# Generating Embroidery Patterns using Image-to-Image Translation

Mohammad Akif Beg

A Thesis
in
The Department
of
Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Applied Science Software Engineering
Concordia University
Montréal, Québec, Canada

October 2019

# Concordia University
School of Graduate Studies

This is to certify that the thesis prepared

By: **Mohammad Akif Beg**

Entitled: **Generating Embroidery Patterns using Image-to-Image Translation**

and submitted in partial fulfillment of the requirements for the degree of

**Master of Applied Science Software Engineering**

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining commitee:

| | |
|---|---|
| Dr. Aiman Hanna | Chair |
| Dr. Adam Krzyzak | Examiner |
| Dr. Tien D. Bui | Examiner |
| Dr. Jia Yuan Yu | Supervisor |

Approved by: Lata Narayan
Chair of Department

September 17th 2019

Dr. Amir Asif
Dean
Faculty of Engineering and Computer Science

# Abstract

Generating Embroidery Patterns using Image-to-Image Translation

Mohammad Akif Beg

In many scenarios in computer vision, machine learning, and computer graphics, there is a requirement to learn the mapping from an image of one domain to an image of another domain, called *Image-to-image translation*. For example, *style transfer*, *object transfiguration*, visually altering the appearance of weather conditions in an image, changing the appearance of a day image into a night image or vice versa, photo enhancement, to name a few. In this paper, we propose two machine learning techniques to solve the embroidery image-to-image translation. Our goal is to generate a preview image which looks similar to an embroidered image, from a user-uploaded image. Our techniques are modifications of two existing techniques, *neural style transfer*, and *cycle-consistent generative-adversarial network*. Neural style transfer renders the semantic content of an image from one domain in the style of a different image in another domain, whereas a cycle-consistent generative adversarial network learns the mapping from an input image to output image without any paired training data, and also learn a loss function to train this mapping. Furthermore, the techniques we propose are independent of any embroidery attributes, such as elevation of the image, light-source, start, and endpoints of a stitch, type of stitch used, fabric type, etc. Given the user image, our techniques can generate a preview image which looks similar to an embroidered image. We train and test our propose techniques on an embroidery dataset which consist of simple 2D images. To do so, we prepare an unpaired embroidery dataset with more than 8000 user-uploaded images along with embroidered images. Empirical results show that these techniques successfully generate an approximate preview of an embroidered version of a user image, which can help users in decision making.

# Acknowledgments

Throughout the writing of this thesis, I have received a great deal of support and assistance. I would first like to thank my supervisor, Dr. Jia Yuan Yu, for introducing me to the field of machine learning and artificial intelligence. Also, his support and feedback during this research and in writing of this thesis. I want to acknowledge the people who have supported me in writing this thesis, whether it's providing my knowledge about writing or proofreading my work. I would especially like to thank Syed Eqbal and Shuai Ma for proofreading this work and helping me make this work as good as I can. I would also like to thank my parents and my sisters for their counsel and support. They have been the backbone of emotional, mental and financial support throughout my degree and nothing would have been possible without their support. Finally, my friends, who were of great support in discussing the problems and findings, as well as providing a happy atmosphere outside of my research.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Customizable fashion is on the rise in industry. Customization creates a deeper connection between the customer and the product. Embroidery customization is considered to be the most popular type of embroidery in comparison to the counterparts, screenprint and digital print. For this work we partnered with a firm that personalize apparels with custom embroidery. A customer uploads an image to be embroidered and selects clothing and other details like the positioning and size of the image. Having a real-time approximate preview of their design is a significant factor in the decision making of the client. Based on our industry research and knowledge, we did not find any work which simulates an approximate embroidered version of a user-uploaded image automatically to facilitate the customization experience. The image-to-image translation is one of the techniques that can help us in providing an approximate embroidered version of an image

The image-to-image translation is a process of translating one possible representation of an image to another. In other words, it means to transform an image from its original form to another form while keeping the original structure and semantics of the image (such as a change in style, design alteration, colorization, and others) intact. Before the introduction of this technique, to achieve a similar task a combination of different image processing techniques were required [1] like image quilting [2], image analogies [3], image denoising [4], depth prediction [5], semantic labeling [5], surface normal estimation [5], image colorization [6] and many others [7, 8, 9, 10].

In this work, we propose to solve this problem by modification of two existing

techniques, neural style transfer [11] and cycle-consistent generative adversarial networks (CycleGAN) [12]. Both of these techniques have shown remarkable results in solving different image-to-image translation problems. Gatys et al. [11] introduced neural style transfer to solve an image-to-image translation problem for art. They used a convolutional neural network (CNN) [13] to transfer the style of a painting on a photograph. Since then, many different researchers have proposed different modifications and advancements to the original algorithm, and all of them have achieved groundbreaking results. Our work is directly related to the line of work initiated by Gatys. Another technique that we propose to solve the embroidery image-to-image translation problem is using cycle-consistent generative adversarial networks. Goodfellow et al. [14] introduced generative adversarial networks (GAN). They proposed a framework which uses an adversarial process for estimating the generative model. Different architectures and modification of generative adversarial networks are used to solve image-to-image translation problems. Isola et al. [1] as a general-purpose solution to image-to-image translation problem proposed conditional adversarial networks. They have tested this framework on a paired set of images, and the results were quite remarkable. Zhu et al. [12] used an unpaired set of images and cycle-consistent adversarial network to solve the image-to-image translation problem. The use of an unpaired set of images has given this framework an advantage to be widely used in different applications. Though there are many different approaches to solving an image-to-image translation problem, our work is closely related to the work of Zhu et al. [12] because of the freedom of using unpaired images.

In the following chapters of this document, we discuss how we can use these techniques to solve our problem and to transform an image to its subsequent embroidered version. The document is divided into different chapters, namely prerequisites, image preprocessing techniques, neural style transfer, generative adversarial network, training details and results, future work and conclusions. In the second chapter (2), we have explained some essential machine learning and deep learning concepts which are useful in understanding the rest of the document. In the third chapter (3), we explained the neural style transfer. Neural style transfer is the first image-to-image translation technique that we have used to solve our embroidery translation problem. The chapter explains neural style transfer and the loss functions of neural style transfer. We have also explained the modification we propose to neural style transfer to

solve the embroidery translation problem. In the fourth chapter (4), we have introduced the generative adversarial network. GANs are the second technique we have used to solve our problem. We first explain a generative adversarial network. Then we explain its architecture and the objective function. We also explain the cycle-consistent adversarial network (CycleGAN) which is the baseline architecture we use for our problem and lastly, our modified CycleGAN for the embroidery problem, embroidery cycle-consistent adversarial network (EmbGAN). In the fifth chapter (5) we explain the dataset that we prepare for this project, the preprocessing techniques we use and we also provide the training details and the results of both the techniques. In the last chapter (6), we propose some future work and the conclusions of this research.

# Chapter 2

# Prerequisites

To have a proper understanding of the technologies used in the research to achieve our task, we need to have some basic knowledge about a few concepts of machine learning and deep learning. This chapter introduces these concepts in detail, along with examples. The content of the chapter is a compressed version of the resources like the deep learning book [15], Introduction to Statistical Learning book [16], and many other books like [17, 16, 18]

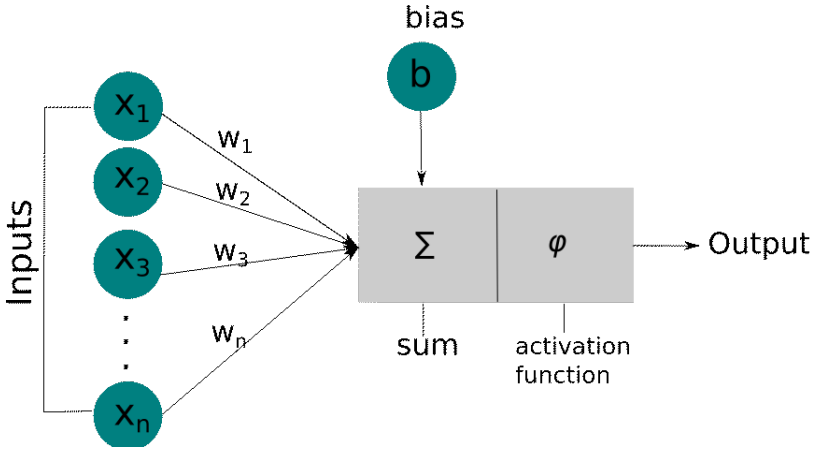



Figure 1: An Artificial Neuron with $n$ input values $x_1, \cdots, x_n$ and $n$ corresponding weights $w_1, \cdots, w_n$ and $b$ is the bias node. The $\sum$ is the summation symbol and $f$ is the activation function.

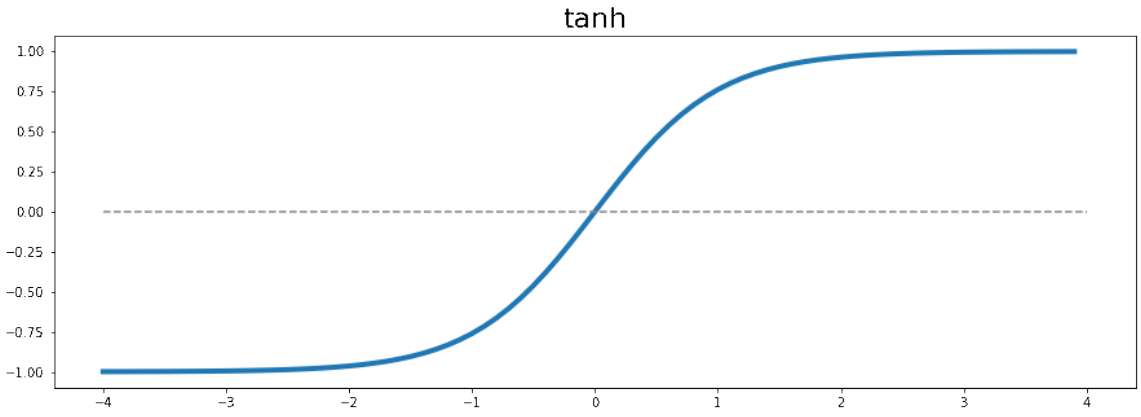
## 2.1 Neural Network

To understand the proper functioning of an artificial neural network (ANN), we first have to know the building block of an ANN, that is a neuron. Simon hayken [19] defined an artificial neuron as a processing unit that follows the paradigm of a neuron in a human brain and is interconnected by synapses . In other words, a neuron is a computational block that performs some sort of operation on the inputs and forwards the result to another neuron. In Figure (1), there are $\boldsymbol{n}$ inputs to the neuron, it performs some operation on these values and then passes the output value forward. When a neuron or a node receives an input value, it goes through two steps of computation. First, all the input values are multiplied to the weights assigned to the synapses and a bias is added, and then a summation over the number of input values is perfomed. Second, the summation is applied to an activation function, and the output is then propagate to a forward node. Let $i = \{1, \cdots, n\}$ be the number of nodes in a neuron, $x_i \in \mathbb{R}$ be an input of a neuron, $w_i \in \mathbb{R}$ is the weight of the edge, $b \in \mathbb{R}$ be a bias, $y \in \mathbb{R}$ be the output of a neuron in a layer, $\phi(x) : \mathbb{R} \to \mathbb{R}$ is an activation function, which is applied componentwise. $f(x) : \mathbb{R} \to \mathbb{R}$ be an affine function defined as $f(x) = w_i \cdot x_i + b$. The notation $\circ$ denotes the composition of functions. Mathematically, we can represent an output of a neuron, as a function $g : \mathbb{R}^0 \to \mathbb{R}^n$:

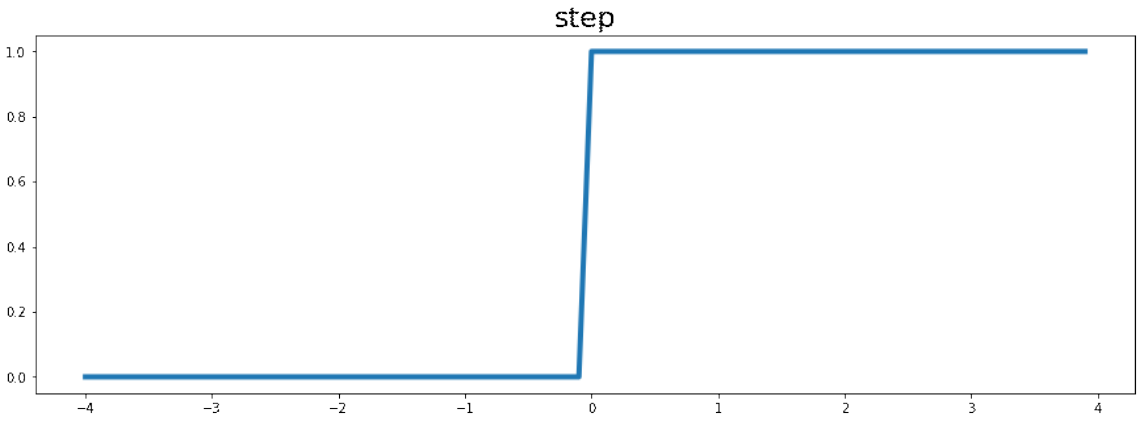
$$y = g = \phi \circ f(x) \qquad \forall i \in \{1, \cdots, n\}.$$

Table 1: Types of Activation Function

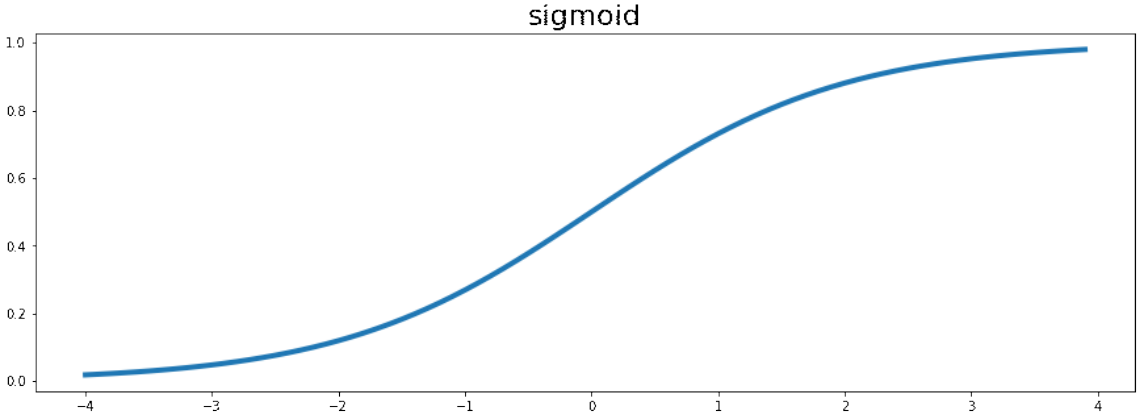
| | |
|---|---|
| Sigmoid Function | $\phi(x) = \frac{1}{1+exp^{-x}}$ |
| Step Function | $\phi(x) = \begin{cases} 1 \text{ if x} \geq 0, \\ 0 \text{ if x} < 0, \end{cases}$ |
| Rectifier Linear Unit (ReLU) | $\phi(x) = max(x, 0)$ |
| Hyperbolic Tangent Function | $\phi(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ |


tanh
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−4
−3
−2
−1
0
1
2
3
4

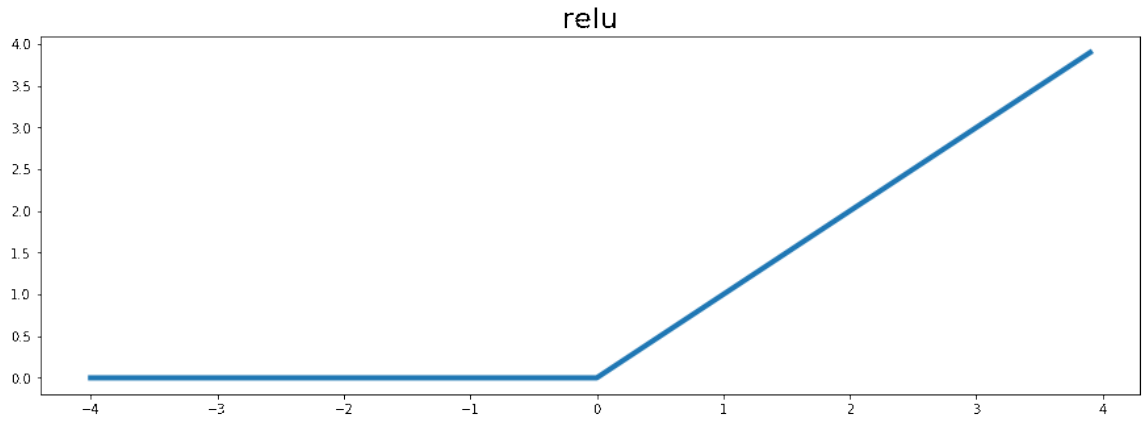
(1)


step
1.0
0.8
0.6
0.4
0.2
0.0
−4
−3
−2
−1
0
1
2
3
4

(2)


sigmoid
1.0
0.8
0.6
0.4
0.2
0.0
−4
−3
−2
−1
0
1
2
3
4

(3)




(4)

Figure 2: Different types of activation function : (1) is the hyperbolic tangent function, tanh. (2) is the step activation function. (3) is the sigmoid activation function, and (4) is the ReLU activation function

**Activation Function**

The activation function $\phi(x) = \mathbb{R} \to \mathbb{R}$, is a function used to delimit or bound the output of a neuron, and it is applied componentwise. In [19], the author Simon Hayken has referred to it as a squashing function as it squashed the amplitude range of the output signal to a finite value. It can be linear or non-linear depending upon the function it is representing [20]. Table (1) and Figure (2) shows the most widely used activation functions.

### 2.1.1 Artificial Neural Network

An artificial neural network is a weighted directed graph of neurons that have interconnecting synaptic and activation link [19], where each neuron represents a node in the graph. These nodes are clustered into different layers, namely one input layer, one or more hidden layer, and one output layer. The input value of every node can either be a binary, integer, or floating point number. In cases where the inputs are not numerical, they are usually encoded with a set of integer values. The practice of scaling the input and output values is to make computation easy for the neural network and to ensure stability.

We can represent a neural network of layers $L$, with only feed forward propagation as a composition of $L$ functions $g_i$. The set $L = \{0, \cdots, L\}$ denotes the number of

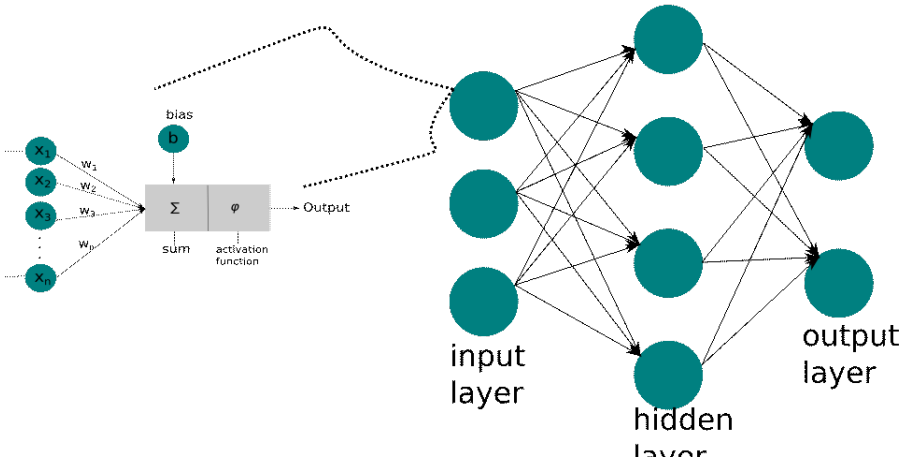



Figure 3: A simplified version of a feed forward neural network with one input, one hidden and one output layer. The dotted lines followed by a small figure of a neuron denotes that each node in a layer is a neuron.

layers in a neural network. The set $i^l = \{1^l, \cdots, n^l\}$ denotes the number of nodes in a layer $l$. Let $x_i \in \mathbb{R}$, $w_i \in \mathbb{R}$ and $b \in \mathbb{R}$ , the set $X_{i^l} = \{x_{1^l}, x_{2^l}, \cdots, x_{n^l}\}$ represents a set of input values in a layer $l$, $W_{i^l} = \{w_{1^l}, w_{2^l}, \cdots, w_{n^l}\}$ represents a set of corresponding weights in a layer $l$ and $B_{i^l} = \{b_{1^l}, b_{2^l}, \cdots, b_{n^l}\}$ be a set of bias node in a layer $l$. The activation function is represented as $\phi(x) : \mathbb{R} \to \mathbb{R}$ is applied componentwise and $f_{i^l}(x) : \mathbb{R}^{i^{l-1}} \to \mathbb{R}^{i^l}$ be an affine function, for any $l = \{1, \cdots, L\}$ and for any $i = \{1, 2, \cdots, n\}$ it is defined as $f_{i^l}(x) = w_{i^l} \cdot x_{i^l} + b_{i^l}$. For any $l = \{1, \cdots, L-1\}$, let $g^l(x) = \phi \circ f_{i^l}(x)$. We define a feed forward neural network, a function $g^l : \mathbb{R}^{i^0} \to \mathbb{R}^{i^L}$ as:

$$g = f^L \circ g^{L-1} \circ \cdots \circ g^1 \tag{1}$$

### 2.1.2 Learning Process

Learning process of a neural network are processes used to update the weights of a neural network. Almost every learning method in a neural network is dependent on gradient descent. Gradient descent is a process which iteratively aims to find the local minima of a loss function by adjusting its coefficients and hence lowering the prediction error of the model.

### Parameter Initialization

Initialization of parameters is a very important step in the learning process. We initialize weights and bias. We should randomly initialize all the parameters to non-zero because initializing them to zero would always result in an equal gradient and the output after each iteration would be the same, and the algorithm would not learn anything. All the parameters are randomly initialized with weights between 0 and 1.

### Forward Propagation

Forward propagation is the next phase of the learning process. In this phase, the network is introduced with our input training data, and the data is forwardly propagated to the entire network to make appropriate predictions. Each neuron receives an input from the previous layer and performs an affine transformation with the already initialized weights and bias and then applies an activation function to the output of the affine transformation, elementwise. After the propagation of data through all the neurons in all the layers, the final layer has a prediction value for the given set of inputs.

### Loss Function

A loss function, also known as an objective function is function which hlps us in evaluating whether or not an algorithm models our dataset. Let $\{x(1), ..., x(n)\}$ be the given input vectors and $\{y(1), ..., y(n)\}$ be the desired output vectors. To map each input vector to the corresponding output vector, we need to tune the hyperparameters that are weight and threshold or bias. The learning rate determines how quickly these parameters are determined. It is often considered to be the most important hyperparameter [15]. The higher the value of a loss function, the more our predictions are not mapping to the desired output vector. In the third chapter (3) and thhe fourth chapter (4), we explain our proposed solution for the embroidery image to image translation problem presented in the first chapter (1). Both the techniques have a loss function that we want to optimize for the algorithm to perform on our dataset. Two of the most popular loss functions used in a neural network are the sum of square error and cross entropy, and they are defined as follows:

### Sum of Square Error

The sum of square error is a loss function that is used to find the difference between the actual value and the predicted value. In some cases, we also used the mean value of the error function is known as a mean square error, MSE.

$$E = \sum_{n=1}^{N} E(n),$$

$$E(n) = \|y(n) - \hat{y}\|^2,$$

where, $y(n)$ is the desired output and $\hat{y}$ is the output from the network when the input is $x(n)$.

**Cross Entropy**

Cross entropy loss measures the performance of a model whose output is a probability between 0 and 1. These are generally classification models. If the predicted probability diverges from the actual label, cross entropy loss increases.

$$E(n) = -\Big(y(n) \odot \ln(\hat{y}) + (1 - y(n)) \odot \ln(1 - \hat{y})\Big),$$

$x \odot y$ is a Hadamard product which means element-wise product of matrices.

**Backpropagation**

Rumelhart et al. [21] and Werbos [22] introduced the breakthrough concept of backpropagation. It is a technique used to find a gradient of a loss function of a network. Backpropagation is sometimes confused with being a complete learning algorithm, but it is only a technique to calculate the gradient [15]. The algorithm calculates the gradient of the loss function with respect to the weights and bias in every layer by using the chain rule and following the steepest descent manner. We start the algorithm from the output of the neural network, that means by adjusting the weight and bias of the last layer and working backward to the hidden layers.

$$\delta_j^N = f(\sum_{m=1}^{n-1} w_{jm}^N y_m^{N-1} + b_j^N)(y(n) - \hat{y}),$$

where, $\delta_j^N$ is the error term for $j^{th}$ neuron in the last layer. We propogate backwards, to calculate $\delta_j^n$'s for the following layers:

$$\delta_j^n = f(\sum_{m=1}^{n-2} w_{jm}^{n-1} y_m^{n-2} + b_j^{n-1})(\sum_{m=1}^{n} \delta_m^n w_{jm}^n),$$

$$\delta_j^n = \frac{\partial E(n)}{\partial W},$$

where, $E(n)$ is the error and $W$ is the weight.

### 2.1.3 Convolution Neural Network

The Convolution Neural Network or CNN or ConvNet is a particular type of artificial neural network which uses a mathematical operation known as convolution in at least one of its layer. They are used for processing data which usually has a gridlike structure [15] for example, images. CNNs are also known as a space or shift invariant artificial neural network because they have a property of detecting translation invariance with the help of convolution and pooling layers. CNNs were first discovered by David Hubel and Torsten Weisel in an experiment where they monitored the cat visual cortex [23], but it was Yan LeCun et al. in 1998 who used backpropagation and gradient descent to train a convolution neural network and successfully manage to do document recognition. The first CNN was named LeNet [24]. Alex Krizhevsky et al. in 2012 has revolutionized convolution neural network in 2012 with the paper [25], and improve the image classification task, exponentially. Convolution neural network has similar architecture rules of an artificial neural network. It also has an input and an output layer and several hidden layers, but hidden layers in CNNs are very different. It consists of a convolution layer, a pooling layer, a normalization layer, an activation layer, and a fully connected layer.

#### Convolution

The mathematical definitions in this section are heavily inspired by the deep learning book [15]. Mathematically, convolution is an operation on two functions which shows how one function has affected the shape of another. The integral of the products of functions is done after one function is reversed and shifted. In the case of image processing, one function acts as an input image and the second function is a kernel.

$$(f * h)(t) = \int_{-\infty}^{\infty} f(x)h(t-x)dx, \tag{2}$$

where, the notation $*$, is the symbol of the convolution operation. We can express the equation (2) in discrete time.

$$(f * h)(t) = \sum_{x=-\infty}^{\infty} f(x)(t-h), \tag{3}$$

Since we are discussing convolution neural network and the convolution process is primarily used in the context of images, we need to express the convolution process (2) in 2D.

$$S(i,j) = (I * K)(i,j) = \sum_{m}\sum_{n} I(m,n)K(i-m, j-n) \tag{4}$$

Equation (4) is commutative, which means $(I * K) = (K * I)$ as proved in [15], where I is a two dimension input and K is the kernel.

### 2.1.4 Explanation of Convolution Neural Network

For proper understanding and simplicity, let us consider a specific example, where we are just considering a simple black and white image. Suppose we want to train a model to classify between an image of **X** and an image of **O**.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | 1 | -1 | -1 | -1 | -1 | -1 | 1 | -1 |
| -1 | -1 | 1 | -1 | -1 | -1 | 1 | -1 | -1 |
| -1 | -1 | -1 | 1 | -1 | 1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | 1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | 1 | -1 | 1 | -1 | -1 | -1 |
| -1 | -1 | 1 | -1 | -1 | -1 | 1 | -1 | -1 |
| -1 | 1 | -1 | -1 | -1 | -1 | -1 | 1 | -1 |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |

Representation of an image of **X** as a matrix in memory. The pixels with value 1 denotes that these pixels represents the image of X while the pixel value $-1$ denotes the transparent background of the image.

Figure (4) shows the convolution steps for a part of the image, similarly, the kernel will convolve through the entire image, the number of pixels we convolve at every step is known as **stride**. In this example, the stride is 1 but we can use a stride

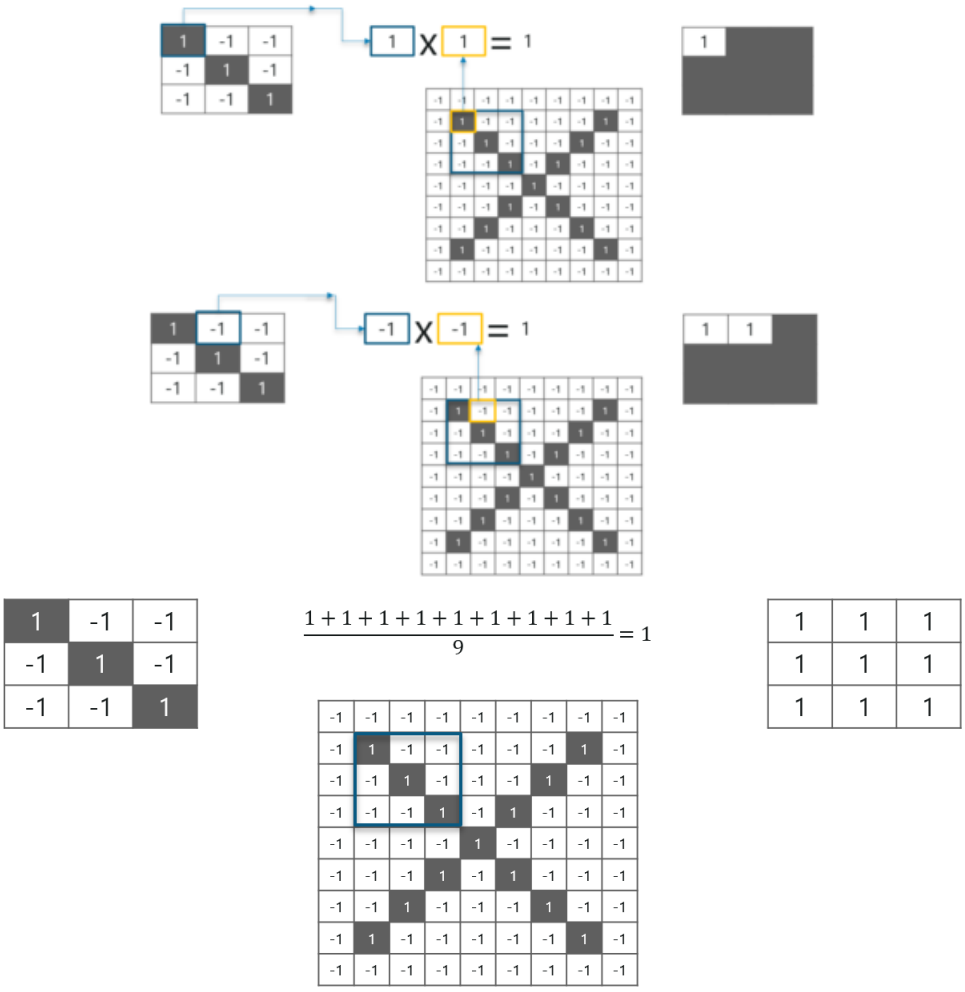



Figure 4: image demonstrates the convolution steps, align the image and the kernel, then multiply corresponding pixels from the image and the kernel. We then add the multiplication result and divide it with number of pixels in the filter.

of 2 or 3. The important thing to keep in mind is that the higher value of stride will downsample the image. The convolution operation results in a feature map, for the above example the feature map is shown in Figure (5).

### ReLU Layer

The feature map for the input is created through convolution layer, the output is then passed to a non-linear activation function, ReLU. Earlier, sigmoid and Tanh activation function were used, but later the researchers discovered that ReLU learns a lot faster in comparison to other functions. The papers [26, 27], are excellent sources

| | | | | | | |
|---|---|---|---|---|---|---|
| 0.77 | -0.11 | 0.11 | 0.33 | 0.55 | -0.11 | 0.33 |
| -0.11 | 1.0 | -0.11 | 0.33 | -0.11 | 0.11 | -0.11 |
| 0.11 | -0.11 | 1.0 | -0.33 | 0.11 | -0.11 | 0.55 |
| 0.33 | 0.33 | -0.33 | 0.55 | -0.33 | 0.33 | 0.33 |
| 0.55 | -0.11 | 0.11 | -0.33 | 1.00 | -0.11 | 0.11 |
| -0.11 | 0.11 | -0.11 | 0.33 | -0.11 | 1.00 | -0.11 |
| 0.33 | -0.11 | 0.55 | 0.33 | 0.11 | -0.11 | 0.77 |

Figure 5: Feature map for the image of an **X** after the convolution layer.

to understand the convolution neural network and use of ReLU as an activation function. Table (1) shows that a ReLU function is $\phi(x) = max(x, 0)$, which basically means it converts all the negative value to 0. After propagating through the ReLU layer, our feature map looks like (6).

| | | | | | | |
|---|---|---|---|---|---|---|
| 0.77 | -0.11 | 0.11 | 0.33 | 0.55 | -0.11 | 0.33 |
| -0.11 | 1.0 | -0.11 | 0.33 | -0.11 | 0.11 | -0.11 |
| 0.11 | -0.11 | 1.0 | -0.33 | 0.11 | -0.11 | 0.55 |
| 0.33 | 0.33 | -0.33 | 0.55 | -0.33 | 0.33 | 0.33 |
| 0.55 | -0.11 | 0.11 | -0.33 | 1.00 | -0.11 | 0.11 |
| -0.11 | 0.11 | -0.11 | 0.33 | -0.11 | 1.00 | -0.11 |
| 0.33 | -0.11 | 0.55 | 0.33 | 0.11 | -0.11 | 0.77 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 0.77 | 0 | 0.11 | 0.33 | 0.55 | 0 | 0.33 |
| 0 | 1.00 | 0 | 0.33 | 0 | 0.11 | 0 |
| 0.11 | 0 | 1.00 | 0 | 0.11 | 0 | 0.55 |
| 0.33 | 0.33 | 0 | 0.55 | 0 | 0.33 | 0.33 |
| 0.55 | 0 | 0.11 | 0 | 1.00 | 0 | 0.11 |
| 0 | 0.11 | 0 | 0.33 | 0 | 1.00 | 0 |
| 0.33 | 0 | 0.55 | 0.33 | 0.11 | 0 | 1.77 |

Figure 6: Feature map for the image of an **X** after the convolution layer and ReLU layer.

### Pooling Layer

The pooling layer in a CNN downsamples the feature map and focuses only on the important features. This layer makes use of spatial invariance. We know that the goal in hand is to determine whether or not a feature is present in an image, even if its positioning is different. The relative positioning of other features is more important than absolute positioning. There are different pooling techniques available, but the most popular are **maxpooling** or **average pooling**. In maxpooling, we take a filter and a stride of similar length. The filter then convolves around the input value, and

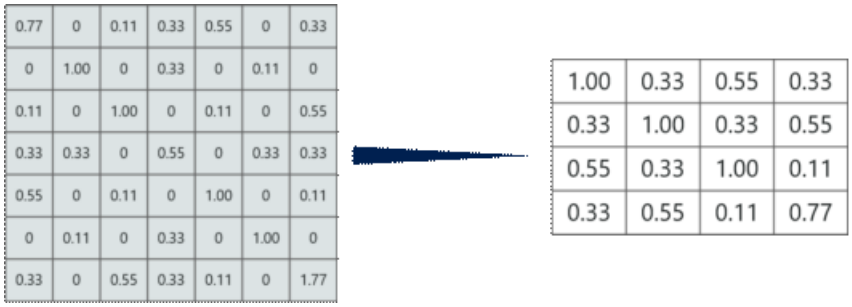



Figure 7: Feature map for the image of an **X** after the convolution layer, ReLU layer and pooling layer.

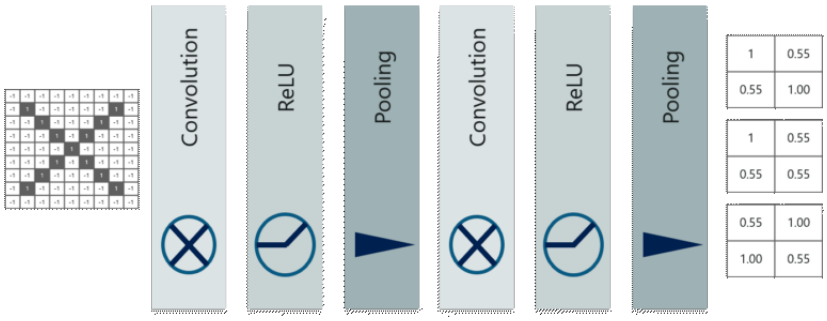
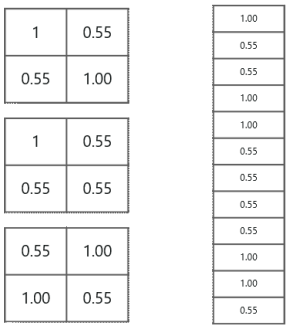



Figure 8: Feature map for the image of an **X** after multiple blocks of convolution, ReLU and pooling layer.

the output is maximum in that region. The convolution, ReLU, and pooling layers are often stacked up to downsample an image further and focus on only a few important features. The number of blocks of these layers present CNN is usually depend upon the size of the input sample.

### Fully Connected Layer

The final layers of a convolution neural network mimic an artificial neural network, and these are fully connected layers, which means every neuron in a preceding layer is connected to every neuron in a succeeding layer. The classification process happens in these fully connected layers but before transmitting the results from the pooling layer, we flatten the matrices into a vector.




Flattening of Feature Map for Fully Connected Layer.

### 2.1.5 Mathematical Formulation: Convolutional Neural Network

In the section, we mathematically formulate the architecture of a convolutional neural network. Let $L = \{0, \cdots, L\}$ denotes the number of layers in the convolutional neural network. Let $u \in Z^+$ be the height, $v \in Z^+$ be the width and $c \in Z^+$ be the color channel of an image. We represent the image as $x \in \mathbb{R}^{u \times v \times c}$. For all $i \in \{1, \cdots, n\}$, $W_i^l = \{w_1^l, \cdots, w_n^l\}$ represents a set of weight matrices in layer l and $B_i^l = \{b_1^l, \cdots, b_n^l\}$ represent a vector of biases. An activation function $\phi(x) : \mathbb{R} \to \mathbb{R}$. Let $k \times k \in \mathbb{R}$ be the size of square-kernel, we represent a weight kernel $w_i^l \in W^l$, such that $w_i^l \in \mathbb{R}^{k \times k \times c}$. At any given layer $l$, $d^l$ denotes the number of filters in the layer, $\hat{u}^l$ and $\hat{v}^l$ denotes the height and width of the feature map, $p^l$ denotes the size of the feature, which is the product of $\hat{u}^l$ and $\hat{v}^l$. We define the feature map as $H(x) \in \mathbb{R}^{d^l \times p^l}$. The convolution neural network has different type of layers namely, convolution layer, pooling layer, and fully connected layer. The first layer, i.e. $l = 0$ takes an image $x$ as the input, and given $n^0$ number of weight matrices $W_i^0 = \{w_1^0, \ldots, w_n^0\}$ and bias vectors $B_i^0 = (b_1^0, \ldots, b_n^0)$, it computes $n$ feature maps $H_i^0 = \{h_1^0, \ldots, h_n^0\}$ using a convolution and activation function. We can now define the different type of layers from $2 \leq l \leq L$. For simplicity, we will be using the same notation $f$ to define every layer. First, the convolutional layer uses a convolution function, $z^l(x, w_i^l) : \mathbb{R}^{k \times k} \to \mathbb{R}$ on the input image to produce a feature map, $h_i'$. We define the convolution function in a convolutional layer $l$ as :

$$z^l(x, w_i^l) = \sum_{a=0}^{k-1} \sum_{b=0}^{k-1} \sum_{c=1}^{C} [x]_{a,b,c} \cdot [w_i^l]_{a,b,c} + b_i^l \tag{5}$$

The activation function, $\phi$ is then applied to the output of the convolution function from the Equation (5), we then define the convolutional layer, $f^l(x, w_i^l) \in \mathbb{R}^{d \times d \times c}$ as :

$$f^l(x, w_i^l) = \phi \circ z(x, w_i^l) \tag{6}$$

We can now define the pooling layer which takes the output from the activation layer as input and then outputs the maximum value of the input at any given position, that is within the kernel. For all $1 \leq a \leq k$ and $1 \leq b \leq k$, we defined the output of pooling layer as:

$$f^l(x, w_i^l) = \max(\phi \circ z(x, w_i^l))_{q,r} \tag{7}$$

The convolutional neural network is a stacking up of all these layers together and repeating these blocks again as shown in Figure (8). After multiple blocks of convolution, activation, and pooling layer, the final layer of a convolutional neural network is a fully connected layer similar to that of an artificial neural network defined in Section (2.1.1). We define the convolutional neural network $g(x, w_i^l) : \mathbb{R}^0 \to \mathbb{R}^L$ as :

$$g(x, w_i^l) = f^L(f^{L-1} \circ f^{L-2} \circ \cdots \circ f^1 \circ x) \tag{8}$$

where $f^L$ is a fully connected layer, $f^1, \cdots, f^{L-1}$ are either convolutional layer, activation layer, or pooling layer.

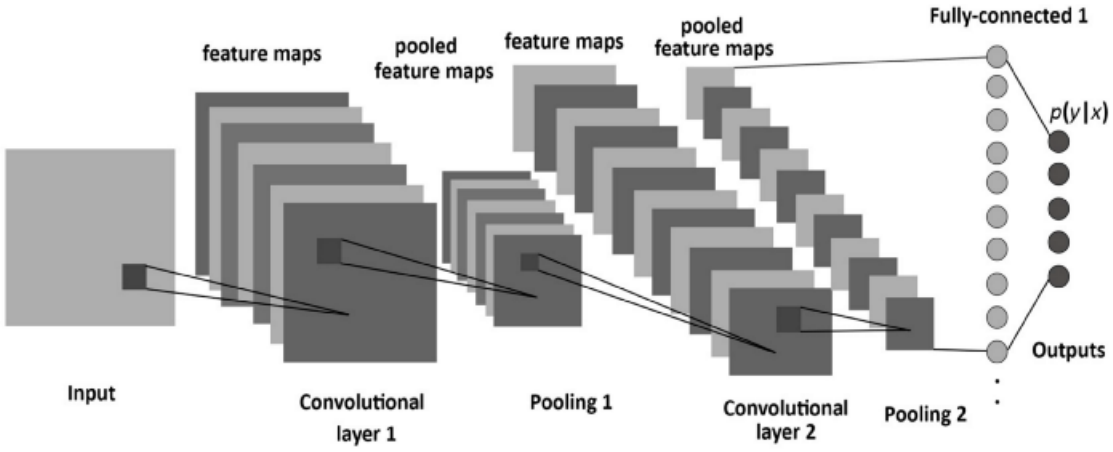



Figure 9: A simplified convolution neural network architecture with two blocks of convolutional and pooling layer and a fully connected layer at the end.

# Chapter 3

# Neural Style Transfer

Neural style transfer is a technique proposed by Gatys et al. [11]which apply textures synthesis from one image onto another image without changing the high-level content of the original image. Neural style transfer generates a stylized image that contains the texture style of one image and the semantic content of another image. In [3], the author presented a technique, image analogies which is one of the earliest solution of image-to-image translation. The neural style transfer has provided a wide variety of applications in modern time, changing the appearance of an image from day to night or from summer to winter [28, 7, 8], texture transfer of a painting [11, 3, 29, 30], modifying human face attributes [31], changing clothing on a photo of a model [32], synthesizing streets and building from labels and converting edges to photo [1]. The basic approach of neural style transfer is to jointly minimize the distance of the style representation and the content representation learned in different layers of the convolution neural network. In [33], the author proposed the use of feature maps to represent the content and Gram matrices to represent the style of another image. The author used the VGG-16 network to reconstruct the stylized image. The layers of the convolutional neural network (*also known as ConvNet*) are learning distinct filtered features of an image at different layers, and the complexity of learning increases with every layer. Every layer of ConvNet outputs a feature map unit of the input image. The feature map [11] at the lower layer of the ConvNet might consist of absolute pixel values or small features, whereas the higher layers will have more complex content. They used gradient descent and back propagation to minimize the losses. Neural style transfer is considered to be successful if the stylized image has the objects of the

content image and the visual look of the style image. The author of [11] also found that replacing the max-pooling with average pooling in CNN yields better results.

## 3.1 Problem Formulation

The feature map of a convolutional neural network are generally a very good representation of the features of an image. They capture spatial information of an image without containing the style information. Flattened feature vectors from a convolutional feature map representing features of the input space, and their dot product give us the information about the texture of a given image. We have introduced the feature map in the section (2.1.5) of (2). Let $u \in Z^+$ be the height, $v \in Z^+$ be the width and $c \in Z^+$ be the color channel of an image. Let $x$ and $y$ represent two images. $x \in \mathbb{N}^{u \times v \times c}$ represents the content image we want to synthesize, and $y \in \mathbb{N}^{u \times v \times c}$ represents the image whose style we want to transfer on image $x$. Let $\hat{y} \in \mathbb{N}^{u \times v \times c}$ be the styled image which represents the style of image $y$ and the principal content of image $x$. Let $d^l$ is the number of filters in the $l^{th}$ layer of the ConvNet. $\hat{u}^l$ and $\hat{v}^l$ be the height and weight of the feature map at layer $l$, $p^l$ represents the size of the feature map at layer $l$. $p^l$ is the product of $\hat{u}^l \times \hat{v}^l$. Let $z$ be any given image, we define the feature map as $H^l_{ij}(z) \in \mathbb{R}^{d^l \times p^l}$. The feature map stores the result of every layer of the ConvNet as a two-dimensional matrix. $H^l_{ij}$ is the activation of the $i^{th}$ filter at position $j$ in layer $l$. These matrices at different layers of the ConvNet are the filter responses, which helps in representing the image.

### 3.1.1 Content Reconstruction

The content reconstruction step is visualizing the image information encoded at different layers of hierarchy. The author performs gradient descent on a white noise image to find another image that matches the feature response of the content image. The author proposed that the feature responses of the deeper layers of VGG-16 convolutional neural network are used to represent the content of an image. For the styled image $\hat{y}$ to have the content of the original image $y$, we need to minimize the mean squared difference of the feature map $H^l_{ij}(z)$ of both the original and the styled image. This signifies that the loss is minimal. Given any two images $x, \hat{y}$ the author

of [11] define the content loss $L_c^l(x, \hat{y})$ at layer $l$ as:

$$L_c^l(x, \hat{y}) = \frac{1}{d^l p^l} \sum_{i=1}^{d^l} \sum_{j=1}^{p^l} (H(\hat{y})_{ij}^l - H(x)_{ij}^l)^2. \tag{9}$$

### 3.1.2 Style Reconstruction

Style reconstruction is obtained by calculating the correlations between filter responses across different layers of the convolutional neural network. Similar to content reconstruction, style reconstruction is performed using gradient descent. The author uses gradient descent from a white noise image to find another image that matches the texture of the style image. Our main goal is to extract the embroidery design from the style image $Y$ and transfer it to our output image $\hat{y}$. Every difference in texture, luminance, stitch pattern, etc. increases the style loss. To formulate the style loss function, we need to define the gram matrix. Let $z$ be an image, we define the gram matrix in 10, $G_{ij}^l(z) \in R^{d^l \times d^l}$ which is the dot product of the feature map, defined in section (3.1). The mean square difference between the gram matrices of the output image and the style image is calculated and needs to be minimized for having a successful style transferred image.

$$G(y)_{ij}^l = \sum_{k=1}^{p^l} H_{ik}^l \cdot H_{jk}^l. \tag{10}$$

Given two images $y$ and $\hat{y}$, the author defined the style loss $L_s^l(y, \hat{y})$ at *layer* l as:

$$L_s^l(y, \hat{y}) = \frac{1}{(2d^l)^2} \sum_{i=1}^{d^l} \sum_{j=1}^{p^l} (G(\hat{y})_{ij}^l - G(y)_{ij}^l)^2. \tag{11}$$

We can also express the style loss function in respect to (10)

$$\begin{aligned} &L_s^l(y, \hat{y}) = \\ &\frac{1}{(2d^l)^2} \sum_{i=1}^{d^l} \sum_{j=1}^{p^l} (\frac{1}{p^l}(H(\hat{y})_i^l \cdot H(\hat{y})_j^l) - \frac{1}{p^l}(H(y)_i^l \cdot H(y)_j^l))^2. \end{aligned} \tag{12}$$

#### Total Loss Function

The total loss function (TCF) for embroidery style transfer is the addition of content loss $L_c^l$ and the style loss $L_s^l$. $\alpha$ and $\beta$ are the weights of content and style loss,

respectively. By controlling $\alpha$ and $\beta$ we can control the amount of content and style present in the styled image.

$$L_t^l(x, y, \hat{y}) = \alpha L_c^l(x, \hat{y}) + \beta L_s^l(y, \hat{y}). \tag{13}$$

**TCF in Matrix Form**

From the Equations (9) and (11), we can express the total loss function in the form of feature matrices :

$$L_t^l(x, y, \hat{y}) = \alpha(\frac{1}{d^l p^l} \sum_{i=1}^{d^l} \sum_{j=1}^{p^l} (H(\hat{y})_{ij}^l - H(x)_{ij}^l)^2) +$$
$$\beta(\frac{1}{(2d^l)^2} \sum_{i=1}^{d^l} \sum_{j=1}^{p^l} (\frac{1}{p^l}(H(\hat{y})_i^l \cdot H(\hat{y})_j^l) - \frac{1}{p^l}(H(y)_i^l . H(y)_j^l))^2).$$

To achieve our task of generating an embroidered version of an image, we have used different stitch patterns as the style image and different logos or images as the content image. We propose a modification to this framework by segmenting the images into sub-images of different colors and then applying the style transfer technique. This has increased the quality of the output image in comparison to neural style transfer.

## 3.2 Split Style Transfer

We observe that using neural style transfer for translating the embroidery style on a given image has some limitations. For instance, Figure 10 shows the result of embroidery translation using neural style transfer in which the output image has some qualitative problems. We know that the output image in a neural style transfer is a blend of the style image and the content image, but the output image has no definite proportion of the blend but in this example the proportion of style image is more prominent than the content image. Though there are a few methods available for hyperparameter tuning to control the proportion of style and content in the output image, it still is not a probable solution for translating embroidery. The other drawback of neural style transfer is that we can only use one style image in the process, but in reality, an embroidered image usually have multiple distinct stitch patterns in one single image. Figure (11) shows the split style transfer method we propose to translate embroidery on a given image.

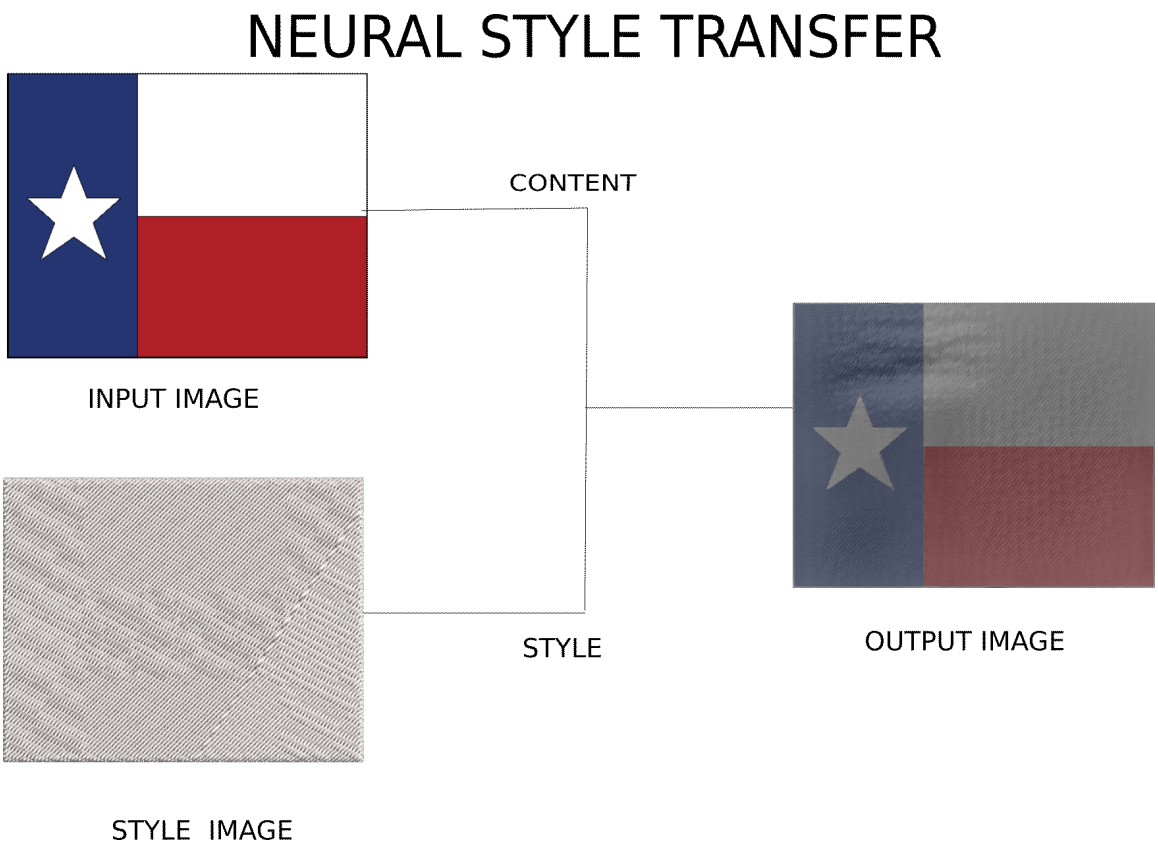



Figure 10: Neural Style Transfer: This is the basic example of a neural style transfer where we translate the style of one image on a given input image.

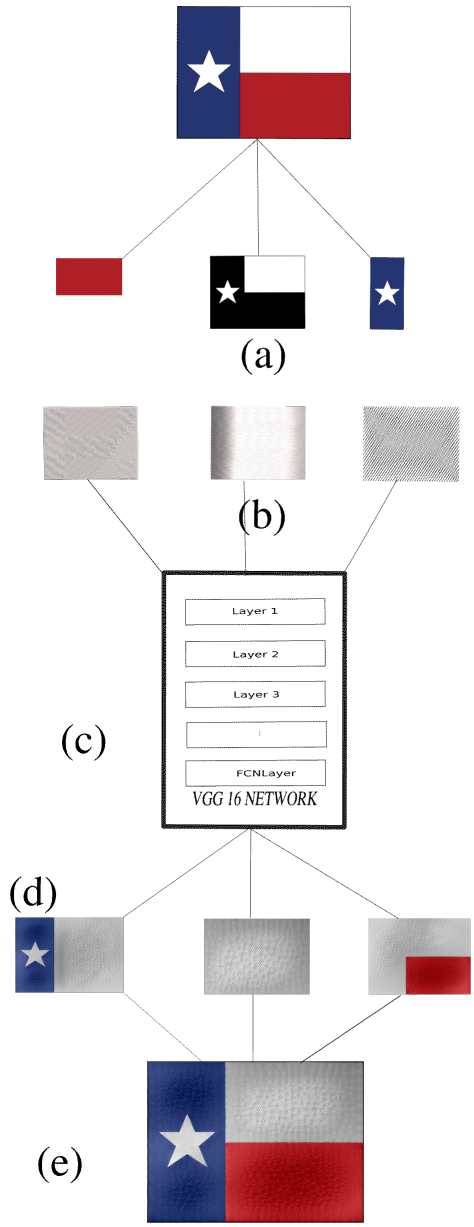



Figure 11: Split Style Transfer: this figure demonstrate the technique we propose to create an embroidered version of an image. The algorithm is separated into 5 steps. (a) split the image into different sub-images of different colors. (b) select different types of stitch for each sub-image. (c) perform style transfer on sub-images using different stitches. (d) styled sub-images using different stitches. (e) combine the styled sub-images. Note: The sub-image in (b) is made with a black background on purpose for understanding.

### 3.2.1 Problem Formulation

Let $u \in Z^+$ be the height, $v \in Z^+$ be the width and $c \in Z^+$ be the color channel of each image. For $r = \{1, \cdots, e\}$, let $x$ be the content image, such that $x \in \mathbb{N}^{u \times v \times c}$ and $Y = \{y_1, \cdots, y_e\}$ represents the set of style image (*embroidery images in our case*), such that $y_i \in \mathbb{N}^{u \times v \times c}$ similar to neural style transfer. In split style transfer, we first split our content image $x$, into a set of images of distinct color $X' = \{x'_1, \cdots, x'_e\}$, where $e$ is the total number of distinct colors in an image. Then, every sub image $x'_r$ is styled using a different style image, $y_r \in Y$. The final styled images, $\hat{Y}' = \{\hat{y}_1, \cdots \hat{y}_e\}$ are then combined to one image $\hat{y}$ at the end after style transfer is performed separately. Similar to Equations (9), (11), and (13) of neural style transfer we define the content $L_c^l$, style $L_s^l$, and total loss $L_t^l$ of split style transfer for a layer $l$ as :

**Content Loss**

$$L_c^l(x'_r, \hat{y}'_r) = \sum_{r=1}^{e} \left( \frac{1}{d^l p^l} \sum_{i=1}^{d^l} \sum_{j=1}^{p^l} (H(\hat{y}'_r)_{ij}^l - H(x'_r)_{ij}^l)^2 \right) \tag{14}$$

**Style Loss**

$$L_s^l(y_r, \hat{y}'_r) = \sum_{r=1}^{e} \left( \frac{1}{d^l p^l} \sum_{i=1}^{d^l} \sum_{j=1}^{p^l} (G(\hat{y}'_r)_{ij}^l - G(y_d)_{ij}^l)^2 \right) \tag{15}$$

**Total Loss**

$$L_t^l(x'_r, y_r, \hat{y}'_r) = \alpha L_c^l(x'_r, \hat{y}'_r) + \beta L_s^l(y_r, \hat{y}'_r) \tag{16}$$

The main benefit of using a split style transfer is to provide an option of using different types of stitches for a single image, which is an ideal scenario in embroidery customization. Though, in [34], the author has introduced a multi style transfer (MST) method. Theoretically, multi style transfer can be used to translate more than one style to an output image, but there is one limitation to this. There is no spatial control over which area of the given image will receive which style. Splitting the image into different layers of distinct colors will ensure that there is no overlap among the different stitch styles. The split style transfer produces some promising results, but it still has some limitations. Similar to neural style transfer [11], we can only translate a single image in one iteration, and this could be a costly process with respect to time. To create an embroidered version of multiple images at a time, we decided to use a generative algorithm. Since we were able to prepare an unpaired dataset for embroidery, we chose cyclegan and modified its architecture for

**Algorithm 1:** Split Style Transfer

Pre-processing: content images $x$, set of style images $Y = \{y_1, y_2, \ldots, y_e\}$, $u \times v \times c \leftarrow 256 \times 256 \times 3$, $e$ number of different style of embroidery images and the number of distinct colors in the input image, split images $X' = \{x'_1, \cdots, x'_e\}$.
$Z = \{z_1, z_2, \ldots, z_e\}$ is a random noise image similar to the size of $y_r$.
Input : $(x'_1, y_1), (x'_2, y_2), \ldots, (x'_e, y_e)$, for all $r \in \{1, 2, \ldots, e\}$
Output: styled images $\hat{y}'_r = \{\hat{y}_1, \cdots \hat{y}_e\}$, later combined to one image,
Initialization: Learning rate $\lambda \leftarrow 1e1$, content weight $\alpha \leftarrow 5e0$, style weight $\beta \leftarrow 1e2$, number of iterations $T \leftarrow 1000$

**for** $i \leftarrow 1$ *to* $e$ **do**
  **for** $t \leftarrow 1$ *to* $T$ **do**
    $L_c^l(x'_r, \hat{y}'_r) \leftarrow \sum_{r=1}^{e} \left( \frac{1}{d^l p^l} \sum_{i=1}^{d^l} \sum_{j=1}^{p^l} (H(\hat{y}'_r)_{ij}^l - H(x'_r)_{ij}^l)^2 \right)$
    $L_s^l(y_r, \hat{y}'_r) \leftarrow \sum_{r=1}^{e} \left( \frac{1}{d^l p^l} \sum_{i=1}^{d^l} \sum_{j=1}^{p^l} (G(\hat{y}'_r)_{ij}^l - G(y_r)_{ij}^l)^2 \right)$
    $L_t^l(x'_r, y_r, \hat{y}'_r) \leftarrow \alpha L_c^l(x'_r, \hat{y}'_r) + \beta L_s^l(y_r, \hat{y}'_r)$
  **end**
  $minimize(L_t^l(x'_r, y_r, \hat{y}'_r))$
**end**

embroidery generation. In the next section, we introduce our modified CycleGAN which can be used to translate multi-images in a much faster way in comparison to split style transfer.

# Chapter 4

# Generative Adversarial Networks

In this chapter, we first introduce generative adversarial networks (GANs) and its architecture and the loss functions, we then introduce cycle-consistent generative adversarial network (CycleGAN) which is the baseline architecture for the model we propose to solve embroidery image-to-image translation. We introduce its architecture and loss functions. We then introduce our propose model, the modification we propose in the architecture and loss function of CycleGAN. GANs are a subset of algorithms in machine learning known as generative models, which belongs to the domain of unsupervised learning. The critical feature of algorithms belonging to this domain is to learn the underlying structure of a given dataset, without specifying a target value. Goodfellow et al. in [14], extensively formulate and presented a framework of generative models which is constrained to work through an adversarial process. In an adversarial technique or methodology, there are two agents, where each has a target of outrun the other agent, and during this to and fro process of adversary, we reach an equilibrium where both the agents are at their most exceptional possible state. In GANs, the two agents are two neural network models, typically convolution neural networks where one is called a discriminator and the other is called a generator.

The Generative Adversarial Networks consists of two models. The first model is a generative network model, which is typically a convolution neural network with some deconvolution layers, takes random noise as an input vector and generates an output image. During training, the network learns how to improve the image so that it looks more like the original image, sometimes focusing only on part of images to improve for multiple iterations or epochs and hence making the work of discriminator difficult

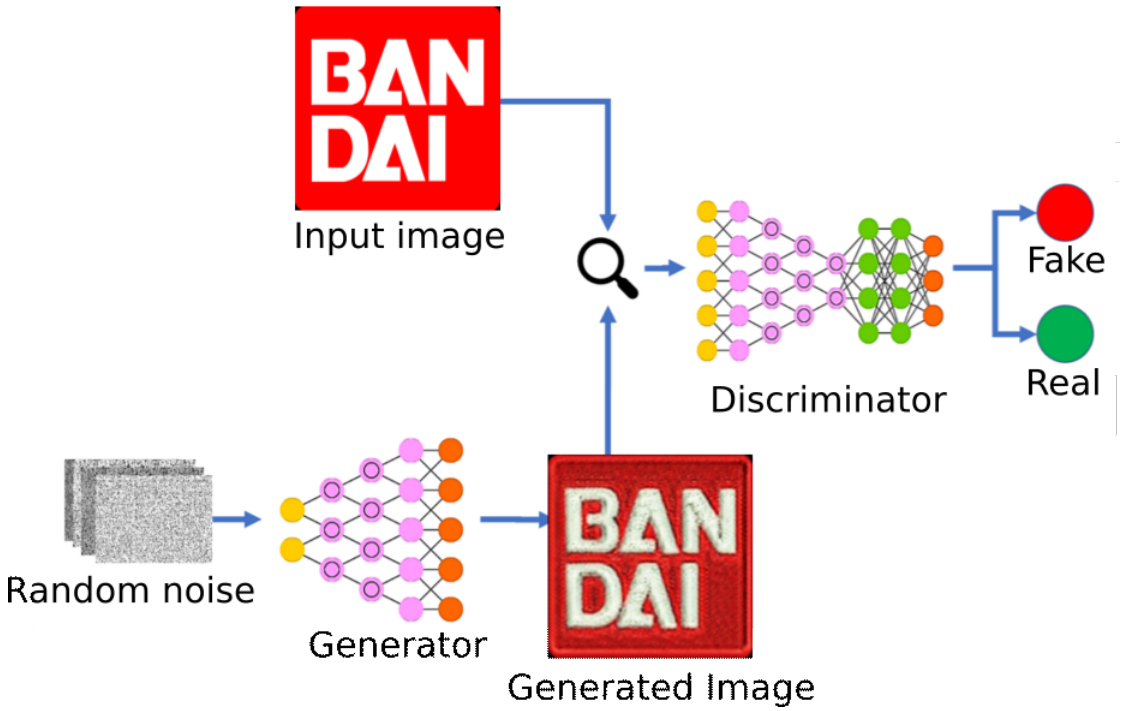



Figure 12: This is a simplified architecture of a generative adversarial network. The generator is a deconvolutional network which generates image from a random noise. The discriminator is a convolutional neural network which takes the input image and the generated image and classifies them as fake or real.

and hence comes the adversary. The second model is a discriminator network model, which is also a convolution neural network in most cases, and its task is to classify between the generated image and the input sample. As you can see in the above image, the input to the discriminator comes from the original input samples and the generated images from the generator model. This cycle goes on where the inputs are mixed from both the samples and the generator had to identify between the fake and the real image.

## Mathematical Formulation

Let the dataset consists of input images from one domain and the output images from another domain, both of the same dimension. Let $u \in Z^+$ be the height, $v \in Z^+$ be the width and $c \in Z^+$ be the color channel of each image. Suppose n be the total

number of images in the dataset and $N = \{1, 2, \cdots, n\}$. Let $X = \{x_1, x_2, \cdots, x_n\}$ be the set of user images, where $x_i \in \mathbb{N}^{u \times v \times c}$ for $i \in N$. Let $Y = \{y_1, y_2, \cdots, y_n\}$ be the set of embroidered images, where $y_i \in \mathbb{N}^{u \times v \times c}$ for $i \in N$. Let $Z = \{z_1, z_2, \cdots, z_n\}$ be a set of independent and normally distributed noise vectors with 0 mean 1 variance. Suppose that $G$ be a generator and $D$ be a discriminator, they consist of neural networks with parameters $\theta_G$ and $\theta_D$, respectively, $\theta_G$ and $\theta_D$ are the set of weights and biases of neural networks. Let $Q \times Q$ be the dimension of the noise vector $z_i \in Z$. We define differentiable function $f_G : \mathbb{N}^{u \times v \times c} \times \mathbb{R}^{Q \times Q} \to \mathbb{N}^{u \times v \times c}$. We denote the set of images generated by the generator as $\hat{Y} = \{\hat{y_1}, \hat{y_2}, \cdots, \hat{y_n}\}$, where $\hat{y}_k \in \mathbb{N}^{u \times v \times c}$ for $k \in N$. Let $f_D : \mathbb{N}^{u \times v \times c} \times \mathbb{N}^{u \times v \times c} \to [0, 1]$ represents discriminator functions. GANs are structured probabilistic models with latent variable $z_i$ and observed variable $x_i$. In mathematical notation, a generator $f_G(\theta_G)$ and a discriminator $f_D(\theta_D)$ are playing a 2 player min-max game. The generator is a differentiable function $f_G(\theta_G, z_i)$. Its role is mapping input noise variables $z_i$ to the desired data space $x_i$ (say images). Conversely, a discriminator is a discriminative function $f_D(\theta_D, x_i)$ outputs the probability that the data came from the real dataset, in the range (0,1). For $i = \{G, D\}$, $\theta_i$, represents the weights or parameters that define each neural network. As a result, the discriminator is trained to classify the input data as either real or fake correctly. This means its weights are updated as to maximize the probability that any real data input $x_i$ is classified as belonging to the real dataset while minimizing the probability that any fake image is classified as belonging to the real dataset. In more technical terms, the loss function used maximizes the function $f_D(\theta_D, x_i)$, and it also minimizes $f_D(f_G(\theta_G, z_i))$. Furthermore, the generator is trained to fool the discriminator by generating data as realistic as possible, which means that the weights of generator are optimized to maximize the probability that any fake image is classified as belonging to the real dataset. In practice, the logarithm of probability (e.g. $\log f_D(\cdot)$) is used in the loss functions instead of the raw probabilities, since using a log loss heavily penalizes classifiers that are confident about an incorrect classification. After several steps of training, if the generator and discriminator have enough capacity (if the networks can approximate the objective functions), they will reach a point at which both cannot improve anymore. At this point, the generator generates realistic synthetic data, and the discriminator is unable to differentiate between the two types of input. Since during training, both the discriminator and generator are trying to

optimize different loss functions, they can be thought of two agents playing a minimax game with value function $V(f_G, f_D)$. In this minimax game, the generator is trying to maximize its probability of having its outputs recognized as real, while the discriminator is trying to minimize this same value[14]. Let the probability distribution over real data(images) is $x \sim p(x)$, and probability distribution over generated data by generator, $z \sim p(z)$. We define the objective function of a generative adversarial network as :

$$\begin{aligned} &\underset{\theta_G}{min}\underset{\theta_D}{max}V(f_D, f_G) = \\ &\qquad \underset{\theta_G}{min}\underset{\theta_D}{max}\mathbb{E}_{x\sim p(x)}[\log f_D(\theta_D, x)] + \mathbb{E}_{z\sim p(z)}[\log(1 - f_D(\theta_D, f_G(\theta_G, z)))]. \end{aligned} \tag{17}$$

## 4.1 Cycle-Consistent Adversarial Network

The second method we propose is inspired by cycle-consistent adversarial network (CycleGAN) [12]. Zhu et al. introduced CycleGAN, which is an extension of generative adversarial network (GAN) [14]. CycleGAN has achieved substantial success in the image-to-image translation problem domain because it uses an unpaired image dataset. However, Isola et al. [1] have already introduced a conditional adversarial network which provides promising results, but it requires a paired image dataset and creating paired image dataset for a new task like ours is a tedious process. CycleGAN uses a pair of generators to achieve the translation problem of images from input domain X to an output domain Y and also it can translate the images from output domain Y to input domain X. The main concept of CycleGAN is to provide translation from the original domain to the target domain and vice versa.

### 4.1.1 Mathematical Formulation

Similar to the generative adversarial network, let the dataset consists of input images from one domain and the output images from another domain, both of the same dimension. Let $u \in Z^+$ be the height, $v \in Z^+$ be the width and $c \in Z^+$ be the color channel of each image. Suppose n be the total number of images in the dataset and $N = \{1, 2, \cdots, n\}$. Let $X = \{x_1, x_2, \cdots, x_n\}$ be the set of user images, where $x_i \in \mathbb{N}^{u\times v\times c}$ for $i \in N$. Let $Y = \{y_1, y_2, \cdots, y_n\}$ be the set of embroidered images, where

$y_i \in \mathbb{N}^{u \times v \times c}$ for $i \in N$. For $i \in \{1, 2\}$, let $G_i$ be a generator and $D_i$ be a discriminator, they consist of neural networks with parameters $\theta_{G_i}$ and $\theta_{D_i}$, respectively, $\theta_{G_i}$ and $\theta_{D_i}$ are the set of weights and biases of neural networks. Let $Q \times Q$ be the dimension of the noise vector $z_i \in Z$. For $i \in \{1, 2\}$, we define differentiable function $f_{G_i} : \mathbb{N}^{u \times v \times c} \times \mathbb{R}^{Q \times Q} \to \mathbb{N}^{u \times v \times c}$. We denote the set of images generated by the generator as $\hat{Y} = \{\hat{y_1}, \hat{y_2}, \cdots, \hat{y_n}\}$, where $\hat{y}_k \in \mathbb{N}^{u \times v \times c}$ for $k \in N$. Similarly, $f_{D_i} : \mathbb{N}^{u \times v \times c} \times \mathbb{N}^{u \times v \times c} \to [0, 1]$ represents discriminator functions. CycleGAN uses two generators, let $f_{G_1}(\theta_{G_1})$ and $f_{G_2}(\theta_{G_2})$ be the two generators and $f_{D_1}(\theta_{D_1})$ and $f_{D_2}(\theta_{D_2})$ be the two discriminators. The generators do the inter-domain translation from original domain to target and vice versa and the model does two separate mapping $f_{G_1}(\theta_{G_1}) : X \to Y$ and $f_{G_2}(\theta_{G_2}) : Y \to X$. An adversary to these two generators, the two discriminators, perform an inspection of the generated image. $f_{D_1}(\theta_{D_1})$ try to distinguish between the image $x_i \in X$ and the image generated by $f_{G_2}(\theta_{G_2}, y_i)$ and the discriminator $f_{D_2}(\theta_{D_2})$ distinguishes between the image $y_i \in Y$ and the image generated by $f_{G_1}(\theta_{G_1}, x_i)$.

## Loss Functions

The CycleGAN model has two type of losses :

1. Adversarial Loss: Each generator in the model tries to minimize its loss, whereas each discriminator in the model tries to maximize its loss. The adversarial loss focuses on the fact that the data distribution between the output domain and generated domain measures up with each other. Let the set of input images be X $= \{x_1, x_2, \ldots, x_n\}$ and the set of output images be Y $= \{y_1, y_2, \ldots, y_n\}$. Suppose $L_j(\theta_{G_j}, \theta_{D_j})$ be the adversarial loss for the generative adversarial network $j$, for $j \in \{1, 2\}$ and $x_i \in X, y_i \in Y, z_i \in Z$ for all $i \in \{1, 2, \ldots, N\}$. We define adversarial loss as follows:

$$\begin{aligned} L_1(\theta_{G_1}, \theta_{D_1}) = \frac{1}{N} \sum_{i=1}^{N} \log f_{D_1}(\theta_{D_1}, x_i, y_i) P(Y = y_i) \\ + \frac{1}{N} \sum_{i=1}^{N} \log(1 - f_{D_1}(\theta_{D_1}, x_i, f_{G_1}(\theta_{G_1}, x_i))) P(X = x_i), \end{aligned} \tag{18}$$

for all $\theta_{D_1}$ and $\theta_{G_1}$, and

$$\begin{aligned} L_2(\theta_{G_2}, \theta_{D_2}) = \frac{1}{N}\sum_{i=1}^{N} \log f_{D_2}(\theta_{D_2}, y_i, x_i) P(X = x_i) \\ + \frac{1}{N}\sum_{i=1}^{N} \log(1 - f_{D_2}(\theta_{D_2}, y_i, f_{G_2}(\theta_{G_2}, y_i))) P(Y = y_i), \end{aligned} \tag{19}$$

for all $\theta_{D_2}$ and $\theta_{G_2}$. The goal here is to find the optimal value of $\theta^*_{G_j}$ and $\theta^*_{D_j}$, $\forall j$. We can state the objective as follows:

$$\theta^*_{G_j}, \theta^*_{D_j} = \arg \min_{\theta_{G_j}} \max_{\theta_{D_j}} L_j(\theta_{G_j}, \theta_{D_j}), \forall j. \tag{20}$$

2. Cycle Consistency Loss: CycleGAN introduces a cyclic approach to converting the generated image back to its subsequent image from the original domain. The loss incurred during the process is addressed as Cycle Consistency Loss. Since we have two generators and discriminator pairs, we also have two cyclic consistency loss, namely:

   (a) Forward Consistency Loss: An image $x_i \in X$ domain is fed to the generator $f_{G_1}(\theta_{G_1} x_i, y_i)$ which generates $y_i^{'}$ and then this is again fed to generator $f_{G_2}(\theta_{G_2}, y_i^{'}, x_i)$ which generates $x_i^{'}$, ideally close to $x_i \forall i$. Forward consistency loss is defined as:

$$\begin{aligned} &L_{cyc_1}(\theta_{G_1}, \theta_{G_2}) = \\ &\frac{1}{N}\sum_{i=1}^{N} |f_{G_2}(\theta_{G_2}, f_{G_1}(\theta_{G_1}, x_i, y_i), x_i) - x_i| P(X = x_i) \end{aligned} \tag{21}$$

   (b) Backward Consistency Loss : Similarly to forward consistency loss, we define backward consistency loss as follows:

$$\begin{aligned} &L_{cyc_2}(\theta_{G_2}, \theta_{G_1}) = \\ &\frac{1}{N}\sum_{i=1}^{N} |f_{G_1}(\theta_{G_1}, f_{G_2}(\theta_{G_2}, y_i, x_i), y_i) - y_i| P(Y = y_i) \end{aligned} \tag{22}$$

Our goal is to obtain optimal parameters of the sum of all losses :

$$\begin{aligned} \theta^*_{G_j}, \theta^*_{D_j} = \arg \min_{\theta_{G_j}} \max_{\theta_{D_j}} \sum_{j=1}^{2} (L_j(\theta_{G_j}, \theta_{D_j}) + L_{cyc_j}(\theta_{G_j}, \theta_{G_k}) \\ , \forall j, \forall k \in \{1, 2\} \wedge j \neq k \end{aligned} \tag{23}$$

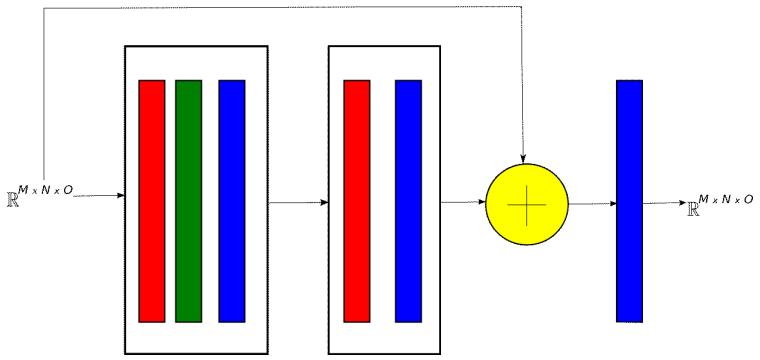



Figure 13: A single residual block. The red layers are convolutional layers. The green layer is instance normalization, the blue layers are ReLU activation layers. The yellow addition symbol adds inputs up.

## 4.2 Modified Cycle Consistent Adversarial Network

Our method is inspired by the architecture of CycleGAN [12]. We propose some modifications to the architecture of CycleGAN for embroidery translation problem. Instead of using just instance normalization [35] like the original CycleGAN, we have added spectral normalization [36] similar to [37] to improve the quality of the generated image. We also took inspiration from [38], which uses a regularized embedded channel for both input and output of the generators. The embedded channel helps the generator memorize the important structures necessary for the reconstruction of the image. As the embedded channel is regularized, it eliminates the possibility of the generator learning the entire image and hence not depreciating the cycle-consistency of CycleGAN. Combining these two modifications to the original architecture of CycleGAN definitely help a lot in improving the quality of the generated images in comparison to original CycleGAN, which we have shown in the following chapter. Let us mathematically formulate the embroidery translation problem. Each image from the input domain has multiple visual attributes like color, shape, size, luminance, texture, and many others. In the CycleGAN [12] paper, the author has mentioned that there is a primal interconnection between the input image and the generated image. In the embroidery translation problem, the input image and the generated image have many similarities. Ideally, almost everything except the texture of the images is similar. We will introduce the modified loss function in the following section.

### 4.2.1 Architecture of Modified CycleGAN

Our network is a modified version of CycleGAN, for generating an embroidered version of images. The network has two generators $f_{G_i}(\theta_{G_i}), \forall i \in \{1, 2\}$ and two Discriminators $f_{D_i}(\theta_{D_i}), \forall i \in \{1, 2\}$. The generator used in the network is almost similar to the generator of CycleGAN [12], which is shown in figure 14. The structure is originally adopted from [39], which showed remarkable results in style transfer and image super resolution. The network contains two stride-2 convolutions, 9 residual blocks [40] and two fractionally strided convolutions with stride $\frac{1}{2}$, which is the typical structure for CycleGAN [39, 12]. In [12], only instance normalization [35] is used in the discriminator but we have also added spectral normalization [36] similar to [37] which increases the quality of the generated images. We also propose to add an additional regularized embedded channel to both the input and the output of the generator similar to [38]. The reason behind adding an embedding channel is to encourage the generator to generate an image which has the least amount of structural loss [38]. An embedding channel of zeros has been added to the input of the generator along with an input image. The $L_1$ regularized channel of zeros indicates the generator that there are no restrictions on how the translation should be done. The residual block shown in figure 13, usually perform better or equal to identity mapping because the input is available at all times. The residual block represent a function $g(x) = f(x) + x$. The residual block also helps reducing the gradient vanishing problem in deep networks. The discriminator networks used are $70 \times 70$ PatchGAN classifier [1, 41, 42], which aims to classify whether a $70 \times 70$ overlapping image patches are original or generated. In [1] results, we have seen that a patch-level discriminator outperforms the full discriminator as it has fewer parameters.

### 4.2.2 Problem Formulation

Let the dataset consists of user images and embroidered images, both of the same dimension. Let $u \in Z^+$ be the height, $v \in Z^+$ be the width and $c \in Z^+$ be the color channel of each image. Suppose n be the total number of images in the dataset and $N = \{1, 2, \cdots, n\}$. Let $X = \{x_1, x_2, \cdots, x_n\}$ be the set of user images, where $x_i \in \mathbb{N}^{u \times v \times c}$ for $i \in N$. Let $Y = \{y_1, y_2, \cdots, y_n\}$ be the set of embroidered images, where $y_i \in \mathbb{N}^{u \times v \times c}$ for $i \in N$. For $i \in \{1, 2\}$, let $G_i$ be a generator and $D_i$ be a discriminator, they consist of neural networks with parameters $\theta_{G_i}$ and $\theta_{D_i}$, respectively, $\theta_{G_i}$ and

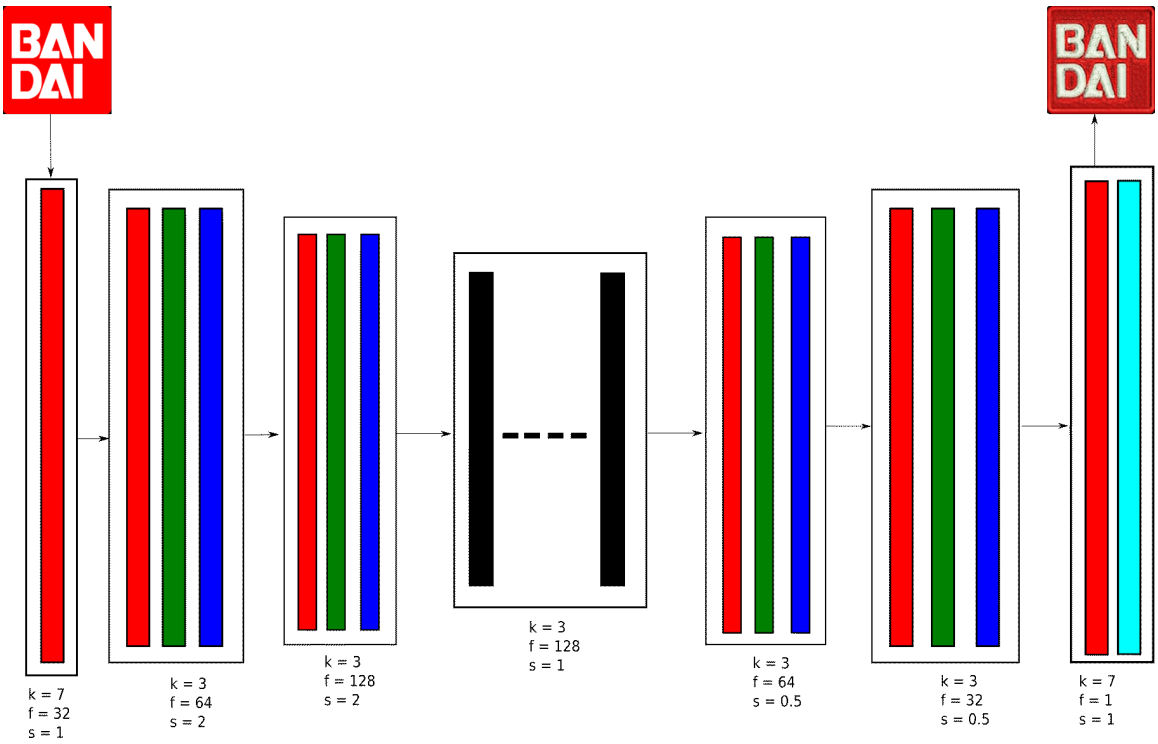



Figure 14: Generator with residual blocks. The red layer is a convolutional layer. The green layer is an instance normalization node. The blue layer is a ReLU activation layer. The yellow layer is a residual block. The cyan layer is a tahn activation layer. Convolutional layers have a kernel of size $k \times k$, $f$ filters and a stride of $s$. For images of size $256 \times 256$ or higher, the number of residual blocks is nine. For smaller images six residual blocks are used.

$\theta_{D_i}$ are the set of weights and biases of neural networks. Let $Q \times Q$ be the dimension of the noise vector $z_i \in Z$. For $i \in \{1, 2\}$, we define differentiable function function $f_{G_i} : \mathbb{N}^{u \times v \times c} \times \mathbb{R}^{Q \times Q} \rightarrow \mathbb{N}^{u \times v \times c}$. We denote the set of images generated by the generator as $\hat{Y} = \{\hat{y_1}, \hat{y_2}, \cdots, \hat{y_n}\}$, where $\hat{y}_k \in \mathbb{N}^{u \times v \times c}$ for $k \in N$. Similarly, $f_{D_i} : \mathbb{N}^{u \times v \times c} \times \mathbb{N}^{u \times v \times c} \rightarrow [0, 1]$ represents discriminator functions. Here, $f_{D_i}(\theta_{D_i})$ and $f_{G_i}(\theta_{G_i})$ are neural networks. $f_{D_i}(\theta_{D_i})$ used the sigmoid activation function, and $f_{G_i}\theta_{G_i}$ uses the hyperbolic tangent function as activation to get the desired outputs. We define the sigmoid function to classify the set of data points in two desired labels. It predicts the probability of occurrence of a particular label say $y \in \{0, 1\}$. Let $x$ be the input, $w$ be the weight vector and $b$ be the bias of the neural network. We

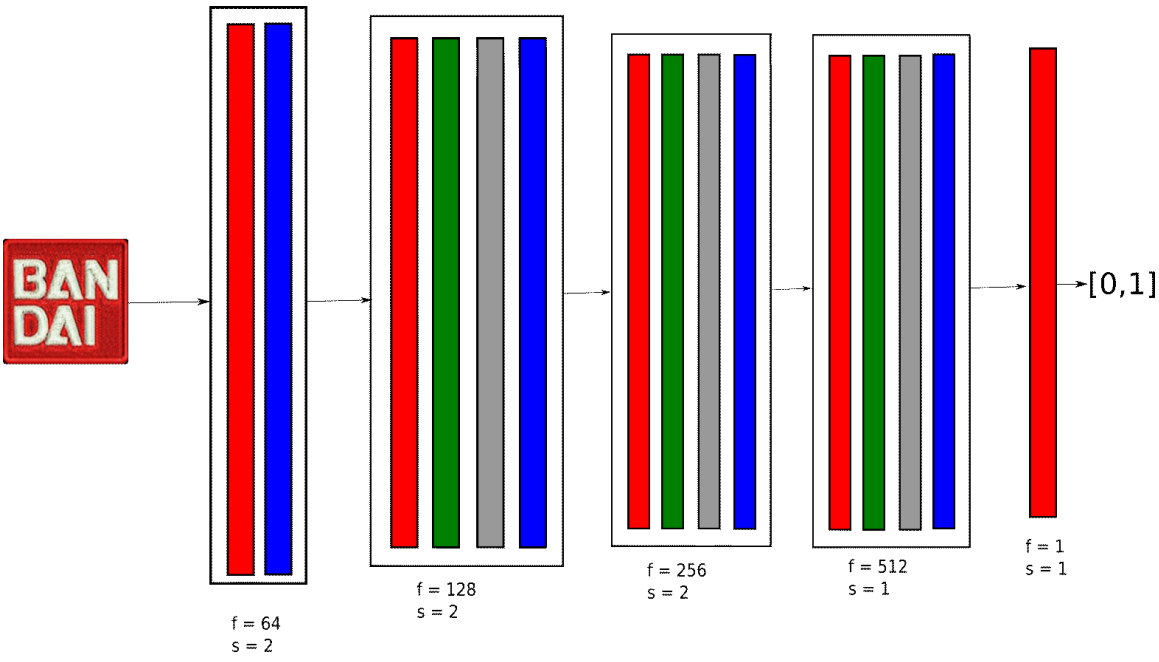



Figure 15: PatchGAN discriminator. The red layer is a convolutional layer with a kernel of size $4 \times 4$, f filters and a stride of s. The blue layer is a LeakyReLU activation layer with a slope of 0.2. The green layer is an instance normalization node. The gray layer is a spectral normalization node. The receptive field for every entry in the output is a $70 \times 70$ patch of the input

defined the sigmoid function $s : \mathbb{R} \to [0, 1]$ as

$$s(z) = \frac{1}{1 + \exp^{-z}},$$

where $z = w \cdot x + b$. If the probability of occurrence of the label is 1 is $s(z)$ then the probability of occurrence of label 0 is $1 - s(z)$. Let $x^{'}$ be the input, $w^{'}$ be the weight vector and $b^{'}$ be the bias of another neural network, we define the hyperbolic tangent function $tanh : \mathbb{R} \to [-1, 1]$ as following

$$tanh(u) = \frac{e^{u} - e^{-u}}{e^{u} + e^{-u}},$$

where $u = w^{'} \cdot x^{'} + b^{'}$ [19]. We use cross entropy to compute the similarity measure between the probability of occurrence of labels based on the true data points from the training dataset and the probability of occurrence of labels from the generated data points. Let $P = \{P_1, P_2, \ldots, P_n\}$ be the visual attributes of an image $\in \{x, y\}$ and, $Q = \{Q_1, Q_2, \ldots, Q_n\}$ be the values of these attributes. Our goal is to find two

optimized mappings $f_{G_1^*} : X \to Y$ and $f_{G_2^*} : Y \to X$. For $\forall x \in X$, the generated image $\hat{y} = f_{G_1}(\theta_{G_1}, x)$, the values $Q_x$ and $Q_y$ of all attributes $P$ should be kept same except the embroidery attribute. That means the optimal mapping $f_{G_1^*}$ transferred only the embroidery attribute without destroying any other attribute [37].

**Loss Function**

The loss function of our model is different from our baseline CycleGAN [12]. To achieve better quality results, we have taken inspirations from different researchers and modified the architecture of CycleGAN to achieve the best results for embroidery translation problem. The loss function for our model is as follows: The adversarial loss and the cyclic loss is similar to CycleGAN (18), (19), (21), (22) which we have previously seen in section 4.1.1. We have also introduced identity loss, and embedded loss. We have used a L1 norm loss function for the additional channel to the input image which is defined as the absolute difference between the target value and the estimated value. For $i = \{1, 2, \cdots, n\}$, let $S$ be the difference, $t_i$ be the target value and $p_i$ be the estimated value, we define L1-norm as:

$$S = \sum_{i=1}^{N} |t_i - p_i|.$$

**Identity Loss**: Identity loss is an optional loss in the original CycleGAN paper, but during our training process, we learned that adding an identity loss helps in training the network efficiently similar to [37]. Identity loss helps the generator identify whether the image is from the input domain or the output domain and hence reducing the chances of making it just a mapping algorithm. We define identity loss as:

$$L_{idt}(\theta_{G_1}, \theta_{G_2}) = \frac{1}{N} \sum_{i=1}^{N} f_{G_2}((\theta_{G_2}, x_i) - x_i) + f_{G_1}((\theta_{G_2}, y_i) - y_i). \tag{24}$$

**Embedding Loss**: We have added an embedded channel to the input and output of both the generators to assist in the process of learning the structure of the input image as shown in the figure (16), is able to learn a separate channel, which can be thought of as an embedding, that helps in the reconstruction of the input. The idea is to encourage the generator to generate a properly translated image and an embedding of the changes that were made are required to reconstruct the input. An L1 distance is used with the embedding loss, similar to [12]. The reason of using an

L1 distance instead of L2 is that L1 produces less blurring images [1]. Also, since a L2-norm squares the error (increasing by a lot if error $> 1$), the model will see a much larger error ( e vs $e^2$ ) than the L1-norm, so the model is much more sensitive to this example, and adjusts the model to minimize this error. If this example is an outlier, the model will be adjusted to minimize this single outlier case, at the expense of many other common examples, since the errors of these common examples are small compared to that single outlier case. Let the extra embedded channel to the input of the image is donated by H, we define the embedding loss as

$$L_{emb}(\theta_{G_1}, \theta_{G_2}) = \frac{1}{N}\sum_{i=1}^{N} ||f_{G_1}(\theta_{G_1}, (x_i + H))||_1 + ||f_{G_2}(\theta_{G_2}, y_i + H)||_1. \tag{25}$$

Our goal is to obtain optimal parameters for the sum of all losses of original CycleGAN which is defined in section 4.1.1, equation (23) combined with (24),(25). Our final loss is defined as:

$$\begin{aligned} \theta^*_{G_j}, \theta^*_{D_j} = \underset{\theta_{G_j}}{argmin}\underset{\theta_{D_j}}{max} \sum_{j=1}^{2} (L_j(\theta_{G_j}, \theta_{D_j}) + L_{cyc_j}(\theta_{G_j}, \theta_{G_k}) \\ + \lambda_1 L_{idt}(\theta_{G_j}) + \lambda_2 L_{emb}(\theta_{G_j}), \\ \forall j, \forall k \in \{1, 2\} \wedge j \neq k. \end{aligned} \tag{26}$$

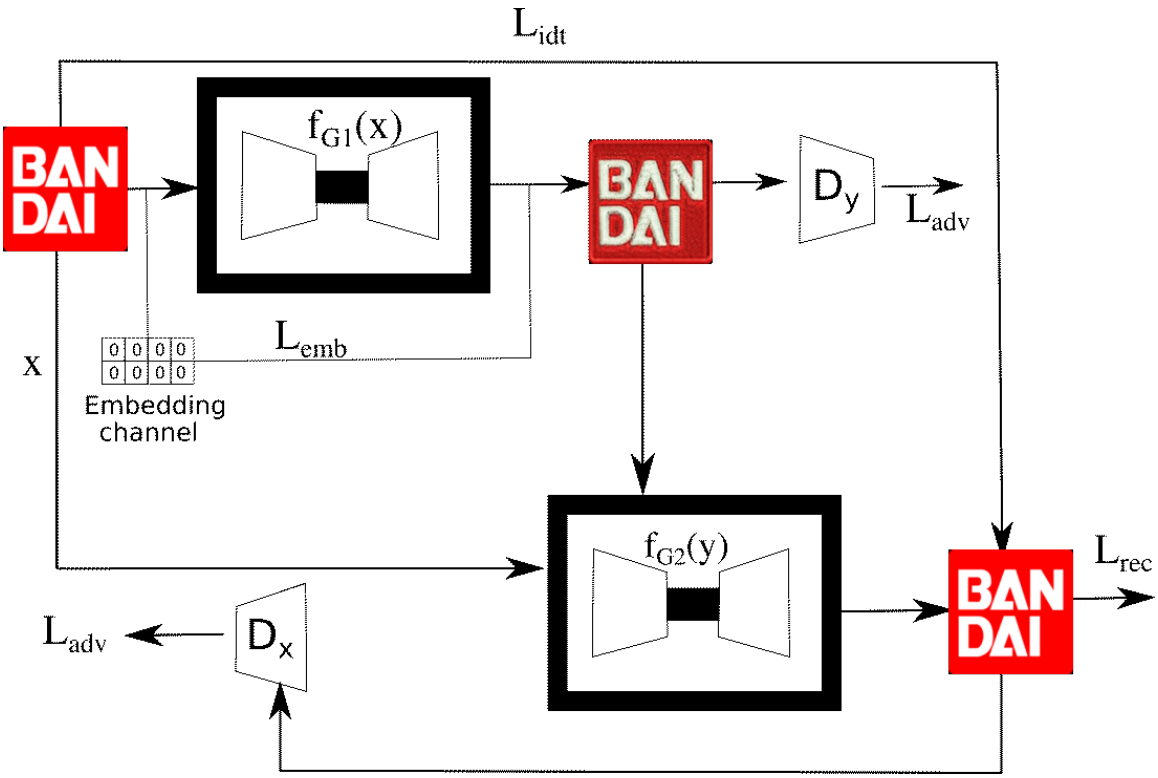



Figure 16: A simplified architecture of our Modified CycleGAN. For simplicity, we have just included one cycle from $x$ to $\hat{y}$ to $x'$. The eight zero's represents the embedding channel which is added to the input and output of the generator. $L_{adv}$ are the adversarial losses, $L_{emb}$ is the embedding loss, $L_{idt}$ is the identity loss. The complete architecture along with loss functions are explained in section (4.2), page (32).

**Algorithm 2:** Algorithm to train the EmbGAN model

Pre-processing: user images $X = \{x_1, x_2, \ldots, x_N\}$, embroidered images $Y = \{y_1, y_2, \ldots, y_N\}$, $u \times v \times c \leftarrow 256 \times 256 \times 3$, $N$ number of images in training dataset.

Input for generator $G : (x_1), (x_2), \ldots, (x_N)$, $x_i \in X$ for all $i \in \{1, 2, \ldots, N\}$

Output: generated embroidery like image $\hat{Y} = \{\hat{y_1}, \hat{y_2}, \ldots, \hat{y_N}\}$,

where $\hat{y_i} \leftarrow f_G(x_i, z_i; \theta_G)$ for all $x_i \in X$, $z_i \in Z$, $\hat{y_i} \in \hat{Y}$ and $i \in \{1, 2, \ldots, N\}$

Input for discriminator $D : \{y_1, y_2, \ldots, y_N\}$ and $\{\hat{y_1}, \hat{y_2}, \ldots, \hat{y_N}\}$, $y_i \in Y, \hat{y_i} \in \hat{Y}$ for all $i \in \{1, 2, \ldots, N\}$.

Initialization: Learning rate $\lambda \leftarrow 0.0002$, normalization factor $\alpha \leftarrow 10$

**if** *network layer is Convolutional* **then**
  $weights \leftarrow \mathcal{N}(0, 0.02)$
  $biases \leftarrow 0$,
**end**
**else if** *network layer is normalization* **then**
  $weights \leftarrow \mathcal{N}(1, 0.02)$
  $biases \leftarrow 0$,
  **if** *norm=instance* **then**
    use InstanceNormalization
  **end**
  **else if** *norm=batch* **then**
    use SpatialBatchNormalization
  **end**
  use SpectralNormalization
**end**
// $T$ is the total number of epochs the model is trained for.
**for** $t \leftarrow 1$ *to* $T$ **do**
  **for** $j \leftarrow 1$ *to* $N$ **do**
    update the parameters using stochastic gradient descent (SGD),
    $\theta_{D_1} = \theta_{D_1} - \lambda \frac{\partial L_1(\theta_{G_1}, \theta_{D_1})}{\partial \theta_{D_1}}$;
    $\theta_{G_1} =$
    $\theta_{G_1} - \lambda \frac{\partial \Big( L_1(\theta_{G_1}, \theta_{D_1}) + L_{cyc_1}(\theta_{G_1}, \theta_{G_2}) + L_{cyc_2}(\theta_{G_2}, \theta_{G_1}) + L_{emb}(\theta_{G_1}, \theta_{G_2}) + L_{idt}(\theta_{G_1}, \theta_{G_2}) \Big)}{\partial \theta_{G_1}}$;
    $\theta_{D_2} = \theta_{D_2} - \lambda \frac{\partial L_2(\theta_{G_2}, \theta_{D_2})}{\partial \theta_{D_2}}$;
    $\theta_{G_2} =$
    $\theta_{G_2} - \lambda \frac{\partial \Big( L_2(\theta_{G_2}, \theta_{D_2}) + L_{cyc_1}(\theta_{G_1}, \theta_{G_2}) + L_{cyc_2}(\theta_{G_2}, \theta_{G_1}) + L_{emb}(\theta_{G_1}, \theta_{G_2}) + L_{idt}(\theta_{G_1}, \theta_{G_2}) \Big)}{\partial \theta_{G_2}}$;
  **end**
**end**

# Chapter 5

# Experiment

In this chapter, we briefly explain the dataset we use in this research and a preprocessing technique that we have done on the images in the dataset to improve the results of image-to-image translation. We prepare an unpaired dataset with images from two domains. One set of images are user-uploaded images which are the inputs to our algorithms. The other set of images are the final embroidered version of the user-uploaded images. These final embroidered version are design using a proprietary software in the fashion industry. A professional digitize the user-uploaded images and then make an embroidered version of it manually. CapBeast, a Montreal based firm has been a part of this research and very helpful in providing these images for this research [1]. We will then breifly describe the training details for style transfer and then describe details of how we trained our modified CycleGAN (EmbGAN) and in the end we will see the results generated by our methods and baseline methods, we have also performed perceptual studies on our result using AMT(Amazon Mechanical Turk) and compare our architectures with the baseline architectures.

## 5.1 Dataset

We have prepared a dataset for embroidery image-to-image translation. The image used in the dataset are simple two dimensional images which can be embroidered have fairly less complex semantic content. A photographic image of a skyline or an image with complex semantic content would be feasible for embroidery and hence is

[1] CapBeast owns a copyright on the dataset used in this research.

not used in preparing the dataset. Most of the images are simple flags, texts, logs or other simple structural images as shown in 17. We can broadly divide the dataset into two categories, one of which is the textual image which consists of word(s) that a user has uploaded to be customized on their apparel and the other category are any non-textual images. The other division of the dataset is based on the color of the images, we have multi-color images, single-color images and gray-scale images. The total number of images in the dataset is 8668, out of which 4643 are user uploaded images, and 4025 are manually embroidered version of these images.


Figure 17: Sample images from our dataset. 1st row : User uploaded images, 2nd row : manually embroidered images using proprietary software.

Table 2: Data distribution in embroidery dataset.

| **Category** | **Total** | **Train** | **Validation** | **Test** |
|---|---|---|---|---|
| **Textual Images** | 2713 | 1408 | 712 | 593 |
| **Non-Textual Images** | 5955 | 3212 | 1433 | 1310 |
| **overall** | 8668 | 4620 | 2145 | 1903 |

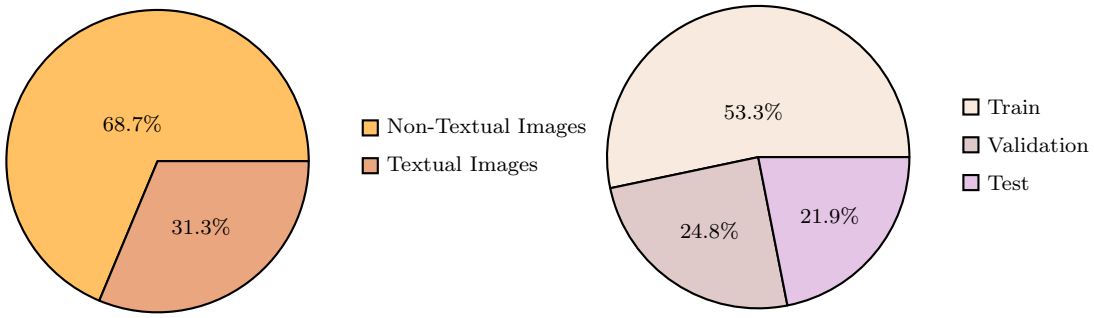



Figure 18: Statistics of the embroidery dataset. First piechart denotes the categorical distribution i.e., the number of textual and non textual images present in the dataset. The second piechart denotes the train-validation,test distribution i.e., the number of images used to train, validate and test the model.

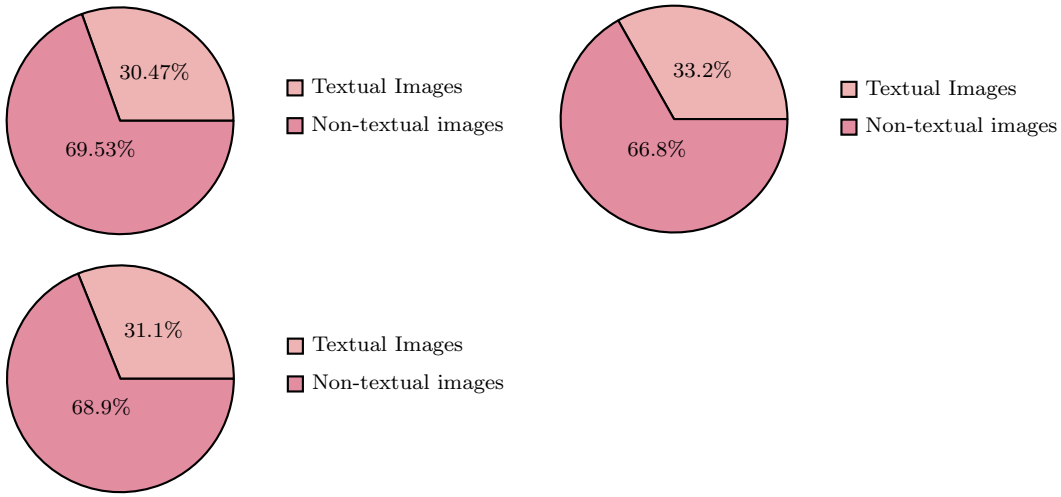



Figure 19: Statistics of the embroidery dataset

## 5.2 Preprocessing

The input images in our dataset are user-uploaded and hence, their size is not uniform. The non-uniformity in the size of the image can be an issue while training the GANs. We did preprocessing on all the images from our dataset. We run a batch script to resize all the images to $256 \times 256$. During our research and experiments, we find out that the non-uniformity in the value of one colored pixel can lead to washed out colors in the generated image. We know there are many different color representing schemes,

RGB is one of them and the most popular one. RGB stands for the color component


Figure 20: An image with two visible colors, red and white but the total number of colors present in the color palette of the above image is 472.

of red, green, and blue in an image. Each component have a value between $\{0-255\}$, where 0 represents that absence of the component and 255 represents that maximum saturation of a color component. Though human eyes cannot discriminate among all the colors in the RGB system, in total we have $256 \times 256 \times 256 = 16,777,216$ colors. In a standard image, the color palette can be extensive in comparison to the visible colors in the image. A color palette is all the colors that are present in the image irrespective of their individual percentage. The human eye cannot discriminate between easily between $\{255, 0, 0\}$ and $\{255, 69, 0\}$ because both of them appear to be red but one of them is red and the other is orange red.

### Color Quantization

Figure 20 is an example of the non-uniformity in the value of the colored pixels. We did preprocessing to remove the non-uniformity in the value of pixels, we use k-means color quantization to make all the adjacent color values in an image to be clustered together to one. However, color quantization might reduce the quality of images by a small fraction k-means manage to keep the quality intact for more than 90% of the images. We have performed k-means color quantization on our dataset [2]. and the following are a few results. We performed the color quantization as a batch processing for our entire dataset and for uniformity we have decided the value of k to be 8 almost more than 90% of the images have similar visual appearance as their original counterpart.

[2]CapBeast owns a copyright on the dataset used in this research

**K-means Color Quantization**

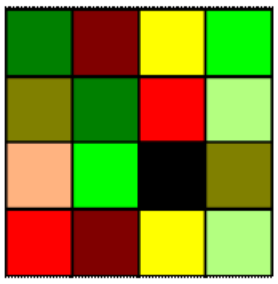
Let $H \in Z^+$ be the height, $W \in Z^+$ be the width and $C \in Z^+$ be the color channel of each image. Suppose n be the total number of images in the dataset and $N = \{1, 2, \cdots, n\}$. Let $X = \{x_1, x_2, \cdots, x_n\}$ be the set of user images, where $x_i \in \mathbb{N}^{W \times H \times C}$ for $i \in N$. The k-means clustering algorithm minimized the sum of squared error (SSE) between the clusters. K-means algorithm divide $x_i$ in mutually exclusive clusters $R = \{R_1, R_2, \cdots, R_K\} \cup_{k=1}^{K} R_K = x_i \in \mathbb{N}^{W \times H \times C}$ to perform pixel-wise vector quantization.

$$SSE = \sum_{k=1}^{K} \sum_{x_i \in R_K} ||x_k - c_k||_2^2,$$

where $|| \cdot ||_2$ denotes the Euclidean distance and $c_k$ centroid of the cluster [43]. A heuristic method developed by Lloyd [44] offers a simple solution for k-means. Lloyd's algorithm starts with $K$ centers, typically chosen uniformly at random from the data points [45]. Each point is then assigned to the nearest center, and each center is recalculated as the mean of all points assigned to it. These two steps are repeated until a predefined termination criterion is met. The pseudocode for this procedure is given below. Here, $m[i]$ denotes the membership of point $\mathbf{x}_i$, i.e. index of the cluster center that is nearest to $\mathbf{x}_i$.

**Algorithm 3:** K-Means Algorithm

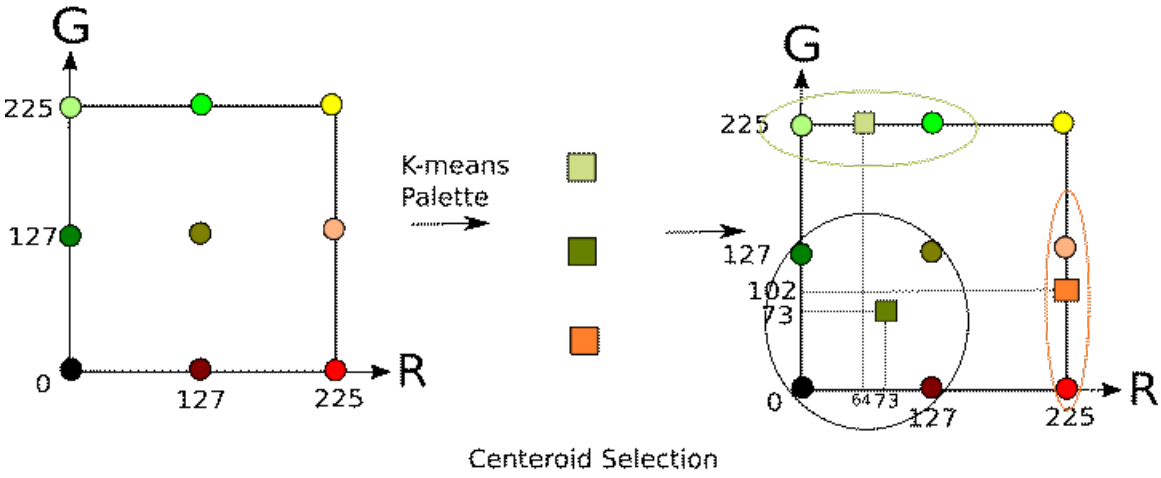
**input :** $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\} \in \mathbb{R}^D$ ($N \times D$ input data set)
**output:** $C = \{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_K\} \in \mathbb{R}^D$ ($K$ cluster centers)
Select a random subset $C$ of $X$ as the initial set of cluster centers;
**while** *termination criterion is not met* **do**
    **for** $(i = 1; i \leq N; i = i + 1)$ **do**
        `Assign` $\mathbf{x}_i$ `to the nearest cluster`;
        $m[i] = \underset{k \in \{1,2,\ldots,K\}}{\operatorname{argmin}} \|\mathbf{x}_i - \mathbf{c}_k\|^2$;
    **end**
    `Recalculate the cluster centers`;
    **for** $(k = 1; k \leq K; k = k + 1)$ **do**
        `Cluster` $S_k$ `contains the set of points` $\mathbf{x}_i$ `that are nearest to the center` $\mathbf{c}_k$;
        $S_k = \{\mathbf{x}_i \,|m[i] = k\}$;
        `Calculate the new center` $\mathbf{c}_k$ `as the mean of the points that belong to` $S_k$;
        $\mathbf{c}_k = \frac{1}{|S_k|} \sum_{\mathbf{x}_i \in S_k} \mathbf{x}_i$;
    **end**
**end**



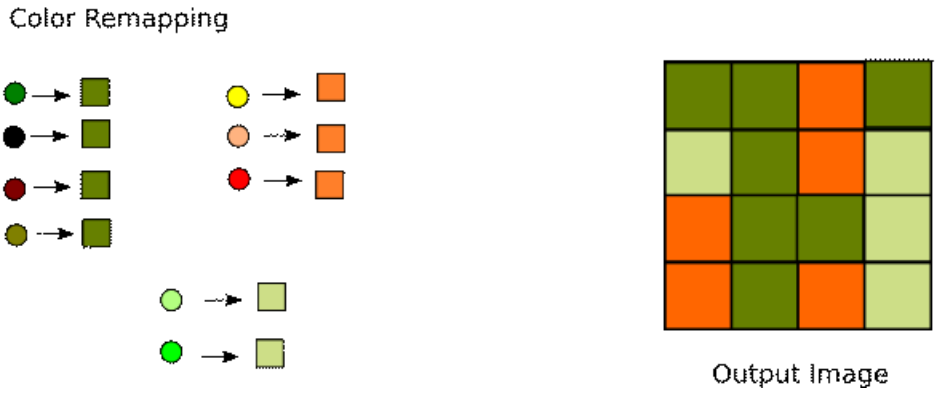
Input Image

| | | | |
|---|---|---|---|
| (0,127) | (127,0) | (255,255) | (127,127) |
| (127,255) | (0,127) | (255,0) | (0,255) |
| (255,127) | (127,127) | (0,0) | (127,255) |
| (255,0) | (127,0) | (255,255) | (0,255) |

Pixel values




Centeroid Selection




Output Image

Figure 21: A simplified example to explain the k-means algorithm (3). For simplicity, we have shown an example of an RGB colorspace and we have kept the value of B channel to be a constant 0. The value of k selected is three and the final output has three color in the palette, whereas the original had nine.

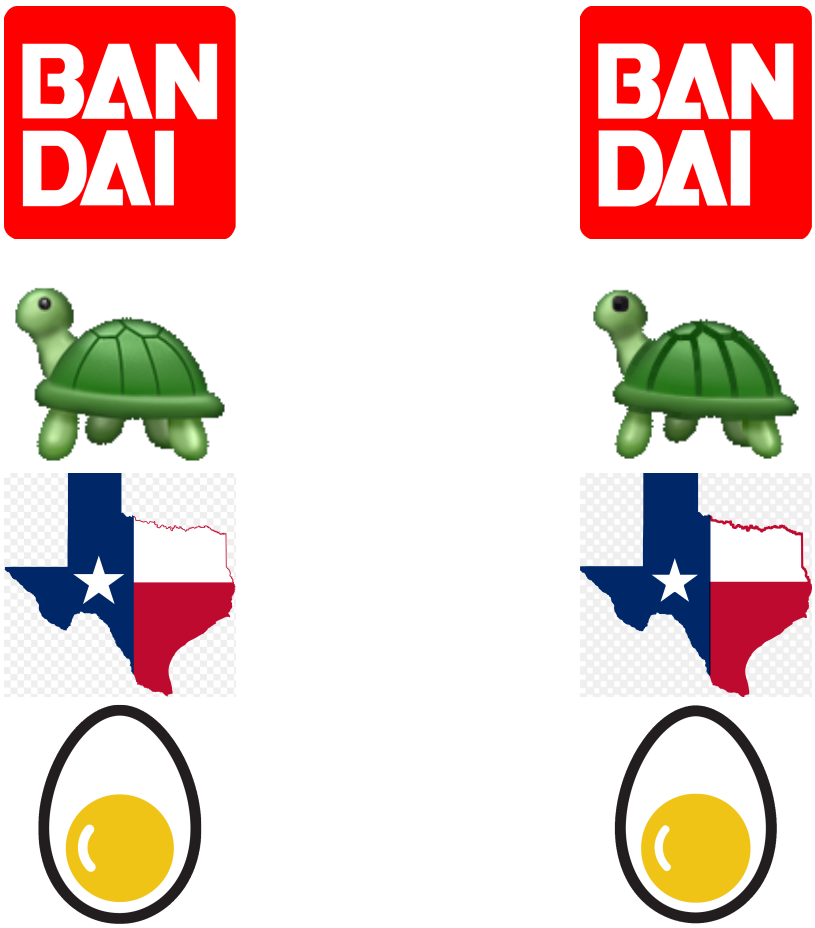



Figure 22: K-means color quantization result. Left column: the user uploaded images. Right column: k-means color quantized version. The value of k is 8.

## 5.3 Implementation Details of Style Transfer

We used an NVIDIA GTX GeForce 1080 GPUs to implement the style transfer. We have used a pretrained VGG-16 convolutional neural network [13] similar to the original paper [11]. We have compared different optimizers to be used but we finally choose the L-BFGS optimizer similar to the original paper [11].

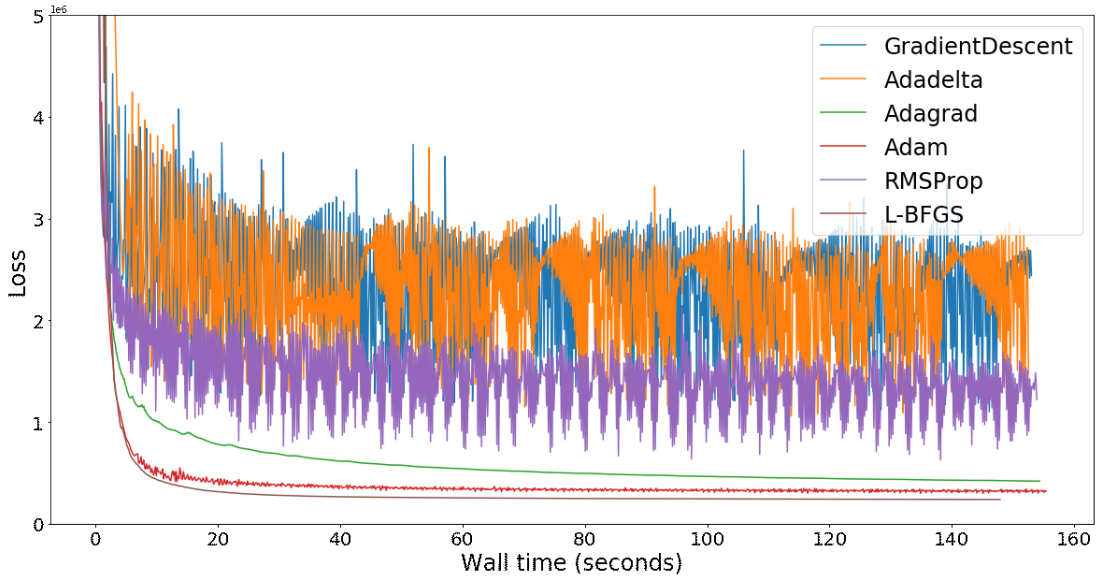



Figure 23: Comparison of different optimizers.

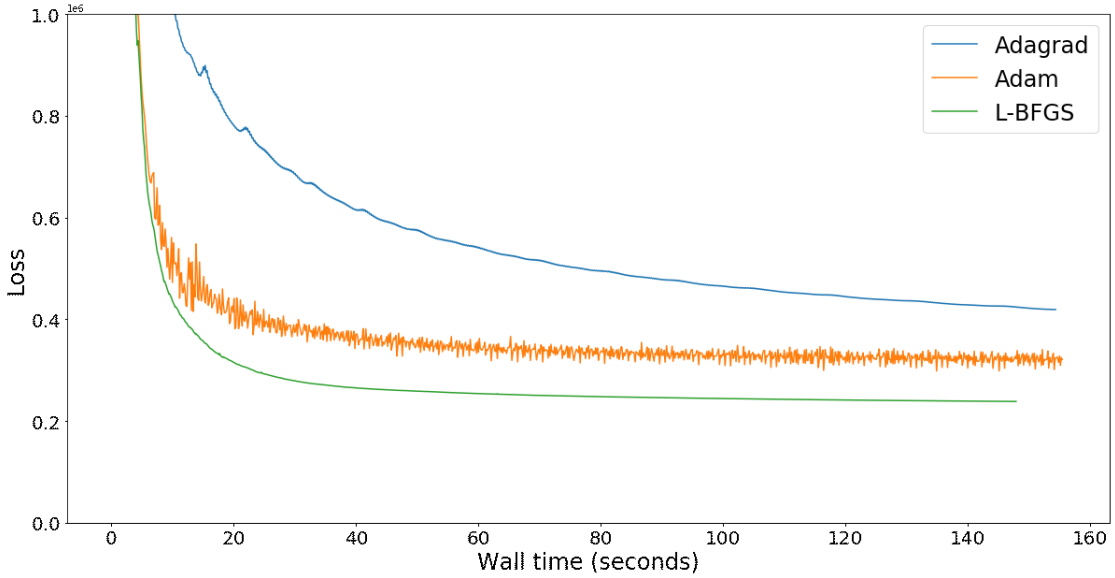



Figure 24: Comparison of Adam, Adagrad and L-BFGS optimizers.

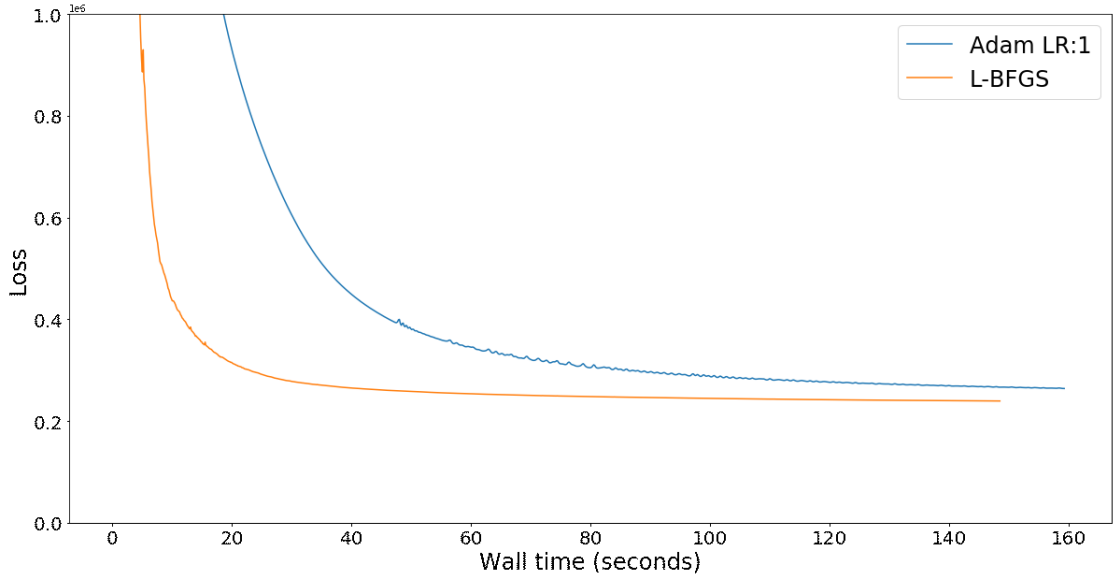



Figure 25: Comparison of Adam with learning rate 1 and L-BFGS optimizers.

For the comparison experiment, we decided to use the learning rate 10, and the number iterations are 1000. The size of images used is $256 \times 256$. We have compared Gradient Descent, Adadelta, Adagrad, Adam, RMSProp, L-BFGS. In figure 23, we can see the comparison of different optimizers. The gradient descent, Adadelta, and RMSProp were the worst optimizers for this task. They were unable to converge with this learning. We further compare the three best optimizers in figure 24. We compared the Adam optimizer and L-BFGS once more but we used lower learning rate of 1 for Adam to see if it outperforms L-BFGS. But even with a lower learning rate, L-BFGS performs better than Adam for style transfer 25.

## 5.4 Training Details for EmbGAN

We used an NVIDIA GTX GeForce 1080 GPUs for the training process. The initial learning rate is set as 0.0002. The $\lambda_1$ and $\lambda_2$ are set as 7 and 3.5, respectively, in equation [26] from section 4.2.2. To upgrade the parameters, we have used Adam solver[46], and WGAN gradient penalty [47] during the training process, and the momentum is set to 0.6. These values and hyperparameters are explicitly tuned for generating good quality embroidered images, and the results show these values are most suitable. For comparison, we have some images digitized by a professional

graphic designer using proprietary software, these images are considered as ground truth images for the network. The ground truth images are used to have a qualitative comparison of the generated images with the professionally digitized images from proprietary software. There are some expensive software used by apparel customization companies to digitize a logo or an image to be ready for machine embroidery.

## 5.5 Result

In this section, we will see the results from our proposed methods and compare them with their baseline counterparts. Our final task is to generate an embroidered version, Y of any input image, X. We want our results to be as similar to a machine embroidered image as possible, and hence we have used the machine embroidered image along with images from domain X to train our model. We have used the original methods as baselines for the qualitative comparison of our results. We will first make a qualitative comparison of images from our method Split Style Transfer to its baseline counterpart Neural Style Transfer and then make a similar comparison of images from our Modified-CycleGAN with the baseline counterpart CycleGAN.

Table 3: AMT "Real vs. Fake" Test. We compare the images generated from CycleGAN and EmbGAN with the ground truth. A turker receives a pair of images where one image is from ground truth, and one is a generated image from either CycleGAN or EmbGAN. The table displays how many times an algorithm was able to fool the turkers.

| Algorithm | % Turkers marked as real |
|---|---|
| CycleGAN | 4.08 |
| EmbGAN | 14 |

Table 4: AMT Comparison Test. We compare the images styled by neural style transfer and split style transfer to one another. A turker receives a pair of images where one image is a styled image by neural style transfer, and the other image is a styled image by split style transfer. The table displays which results were more realistic as an embroidered image, according to the turkers.

| Algorithm | % Turkers marked as real |
|---|---|
| Neural Style Transfer | 32.5 |
| Split Style Transfer | 67.5 |

Table 5: AMT Comparison Test. We compare the images generated from CycleGAN and EmbGAN to one another. A turker receives a pair of images where one image is a generated image from CycleGAN, and the other image is a generated image from EmbGAN. The table displays which results were more realistic as an embroidered image, according to the turkers.

| Algorithm | % Turkers marked as real |
|---|---|
| CycleGAN | 8.5 |
| EmbGAN | 91.5 |

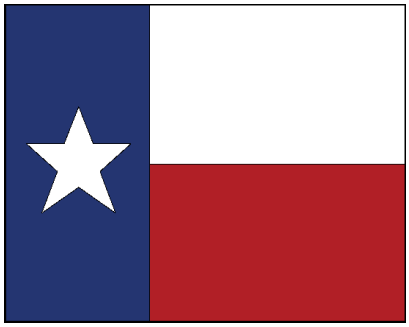
### 5.5.1 Perceptual Studies

The absolute best metric for evaluating the results of any image-to-image translation problem or any other image synthesization task in the field of computer graphics and computer vision is to check how compelling the results are to a human observer. Similar to the baseline architecture, we have run perceptual studies of "real vs. fake" on Amazon Mechanical Turk, the protocol used is identical to [[1], [12], [6]]. We have also run a comparison perceptual studies comparing the results of baseline architecture and our modified architecture. For the "real vs. fake" perceptual studies, the turkers were presented with a series of pair of images, where one of the images is ground truth which is considered "real," and the other image is generated by our algorithm which is considered as "fake" in our study. The pair of images appeared for 1 second, and after the images disappeared, the turkers were given an unlimited amount of time to decide

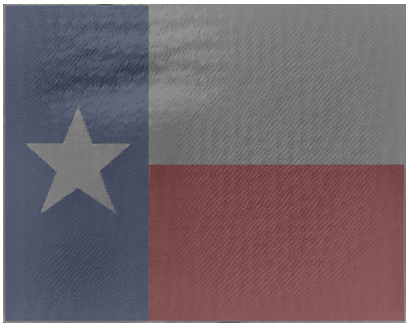
which one was real or fake. The Table 3, displays the result of the first perceptual study. We can see that the images generated from EmbGAN were manage to fool the turkers 4 times more than the images generated from CycleGAN. For the comparative perceptual study, we have increased the time to 2 seconds, so that users will have more time to make the comparison. The Table 5, displays the result of comparison test between CycleGAN and EmbGAN. 91.5% of the turkers choose the images generated from EmbGAN to be more realistic as an embroidered image in comparison to the image generated from CycleGAN. Approximately 50 turkers evaluated each algorithm. For every algorithm, we provided a pair of 50 images, but for the comparative study of neural style transfer and split style transfer, we have used a pair of 40 images. The Table 4, displays the result of the comparison test between neural style transfer and split style transfer. 67.5% of the turkers choose the images styled by split style transfer to be more realistic as an embroidered image in comparison to the image styled by neural style transfer. To ensure that the participants were competent at this task, we have used the strategy similar to [6], 10% of the trials pitted the ground truth with the images generated from initial epochs and the turkers were able to identify the generated image as fake 94% of the time, indicating that the turkers understand the task in hand. To ensure that there is no bias and all the algorithms were tested in the similar environment, we ensure that all the experiments were carried out at the same time of the day and all the sessions were independent and identically distributed to the turkers simultaneously.

## 5.6 Split Style Transfer Results

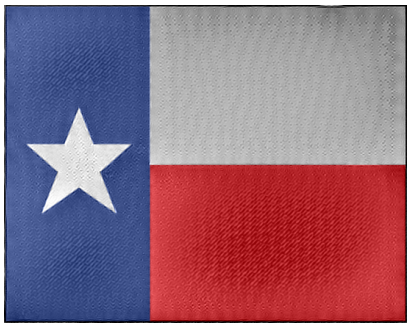
In this section, we display the images styled by neural style transfer and our propose method split style transfer. Every image has three versions, the first is the user uploaded image, the second is a styled image using neural style transfer and one embroidery image, the third is a style image using split style transfer and three different embroidery image for each distinct color. The results will help us in perceptually compare the algorithms based on the visual quality of the styled images.



(1)

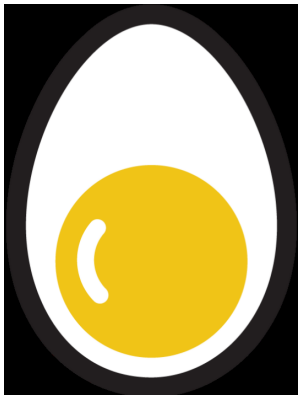


(2)

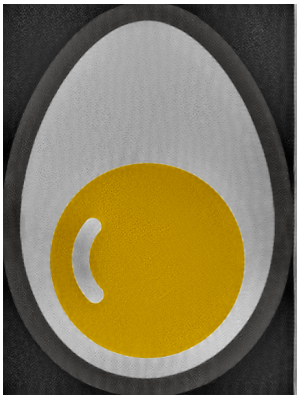


(3)

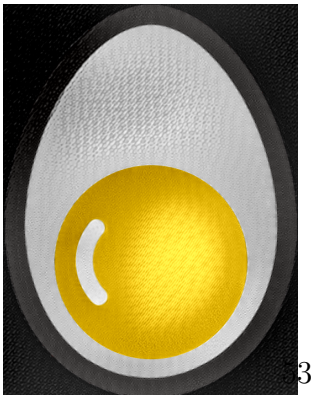
Figure 26: Style Transfer : (1) is the user uploaded image, (2) is a styled image using neural style transfer and one embroidery image, (3) is a style image using split style transfer and three different embroidery image for each distinct color.



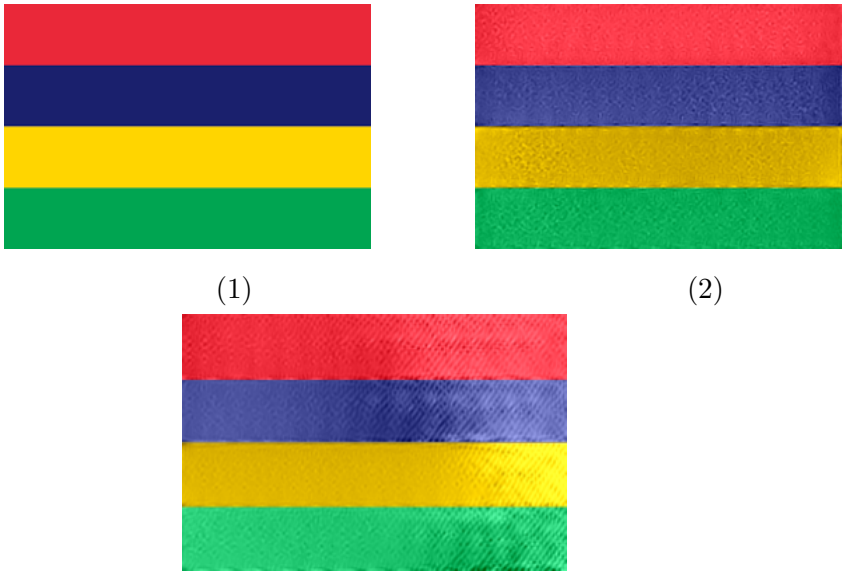
(1)



(2)



(3)



(1) (2)

(3)

Figure 28: Style Transfer : (1) is the user uploaded image, (2) is a styled image using neural style transfer and one embroidery image, (3) is a style image using split style transfer and three different embroidery image for each distinct color.

## 5.7 Limitation of Style Transfer

Style transfer whether it is neural style transfer or the modification that we did to the baseline architecture to propose split style transfer did manage to generate some embroidery pattern in their final outputs. However, the results from the split style transfer were better than neural style transfer, the style transfer as a technique to perform embroidery image-to-image translation is not sufficient. There are some drawbacks to this approach, as the objective of style transfer is to jointly minimize the distance between the style representation and content representation of an image, it fails to capture the spatial detail necessary to generate an approximate embroidery preview of a given two-dimensional image. Features like the difference in the boundary stitch pattern and the background stitch pattern were not visible prominently in the style transfer results. The results did not manage to capture some sort of difference in the boundaries within the image that will separate two separate regions. Also, the results from style transfer techniques are like imprints of an image on another image and blending them, and using some sort of noise to generate the embroidery pattern

and this for sure will not help the customer in the decision-making process as the results are far away from a realistic depiction of what an embroidered version of an image will look like.

## 5.8 EmbGAN Results

In this section, we will see the images generated by our modified version of CycleGAN (EmbGAN) and compare them with the images generated by the original CycleGAN. Also, we will compare both these generated images with the images digitized by proprietary software which is considered to be the ground truth and see how our well both methods perform in generating an approximate embroidered version of an image. Each subfigure has four versions. One of them is the user-uploaded input image. Second, is the manually digitized embroidered version of the image used as ground truth. Third, is the result generated by original CycleGAN. Lastly, the result generated by our modified EmbGAN. The results will help us in perceptually compare the algorithms based on the visual quality of the generated images.


(1) user image


(2) ground truth


(3) embgan

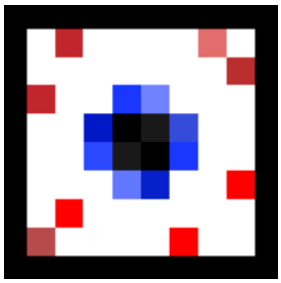

(4) cyclegan

(5) user image

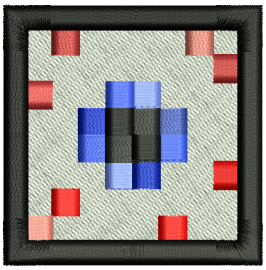
(6) ground truth

(7) embgan

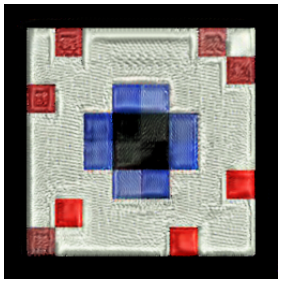
(8) cyclegan



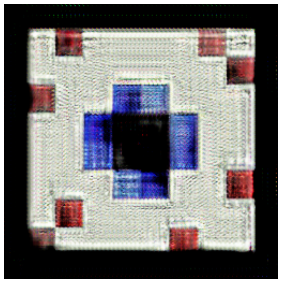
(9) user image



(10) ground truth



(11) embgan



(12) cyclegan


(13) user image


(14) ground truth


(15) embgan


(16) cyclegan


(17) user image


(18) ground truth


(19) embgan


(20) cyclegan


(21) user image


(22) ground truth


(23) embgan


(24) cyclegan

Figure 29: EmbGAN Result: Every subfigure has four versions. One of the them is the user-uploaded input image. Second, is the manually digitized embroidered version of the image used as ground truth. Third, is the result generated by original CylceGAN. Lastly, the result generated by our modified EmbGAN.

# Chapter 6

# Conclusions and Future Work

In this work, we propose two techniques to solve our embroidery image-to-image translation problem. The techniques we propose are a modification of two existing machine learning techniques which are popular in producing good results for any given image-to-image translation task, *neural style transfer* and *cycle-consistent generative adversarial network*. We have done an embroidery image-to-image translation to generate an approximate real-time preview for a customer who wants to have customized embroidery on their apparel. The approximate preview of the final embroidered version of their uploaded two-dimensional image will help the customer in the decision-making process and hence will help in reducing the amount of product returned because of customer's dissatisfaction. The results from both the techniques were very satisfactory. We used perceptual studies to compare the results with the baseline architectures and the results from the split style transfer were qualitatively better than neural style transfer, and the generated images from EmbGAN were qualitatively better than CycleGAN. We also compared the generated images from both the techniques which conclude that the modified cyclegan (EmbGAN) generates the best quality embroidered image for any given image. We have used the digitized embroidered version of images using industry software as the baseline for comparison. After our final comparison between the generated images, we find out that, the style transfer technique is not the best approach for solving embroidery image-to-image translation. The images generated by style transfer feel less realistic visually, and hence, would not be the best approximate preview of the embroidered version of a

user-uploaded image. Therefore, for future work, we propose to continue with EmbGAN to improve the quality of the final generated images. We propose to make some advancements in the dataset. We propose to develop a better dataset with a lot more images of higher quality and different semantic content, and we also propose to structure the dataset in a way that similar type of images are together so that it's more feasible for the EmbGAN to learn an embroidery pattern. We also propose to use different GANs or translation techniques to solve this problem, namely StarGAN, cyCADA(Cycle-Consistent Adversarial Domain Adaptation), and SPADE (Spatially Adaptive Normalization).

# Bibliography


[1] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1125–1134, 2017.

[2] A. A. Efros and W. T. Freeman, "Image quilting for texture synthesis and transfer," in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pp. 341–346, ACM, 2001.

[3] A. Hertzmann, C. E. Jacobs, N. Oliver, B. Curless, and D. H. Salesin, "Image analogies," in *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, pp. 327–340, ACM, 2001.

[4] A. Buades, B. Coll, and J.-M. Morel, "A non-local algorithm for image denoising," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2, pp. 60–65, IEEE, 2005.

[5] D. Eigen and R. Fergus, "Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2650–2658, 2015.

[6] R. Zhang, P. Isola, and A. A. Efros, "Colorful image colorization," in *European Conference on Computer Vision*, pp. 649–666, Springer, 2016.

[7] Y. Shih, S. Paris, F. Durand, and W. T. Freeman, "Data-driven hallucination of different times of day from a single outdoor photo," *ACM Transactions on Graphics*, vol. 32, no. 6, p. 200, 2013.

[8] P.-Y. Laffont, Z. Ren, X. Tao, C. Qian, and J. Hays, "Transient attributes for high-level understanding and editing of outdoor scenes," *ACM Transactions on Graphics*, vol. 33, no. 4, p. 149, 2014.

[9] S. Xie and Z. Tu, "Holistically-nested edge detection," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1395–1403, 2015.

[10] T. Chen, M.-M. Cheng, P. Tan, A. Shamir, and S.-M. Hu, "Sketch2photo: Internet image montage," in *ACM transactions on graphics*, vol. 28, p. 124, ACM, 2009.

[11] L. A. Gatys, A. S. Ecker, and M. Bethge, "A neural algorithm of artistic style," *arXiv preprint arXiv:1508.06576*, 2015.

[12] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," *2017 IEEE International Conference on Computer Vision*, Oct 2017.

[13] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[14] 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* (Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, eds.), pp. 2672–2680, Curran Associates, Inc., 2014.

[15] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. The MIT Press, 2016.

[16] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning*, vol. 112. Springer, 2013.

[17] A. Géron, *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems.* " O'Reilly Media, Inc.", 2017.

[18] C. M. Bishop *et al.*, *Neural Networks for Pattern Recognition*. Oxford university press, 1995.

[19] S. Haykin and N. Network, "A comprehensive foundation," *Neural networks*, vol. 2, no. 2004, p. 41, 2004.

[20] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning," *arXiv preprint arXiv:1811.03378*, 2018.

[21] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," tech. rep., California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[22] P. Werbos, "Beyond regression:" new tools for prediction and analysis in the behavioral sciences," *Ph. D. dissertation, Harvard University*, 1974.

[23] D. H. Hubel and T. N. Wiesel, "Receptive fields and functional architecture of monkey striate cortex," *The Journal of physiology*, vol. 195, no. 1, pp. 215–243, 1968.

[24] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[25] 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.

[26] C.-C. J. Kuo, "Understanding convolutional neural networks with a mathematical model," *Journal of Visual Communication and Image Representation*, vol. 41, pp. 406–413, 2016.

[27] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.

[28] J. R. Gardner, P. Upchurch, M. J. Kusner, Y. Li, K. Q. Weinberger, K. Bala, and J. E. Hopcroft, "Deep manifold traversal: Changing labels with convolutional features," *arXiv preprint arXiv:1511.06421*, 2015.

[29] C. Li and M. Wand, "Combining markov random fields and convolutional neural networks for image synthesis," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2479–2486, 2016.

[30] A. Selim, M. Elgharib, and L. Doyle, "Painting style transfer for head portraits using convolutional neural networks," *ACM Transactions on Graphics*, vol. 35, no. 4, p. 129, 2016.

[31] Y. Choi, M. Choi, M. Kim, J.-W. Ha, S. Kim, and J. Choo, "Stargan: Unified generative adversarial networks for multi-domain image-to-image translation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8789–8797, 2018.

[32] S. Zhu, R. Urtasun, S. Fidler, D. Lin, and C. Change Loy, "Be your own prada: Fashion synthesis with structural coherence," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1680–1688, 2017.

[33] L. A. Gatys, A. S. Ecker, and M. Bethge, "Image style transfer using convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2414–2423, 2016.

[34] H. Zhang and K. Dana, "Multi-style generative network for real-time transfer," *arXiv preprint arXiv:1703.06953*, 2017.

[35] D. Ulyanov, A. Vedaldi, and V. Lempitsky, "Instance normalization: The missing ingredient for fast stylization," *arXiv preprint arXiv:1607.08022*, 2016.

[36] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, "Spectral normalization for generative adversarial networks," *arXiv preprint arXiv:1802.05957*, 2018.

[37] S. Zhang and D. Yang, "Pet hair color transfer based on cyclegan," in *2018 5th International Conference on Systems and Informatics (ICSAI)*, pp. 998–1004, IEEE, 2018.

[38] R. Longman and R. Ptucha, "Embedded cyclegan for shape-agnostic image-to-image translation," in *2019 IEEE International Conference on Image Processing (ICIP)*, pp. 969–973, Sep. 2019.

[39] 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.

[40] 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.

[41] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, *et al.*, "Photo-realistic single image super-resolution using a generative adversarial network," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 4681–4690, 2017.

[42] C. Li and M. Wand, "Precomputed real-time texture synthesis with markovian generative adversarial networks," in *European Conference on Computer Vision*, pp. 702–716, Springer, 2016.

[43] M. E. Celebi, Q. Wen, S. Hwang, and G. Schaefer, "Color quantization of dermoscopy images using the k-means clustering algorithm," in *Color Medical Image Analysis*, pp. 87–107, Springer, 2013.

[44] S. Lloyd, "Least squares quantization in pcm," *IEEE transactions on information theory*, vol. 28, no. 2, pp. 129–137, 1982.

[45] E. Forgy, "Cluster analysis of multivariate data: Efficiency versus interpretability of classification," *Biometrics*, vol. 21, no. 3, pp. 768–769, 1965.

[46] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[47] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of wasserstein gans," in *Advances in Neural Information Processing Systems*, pp. 5767–5777, 2017.