|---------------|---------------|
| Traditional Method | Style Name | Best Accuracy | Training Time |
| Flipping | None | 0.8305 | 41mins |
| Flipping | Scream | 0.8454 | 50mins |
| Flipping | Wave | 0.8486 | 56mins |
| Flipping | ScreamWave | 0.845 | 2h22min |
| FlippingRotation | None | 0.7521 | 1h7min |
| FlippingRotation | Wave | 0.7837 | 1h48mins |
| FlippingRotation | Scream | 0.7862 | 1h45mins |
| FlippingRotation | ScreamWave | 0.7942 | 2h42mins |

Table 5.3: Combined Style Transfer Augmentation

We can see that, compared with traditional methods(Flipping and Rotation), the combined style(ScreamWave) has considerably better performance but has similar performance with single style method. Since there are more training images, the training takes almost two hours. Another finding is that any training that involves Rotation cannot get a good performance.

5.4 Content Weights Change

We also change the proportion of content weights as well as style weight to figure out if the content weights in transformation network impact the performance. The original proportion is $Content_Weight : Style_Weight = 9 : 100$. We increased the content weight to $Content_Weight : Style_Weight = 9 : 80$.

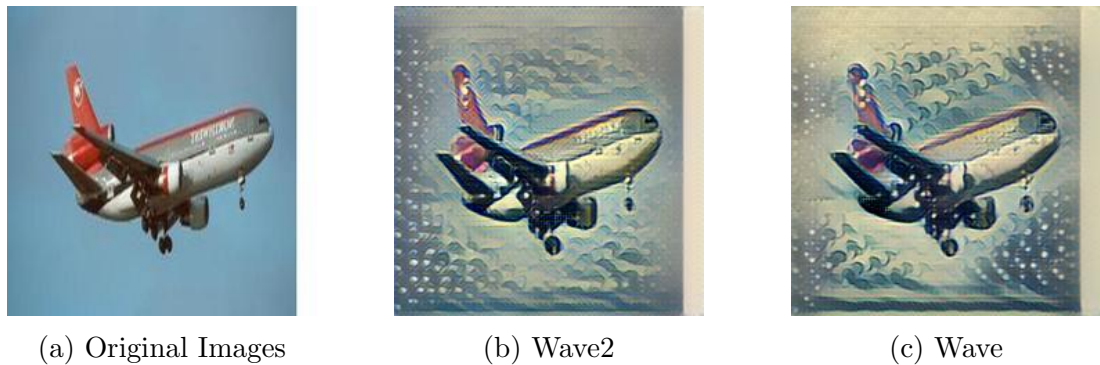


Figure 5.2: Comparison between the original image(airplane) and stylized images with different content weights. The images look similar to each other.

As can be seen from the table 5.4, there is no significant change, and we get the similar results. The figure 5.2 shows that the images look very similar to each other, so the content weight change has a minimal impact on our experiments. Maybe in the future, we can add more content weight and see how the stylized images change accordingly.

| Different Content Weights With VGG16 and Caltech101 | | | |
|---|------------|---------------|---------------|
| Traditional Method | Style Name | Best Accuracy | Training Time |
| None | Wave | 0.8468 | 1h17mins |
| Flipping | Wave | 0.8486 | 56mins |
| N/A | Wave2 | 0.8417 | 1h12mins |

Table 5.4: Different Content Weights

5.5 VGG19

We also did the same experiments on VGG19 and got more observations to see if our approach can generalise well for more network architectures. The results can be found in table 5.5.

| Experiments on VGG19 | | | | |
|----------------------|----------------------|------------|---------------|---------------|
| Dataset | Traditional Method | Style Name | Best Accuracy | Training Time |
| Caltech101 | None | None | 0.8450 | 42mins |
| Caltech101 | Flipping | None | 0.8367 | 43mins |
| Caltech101 | None | Wave | 0.8425 | 1h |
| Caltech101 | Flipping | Wave | 0.8581 | 58mins |
| Caltech101 | None | Scream | 0.8446 | 1h5mins |
| Caltech101 | Flipping | Scream | 0.8465 | 49mins |
| Caltech101 | Flipping Rotation | ScreamWave | 0.8029 | 3h |
| Caltech256 | None | None | 0.62 | 2h26mins |
| Caltech256 | Flipping | None | 0.6666 | 2h9mins |
| Caltech256 | None | Scream | 0.6313 | 3h7mins |
| Caltech256 | Flipping | Scream | 0.6728 | 2h52mins |
| Caltech256 | None | Wave | 0.6272 | 2h57mins |
| Caltech256 | Flipping | Wave | 0.6632 | 2h58mins |
| Caltech256 | Flipping Rotation | None | 0.5853 | 3h47mins |

Table 5.5: Experiments on VGG19 with both Caltech101 and Caltech256

The baseline classification accuracy for Caltech101 is 84.5%. Based on this number, there are some interesting findings in line with the VGG16.

Using Wave or Flipping itself does not improve the performance of VGG19, but if we combine Flipping with Wave we can get a considerable improvement, with accuracy reaching 85.81%. And more complex combination(FlippingRotation with ScreamWave) leads a decrease of the classification accuracy to 80.29%.

Similar result has been found in Caltech256 with VGG19. The single Flipping has an accuracy of 66.66%, while that of the single Screamstands at 63.13%. But if we combine the Flipping with Scream we can get a 67.28% accuracy. The combination between Flipping and Wave gets a 66.32% which is still higher than the single Wave augmentation. Again, if we bring in Rotation as a kind of data augmentation strategy,

the performance is not good, with accuracy at only 58.53%, which is even lower than None augmentation.

The experiments performed on VGG19 show that the style transfer is still an effective data augmentation method, and we can also combine it with some traditional approaches to further improve the performance.

Chapter 6

Conclusion

In this thesis, we proposed a new data argumentation approach based on neural style transfer algorithm. This approach is proven to be an effective way to increase the performance of CNNs on image classification tasks. The accuracy for VGG16 is increased to 85% from 83% while for VGG19 the increase is from 84.50% to 85.81%. We also found that we can combine this new approach with traditional methods like Flipping. Due to the limited time allowed for this dissertation, there exist certain limitations and some future works remains to be done.

6.1 Limitations

In this section, I would like to describe some limitations of this dissertation.

Limited Tasks: We only tested our method on image classification task. Since there are many different kinds of tasks in the Computer Vision field, we can also apply for style transfer as a data augmentation method on tasks like image segmentation and image recognition. We may test more CNN architectures and datasets to verify the

ability of generalisation of this approach.

Styles Category: Even we tried to select images of different styles, we did not classify the images according to their category. We do not know if an image is Abstract, Classical or Expressionism. Maybe in the future, we can choose several images that belong to one category to test if the category has an impact on the image classification performance, or if different images in the same category have different effectiveness.

Slowness: Compared with traditional data augmentation methods, neural style transfer is much slower. The slowness can be attributed to two reasons. Firstly, it takes around 5 hours to get a transformation network well trained with a decent GPU and CPU. But this training can be done once and for all, as long as we have a transformation network we can share it with other researchers and use it for many more experiments. Secondly, creating the stylized images is still slower than traditional methods. For Caltech101, it took around five minutes to augment the whole dataset while the traditional methods can complete it in a few minutes. So it is still very important to find more efficient style transfer algorithms.

6.2 Future Work

As we have many limitations in this dissertation, one of the major future works is to spend more time to solve them. Besides, another future work we can do is to use image classification loss to optimize the transformation network. During our experiments, we did figure out how content weights impact the augmentation performance. But if we consider the image classification accuracy as a way to update the transformation network, we may have a way to identify the uncertain hyper parameters or even optimize the weights for the transformation network. For example, when training VGGNet for

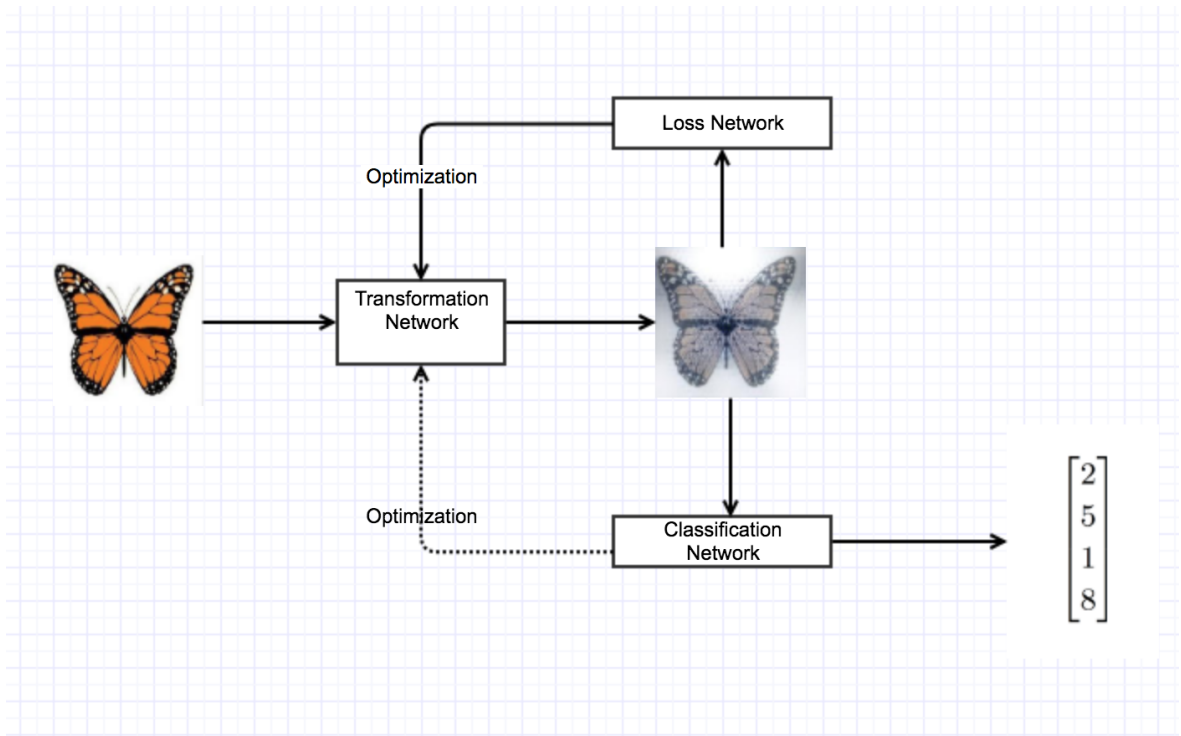


Figure 6.1: The process of using classification loss to optimize the transformation network

image classification, we only perform backpropagation with the training loss on the VGGNet itself. But we think it is possible that we use the same backpropagation to train the transformation network and thus to identify the best weights and even hyper parameters that suit the dataset as well as the specific task best. Figure 6.1 shows the process of how a transformation network learns augmentations that best improve the ability to classify images correctly.

Another future work is to explore how data augmentation impacts the training time of an image classification algorithm. As mentioned in Chapter 5, the table 5.1 shows that the training time is quite different for different approaches. For Node and Flipping, training only takes less than one hour, but for FlippingRotation it takes around 4 hours. If given more time, we can look into how the model converges during

the training process and understand the algorithm to figure out if we can reduce the training time.

Bibliography

- [1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in neural information processing systems*, pp. 1097–1105, 2012.
- [2] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.
- [3] L. A. Gatys, A. S. Ecker, and M. Bethge, “Image style transfer using convolutional neural networks,” in *Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on*, pp. 2414–2423, IEEE, 2016.
- [4] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, “ImageNet Large Scale Visual Recognition Challenge,” *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.
- [5] J. Johnson, A. Alahi, and L. Fei-Fei, “Perceptual losses for real-time style transfer and super-resolution,” in *European Conference on Computer Vision*, pp. 694–711, Springer, 2016.
- [6] L. Fei-Fei, R. Fergus, and P. Perona, “One-shot learning of object categories,”

- IEEE transactions on pattern analysis and machine intelligence*, vol. 28, no. 4, pp. 594–611, 2006.
- [7] G. Griffin, A. Holub, and P. Perona, “Caltech-256 object category dataset,” 2007.
- [8] J. Schmidhuber, “Deep learning in neural networks: An overview,” *Neural networks*, vol. 61, pp. 85–117, 2015.
- [9] J. E. Kyprianidis, J. Collomosse, T. Wang, and T. Isenberg, “State of the” art”: A taxonomy of artistic stylization techniques for images and video,” *IEEE transactions on visualization and computer graphics*, vol. 19, no. 5, pp. 866–885, 2013.
- [10] C. N. Vasconcelos and B. N. Vasconcelos, “Increasing deep learning melanoma classification by classical and expert knowledge based image transforms,” *CoRR*, *abs/1702.07025*, vol. 1, 2017.
- [11] V. Dumoulin, J. Shlens, and M. Kudlur, “A learned representation for artistic style,” *Proc. of ICLR*, 2017.
- [12] D. Erhan, Y. Bengio, A. Courville, and P. Vincent, “Visualizing higher-layer features of a deep network,” *University of Montreal*, vol. 1341, no. 3, p. 1, 2009.
- [13] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell, “A deep convolutional activation feature for generic visual recognition. arxiv preprint,” *arXiv preprint arXiv:1310.1531*, 2013.
- [14] M. D. Zeiler and R. Fergus, “Visualizing and understanding convolutional networks,” in *European conference on computer vision*, pp. 818–833, Springer, 2014.
- [15] “Visualizing what convnets learn.” URL <http://cs231n.github.io/understanding-cnn/>. Accessed: 2018-08-24.

- [16] A. Mahendran and A. Vedaldi, “Understanding deep image representations by inverting them,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5188–5196, 2015.
- [17] L.-Y. Wei, S. Lefebvre, V. Kwatra, and G. Turk, “State of the art in example-based texture synthesis,” in *Eurographics 2009, State of the Art Report, EG-STAR*, pp. 93–117, Eurographics Association, 2009.
- [18] J. Portilla and E. P. Simoncelli, “A parametric texture model based on joint statistics of complex wavelet coefficients,” *International journal of computer vision*, vol. 40, no. 1, pp. 49–70, 2000.
- [19] L. Gatys, A. S. Ecker, and M. Bethge, “Texture synthesis using convolutional neural networks,” in *Advances in Neural Information Processing Systems*, pp. 262–270, 2015.
- [20] Y. Jing, Y. Yang, Z. Feng, J. Ye, and M. Song, “Neural style transfer: A review,” *CoRR*, vol. abs/1705.04058, 2017.
- [21] L. A. Gatys, A. S. Ecker, and M. Bethge, “A neural algorithm of artistic style,” *arXiv preprint arXiv:1508.06576*, 2015.
- [22] V. Dumoulin, J. Shlens, and M. Kudlur, “A learned representation for artistic style,” *arXiv e-prints*, vol. abs/1610.07629, 2016.
- [23] P. Y. Simard, D. Steinkraus, and J. C. Platt, “Best practices for convolutional neural networks applied to visual document analysis,” in *null*, p. 958, IEEE, 2003.
- [24] A. Dertat, “Applied Deep Learning - Part 4: Convolutional Neural Networks.” URL <https://towardsdatascience.com/>

- applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2.
Accessed: 2018-08-17.
- [25] L. Perez and J. Wang, “The effectiveness of data augmentation in image classification using deep learning,” *arXiv preprint arXiv:1712.04621*, 2017.
- [26] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in *Advances in neural information processing systems*, pp. 2672–2680, 2014.
- [27] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, “Unpaired image-to-image translation using cycle-consistent adversarial networks,” *arXiv preprint*, 2017.
- [28] H. Inoue, “Data augmentation by pairing samples for images classification,” *arXiv preprint arXiv:1801.02929*, 2018.
- [29] “About TensorFlow.” URL <https://www.tensorflow.org/>. Accessed: 2018-08-24.
- [30] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, “Tensorflow: a system for large-scale machine learning,” in *OSDI*, vol. 16, pp. 265–283, 2016.
- [31] F. Chollet *et al.*, “Keras: Deep learning library for theano and tensorflow,” *URL: https://keras.io/k*, vol. 7, no. 8, 2015.
- [32] N. Ketkar, “Introduction to keras,” in *Deep Learning with Python*, pp. 97–111, Springer, 2017.

- [33] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.
- [34] L. Bottou, “Large-scale machine learning with stochastic gradient descent,” in *Proceedings of COMPSTAT’2010*, pp. 177–186, Springer, 2010.
- [35] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255, Ieee, 2009.
- [36] L. Fei-Fei, J. Johnson, and Y. Serena, “Cnn architectures.” URL http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture9.pdf. Accessed: 2018-08-19.
- [37] L. Engstrom, “Fast style transfer.” <https://github.com/lengstrom/fast-style-transfer/>, 2016.
- [38] G. Sam and W. Michael, “Training and investigating residual nets.” <http://torch.ch/blog/2016/02/04/resnets.html>, 2016.
- [39] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, “Microsoft COCO: common objects in context,” *CoRR*, vol. abs/1405.0312, 2014.
- [40] D. Kinga and J. B. Adam, “A method for stochastic optimization,” in *International Conference on Learning Representations (ICLR)*, vol. 5, 2015.

Appendix A

Image Styles



Figure A.1: The Great Wave off Kanagawa, Hokusai, 1829 - 1832



Figure A.2: A Snow picture from the Internet, simple but beautiful

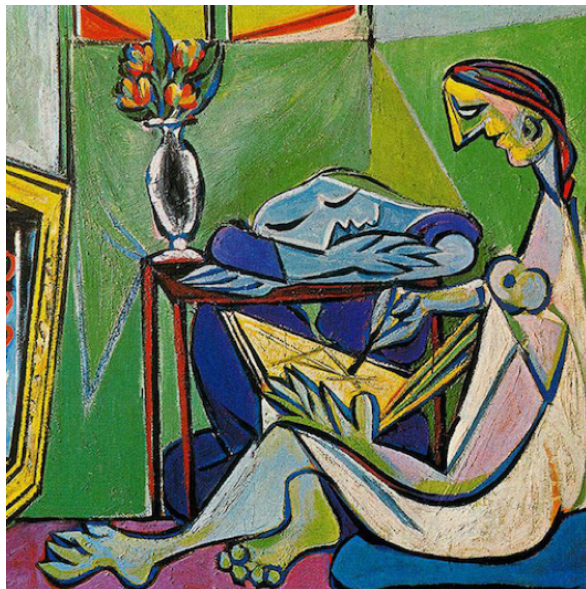


Figure A.3: The Muse, Pablo Picasso, 1935



Figure A.4: Rain Princess, a painting by Leonid Afremov



Figure A.5: Sunflower from one of Van Gogh's series of paintings



Figure A.6: A typical udnie painting



Figure A.7: A screenshot from the Japanese Animation: Your Name (2016)

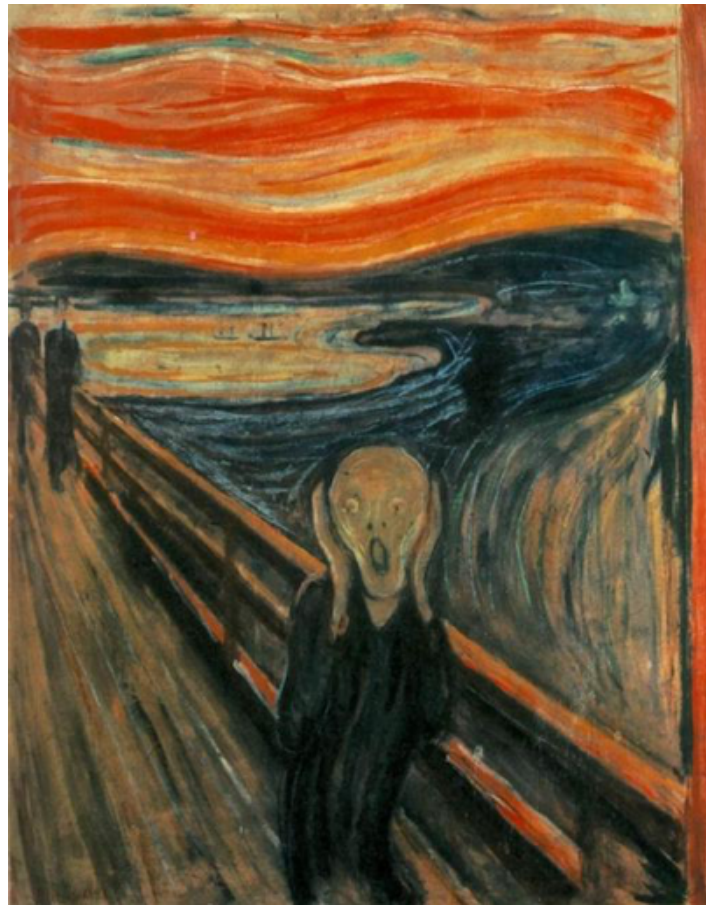


Figure A.8: The Scream, Edvard Munch, 1893 - 1910

Appendix B

GitHub Link

The entire implementation code can be found on GitHub: <https://github.com/zhengxu001/neural-data-augmentation>.