Deep Learning Concepts for Evolutionary Art

Fazle Rabbi Tanjil

Submitted in partial fulfilment
of the requirements for the degree of

Master of Science

Department of Computer Science
Brock University
St. Catharines, Ontario

Abstract

A deep convolutional neural network (CNN) trained on millions of images forms a very high-level abstract overview of any given target image. Our primary goal is to use this high-level content information of a given target image to guide the automatic evolution of images. We use genetic programming (GP) to evolve procedural textures. We incorporate a pre-trained deep CNN model into the fitness. We are not performing any training, but rather, we pass a target image through the pre-trained deep CNN and use its the high-level representation as the fitness guide for evolved images. We develop a preprocessing strategy called Mean Minimum Matrix Strategy (MMMS) which reduces the dimensions and identifies the most relevant high-level activation maps. The technique using reduced activation matrices for a fitness shows promising results. GP is able to guide the evolution of textures such that they have shared characteristics with the target image. We also experiment with the fully connected "classifier" layers of the deep CNN. The evolved images are able to achieve high confidence scores from the deep CNN module for some tested target images. Finally, we implement our own shallow convolutional neural network with a fixed set of filters. Experiments show that the basic CNN had limited effectiveness, likely due to the lack of training. In conclusion, the research shows the potential for using deep learning concepts in evolutionary art. As deep CNN models become better understood, they will be able to be used more effectively for evolutionary art.

Acknowledgements

I would like to thank my amazing supervisor, Dr. Brian Ross, for his continuous guidance and instant advice throughout all of our research. Special thanks to Micheal Gircys for his amazing work on the GP system and helping me understand the system. I am also thankful to Cale Fairchild for his support. I would also like to thank NVIDIA for the donation of the Titan XP GPU unit which enabled us to run our experiments faster. Special thanks to Brock University and the Department of Computer Science for giving me the opportunity to undertake this research.

Finally, I would like to express my immense gratitude towards my parents for their endless support, encouragement, and prayer during the entire time of my graduate studies. This research would not have been possible without their unconditional support.

Contents

1 Introduction **1**

1.1 Goals and Motivation 2

1.2 Thesis Structure 3

2 Background **4**

2.1 Deep Learning 4

2.1.1 Definitions 4

2.1.2 Background 5

2.2 Convolutional Neural Networks 9

2.2.1 Architecture of CNN 10

2.3 VGG and ImageNet 16

2.3.1 Training 18

2.4 Genetic Programming 18

2.4.1 Representation and Initialization 19

2.4.2 Fitness 19

2.4.3 Selection 20

2.4.4 Reproduction Operators: Crossover and Mutation 20

2.5 Procedural Textures 21

2.5.1 Noise 22

2.6 Evolutionary Art 23

3 Literature Review **24**

3.1 Deep CNN in Artistic Image Generation 24

3.1.1 Image Style Transfer Using Convolutional Neural Networks . . 24

3.2 Evolutionary Art 27

3.3 Deep Learning and Evolutionary Art 29

4 System Design **30**
4.1 Control Flow of the Overall Architecture . . . 31
4.2 Genetic Programming . . . 33
4.3 Deep Convolutional Neural Network . . . 33
4.3.1 Fitness . . . 34
4.4 Software and Hardware Details . . . 34

5 Experiment Details **36**
5.1 General Setup . . . 36
5.2 Naive Strategy . . . 37
5.3 Dimensionality Reduction . . . 42
5.4 Statistical Analysis and Reducing the Number of Feature Maps . . . 44
5.5 The Mean Minimum Matrix Strategy
(MMMS) . . . 46
5.5.1 GP Results . . . 51
5.5.2 Further Discussion on MMMS . . . 58
5.6 Target Combinations using MMMS . . . 58
5.6.1 Configuration of the System . . . 58
5.6.2 Experiment Details and Analysis . . . 59
5.6.3 Future Work . . . 61

6 Fully Connected Layer **65**
6.1 Background . . . 65
6.1.1 Fully Connected Layer System Overview . . . 66
6.1.2 Using Fully Connected Layer As a Guide to Evolution . . . 66
6.2 System Design and Initialization . . . 67
6.2.1 Fitness . . . 69
6.3 Experiment Details and Analysis . . . 70
6.4 Target Combinations Using the FC Layer . . . 78
6.5 Conclusion . . . 81

7 Basic CNN **82**
7.1 Introduction . . . 82
7.2 GP Integration . . . 87
7.3 Experiments and Result Analysis . . . 87
7.3.1 Greyscale . . . 87
7.3.2 RGB Image . . . 92

8 Conclusion **96**

8.1 Summary 96

8.2 Future Work 97

Bibliography **99**

List of Tables

2.1 The convolutional layer parameters are denoted as “conv(receptive field size)-(number of channels)” [50]. FC-(number of channels) stands for fully connected layer. In system labels small “r” means ReLU layer and “p” means pooling layer. 17

5.1 GP parameters 37
5.2 GP texture language overview 40
5.3 Mean Min activation values for Group 1 and Group 2. Green boxes indicate common activations in the same level. 50
5.4 Image groupings and selected common activations numbers 53
5.5 Image groupings and selected common activations numbers 54
5.6 Target combinations. Top table shows the activation for each separate target image. Bottom table combines the top activations. Notice how combined selected activations are chosen. First half comes from first content image activation. Second half comes from second content image activations. 60

6.1 Class description and class number defined by the ILSVRC 2012 dataset. The confidence scores from deep CNN (VGG-19) are listed for reference. 71

7.1 Image kernels and result after convolution (greyscale) 84
7.2 Mean activations of the 14 feature maps in the final layer(Layer 7). . 90

List of Figures

2.1 A single neuron. Image source [1] 6
2.2 A simple multilayer perceptron 7
2.3 A simple deep learning model for object recognition. Image inspired by [28] 8
2.4 Typical CNN architecture. 10
2.5 Matrices required for performing convolution operation 11
2.6 The convolution operation. Image source [3] 12
2.7 Matrices required for performing convolution operation 13
2.8 The connection between the input volume and the convolution layer. Notice how the convolution layer performs convolution on a small local region of the input volume, not the total volume. This small local region is called the *local receptive field*. 14
2.9 Depth of convolutional layer. 15
2.10 The max-pooling operation. Each colour block represents 2x2 window size. Largest element of each block is taken. The windows stride is 2. The operation reduces the dimension from 4x4 to 2x2. 15
2.11 Control flow of genetic programming. 19
2.12 GP tree representation of the expression $(x - y) + \sin(z)$ 20
2.13 Reproduction operators: crossover and mutation 22
2.14 Simple Perlin noise 23

3.1 Style transfer algorithm. Left side depicts style representation and right side depicts content representation (image source [22]). 26

4.1 Overall architecture design 32

5.1 Red circle content image 38
5.2 Mean activations for the target circle image. Notice the wide range of values. 41

5.3 Some results from the final generation. All the images from the final generation are almost white. 42
5.4 Max pooling vs average pooling applied to circle image. 43
5.5 Visualization of first 135 feature maps of red circle image in layer r42. Notice how some feature maps are activated to varying levels, while others are completely turned off. 45
5.6 Similar content images. 46
5.7 Content images grouped into pairs. 47
5.8 Best results of 10 runs. Standardized fitness plots are averaged over 10 runs. 55
5.9 Best results of 10 runs. Standardized fitness plots are averaged over 10 runs. 56
5.10 Best results of 10 runs. Standardized fitness plots are averaged over 10 runs. 57
5.11 Best results of 10 runs. Standardized fitness plots are averaged over 10 runs. 62
5.12 MMMS combination of horizontal and vertical stripes. Lower MSE in any axis means they are close to the corresponding target image in that axis. 63
5.13 MMMS combination of the vertical stripes and the circle. Lower MSE in any axis means they are close to the corresponding target image in that axis. 64

6.1 System design of fully connected layer with GP. Green box is the new FC layer output we included. The FC layer output is a 1000x1 vector. 68
6.2 Best results of 8 runs. Standardized fitness plots are averaged over 8 runs. 74
6.3 Best results of 8 runs. Standardized fitness plots are averaged over 8 runs. 75
6.4 Best results of 8 runs. Standardized fitness plots are averaged over 8 runs. 76
6.5 Best results of 8 runs. Standardized fitness plots are averaged over 8 runs. 77
6.6 Best results of 8 runs. Standardized fitness plots are averaged over 8 runs. 80

7.1 Input image 85

7.2 Basic CNN architecture 86
7.3 14 feature maps from the first layer after convolution + ReLU + Max pooling 88
7.4 Visualization of activations of 14 feature maps from 6th layer 89
7.5 Best results of 10 runs. Standardized fitness plots are averaged over 10 runs. 91
7.6 Best results of 10 runs. Standardized fitness plots are averaged over 10 runs. 92
7.7 Best results of 10 runs. Standardized fitness plots are averaged over 10 runs. 94
7.8 Best results of 10 runs. Standardized fitness plots are averaged over 10 runs. 95

Chapter 1

Introduction

Evolutionary art has been the subject of great interest since its inception. There has been a lot of effort to create art with the help of computational intelligence [17, 51, 45, 56]. Due to the exploratory nature of evolutionary computation, evolutionary methodologies have been very successful in this application area. Many of the early attempts using evolutionary computation were interactive, i.e. evolved image fitness is manually assigned a score based on the visual appeal [29]. After that, techniques for automatic evolution have been of interest. The main goal is to improve the artistic quality and sophistication of evolved images.

One of the major challenges in the process of creating automatic art is how to intelligently guide the high-level characteristics of evolved images. It is an important part of creative image generation because, even for human artists, high-level representations are critical. Finding new ways to give higher-level control to the evolutionary art system is therefore worth investigating.

Previously the idea of having a high-level abstract overview of a target object image has been difficult to realize. Many computer vision algorithms have been developed to detect certain low-level features of an image, but these algorithms are usually *ad hoc* solutions that deal with low-level pixel information. Recently, breakthroughs in deep artificial neural networks [50] [20] [33] [22] and the concept of deep learning greatly resolves this problem [22]. A deep convolutional neural network (CNN) trained on millions of images forms a very high-level abstract overview of image content [22]. A deep CNN is made of millions of neurons with weights and activations which fire or activate based on different inputs. They have biases associated with them such that neurons only activate after crossing some threshold value. These neurons along with their all weights, biases, and activations form an abstract representation of the given input.

In short, we can say that a deep CNN is capable of having a high-level abstract overview of an image regardless of the low-level pixel positions or spatial information. It can also retain spatial invariance which means slight changes in low-level pixels does not affect the overall high-level representation. Therefore, this might be used as an intelligent tool to guide evolutionary art. This thesis is based on this idea, and we will investigate whether it can be successfully applied to evolutionary art.

1.1 Goals and Motivation

Our goal is to investigate and develop a creative evolutionary system using genetic programming integrated with a trained deep CNN. The deep CNN will serve as a fitness guide to the evolutionary system. We will not be performing any training, but instead, we will use a pre-trained deep CNN. We will pass a target object image (or content image) to the deep CNN. It will form a high-level overview of that given target image. The high-level content information will be used as a fitness target for evolution so that evolved images will retain some of the characteristics of the content image. This way, the evolutionary system be guided by the content image during its exploration of images.

In the past, there were no novel ways or algorithms to represent the content image in a high-level abstraction. Popular computer vision algorithms like SIFT [37], HOG [16], and SURF [10] try to detect the local features like a key point, an edge, or a blob, but fail to capture overall high-level image representation. They essentially work as feature detectors. However, major problem with those is that they have a very limited generalization capability. On the other hand, deep CNN uses several hidden layers to hierarchically model the high-level representation of an image. For example, the first layer might detect edges in the image. The second layer might detect corners present in the content image based on the previous layer's detected edges. The third layer detects other features. Changes in one feature do not drastically change the whole representation of an image in the higher layer. Each deep layer captures characteristics of multiple features of the previous layer. This will be a very important aid to our evolutionary system. The evolutionary technique we will primarily focus on is Genetic Programming (GP). GP automatically tries to solve problems without knowing the concrete form of the problem in advance. It will be interesting to see how GP generates "creative solutions" while guided by the deep CNN.

Our research will be predominantly focused on customizing the deep CNN architecture to fit with our GP system. We will investigate which components of the deep CNN are more effective when used in GP fitness. We will also experiment with a complete deep CNN system including the final output (or classification) layer to guide evolution. The classification layer is the most abstract representation of the target object within the deep CNN. To investigate the concept further, we will also design our own basic convolutional neural network from scratch and use it in image evolution.

1.2 Thesis Structure

The thesis is organized as follows. Chapter 2 provides background information on the underlying concepts we will be using, such as an introduction to deep learning, convolutional neural network, procedural textures, and genetic programming. In Chapter 3, we will discuss previous work which motivates our research. Chapter 4 describes the overall architecture of the GP/deep CNN system. Chapter 5 presents experiments using the convolution layers of the deep CNN and explores different fitness strategies. It also provides a description of the GP language, parameters, and analysis of the best GP results. Chapter 6 explores the use of the fully connected layer as a fitness target instead of using the convolutional layers. In Chapter 7, we describe our own implementation of a basic CNN and use it for image evolution. Finally, Chapter 6 summarizes the thesis and discusses future work.

Chapter 2

Background

2.1 Deep Learning

2.1.1 Definitions

Deep learning is a type of machine learning, which tries to exploit the unknown structure in the input distribution to find good representations, in multiple levels, with higher-level learned features defined in terms of lower-level features [11]. It enables computer systems to improve with experience and data [28]. The definition of deep learning is somewhat broad. Simply speaking, deep learning allows computers to learn from experience and understand the world in terms of a hierarchy of concepts, with each concept defined in terms of its relation to simpler concepts. The hierarchy of concepts allows the computer to learn complicated concepts by building them out of simpler ones. If we draw a graph showing how these concepts are built on top of each other, the graph is *deep*, with many layers. For this reason, this approach to AI is called deep learning [28].

From another perspective, deep learning can be described as a class of machine learning techniques, where numerous layers of non-linear information processing steps are arranged in hierarchical supervised models/architectures. These models are then exploited for unsupervised feature learning. The process of hierarchical representations of the data is the core of deep learning techniques, where the higher-level features are defined from the lower-level features [19]. It is important to note that there are two key aspects of deep learning: (i) models of multiple layers of non-linear information processings, (ii) methods for supervised or unsupervised learning of feature representation at the higher layers [19].

Deep learning uses a mixture of ideas from other research areas like graphical modeling, optimization, pattern recognition, and signal processing [19]. In fact, in case of artificial neural networks, the concept of deep hierarchy is applied in terms of multiple hidden layers of the network. A feedforward deep neural network or (multi-layer perceptron) is be the most common example of deep learning [28]. Deep learning's accuracy in pattern recognition in a supervised manner comes from the fact that there are massive data sets available nowadays which can be used for rigorous training. Other factors contributing to the accuracy of deep learning methodologies are massively increased computing power and increased availability of affordable GPUS's and improvement in machine learning training algorithms such as backpropagation algorithms.

2.1.2 Background

The basis of deep learning architecture is derived from the artificial neural network. In fact, the most common example of deep learning model is a feedforward deep network or multilayer perceptron (MLP). A perceptron takes many binary inputs $x_1, x_2, ..., x_n$ and produces a single output. Usually, there is a method to compute the output. One such method is the use of real number weights $w_1, w_2, ..., w_n$ associated with each input, which signifies the relationship of input to output. The perceptron's output is calculated by the weighted sum $\sum_j w_j x_j$, which is then evaluated whether it is more than some threshold value. The real number threshold value is a parameter of the perceptron. The mathematical model of a simple binary perceptron can be expressed as:

$$Output = \begin{cases} 0, & \text{if} \quad \sum_j w_j x_j \leq \text{threshold value} \\ 1, & \text{if} \quad \sum_j w_j x_j > \text{threshold value} \end{cases}$$

This is the simplest form of perceptron. In modern neural networks, the weighted sum usually passes through some mathematical function called an activation function (φ). There are many different activation functions, for example, sigmoid, tanh, rectified linear unit and sinusoid. The nodes containing the mathematical functions are called neurons (see Figure 2.1). By arranging many neurons into many layers we get a multilayer perceptron. In summary, an MLP is a mathematical function which maps some sets of input values to output values. This function is formed by composing many simpler functions. Each application of a mathematical function is providing a new representation of the input.

Typically, an MLP consists of three or more layers. The first layer is called the input layer and the final layer is called output layer. All layers in between the input

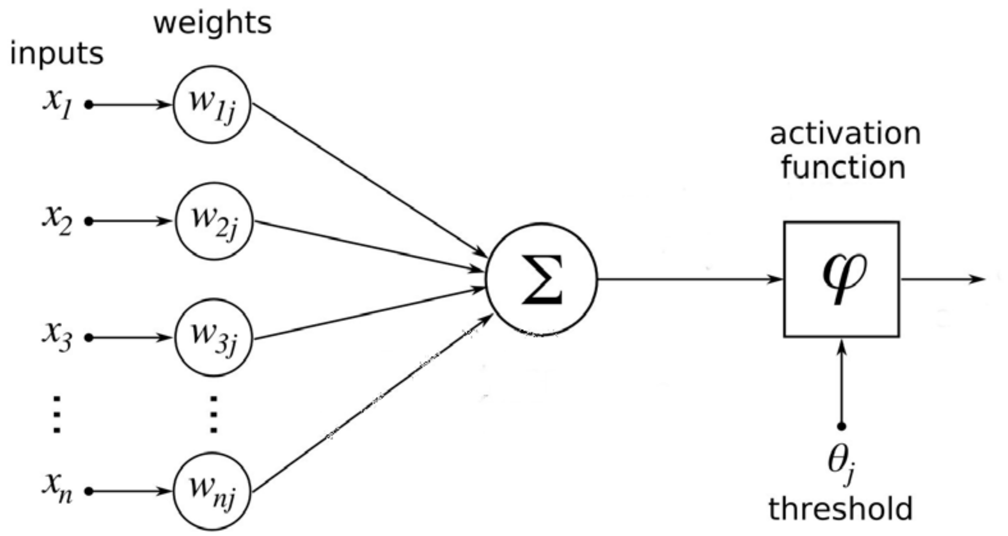


Figure 2.1: A single neuron. Image source [1]

and output layers are called hidden layers (Figure 2.2). In the case of deep neural networks, the total network consists of thousands of neurons arranged in many different layers. The number of intermediate/hidden layers, or depth, varies from architecture to architecture. For example, the VGG (Visual Geometry Group) proposed an architecture which is 19 layers deep [50]. This will be discussed in detail in Section 2.3. The depth enables the overall process to learn a multistep computer program [28]. Each layer representation can be thought of as the state of the process's memory after executing another set of instructions. Networks with greater depth can execute more instructions layer by layer.

One important factor in deep neural networks is the use of non-linear activation functions in hidden layers instead of the linear ones. If linear activations are used for all the hidden layer neurons, the total network will act as a single linear transformation of the input. No matter how many hidden layers are used, the neural network will compute a single linear activation of the input since the composition of many linear functions is just a single linear function. This will make the use of hidden layers pointless. As we already know that deep learning is used for capturing high-level abstract feature representations, non-linearity for the hidden layers is very important.

In order to understand how deep learning models solve complicated classification problems, we will now address the problem of classifying objects of a given image. An image is a collection of pixel values. In digital imaging, a pixel is a smallest physical point in a raster image. The pixels represent an entire object(s). It is difficult for a computer to understand the meaning of raw pixel values. Mapping thousands of

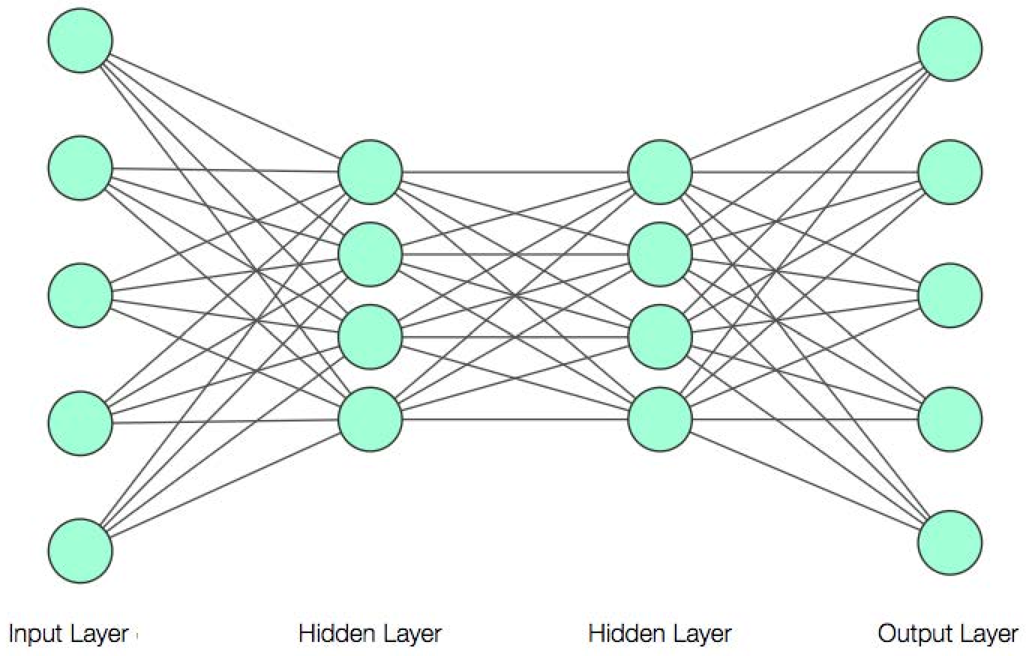


Figure 2.2: A simple multilayer perceptron

pixel value to an object is difficult with traditional computer imaging approach. Deep learning solves this problem by breaking down the complicated inter-pixel mapping into a series of nested simple mappings, each described by the different layer of the model. In order to perform object classification, the image is first fed into the input layer. The series of hidden layers extract increasingly abstract features. From the Figure 2.3, the pixels in the first layer permit easy identification of edges. Given the first hidden layer's description of the edges, the second hidden layer can easily search for corners and extended contours, which are recognizable as collections of edges. From the second hidden layer's description of the image in terms of corners and contours, the third hidden layer can detect entire parts of specific objects, by extending specific collections of contours and corners. Finally, this description of the image in terms of the object components it contains can be used to recognize the overall objects present in the image [28].

Training of Deep Networks

This thesis will not perform any sort of training of deep networks. Rather, we will make use of pretrained deep network model called VGG [50]. In Section 1.3 we will

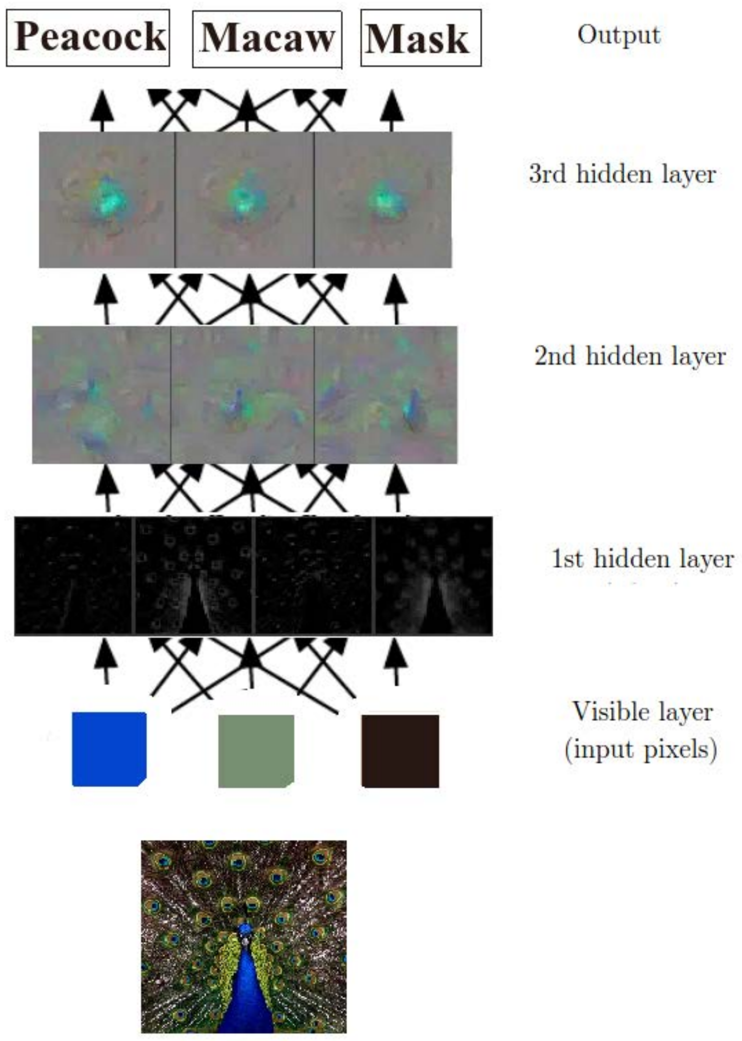


Figure 2.3: A simple deep learning model for object recognition. Image inspired by [28]

discuss VGG in detail. But for now, it is worth mentioning some of the basic concepts of a deep CNN's training process. In supervised learning schemes, training is the process where the network learns to do a particular task from a predefined set of labeled data. In artificial neural networks, training is usually done using a few concepts. First, the aforementioned weights are initialized randomly. Then an input from training set $\hat{x}$ flows through the network to produce output y. The input $\hat{x}$ provides initial information which propagates up to the hidden units at each layer and produces output y . This process is called *forward propagation* [28]. The output is then compared with the output from the training set. The difference is the scalar cost $J(\theta)$. The next concept used is called backpropagation. It allows the information from the cost to flow backward through the network in order to compute the gradient of the cost. The gradient, in this case, is the gradient of the cost function with respect to the parameters/weights $\nabla_\theta J(\theta)$. It means that the gradient is indicating how weight values should be changed so that the cost function is minimized. It is worth pointing out that backpropagation is not a stand alone learning algorithm for neural networks. Rather, it is simply a way of calculating the gradients. Finally, the weights are adjusted in order to minimize the cost. Algorithms such as stochastic gradient descend are used to perform this learning using the gradients from the backpropagation algorithm [28].

During the training phase, all the labeled input data go through the forward and backward propagation process until the cost is greatly minimized. This is the most computationally expensive phase. After the training, the model is ready to be used in an unsupervised manner.

2.2 Convolutional Neural Networks

A convolutional neural network (CNN) is a specialized kind of deep neural network [28]. CNN's have neurons that have learnable weights and biases. Bias is a measure of how easy it is to get a neuron to activate. Like MLP, each neuron receives inputs, performs some operation and based on the architecture, processes it with a non-linear ReLU. However, CNNs slightly differ from regular deep neural networks in some key ways. First, CNN's input data usually has a grid-like shape [28]. A primary example of grid-shaped data is an image. Due to this grid topology, CNN encodes specific characteristics into the architecture. These characteristics make the forward function more efficient and greatly reduce the number of parameters. Second, as the name "convolutional neural networks" suggests, the network applies a mathematical operation called convolution. Convolution is a specialized kind of linear operation. Simply speaking, CNNs are deep networks that use convolution in at least one of their layers instead of traditional matrix multiplication [28]. CNNs were first introduced by Yann LeCun in his "LeNet-5" architecture [34]. Figure 2.4 shows a simple example of that architecture. The CNN is one of the best examples of biologically inspired artificial intelligence.

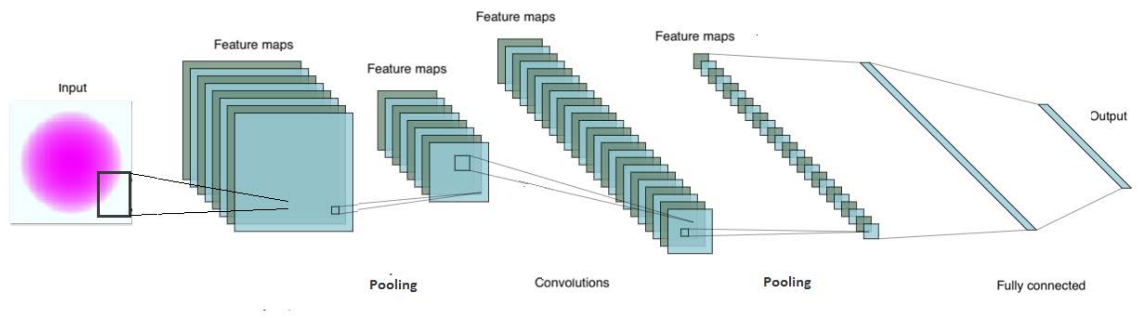


Figure 2.4: Typical CNN architecture.

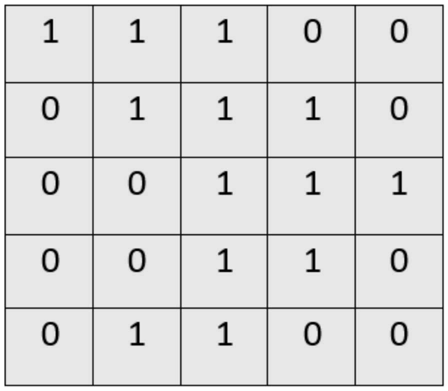
2.2.1 Architecture of CNN

Convolutional neural nets generally employ two major operations: convolution and pooling/downsampling on top of general neural network layers. These operations are applied to a group of neurons separately, and those neurons are divided into convolution layers and pooling layers respectively. Usually, convolutional layers are followed by some types of non-linearity layers. The layers stacked together to form a complete CNN architecture (Figure 2.4).

Convolution Layer

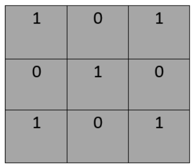
The convolution layer is the core building block of the CNN. This layer applies the convolution operation whose main purpose is to extract features from the input. Before moving into the feature extraction process, we will clarify some of the concepts of the input and output characteristics of the convolution layer. The convolution layer has neurons arranged in 3 dimensions: width, height, depth. It is important to note that depth here means the third dimension of a single convolutional layer, not the the depth of the CNN which is calculated in terms of the number of layers in the total architecture. The first convolutional layer takes an image input. If the image is 256x256 and has 3 colour channels, then the input volume becomes 256x256x3 dimensions. The final layer is usually a classification layer. So, most of the time it is 1x1xK ($1 \leq K \leq$ number of classes). For example, for digit recognition, the final class will be 1 to 10. So, the output volume is 1x1x10.

The feature extraction process can be explained intuitively. The convolution layers have a collection of image filters or kernels which have learnable weights. Convolution preserves the spatial relationship between pixels by learning image features using a small patch of input data [5]. Every filter is a small matrix of fixed width and



1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

(a) Single channel 5x5 input image



1	0	1
0	1	0
1	0	1

(b) A 3x3 filter

Figure 2.5: Matrices required for performing convolution operation

height, and is applied to the full depth of the input volume. To illustrate this, consider an image with a dimension of 32x32x3 (width, height, and the colour channels respectively). The input image is a matrix of pixel values. Filters on the first layer of a CNN might have a size of 3x3x3 (height and width of the filter, and 3 colour channel). During the forward pass, we slide and convolve each filter across the width and height of the input volume, and compute dot products between the entries of the filter and the input at that position. Sliding the filter over the width and height of the input volume will produce a 2D activation map that represents the response of that filter at every position. For simplicity, let us consider a single channel image with the dimension 5x5 and a 3x3 kernel or filter (Figure 2.5). We move the kernel (darker matrix in Fig 5.4b) over the input image matrix (Fig 2.5a) by 1 pixel (called the stride). Next for each position, we perform an element-wise multiplication and add the outputs to get the final integer, which forms a single element of the output matrix (Figure 2.6). In image processing, this 3x3 matrix is called the kernel or the feature detector, and the matrix formed by sliding the filter over the image and computing the dot product is called the convolved feature or activation map or feature map.

It is important to note that filters act as feature detectors from the original input image. For example, consider an input image which contains vertical stripes. Now if we use a vertical line detector kernel and perform the convolution then the resultant feature map would capture the vertical stripe attribute of the input image (Figure 2.7a). On the other hand, if the use the horizontal line detector kernel and perform the convolution with the same image, the resultant feature map is almost dark, meaning this filter did not capture any horizontal attribute of the input image (Figure 2.7b). Since there are no horizontal attribute in the input image. All the feature maps capture different information from the same image by using different filters.

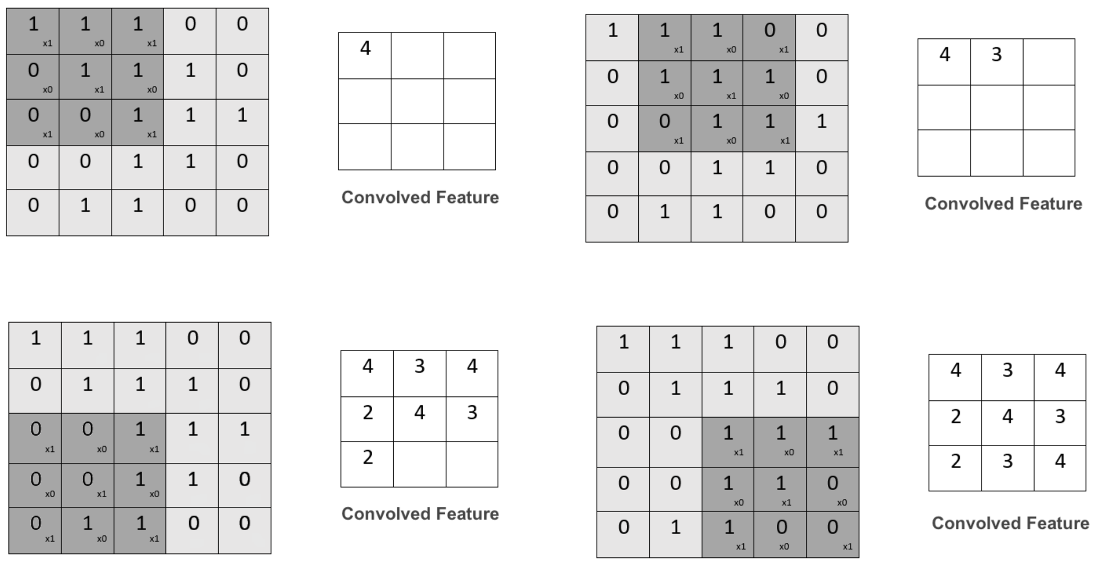
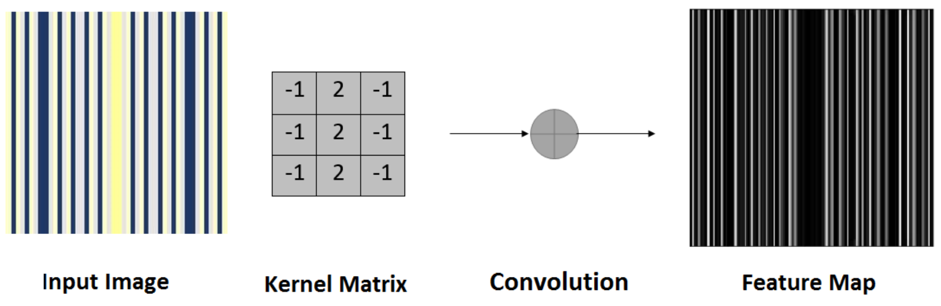


Figure 2.6: The convolution operation. Image source [3]

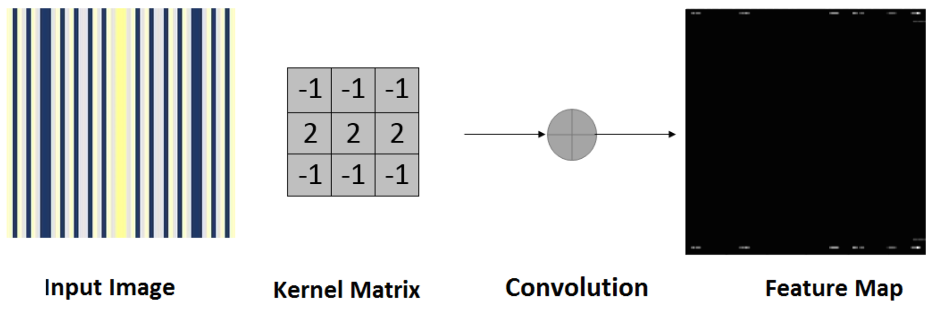
The intuition behind this process is that the CNN will learn to adjust the filters so that they will activate when they see some type of specific visual features, such as an edge of some orientation or a portion of some colour on the first layer. As the depth of convolution layers increases, the complicated features like texture patterns, facial features or object properties will be captured by the filters on the higher layers of the network. After this, we will have an entire set of filters in each convolution layers, and each of them will produce a separate 2-dimensional activation map. We will stack these activation maps along the depth dimension and produce the output volume. Note that the 3x3 matrix sees only a part of the input image in each stride. This concept is called *local receptive field* [34]. Figure 2.8 visually explains this operation. This is very convenient when dealing with high dimensional inputs such as images, because connecting every neuron of a layer to every neuron to the previous becomes impractical due to the tremendous amount of information.

Feature detectors or image kernels learn to detect features during the training phase. The kernel values can be thought of as weights just like in traditional neural networks. The training algorithm adjusts these weights during the training. This makes CNNs powerful because if we have rigorous training, the CNN model will better learn how to detect necessary features. A greater number of filters extract more image features and make the network better at recognizing patterns.

Some parameters are not trainable, such as the number of filters, filter size, archi-



(a) Convolution with vertical line detector kernel.



(b) Convolution with horizontal line detector kernel.

Figure 2.7: Matrices required for performing convolution operation

tecture of the network. These are fixed and should be determined before the training process. These are called hyperparameters. The following section describes some of the hyperparameters that are needed to be defined for CNN before performing the training process. These hyperparameters vary for different architectures.

1. **Depth**: This is the number of filters used in specific convolutional layers. For example, if we use 3 filters and perform convolution the output of these 3 filters will produce 3 distinct feature maps (called the depth). The depth depends on the number of filters used in one particular convolutional layer (see Figure 2.9).

2. **Stride**: This is the number of pixels by which we slide our filter matrix over the input matrix. When the stride is 3 then we move the filters three pixels at a time. Having a larger stride will produce smaller feature maps.

3. **Zero padding**: Zero padding refers to the phenomenon where the border of input matrix is padded with zero. The reason for doing so is that it helps us to control the feature map size.

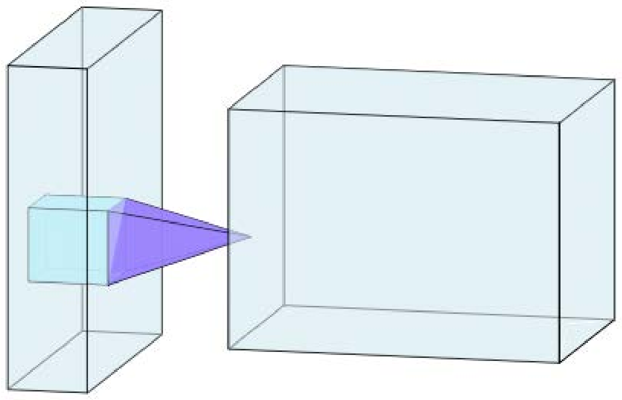


Figure 2.8: The connection between the input volume and the convolution layer. Notice how the convolution layer performs convolution on a small local region of the input volume, not the total volume. This small local region is called the *local receptive field*.

Non Linearity or ReLU Operation

An additional operation called ReLU is used after every convolution operation. ReLU stands for Rectified Linear Unit and is a non-linear operation. It is an activation function in neural networks. Its output is given by the equation:

$$Output = Max(0, Input)$$

ReLU is an element wise operation (applied per pixel) and replaces all negative pixel values in the feature map by zero. The purpose of ReLU is to introduce non-linearity in CNNs. This stage is sometimes referred as the detector stage. Although ReLU is a widely used activation function, there are other types of activation functions as well. For example TanH, Sinusoid, Binary, Sigmoid etc. are pretty common in the field of neural networks.

Pooling Layer

Pooling (also known as subsampling or downsampling) reduces the dimensionality of each feature map but retains the most important information. It is primarily used for reducing the size of the feature maps simply to make the CNN less computationally

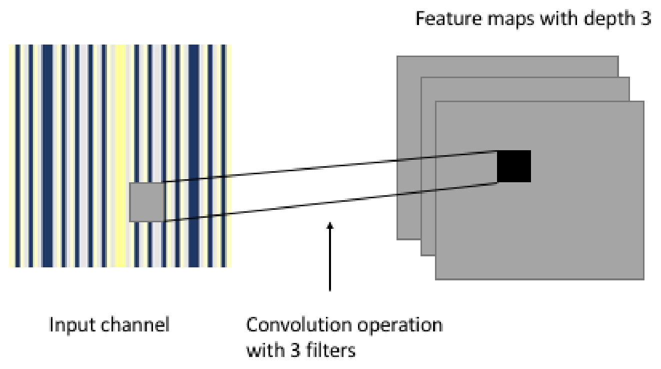

Feature maps with depth 3
Input channel
Convolution operation with 3 filters

Figure 2.9: Depth of convolutional layer.

expensive and less sensitive to smaller changes in the input pixels. Pooling makes the representation approximately invariant to small translations of the input [28]. Spatial pooling can be of different types: max, average, sum, etc. Max pooling is the most commonly used pooling operation. With max pooling, we define a spatial neighbor-

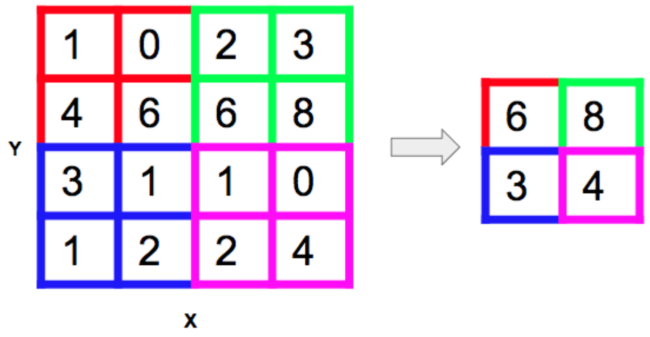


Figure 2.10: The max-pooling operation. Each colour block represents 2x2 window size. Largest element of each block is taken. The windows stride is 2. The operation reduces the dimension from 4x4 to 2x2.

hood (for example, 2x2) and take the largest element from the rectified feature map within that neighborhood. Figure 2.10 shows an example of max pooling operation on a rectified feature map (obtained after convolution + ReLU operation) by using a

2x2 neighborhood. There are some hyperparameters of pooling layer such as *maxpool window size* and *stride* (just like convolution steps).

Fully Connected Layer

The fully connected layer is a traditional multi-layer perceptron where all the neurons from the previous layer are connected to the all the neurons of the next layer. This is mostly used for output classification.

2.3 VGG and ImageNet

CNNs have recently gained popularity in the large-scale image and video recognition [33, 50], thanks to the *ImageNet Large Scale Visual Recognition Challenge (ILSVRC)*, which is held every year [7]. ImageNet is an image database made available to researchers around the world for free access [18]. The database contains millions of images organized into a hierarchy of nodes, and each node is represented by thousands of images [18]. Successful training of a deep CNN using this dataset makes the large-scale image and video recognition possible. Recently, ILSVRC has become a benchmark for new and improved architectures for object recognition.

The CNN architecture we are using is called very deep convolutional neural networks [50] or VGG. Depth-wise, it has several convolutional layers in the overall network design. Details of this architecture can be found in the original paper [50]. The reason for choosing the VGG architecture for this research will be explained in the literature review chapter. The architecture is developed by Visual Geometry Group at University of Oxford. We have already seen that deep CNNs are very good at getting a high-level overview of images, and therefore good for object recognition [20]. VGG was developed on the basis of ImageNet Challenge 2014 [50]. The VGG architecture secured the second position in ILSVRC 2014 classification test which confirms that this architecture is very good at large scale object recognition [50].

There are 5 configurations of the VGG architecture labelled A through E. The 19 layer “E” configuration’s accuracy is higher than any other configurations due to the fact that this is the deepest architecture. It has a total 16 convolution layers and 3 fully connected layers. See Table 2.1 for the arrangement of the layers.

Table 2.1: The convolutional layer parameters are denoted as "conv(receptive field size)-(number of channels)" [50]. FC-(number of channels) stands for fully connected layer. In system labels small "r" means ReLU layer and "p" means pooling layer.

19 weight layers	Our System Labels
input (224 x 224 RGB)	input (224 x 224 RGB)
conv3-64	r11
conv3-64	r12
maxpool	p1
conv3-128	r21
conv3-128	r22
maxpool	p2
conv3-256	r31
conv3-256	r32
conv3-256	r33
conv3-256	r34
maxpool	p3
conv3-512	r41
conv3-512	r42
conv3-512	r43
conv3-512	r44
maxpool	p4
conv3-512	r51
conv3-512	r52
conv3-512	r53
conv3-512	r54
maxpool	p5
FC-4096	
FC-4096	
FC-1000	
soft-max	

2.3.1 Training

Details of the training process of the VGG network is beyond the scope of this thesis, and we are not going to perform any training. We will use the downloaded pre-trained VGG model. Nonetheless, it is worth mentioning some of the key points regarding the training method of this popular architecture. In the VGG architecture, training was performed by optimizing multinomial logistic regression objectives. The architecture used mini-batch gradient descent with momentum [50]. The batch size was 256 and moment was 0.9. The training was regularized by weight decay. Dropout regularization was set for the first two fully connected network. The learning rate was at first was 10^{-2}, and then decreased by factor of 10 when the validation set accuracy stopped improving [50]. The initial process of the weight initialization was done in two steps to rectify the problem of bad weight initialization. They trained the same architecture with random weight initialization which was small enough to training without any learning problems. For the training of the deeper architecture, they initialized the first four convolution layers and last three fully connected layers from the small networked initialized earlier.

2.4 Genetic Programming

Genetic programming (GP) is an evolutionary computation technique that automatically tries solves problems without requiring the user to know or specify the exact form or the structure of the solution in advance [32]. At the most abstract level, GP is a systematic, domain-independent method for solving problems automatically starting from a high-level statement of what needs to be done [43]. GP works by evolving populations of computer programs. This evolution is inspired by the Darwinian principle. It also requires a genetic recombination (crossover) operation appropriate for mating computer programs. After many generations, a succesfull GP run stochastically converts the population of programs/expressions into a new, fitter population of programs. A computer program that solves (or approximately solves) a given problem may emerge from the iterative combination of Darwinian natural selection and genetic operations [32]. Figure 2.11 is a broad overview of how GP operates.

One of the most fundamental aspects of GP is its randomness. This randomness helps bypass traps which tradition deterministic algorithms might fall into, as it helps GP to explore the problem's search space. GP is also very good at exploring the search space of the problem where the structure of the solution is not well defined.

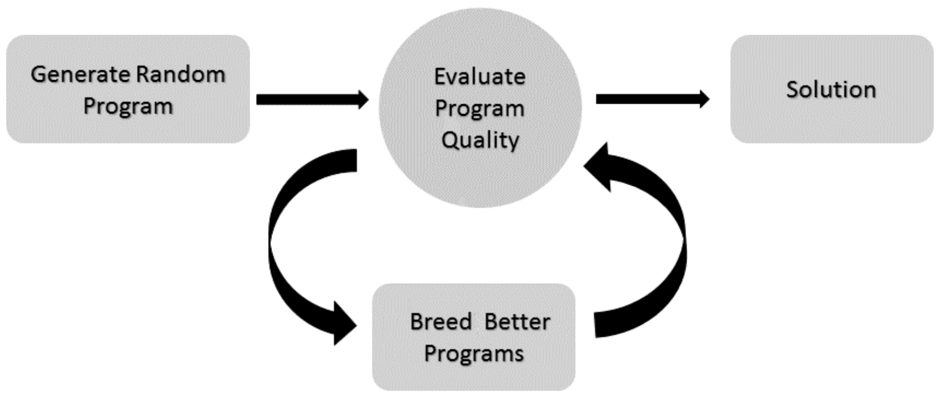


Figure 2.11: Control flow of genetic programming.

GP achieves this by leveraging the random generation of populations, applying genetic operators, and evaluating individuals fitness. This can help solve optimization, regression, and approximation problems. These attributes give GP power to explore the search space for an acceptable solution. Ideally, finding a global optimal solution should terminate the GP search. But in real life problems, often termination occurs when an acceptable solution has been found or after the time limit has been reached.

2.4.1 Representation and Initialization

The most common representation for GP expression is a tree, constructed from functions at the internal nodes, and constants and variables at the leaves [44]. But there are other representations as well. These expressions are treated as individual programs. Figure 2.12 depicts a expression/program representation. The initial population of these individuals is randomly generated at the start of the run. After that a series of steps are performed. Algorithm 1 shows the typical GP procedure.

2.4.2 Fitness

Fitness is the measurement of the quality of individuals with respect to solving some given problem. Fitness can be measured in many different ways. A common fitness measurement is the amount of error between a candidate program's output and the desired output. Since evolved individuals in GP are tree-based programs, fitness eval-

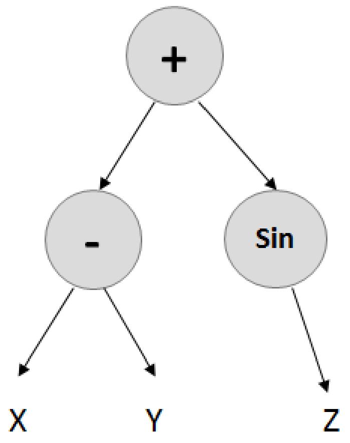


Figure 2.12: GP tree representation of the expression $(x - y) + \sin(z)$

uation normally involves executing a program on training cases for a given problem and evaluating the result during the execution.

2.4.3 Selection

Reproduction is applied to individuals based on their fitness. This means better individuals are preferred over weaker ones (Darwinian "survival of the fittest"). One method for selecting individuals in GP is called tournament selection [26]. In tournament selection, a number of GP individuals are chosen at random from the population. Their fitness is measured and the best of them is chosen to be the parent. Note that tournament selection only looks at which individual is better, and not how much it is better.

2.4.4 Reproduction Operators: Crossover and Mutation

GP reproduction is done with the crossover and mutation operators. Crossover is performed by selecting two individuals as parents. A random node in the tree of each parent is chosen. This is called the crossover point. The crossover points can be any node, for example, it can be the root, internal or terminal nodes. Crossover happens by swapping the entire subtree rooted at the crossover point of one parent tree with that of the other parent tree (see Figure 2.13a).

With mutation, one individual is selected. Mutation is performed by randomly selecting a node in a tree, and replacing the subtree rooted there with a randomly

Algorithm 1: Genetic programming

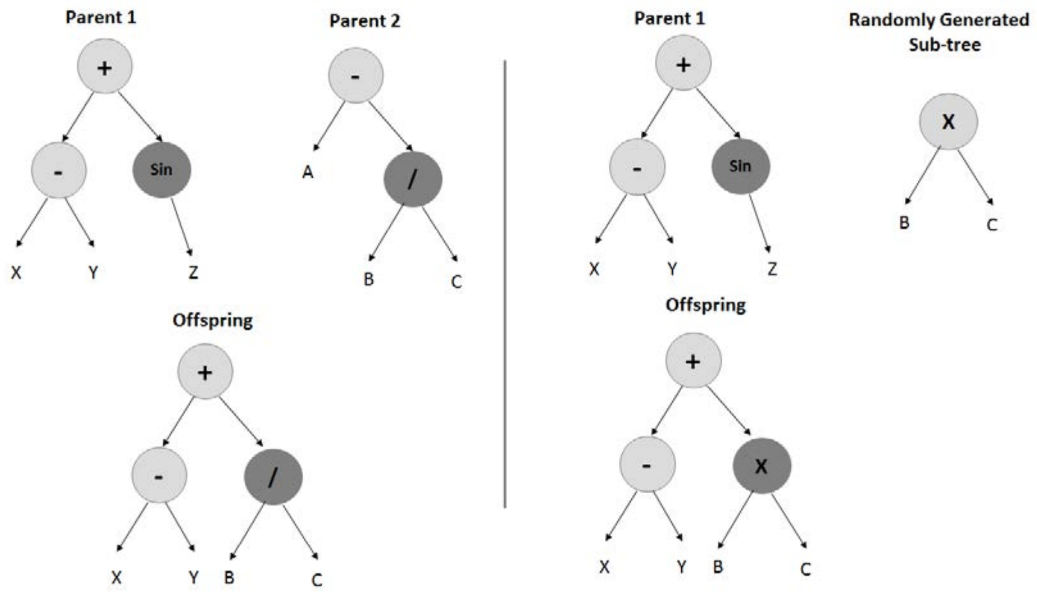
```
1: Randomly create an initial population of program/expression.
2: repeat
3:     repeat
4:         Execute each program from the population and evaluate its fitness.
5:         Select programs from the population with a probability
           based on the fitness to participate in genetic operations.      ▷ Selection
6:         Create new individual by applying genetic operations
           with appropriate probabilities.                                 ▷ Reproduction
7:     until the new population is completely filled
8: until an acceptable solution is found or stopping criterion is met.
9: return best individual program found so far.
```

generated subtree (Figure 2.13b).

Usually, crossover and mutation happen after the selection process described earlier. In "steady state GP", weaker individuals of the population are replaced with new individuals created after crossover and mutation. In the generational GP, a portion of the population is selected and crossover and mutation are performed. Then the resulting offspring are inserted into the population, thus replacing the old individuals. The evolution process is repeated until the stopping criterion is met. In our case the stopping criterion is the number of generations.

2.5 Procedural Textures

A procedural texture is an algorithmic way to represent textures or images instead of directly storing pixel colours. Procedural techniques are algorithmic representations of characteristics of a model, an effect or a texture rather than the actual image bitmaps. For example, a procedural texture for marble surface defines the colour values using an algorithm and mathematical functions [21]. One of the unique features of procedural textures is the abstraction. Rather than directly showing and storing entire image, the procedural texture abstracts them into formulae or algorithms and evaluate the procedure when needed. This saves a lot of storage because, instead of storing the actual pixels, the procedural texture will generate the pixels on the fly. This way textures are not restricted with a fixed resolution since they can evaluate to any resolution when needed. A procedural texture representation is not bounded by any fixed area as well. The textures can be seamlessly repeated to an unlimited extent.



(a) Crossover. (b) Mutation

Figure 2.13: Reproduction operators: crossover and mutation

The power of parametric control is another advantage of procedural texture. This allows assigning parameters to a specific characteristic of a texture. For example, in case of generating cloudy texture, we can control the amount of clouds by just changing the cloud parameter.

There are some downsides of using procedural textures. Evaluation of procedural textures is computationally intensive and potentially slow.

2.5.1 Noise

Noise is a complete procedural texture generation language which is stochastic in nature. Ken Perlin first proposed Perlin noise in 1985 [42]. Due to the stochastic nature of the Perlin noise language, it is mostly used for generating irregularities in procedural textures. Noise is inherently “stochastic” meaning there is pseudo-randomness involved in the process. This breaks the repeated monotonous pattern of some procedural textures. Some of the characteristics of the noise functions are: a repeatable pseudorandom function of its inputs, not exhibiting a regular pattern, translation and rotation invariance and a known range. They can reproduce many natural phenomenon e.g. clouds, minerals, etc.

Although there is randomness involved, the randomness is incorporated in a controlled way [21]. Noise-based procedural textures need to be considered as aestheti-

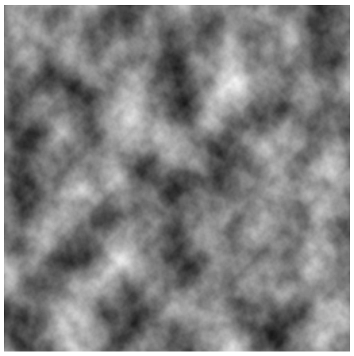


Figure 2.14: Simple Perlin noise

cally pleasing as well. Most of the noise languages we are interested in are lattice-based noise with gradient, such as Perlin noise. To apply randomness in a controlled way, it uses a lattice of a uniformly distributed pseudorandom numbers at each point of the textured surface which are integers. Perlin noise uses interpolation based on the gradient at the eight corners of a single lattice cell to generate smooth noise [21].

Since procedural texture generation is purely algorithmic and uses functions to represent textures, it is quite suitable to incorporate GP to produce textures. We know that procedural textures are parametric. Using GP to evolve these parameters for texture generation is quite intuitive. We use a GP system with a robust language, to evolve textures as part of our image generation process (see Section 4).

2.6 Evolutionary Art

Evolutionary art is characterized by the use of evolutionary computation to generate artistic artifacts [44]. Typically the process can be further broken down into two fundamental ideas: the process of image creation, and evaluating the created artifacts to measure their "quality". In the field of evolutionary art, evolution is used as a tool for exploring the search space for the artistic artifacts [12]. Section 3.2 describes evolutionary art in more detail.

Chapter 3

Literature Review

3.1 Deep CNN in Artistic Image Generation

The application of deep learning in the field of art is very significant and there has been a lot of research using deep CNN in art generation [22, 23, 24, 35, 36]. The first successful attempt to use deep learning in the field of art was by Gatys *et al.* [22]. Our thesis is inspired and partly based on their work. An introductory description of their research is presented in the following section.

3.1.1 Image Style Transfer Using Convolutional Neural Networks

Gatys *et al.* made an early attempt to separate and recombine the content of an image and the specific artistic style using deep CNN [22]. The main goal is to render the semantic content of an image in a different style. The style is captured from another artistic image and is blended in the content image. Previous approaches lacked image representations that explicitly represented high level information of any given image. Because of that, it was hard to separate the content of an image from the style. In other words, previously there was no novel way to get the high-level content representation from the low-level pixels of an image. Gatys *et al.* use the image representations derived from the deep CNN trained for object recognition, which made the high-level image information explicit. They used the pretrained VGG-19 network [50]. Their overall architecture has three distinct segments:

1. Content Representation

Each layer in the network defines a non-linear filter bank whose complexity increases with the position of the layer in the network. Given input image $\overline{x}$ is encoded in each layer of the deep CNN by the filter responses to that image. A layer with N_l distinct filters has N_l feature maps each of size M_l , where M_l is the height times the width of the feature map. Responses in a layer l can be stored in a matrix $F_l \in R^{N_l \times M_l}$, where F^l_{ij} is the activation of the i^{th} filter at position j in layer l. By performing gradient descent on a white noise image on different layers, one can find another image that matches the feature responses of the original image. Let $\overrightarrow{p}$ and$\overrightarrow{x}$ be the original image and the image that is generated, and P^l and F^l their respective feature representation in layer l. The squared-error loss between the two feature representations as defined in [22] is:

$$\mathcal{L}_{content}(\overrightarrow{p}, \overrightarrow{x}, l) = \frac{1}{2}\sum(F^l_{ij} - P^l_{ij})^2$$

They compute the derivative of that loss with respect to the activation in one specific layer and the gradient with respect to the image $\overrightarrow{x}$ was computed using standard error back propagation algorithm (see the right side of Figure 3.1).

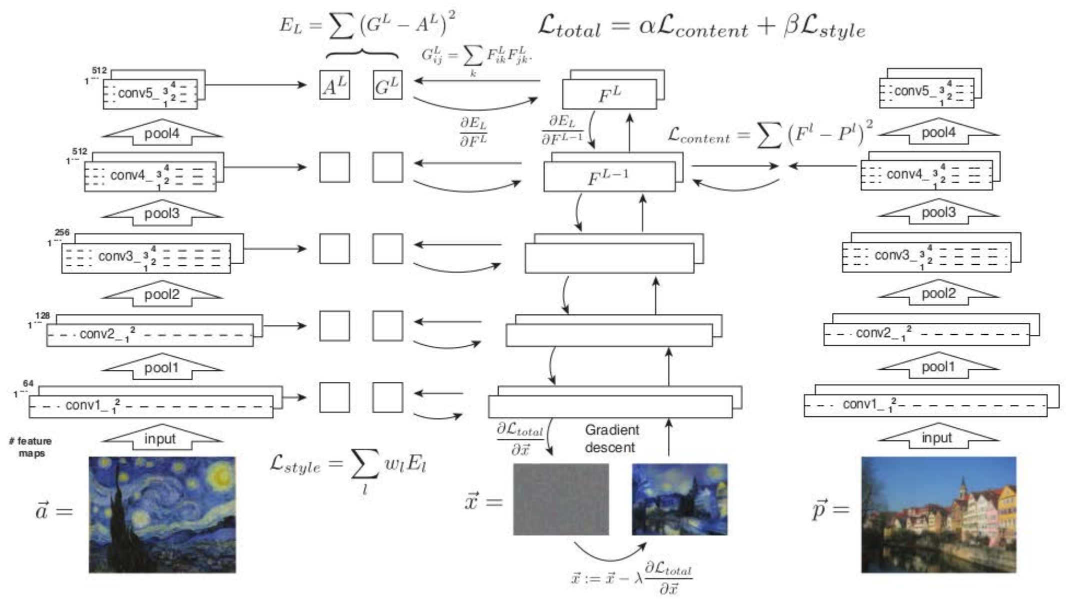
2. Style Representation

To capture the style of an input image, they use a feature space designed to capture texture information [23]. It consists of the correlations between the different filter responses, where the expectation was taken over the spatial extent of the feature maps. These feature correlations are given by the Gram matrix $G^l \in R^{N_l \times N_l}$, where G^l_{ij} is the inner product between the vectorised feature maps i and j in layer l:

$$G^l_{ij} = \sum_k (F^l_{ik} F^l_{jk})$$

By including the feature correlations of multiple layers, they obtain a stationary, multi-scale representation of the input image, which captures its texture information but not the global arrangement (see the left side of Figure 3.1). Let $\overrightarrow{a}$and $\overrightarrow{x}$ be the original image and the image that is generated, and A^l and G^l their respective style representation in layer l. The contribution of layer l to the total loss is:

$$E_l = \frac{1}{4N_l^2 M_l^2}\sum_{i,j}(G^l_{ij} - A^l_{ij})^2$$



$E_L = \sum (G^L - A^L)^2$
$\mathcal{L}_{total} = \alpha\mathcal{L}_{content} + \beta\mathcal{L}_{style}$
$G^L_{ij} = \sum_k F^L_{ik}F^L_{jk}$
$\mathcal{L}_{content} = \sum (F^l - P^l)^2$
conv5_1 conv4_1 conv3_1 conv2_1 conv1_1
pool4 pool3 pool2 pool1
input
# feature maps
Gradient descent
$\mathcal{L}_{style} = \sum_l w_l E_l$
$\vec{x} := \vec{x} - \lambda\frac{\partial\mathcal{L}_{total}}{\partial\vec{x}}$

Figure 3.1: Style transfer algorithm. Left side depicts style representation and right side depicts content representation (image source [22]).

and the total style loss is

$$\mathcal{L}_{style}(\overrightarrow{a}, \overrightarrow{x}) = \sum_{l=0}^{L} w_l E_l$$

Here w_l are the weighing factors of the contribution of each layer to the total loss.

3. Style Transfer:

Transferring the style of an artwork $\overrightarrow{a}$ onto a photograph $\overrightarrow{p}$ requires synthesizing a new image that simultaneously matches the content representation of $\overrightarrow{p}$ and the style representation of $\overrightarrow{a}$. For that they jointly minimize the distance of the feature representations of a white noise image from the content representation of the photograph in one layer and the style representation of the painting defined on a number of layers of the ConvNets. The loss function they minimise is:

$$\mathcal{L}_{total}(\overrightarrow{p}, \overrightarrow{a}, \overrightarrow{x}) = \alpha\mathcal{L}_{content} + \beta\mathcal{L}_{style}(\overrightarrow{a}, \overrightarrow{x})$$

The α and β are the weighting factors for the content and style reconstruction.

It is evident that the content high-level content representation is accurate in representing the content of the given target image. In our thesis, we are interested to use the content representation part and ignore the style capture and transfer part since we will not be performing style transfer. In fact, our architecture design (Figure 4.1) is almost identical to the content representation part of [23]. We will utilize the high-level content representation part as an intelligent guide to our evolutionary process. Details of the overall system design are discussed in Chapter 4.

3.2 Evolutionary Art

The first application of evolutionary algorithms to create shapes or designs was proposed by Richard Dawkins, who evolved 2d shapes called *biomorphs* [17]. After this, Karl Sims successfully applied evolutionary techniques and computer graphics to create artistic images with procedural textures [51]. His expression-based image generation approach produced complex 2D artifacts. He also demonstrated the idea of evolving 3D geometry in some of his work [52, 51]. Many others have since explored the technology. For example, Todd and Lantham applied the biomorph concepts to 3D models [55]. Most of the earlier evolutionary art were interactive i.e they used interactive evolutionary algorithms, where a user decides the fitness of a population and assign scores manually [29]. Some attempted making that collaborative interactive evolution using online communities [49].

Later, automatic image generation was a focus of evolutionary art research. It is important to mention that automatic image generation is the primary focus of our thesis as well. Ibrahim made some of the earliest attempts in his *Genshade* system [31]. The system attempted to replicate image characteristics of a target image by performing wavelet analysis. Wiens and Ross's *Gentropy* [57] used an unsupervised approach, where a target texture image represents the desired texture features, such as colour, shape, and smoothness (contrast). Then the system evolves textures without any user interaction. Gircys used power spectral density of spatial frequencies based on Fourier decomposition to guide the evolutionary texture synthesis [25]. NEvAr system [40] by Machado *et al.* leveraged the parallel evolutionary computations technique where an individual user could independently evaluate certain images from any of the parallel evolution and add them to another specific parallel evolutionary process. It was not fully interactive evolution but it required user intervention.

Recently efforts are attempting to evolve aesthetically pleasing images. Machado and Cardoso propose an aesthetic measurement process which is built on the relation between image complexity and processing complexity [39]. Ross *et al.* propose another measurement of aesthetics based on a bell curve distribution of color gradients [46]. The concept of using the evolution of neural networks to generate images is also used by some systems. Stanley and Miikkulainen developed the NEAT system and employed that system for targeted evolution by interactively evolving networks which gradually refine a spaceship design [6]. Beluja *et al.* use artificial neural nets to learn the user's aesthetic preferences [9]. Subsequently they used this knowledge to evolve aesthetically pleasing images similar to those evolved by the interactive sessions with the user.

3.3 Deep Learning and Evolutionary Art

Both deep learning and evolutionary techniques in image generation were done in the past. Nguyen *et al.* [41] use an evolutionary algorithm to generate images which were passed through to deep neural networks. Those images achieved high prediction score from the DNN. Their primary goal is to fool the DNN employing an evolutionary technique to produce images. Evolutionary art was not their primary goal but the high confidence images generated by the evolutionary algorithm have somewhat abstract characteristics of the specific target object. Which might be considered as abstract art.

Another approach was done by Bontrager *et al.* that combines generative adversarial networks (GANs) with interactive evolutionary computation [13]. They show a GAN trained on a specific target domain can act as an efficient genotype-to-phenotype mapping. A trained GAN, the vector given as input to the GAN's generator network can be controlled using evolutionary techniques, allowing high-quality image generation. Participants were able to evolve images that strongly resemble specific target images. Although GANs are good at generating candidates especially images, there are some key differences between a GAN and an evolution based system. A GAN usually competes with a similar discriminator network and those two networks are bound to a specific domain or data. GAN always generates candidates (generative) and the discriminator evaluates them (discriminative). But in the case of evolutionary art, it is somewhat less bound to the domain/data. As long as the fitness criteria are able the guide the evolutionary system, it can generate candidates independently.

Chapter 4

System Design

This chapter describes the design of the systems involved and provides necessary diagrams for understanding the overall architecture. We also describe necessary tools, languages, and frameworks used to implement the systems.

The overall architecture of the systems can be divided into two distinct subsystems or modules:

1. A robust genetic programming based texture generation module.

2. A pre-trained deep convolutional neural network module.

Figure 4.1 depicts the system architecture. The control flow of overall architecture is discussed in the following section.

4.1 Control Flow of the Overall Architecture

Since we have illustrated the system architecture in Figure 4.1, we now define the overall control flow of the architecture in Algorithm 2:

Algorithm 2: Control Flow of The Overall Architecture

```
1:  select any target content image and preprocess the image.
2:  pass the image through the VGG-19 network until it reaches the r51 layer
3:  save the r51 layer in the memory for later calculation
4:  initiate the GP with proper configuration
5:  repeat
6:      pass each texture image generated by the GP through the same
        VGG-19 network until it reaches r51 layer
7:      calculate the fitness by performing MSE between the r51 layer of the GP
        generated image with the content layer saved in Step 3
8:      return the fitness to the GP
9:  until the maximum number of generations is reached
10: return the best image found so far as a solution
```

Following sections provide more detailed information about each module and explains some of the terminology mentioned in Algorithm 2.

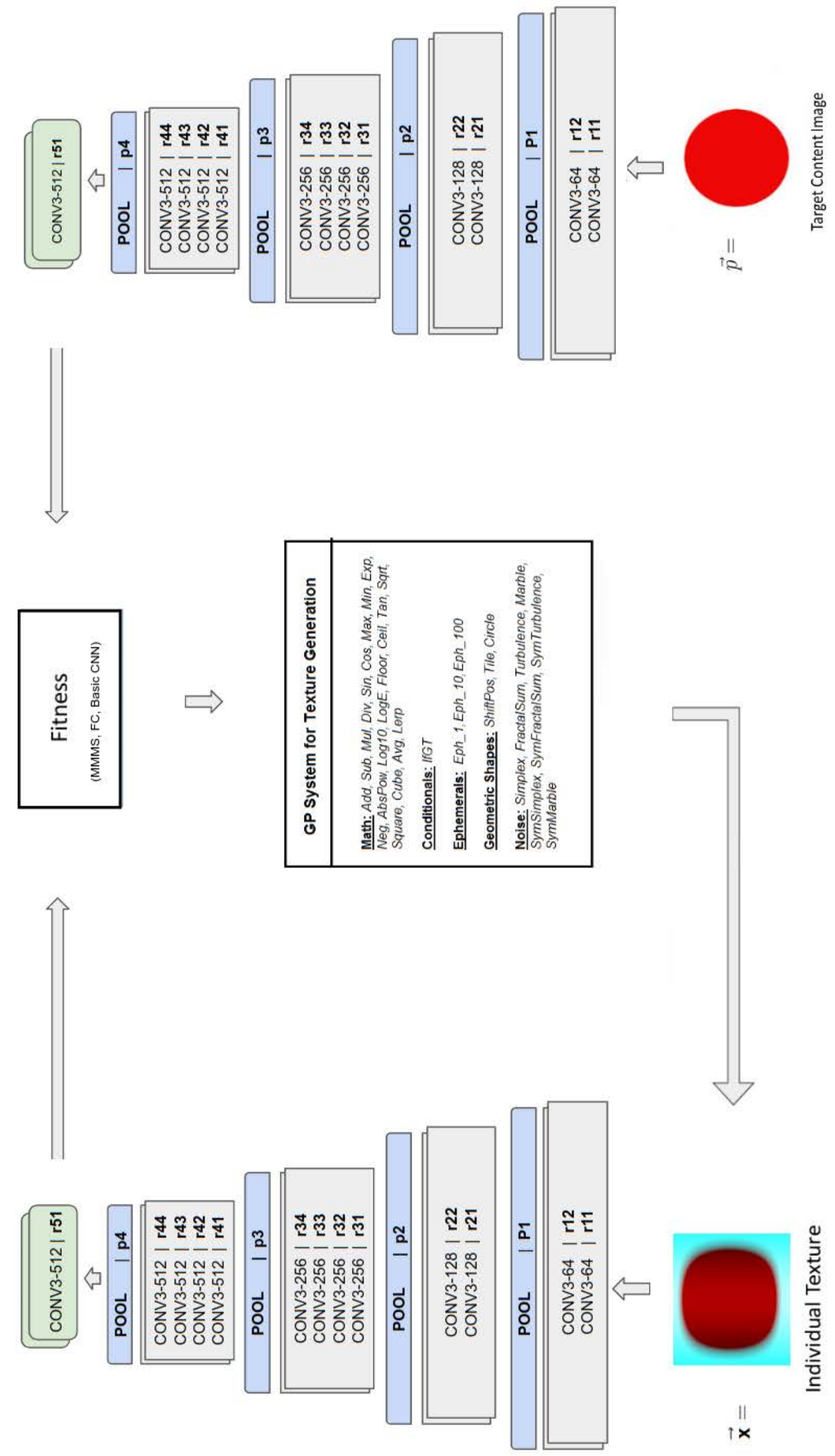


Figure 4.1: Overall architecture design

4.2 Genetic Programming

For the GP system, we are using a GP texture language consisting of a wide range of math, conditional and noise functions to evolve textures/images. The GP system is implemented in ECJ (version 23), an evolutionary computation system written in Java [38]. It is designed to be highly flexible, with nearly all classes and their settings dynamically determined at runtime by a user-provided parameter file. The GP system is solely responsible for the generation and the evolution of the texture images. We will transfer those evolved texture images through an interface to the deep CNN system. The complete description of the GP texture language is described in detail in [25]. We are simply leveraging the GP system of [25] as our principal GP engine.

All the GP parameters and texture language description are discussed in Section 5.1. Except for the maximum number of generations, all the GP parameters are kept the same throughout all experiments. The GP system, along with the entire process of evolving texture images, is completely decoupled and independent from the trained deep CNN module.

4.3 Deep Convolutional Neural Network

This module is inspired by the work by Gatys *et al.* [22]. The deep CNN model we use is VGG-19 [50], which was trained to perform object recognition and localization. We will use the feature space provided by a normalized version of the 16 convolutional and 5 pooling layers of the 19-weight layers VGG network [50]. The most important aspect of this module is that we will not be performing any sort of training. The model is pretrained using the ILSVRC 2012 dataset, and we are simply using this pre-trained model. The reason for choosing this pre-trained model is that it takes too long and too much effort to train one ourselves. This pre-trained model already achieved very good performance on the ImageNet dataset. This model was developed by the Visual Geometry Group at the University of Oxford. They develop the architecture for the submission of Imagenet challenge 2014. This model was significantly more accurate ConvNet architecture compared to the other previous architectures [50] at that time. There are several configurations of the VGG architecture introduced in [50], labeled from A to E. We will be using the configuration **E**. Because this is the deepest architecture compared to the other configurations and [22] also used this configuration. Configuration **E** contains the most significant number of layers. It has

16 weight convolution layers and 3 weight fully connected layers totaling about 19 weight layers.

The input layer of the VGG architecture takes a fixed size 224x224 three channel (RGB) image in the original architecture. However, we will be using a 256x256 RGB image as an input. Initially, the image goes through a preprocessing stage where we subtract the mean RGB value calculated on the training image set from each pixel. This pre-processed image goes through a series of 16 convolution layers. The filters of this architecture use a very small 3x3 receptive field which is sufficient to encode the idea of left/right/up/down [50]. The convolution stride is 1 and padding is 1. Although the original VGG architecture used max pooling in their pooling layer, we change it to average pooling for better results [22]. Originally the training was done using max pooling operation. We will also change the pooling window size and stride. Details of these changes and their motivation behind them will be discussed in Section 4.3.

Apart from that, all the hidden layers used ReLU to introduce non-linearity [33]. The width of convolutional layers starts from 64 in the first layer and increases by the factor of 2 after each pooling layer until it reaches the width of 512. The total number of the weights of this architecture (configuration **E**) is approximately 144 million.

4.3.1 Fitness

The deep CNN will be responsible for calculating the fitness of the evolved texture images. According to [22], the high-level content information is captured at the higher layers in the deep CNN. The layer we are interested in is the one after the fourth pooling layer (see Figure 4.1). We will use this layer (r51) as the high-level content image representation of the given target image. For all our experiments we will use this layer for our fitness calculation. The motivation for selecting this layer and the detailed fitness calculation will be discussed in detail in Chapter 5.

4.4 Software and Hardware Details

The deep learning framework we will use for the implementation of the VGG-19 is PyTorch, which is based on Python [15]. PyTorch is an optimized tensor library for deep learning using GPUs and CPUs [14]. For all our experiments we used the PyTorch version 0.2.0. PyTorch is easily configurable to run on a CUDA based PC system. We performed all our experiments on a NVIDIA TITAN Xp (Compute

Capability 6.1) with 12 GB memory. Our CUDA version is 9.0.176. The GP system (ECJ) runs on 2.80 GHz Intel(R) Core(TM) i7 CPU with 6GB memory. Our OS version is Ubuntu 16.04.3. All the experiments performed with the NVIDIA TITAN Xp are approximately 5 times faster than the CPU-based experiments.

Chapter 5

Experiment Details

This chapter discusses the fundamental approach to achieve our initial goal i.e. getting GP to evolve images having characteristics found in a given target image using the deep CNN as a guidance. This concept of using the trained deep CNN model's high-level content representation as a guide to our evolutionary process involves many different and independent ideas. Deep CNN's content representation and GP's image generation process both have their own ways of dealing with images. For example, in almost all art applications, a deep CNN uses the gradient descent algorithm to change individual pixels on an image to optimize the cost function. These changes are on granular pixel level. On the other hand, GP's procedural texture language tends to produce a texture formula, and the resultant texture image is evaluated against some target criteria to find its fitness. Every evolved texture formula denotes an independent image. Rather than editing individual pixels like a deep CNN (see Gatys *et al.* [22]), the formula is altered during reproduction. This is a much higher level alteration than deep CNN's granular pixel alteration. We will expand on this topic later.

This fundamental difference is one of the intricacies in the overall integration of GP and trained deep CNN. Our experiments primarily deal with examining different approaches to rectify this problem. We will start from the very beginning from a naive approach and will later discuss a more advanced approach that yielded more satisfactory results.

5.1 General Setup

The primary GP setup we use is from Gircys [25] and includes the overall GP engine, texture language, and parameters. We slightly extended the texture language by adding more noise functions to the language. Table 5.1 shows all the GP parameters configured for our experiments. Table 5.2 shows a complete overview of our texture language. The GP parameters will be the same throughout all our experiments. The parameters are based on those in [25]. Note that the population size is smaller than usual, given the higher computation time associated with image generation and CNN analysis.

Parameter	**Value**
Generations	35
Population Size	300
Elitism	1
Tournament Size	3
Crossover Rate	60%
Maximum Crossover Depth	17
Mutation	30%
ERC Mutation	10%
Maximum Mutation Depth	17
Prob. of Terminals in Cross/Mutation	10%
Initialization Method	Ramped Half & Half
Tree Grow Max/Min	[5, 5]
Grow Probability	50%
Tree Half Max/Min	[2,6]

Table 5.1: GP parameters

5.2 Naive Strategy

From Chapter 4, the overall system design lays out the foundation of our architecture. We kept the deep CNN VGG parameters the same as Gatys *et al.* [22]. To aid layer retrieving, Gatys *et al.* conveniently labeled all the layers into proper groups (see Table 2.1 for layer mappings). The deep CNN's target content representation at r42 was used as a fitness to our GP system. This is different from the overall algorithm we formulated before in Chapter 4 which uses r51. The reason we are using r42 instead of r51 is to show how naive approach performs without altering any deep CNN parameters which were initially used in [22]. In the next section we will discuss the justification of using r51. But for now we will use r42. Since we will deal with the r42 content representation, it is important understand how images are represented at this layer. The given input image goes through the network, passes all the layers until it reaches the r42 layer. The reason for choosing this layer was inspired by Gatys *et*

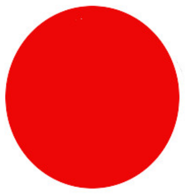


Figure 5.1: Red circle content image

al. This is a high enough layer to capture content information and low enough to keep some resemblance to the content target [22]. In our naive strategy we simply followed Gatys *et al.*'s approach to capture content information of given target image. Details on why Gatys *et al.* selected this layer is explained in the section 3.2 from the original paper by [22]. The r42 layer has total 512 feature maps or activations. Each feature map matrix is a 32x32 matrix. Each element of the 32x32 matrix represents the level of activations. The combined number of parameters are 512x32x32 giving a total of 524,288 different values. If we pass a target input image up to this level, the r42 layer represents the high-level abstract representation of the input target image [22].

Our first idea was to try this final layer directly without altering anything. Initially, we launched the CNN system with the target content target in Figure 5.1. We saved the r42 layer values (512x32x32 matrix) as our target fitness. Then the GP module was initiated with the configuration declared in Section 5.1. Each individual of the GP population is translated to a texture image and passed through all the layers until it reaches the r42 layer. We then compare this layer with the previous content target layer saved earlier. We used the well known mean square error (MSE) between these two layers. Let P and F be the content image and the evolved gp image feature representation at layer r42. If the number of elements in the r42 layer is n(512x32x32), then the mean squared-error loss between the two layer will be:

$$Fitness = \frac{1}{n}\sum(F_n - P_n)^2$$

In principle, if the MSE between these two layers is zero, we will get the same content image produced by the GP. This, however, is not our goal, as we are not interested in synthesising the target image. Rather we want to use GP's exploratory nature to create art while conforming to the characteristics of the target content image. We already had an intuition that this approach might be unsuitable for getting desired results, because the total final layer in its entirety is too large and difficult for GP to handle. The huge matrix of size 512x32x32, totalling about 524,288, has a wide range of values. Depending on the target image some activations are highly activated while others have minimum values (out of 512 feature maps). To give an example of the diversity of the r42 layer, we calculate the mean value for each of the 32x32 feature maps and plot them in Figure 5.2.

Category	Arity	Display	Description
Variables	0	X	Horizontal rendering coordinate
		Y	Vertical rendering coordinate
		Rho	Polar coordinate; distance from {0, 0}
		Phi	Polar coordinate; angle about {0, 0} to X axis
Ephemerals	0	E[1]	Ephemeral in range [0, 1]
		E[10]	Ephemeral in range [0, 10]
		E[100]	Ephemeral in range [0, 100]
Math	1	-	Negation
		abs	Absolute value
		floor	Floor
		ceil	Ceiling
		sin	Trigonometric sine
		cos	Trigonometric cosine
		tan	Trigonometric tangent
		sqrt	Square root
		exp	e (Euler's number) raised to the operand
		pow2	The operand raised to the power 2
		pow3	The operand raised to the power 3
		log E	Natural log
		log 10	Log, base 10
	2	+	Addition
		-	Subtraction
		*	Multiplication
		/	Safe division
		max	The greater of two operands
		min	The lesser of two operands
		avg	The mean of two operands
		pow	arg[0] raised to arg[1]
	3	lerp	Linear interpolation between two arguments
Conditionals	4	IfGT	If greater than conditional
Noise	0	Simplex	Simplex noise generator
		Perlin	Perlin noise generator
		Marble	Marble noise
	1	FractalSum	Fractal Sum/Smooth noise
		Turbulence	Turbulence noise
	3	Cloud	Cloud noise effect

Table 5.2: GP texture language overview

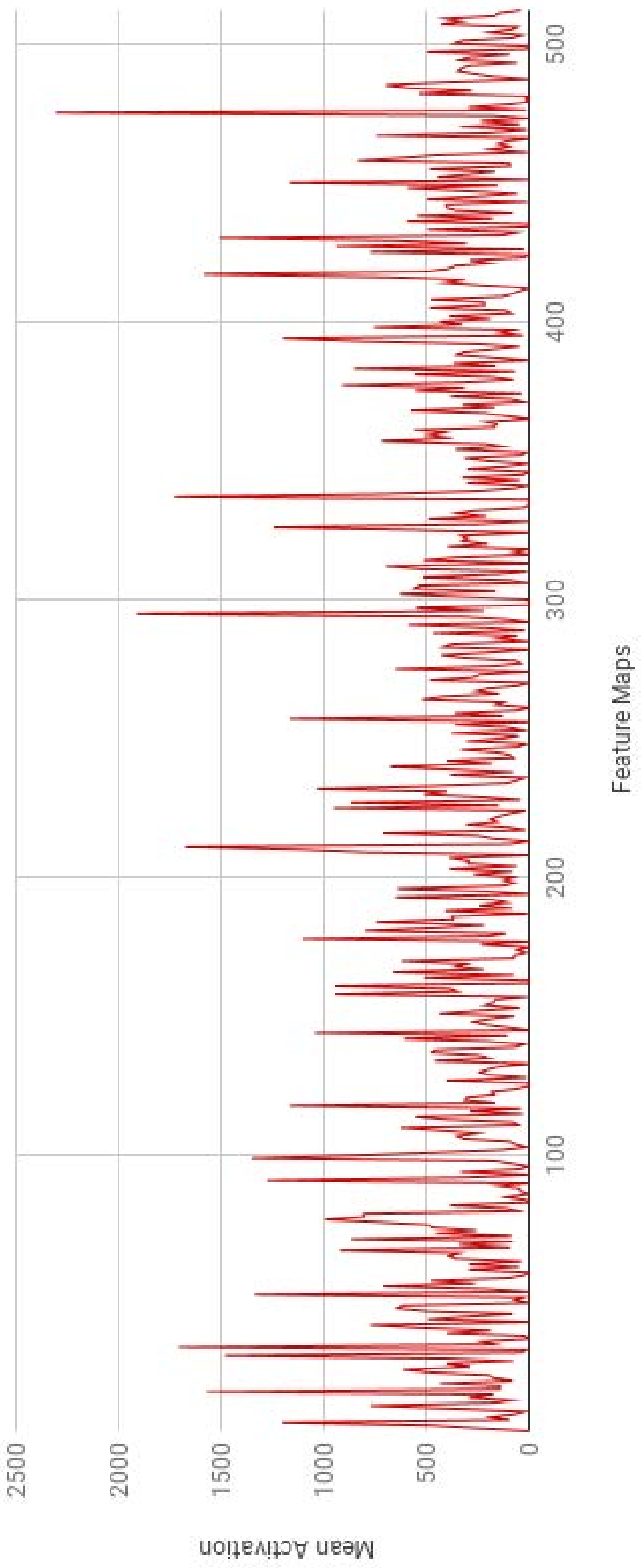


Figure 5.2: Mean activations for the target circle image. Notice the wide range of values.

After running GP for 35 generations with the configuration described in Section 5.2, we obtained a negative result as expected. Figure 5.3 illustrates some results from the final generation. These are randomly picked from the final generation since all the individuals from the final generation are almost identical. In order to minimize the MSE between the activation of content target and GP texture, GP produces an average activation of all the feature maps. The fitness quickly converged and the result was an almost plain white image with slight greyish tone.

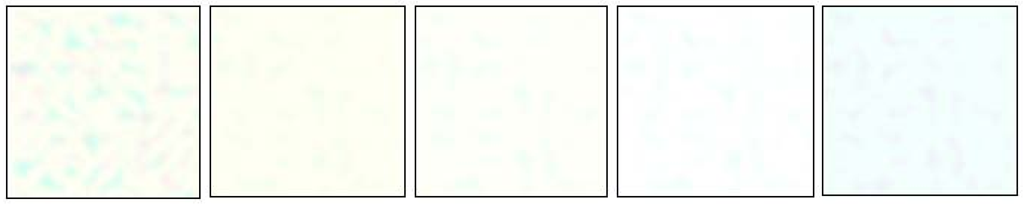


Figure 5.3: Some results from the final generation. All the images from the final generation are almost white.

5.3 Dimensionality Reduction

After our initial trial with naive approach, we started investigating the final layer matrix which was used as our target fitness. As we stated before, this final layer matrix serves quite well when it is used with a gradient descent algorithm. The error between the target matrix and a given input image flows all the way back to the input layer. Starting from white noise, the algorithm can change individual pixels to minimize MSE between input and target . This process is iteratively continued until the MSE is greatly minimized. But in case of GP, the problem is not just the large number of values to minimize in MSE but that GP can only change the high-level tree (formula) and cannot edit low-level pixels like a deep CNN.

There are multiple ways to approach this problem. But before moving into those steps we should reiterate some of the concepts of the pooling layers from deep learning once again. In deep CNN, all the pooling layers are used to gradually reduce the spatial size (downsampling) of the feature maps to reduce the number of parameters, thereby reducing the computation of the network. This in turn controls overfitting and introduced scale invariance. We are interested in the reduction of the spatial size. Since this layer progressively down-samples the feature maps, the deeper we go in the network, the smaller the size of the feature maps will become. This can assist us to resolve our problem with the big matrix. By looking at Table 2.1, instead of using the r42 layer, we can go past the p5 (pooling layer) and use the r51 or r54. If we select any layer after p5, the feature map size will be reduced due to the pooling operation. But we need to be careful because reducing the layer means a loss of content information. To balance this we settled on r51. This reduced the size of the target layer matrix from 512x32x32 to 512x16x16. In other words, 512 feature maps were previously 32x32 but now 16x16. The total number of values was reduced from 524,288 to 131,072.

Another key factor to reduce the feature map size is the window size of the pooling operation. The more we increase the window size and the stride, the smaller the feature maps will become, although with a loss of information. See Figure 2.10 and Section 2.2.1 for details of the operation. After increasing the windows size from 2 to 3 and stride from 2 to 3, final layer matrix reduced from 512x16x16 to 512x3x3. One important fact to remember is that during the initial trial we used max pooling as our pooling function. According to Gatys *et al.* [22] instead of using max pooling, average pooling yields better results. To illustrate this, consider Figure 5.4. We took the red circle target image. We saved the r51 layer as our target image layer. We passed a white image and ran the gradient descent algorithm to lower the MSE between the white image and content image. During max pooling, a lot of information about the content or shape of the image is lost. But if we change the pooling operation from max pooling to average pooling, the resultant image is much better. The reason we mentioned this process is that we wanted to make sure that after all the reduction of the target matrix at layer r51, the matrix is still encoding enough useful information about the content of the image. In summary, we changed the content layer from r42 to r51, increased the window size and stride and used average pooling for better content representation. This helped us reduce the content layer matrix from 512x32x32 to

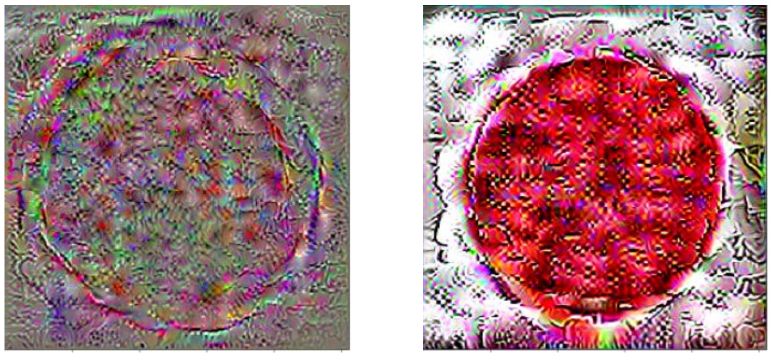

(a) Max Pooled Image (b) Averaged Pooled

Figure 5.4: Max pooling vs average pooling applied to circle image.

512x3x3. This dramatic reduction should not be a major issue when we integrate it with GP, as we are interested in indirectly guiding the evolutionary process. The reduction will actually help GP because this is less computationally expensive than the early naive approach.

5.4 Statistical Analysis and Reducing the Number of Feature Maps

After Section 5.3's dimensionality reduction, next we focus on reducing the number of feature maps. We had 512 feature maps, and each is a 32x32 matrix. We reduced the feature maps from 32x32 to 3x3. Nevertheless, we have 512 of those feature maps to deal with. A reduction of this number is worth undertaking.

Deep neural networks can be thought of as "black boxes". Rigorous training of the massive network makes the inner workings almost impossible to comprehend due to the large number of interacting, non-linear parts. Visualizing neurons is one way of understanding their functionality. Highly activated neurons are the key to understanding the network. Those highly activated neurons, respond to certain properties of the image. Each activation is fired when a certain attribute is present in the target content image. Colour blobs, edges, and specific shapes fire different activations in the hidden layers. In other words, a highly activated neuron represents what that specific filter is sensitive to in an image. If it find that property, it gets highly activated. Deep neural networks use this "strong prior" to classifying the given object. There are a number of papers which investigate this phenomenon in detail, e.g. [58] and [59].

This idea of highly activated neurons inspired us to filter out feature maps. Since for certain features of an image, some neurons get highly activated and others do not, it motivated us to remove inactive feature maps. In order to visualize this, we selected r42 layer for visualization. We preferred this one for visualization because, since the feature maps are of size 32x32, it is easier to scale and convert them into an image that can be viewed (see Figure 5.5). In the r51 layer, the individual feature map dimension is too small (3x3) to properly view the activations.

Since we wish to reduce the number of feature maps, consideration of activation level can aid us. Our first intuition was to utilize feature maps crossing some threshold value: if the mean activation of any feature map is below some small threshold value, we can reject it. We are concerned with the shape or content information of the target

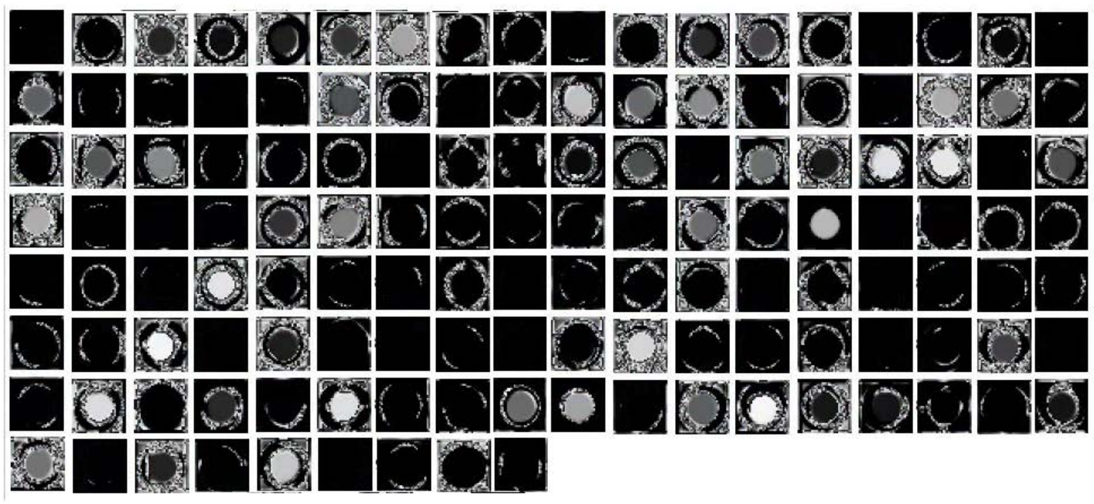
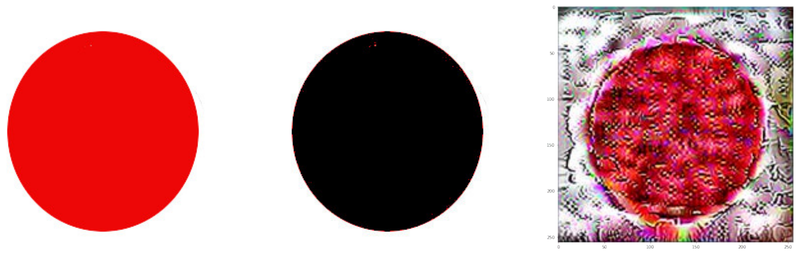


Figure 5.5: Visualization of first 135 feature maps of red circle image in layer r42. Notice how some feature maps are activated to varying levels, while others are completely turned off.

image. Sometimes an activation can have a high value if that particular neuron is looking for color instead of other properties. For example, if we take the red circle as a target image and if there is a highly activated neuron, it is possible that this particular neuron is firing because it is responding to the red color. If we sort the feature maps based on the mean in descending order and select the first K number of activation into consideration, our experiment will work only if the target image properties fire very specific neurons and the activation values are extremely high. For example, vertical and horizontal stripes are straightforward images. In the pool of hundreds of kernels, the line detector kernels will be highly activated. So using a handful of highly activated feature maps will be enough to represent the content of that target image. Our experiments confirmed this observation. However, in cases of circle, triangle, and rectangular images, feature maps properties are less straightforward. Circles have curves while triangles need vertical, positive 45 and negative 45 line detectors. In other words, their representation is more complex than stripes, and they need careful selection of multiple features maps if we want to represent them in an efficient way.

In the following section, we will define a strategy which supports the reduction of the feature maps to use.



(a) Red Circle (b) Black Circle (c) Distorted Circle

Figure 5.6: Similar content images.

5.5 The Mean Minimum Matrix Strategy (MMMS)

The first experiment (Section 5.2) yielded bad results due to the fact that the amount of information was overwhelming for the GP system to produce the desired results. Our goal here is to reduce the number of activation matrices and determine which ones are the most relevant for a particular set of images, i.e., common high activations. The end result is, we identify a small subset of activation matrices to use. As mentioned earlier, we are only interested in those key activations, should they exist that are responsible for capturing the shape of the content image. The strategy that we are going to undertake involves reducing the number of activation matrices down to a small number of activation matrices responsible for retaining the characteristics of object shapes. We will refer to this strategy as the Mean Min Matrix Strategy (MMMS). It is important to mention that MMMS is not a exact algorithm to capture content information since we know that the inner workings of a deep network is hard to formulate. Exactly determining the functionality of activation maps is difficult. The MMMS is simply a heuristic, a statistical preprocessing step in order to heuristically find useful functionality that we can exploit. This preprocessing is performed before running any GP engine with deep CNN.

The overall MMMS steps are listed in Algorithm 3. First, we arrange the set of images of similar objects paired into two groups. For example, for a circle, we take three circle images into account: a red circle, black circle. and a distorted red circle (see Figure 5.6). Those three images are then divided into pairs of images (Figure 5.7).

The intuition behind this selection is that we are trying to arrange the images so that the only thing that is common to these images is their shapes. Next, we pass each image from a pair (e.g. group 1) separately through the deep CNN and extract

Algorithm 3: MMMS algorithm

```
1: select three images with similar content or shapes.
2: pair them into two groups
3: repeat
4:     pass each image separately from the same group through the deep
       CNN.
5:     at layer #r51 take the minimum of two activation matrices of the two images
       respectively from the same group.
6:     calculate the mean of the resultant minimum matrix from the previous step.
7:     store the mean of all the resultant minimum activation matrix in
       descending order along with their activation #.
8: until no groups are left
9: take the top 40 activations from the sorted values from both groups stored in step
   7.
10: divide them into two levels each with 20 activations along with their mean.
11: find common activation # in the same level from both groups.
12: return the list of common activations.
```

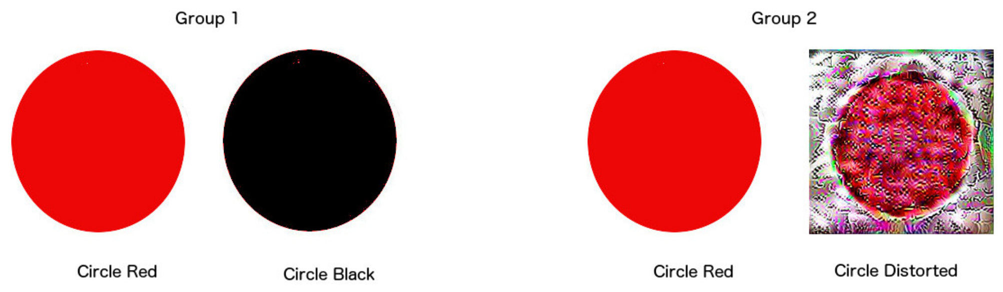


Figure 5.7: Content images grouped into pairs.

the 512 activation matrices from the higher layers. For each image activation, we have two matrices: one for the red circle and another for the black circle (group 1 images). Consider, the activation matrix #38 from both of the images:

$$CirRed: \begin{bmatrix} 43.6896 & 9.7997 & 19.1355 \\ 16.2436 & 47.3683 & 0.0000 \\ 74.1886 & 63.5082 & 38.9489 \end{bmatrix} CirBlack: \begin{bmatrix} 61.8429 & 42.0349 & 39.6617 \\ 28.5306 & 63.1241 & 5.1725 \\ 58.1770 & 64.3592 & 29.1844 \end{bmatrix}$$

Next, we apply the MMMS operation. It compares each corresponding matrix element from the two matrices and produces another matrix with the same dimension which contains the minimum value of the two elements.

$$R_{ij} = min(CirRed_{ij}, CirBlack_{ij}) \ (1 \leq i, j \leq 3)$$

For the above matrices, the resultant matrix looks like this:

$$Min: \begin{bmatrix} 43.68964767 & 9.79972649 & 19.13554955 \\ 16.24355507 & 47.36829376 & 0. \\ 58.17697144 & 63.50819397 & 29.18441772 \end{bmatrix}$$

Next we take the mean of that minimum matrix values and store it. After that, we gather the means of all 512 activations and sort them in descending order based on the mean performed earlier. We do the same with group 2 (red circle and distorted circle). Now we have two lists of activations sorted in descending order by the mean of minimum matrix: one from group 1 and another from group 2. Our goal is to select top activations that are common in both groups. Table 5.3 shows the final sorted values from each group.

The top 40 activations are divided into two levels or batches. The first 20 activations out of those 40 activations are in level 1 and the remaining 20 activations are in level 2. The intuition behind having two levels is to make sure that when we try to match between these 2 groups and find the common activations, the position of the common activation is as close as possible. In other words, an activation that is in level 1 in group 1 should also be in level 1 in group 2. For example, from Table 5.3, activation #369 is in level 1 in both group 1 and group 2. On the other hand, the activation #8 is in level 1 in group 1 but is in level 2 in group 2. So we will not count this as common activation. Similarly, in group 1 the activation #442 is not taken as common activation because in group 1, activation #442 is in level 2. But in group 2 it is in level 1. So this will not be considered as common activation. From the table, we can see we found 12 common activations in level 1 and 7 common activations in level 2 totaling about 19 activations. Those are the following:

$$18, 63, 103, 104, 151, 178, 187, 271, 295, 305, 324, 338, 355, 369, 385, 387, 410, 411, 416$$

So instead of using total 512 feature maps from deep CNN's higher layer, we filtered out most of the feature maps. We will only use this activation maps when calculating MSE between target image and GP image.

Note that both high mean activation values and low mean activation values are important in modelling image features. MMMS helps to find the high and low activations common in both groups (similar images paired into groups). "Uncommon" cases mean these feature maps may not be relevant. Again this is a heuristic that we are trying to expoit. Another important factor to remember that we are not going to use the mean minimum activation matrix as our target fitness when running GP. We are applying the MMMS only to find common activations. Later, when we run the GP engine, we will pass an actual content image through the network and use only the common activations selected earlier by MMMS.

In case of representing the shape characteristics of the circle, the reduced feature maps might be enough. We reduced the initial 512x3x3 target matrix to a 19x3x3 matrix. The deep CNN will only calculate the MSE (mean square error) between those selected feature maps. We can do this by filtering the r51 matrix by keeping only the common activation maps as target. In the following section, we will see the results of GP runs which will give some idea how GP improved after using MMMS strategy.

Level	Red & Black (Group 1)		Red & Distort (Group 2)	
	Activation #	Mean Min Activation	Activation #	Mean Min Activation
Level 1	493	104.137	369	70.0863
	268	101.723	178	63.5016
	63	95.9581	271	60.9847
	385	94.1941	162	51.1977
	178	92.5668	63	50.674
	187	86.8222	187	49.927
	369	78.5026	385	49.7277
	324	75.7872	416	44.2747
	18	67.4085	8	38.6199
	95	64.8462	475	38.604
	416	61.9521	474	38.0422
	355	61.6623	283	36.1778
	103	60.755	346	35.0986
	271	53.4458	103	34.2125
	442	48.5399	18	30.4536
	295	46.1593	324	29.7953
	403	45.7757	439	29.247
	429	43.4773	355	29.0189
	269	42.6636	74	28.4641
	418	40.2662	295	28.3961
Level 2	305	40.2477	504	27.119
	231	40.1401	411	26.6158
	411	38.0863	104	25.9207
	387	36.7168	410	25.6588
	151	33.3331	442	24.52
	326	32.5272	269	24.3924
	262	32.4313	274	22.891
	481	31.9384	493	22.8273
	38	31.9007	286	22.5389
	45	31.1377	418	22.4569
	181	29.1376	387	22.4514
	338	28.7829	305	22.2504
	494	28.4344	95	22.1701
	410	28.3211	445	22.0552
	104	28.2425	268	21.7873
	8	28.212	390	21.6306
	228	27.6231	429	20.7045
	303	27.4466	71	19.1928
	474	26.5793	151	19.0196
	74	26.2086	338	18.9318

Table 5.3: Mean Min activation values for Group 1 and Group 2. Green boxes indicate common activations in the same level.

5.5.1 GP Results

We ran a total of 10 runs for each target content image with activation selected for fitness from MMMS. Image groupings of all the content images and common activation numbers calculated by the MMMS are listed in Table 5.4 and Table 5.5. Again we are simply considering the activation # selected by the MMMS. We are not using the values in the mean minimum activation matrix.

The best results along with their fitness and MSE between the target content images are listed in Figure 5.8, Figure 5.9, and 5.10. Notice the fitness plots have varying generation numbers. This is purely for experimental reasons. We noticed that despite the fitnesses steadily increasing (not converging after some generations) after 25 to 30 generations, the best individuals are producing almost identical images. One can reason that small pixel changes are not affecting the images, but are only slightly affecting the fitness. Nonetheless, the visual results are similar.

If we inspect the best results, we can see the stripes (Figure 5.8a) and rectangle have much lower MSE (higher fitness). Visually, it is evident that images are capturing characteristics of corresponding target images. In other words, the best result images are visually similar to the target content image, and their fitness values are high. This means that good fitness values correlate to image sharing characteristics of the target image. In case of the triangle (Figure 5.9a) and circle (Figure 5.9b), the same can be said. Notice in case of the circle, some of the best results are very close to the target image. This can be explained by GP's procedural texture language. As mentioned in Section 4.1, our GP texture language has polar coordinates along with X, Y Cartesian coordinates. These polar coordinates are centered in the middle. Since the texture coordinates range in $0 \leq X, Y \leq 1$, GP can easily draw a circle. There is a language bias towards circular shapes like target image. Another interesting characteristic of the circle images and stripe images is that red is quite dominant. We can speculate that this happens because the image group we used for common activation was red color dominant, both in the case of circles and vertical stripes.

In case of the concentric rings, although the fitness is lower than other targets, one can clearly see curves and distorted rings in the evolved images (Figure 5.10a), which is one of the main characteristics of concentric rings.

Of all the target images, the spiral (Figure 5.10b) has the lowest fitness. This may happen for a number of reasons. Although our texture language consists of a fairly large number of functions and terminals, maybe these are not enough to easily draw a complex spiral shape. The procedural texture is good at producing repeated texture pattern rather than a single complex object like a spiral. Another key factor is that deep CNN is trained on millions of images. Since the purpose of that training was to recognize and classify daily objects, there is a chance that the deep CNN did not see any spiral objects during training.

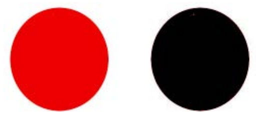
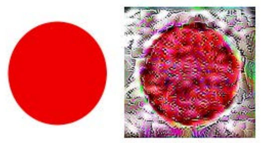
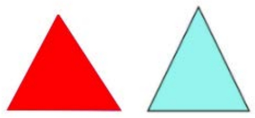
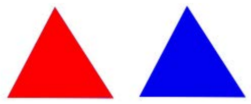
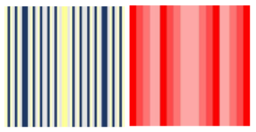
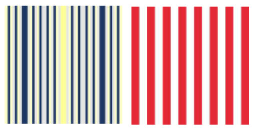
Content	Image Groups	Selected Activation #
Circle	Group 1  Group 2 	369,178,271,63,187,385,104,410 416,103,18,324,355,295,411, 387,305,151,338
Triangle	Group 1  Group 2 	178,136,38,162,401,268,104,428,376 203,75,295,88,200,474,416,299
Vertical Stripe	Group 1  Group 2 	314,178,484,162,274,247,283,350,206, 101,104,155 143,5,156,478,214,354,431

Table 5.4: Image groupings and selected common activations numbers

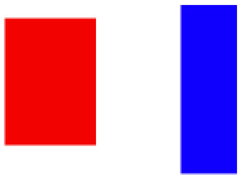
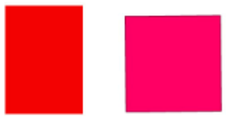
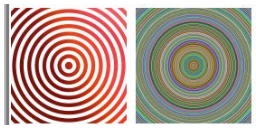
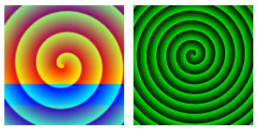
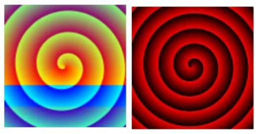
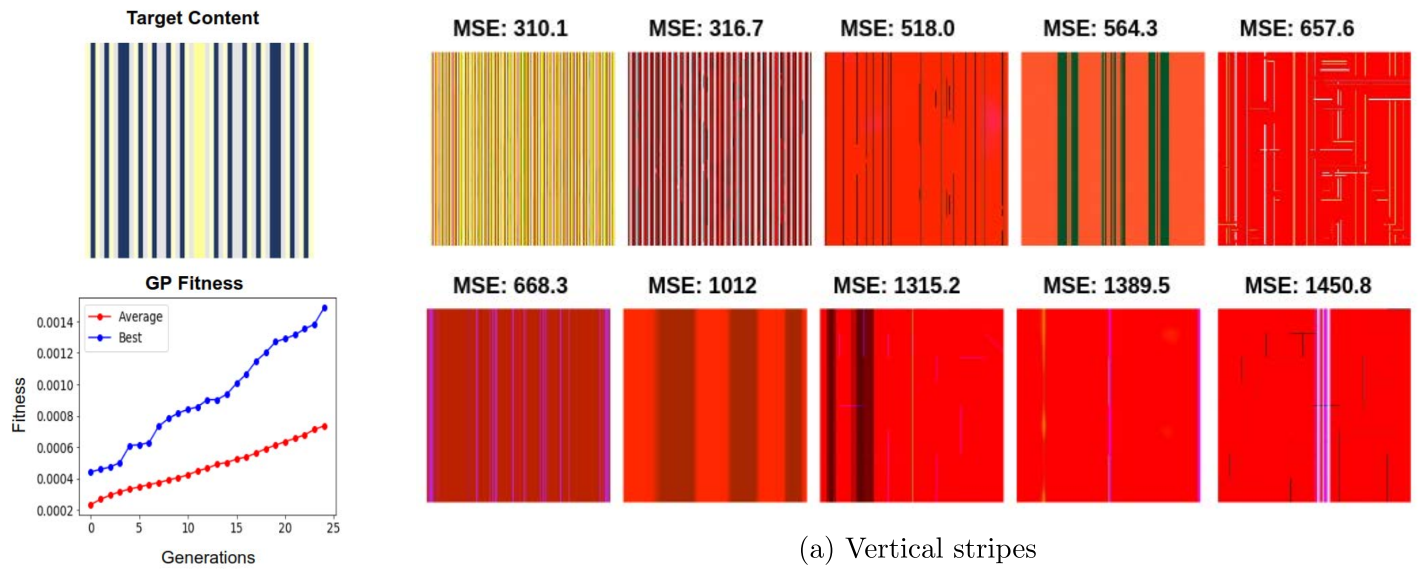
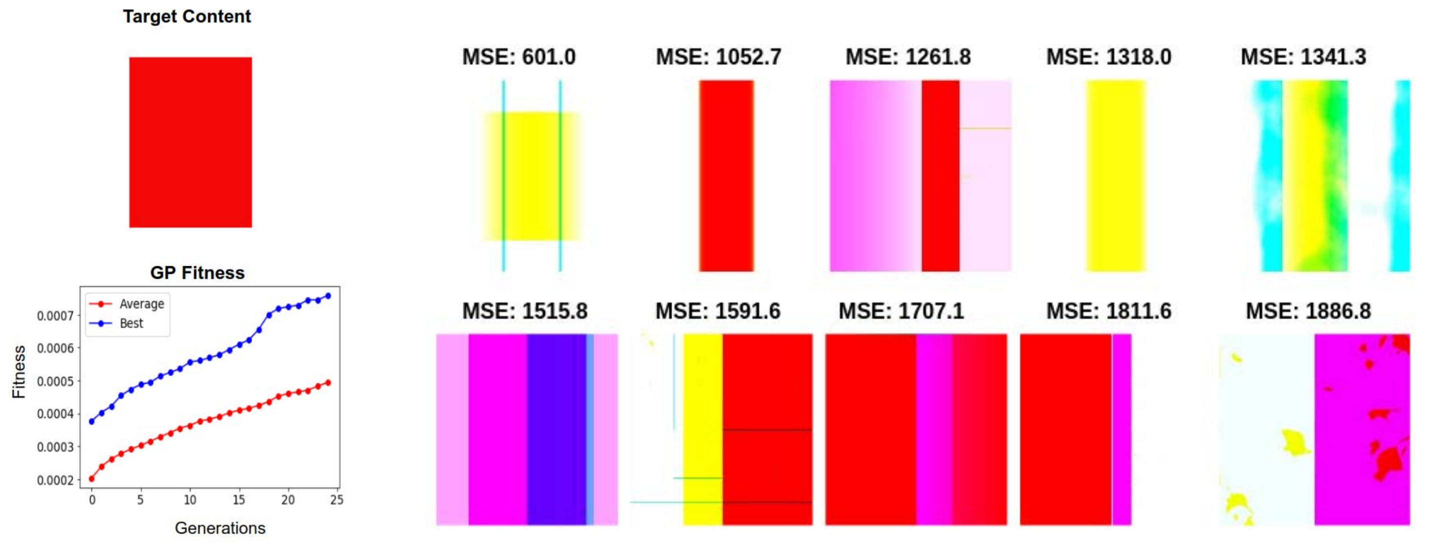
Content	Image Groups	Selected Activation #
Rectangular	Group 1  Group 2 	38,178,162,369,324,104,350,126,63, 509,258,299 210,403,504,416,470,472, 187,203
Concentric Rings	Group 1  Group 2	199,141,120,364,63,129,178,493,7 387,8,251,18,288,503,286,369,390,420
Spiral	Group 1  Group 2 	283,493,156,178,63,403,129,390,503,116 8,18,465,328,364,369,45,355,167

Table 5.5: Image groupings and selected common activations numbers

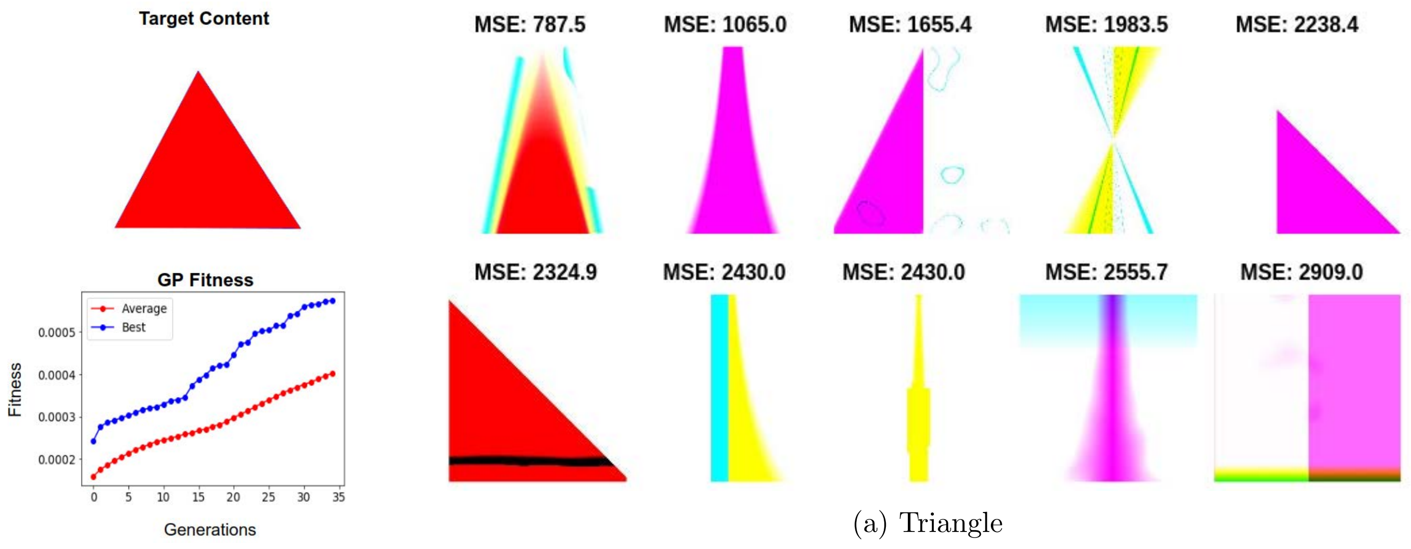


(a) Vertical stripes

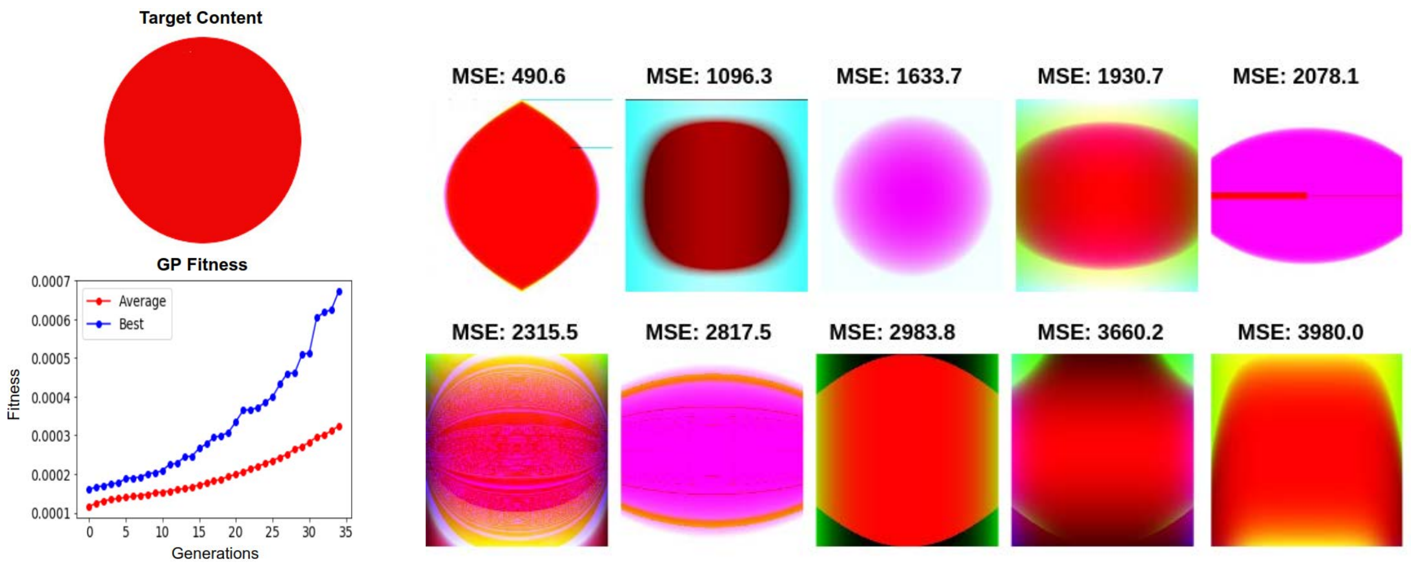


(b) Rectangle

Figure 5.8: Best results of 10 runs. Standardized fitness plots are averaged over 10 runs.

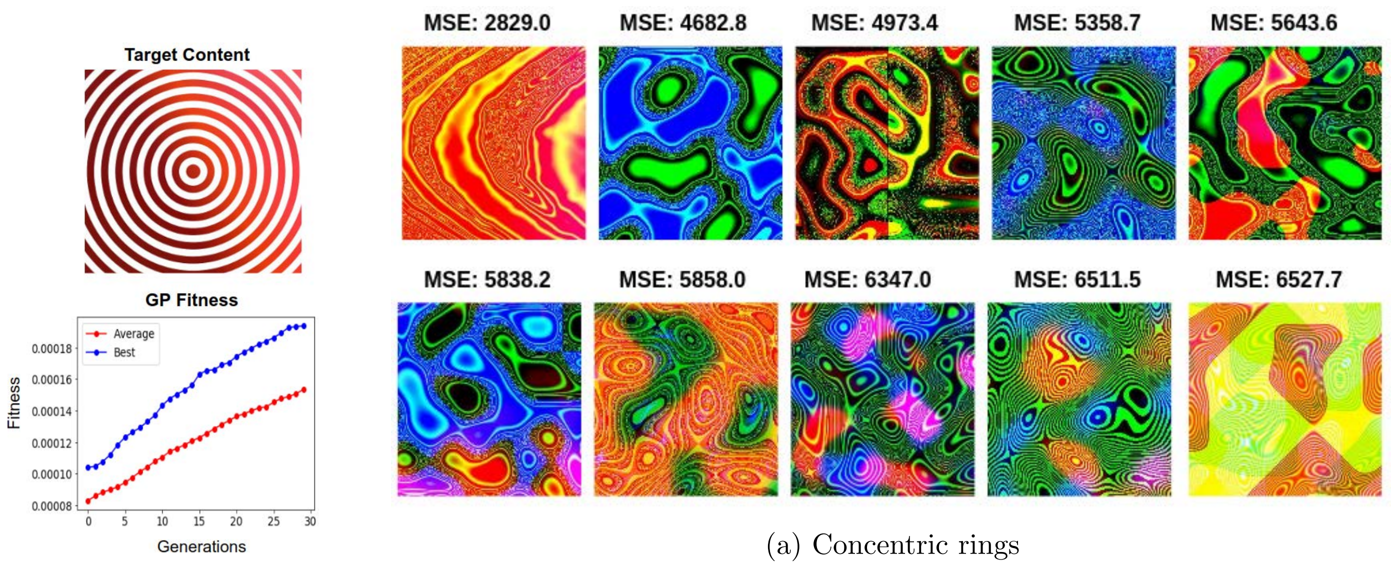


(a) Triangle

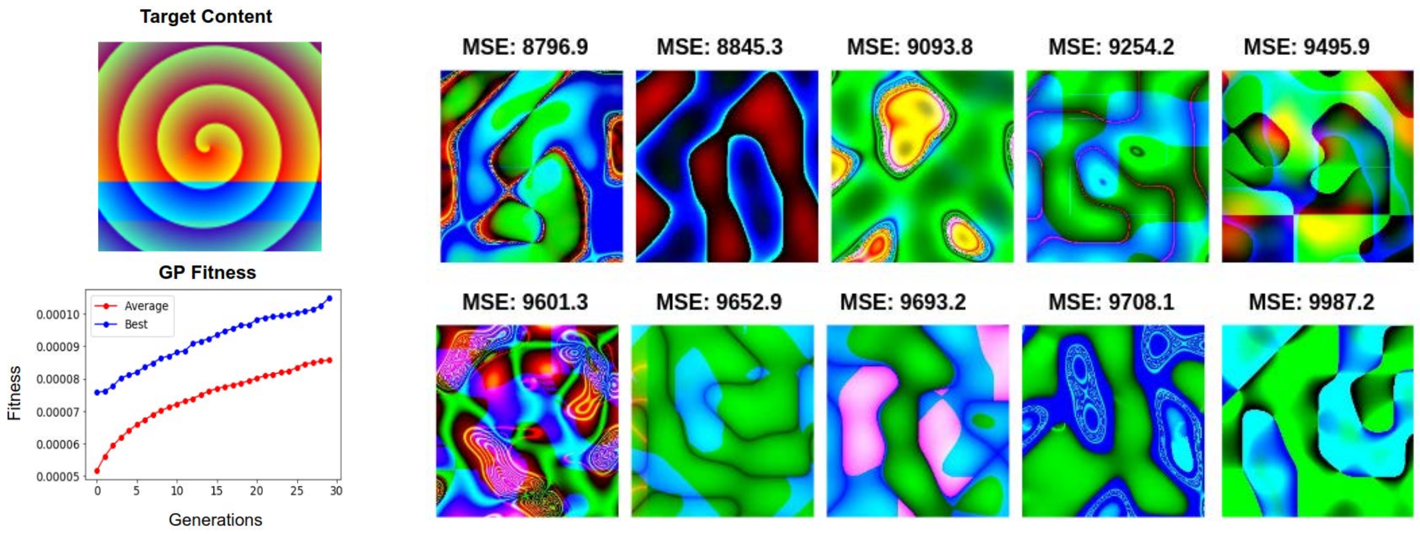


(b) Circle

Figure 5.9: Best results of 10 runs. Standardized fitness plots are averaged over 10 runs.



(a) Concentric rings



(b) Spiral

Figure 5.10: Best results of 10 runs. Standardized fitness plots are averaged over 10 runs.

5.5.2 Further Discussion on MMMS

From the GP result presented in Section 5.5.1, it is evident that our GP engine is exploiting our heuristic preprocessing MMMS. However, there are a number of ways we may improve this. One may use more target content image pairs that would result in more accurate determination of useful maps. Also, shuffling similar image pairs multiple times so that each image will be paired with all other images and then find the common activations. A more vigorous statistical analysis may also help accurately select activations.

5.6 Target Combinations using MMMS

After the initial success with the MMMS with a simple target image, we shifted our focus to multiple target images to achieve target combinations. We know from the previous section that we can specify a single target image with activations calculated by MMMS, and GP can produce a texture capturing aspects of the content of the target image using those activations. Intuitively, we can contemplate specifying multiple images with their corresponding activations from MMMS. This section discusses such experiments to demonstrate whether or not this idea is feasible. However, because of the restricted GP texture language, this might be difficult. GP can only express textures with one single formula and so it might be hard for GP to express multi-image content with a single texture formula. But nonetheless, it is interesting to investigate the effect of multiple target images on the evolutionary process.

For simplicity, each experiment uses two target content images. We will not be experimenting with three or more images because the MMMS process will be too inaccurate and complex. As done earlier, we will not use large activation matrices as a target. Using multiple images will require us to provide more common activation matrices calculated by the MMMS. For example, the MMMS gives us 19 activations for each single target image (see Section 4.5). If we take two target content images, the number will be 38. This is double the size of the single target image activation matrices. In the following section we will discuss how to resolve that problem by properly configuring the total system.

5.6.1 Configuration of the System

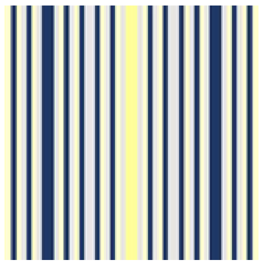
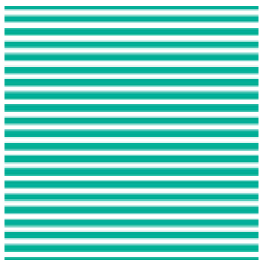
The architecture is exactly the same as described in Chapter 4. All the GP parameters and texture language are listed in Section 5.1 in Tables 5.1 and 5.2. In order to use two target images, we slightly modified the deep CNN system. The initialization of the deep CNN will be done using two content images instead of one. Both of the content images will pass through the deep CNN. The feature maps selected by MMMS at r51 will be saved separately for each of the target content images. Here we need to reduce the number of feature maps or activation matrices to keep the content target dimension consistent with the previous experiments in Section 5.5. For example, from Table 5.4 we know that for vertical stripes there are 19 activations. Similarly, using MMMS we can calculate common 19 activations for the horizontal stripes image. If we want to combine them for the multi-image target, then the total common activation of the two targets will result in double the size of the r51 activation matrix (19 + 19 = 38 total activations). To rectify this, we will use only the first half of the activations (to be precise, 9 activations) from both of the target's common activations list. This way our target r51 matrix will be close to the size of the target in Section 5.5 experiments (see Table 5.6). This may produce less impressive results because of the reduced activations from each target content image. The GP produced image will pass through all the layers until r51 and the MSE between this layer and the other two layers (one for each target image) will be calculated separately. That means GP first considers the first image and uses the activation associated with it to calculate MSE. Then we calculate the MSE with the second target image with the second activations. The two MSEs are combined and will be returned to the GP as the fitness.

$$Total\ Fitness = MSE_{content1} + MSE_{content2}$$

The fitness is a simple sum of the distances of GP produced image from the two target images.

5.6.2 Experiment Details and Analysis

We run 10 GP runs with 30 generations for each experiment. Although from the Table 5.1, the default number of generations is indicated as 35 but for different experiments, we will vary the number slightly. In those cases, we will mention the generation number explicitly in the beginning. Figure 5.11 shows all the best results along with their MSE and fitness plot. The best solution images share the same characteristics of both target content images. But some interesting factors are worth discussing. In case of vertical and horizontal stripes (Figure 5.11a), instead of equally minimizing MSE from both of the targets, GP results tend to lean towards a single target image. That means they are minimizing MSE for only one target. Most of the solution

Content Description	Target Image	Selected Activation
Vertical Stripes		314,178,484,162,274,247,283,350, 206 101,104,155 143,5,156,478,214,354,431
Horizontal Stripes		316,33,143,63,104,274,89,155,38,283,96 178,350,8,484,162,411,298,428

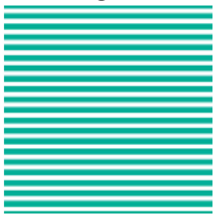
Content Description	Combined Targets	Combined Selected Activations
Vertical and Horizontal Stripes	Target 1 Target 2 	314,178,484,162,274,247,283,350,206 + 316,33,143,63,104,274,89,155,38

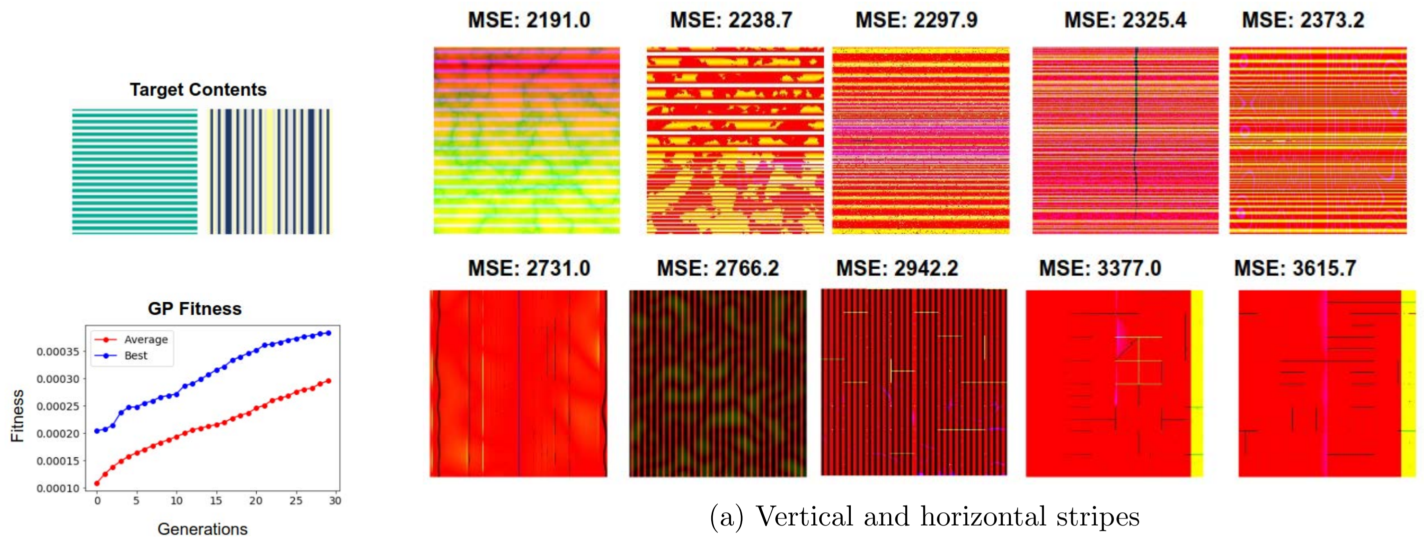
Table 5.6: Target combinations. Top table shows the activation for each separate target image. Bottom table combines the top activations. Notice how combined selected activations are chosen. First half comes from first content image activation. Second half comes from second content image activations.

images are either vertical or horizontal.

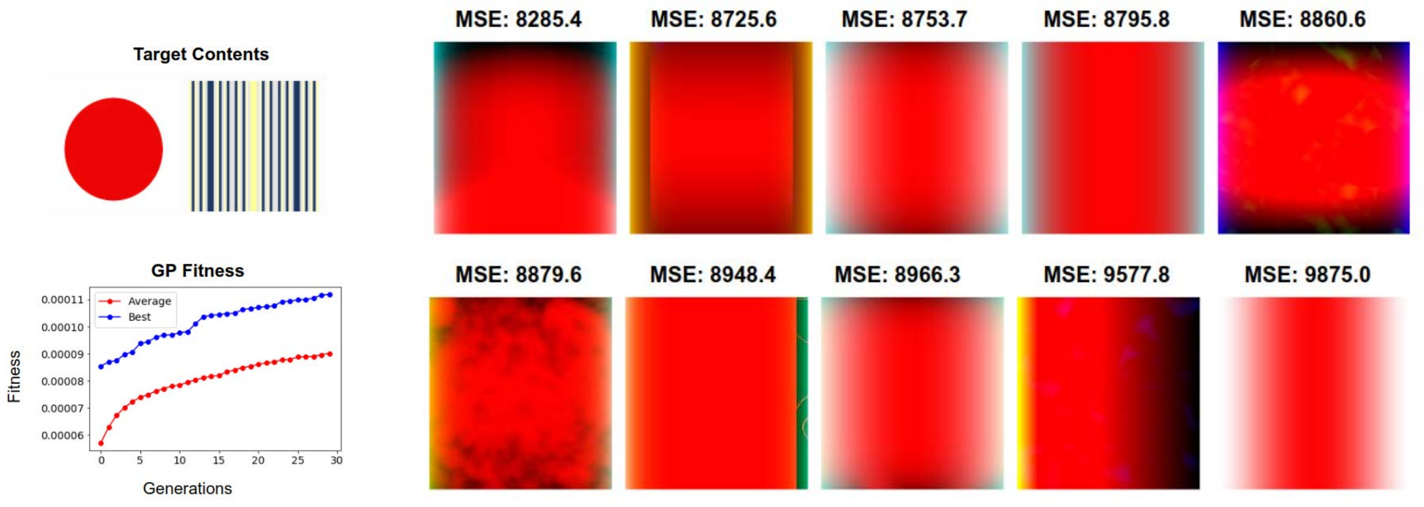
In order to visualize this better, we created a scatter plot with X-axis with MSE from the vertical image and Y-axis with the MSE from the horizontal image (Figure 5.12). We placed the corresponding GP best results on the map. This chart conveys an idea of how far the solutions are from each target content image. Interestingly they are divided into two groups. Only 2 of them are somewhere in the middle. Those two images have both vertical and horizontal stripes. In the case of the circle and the vertical stripes (Figure 5.11b), the same can be said. Results consist of a slight mixture of the red blob in the middle and around the edges there are small stripes. It is reasonable to say, they are capturing vague characteristics of the contents (stripes and circular shapes). We made another scatter plot (Figure 5.13). Solutions that greatly minimized MSE from vertical content target showing characteristics of vertical stripes. Solutions that greatly minimized the circular target image, showing circular shape by having a red blob in the middle. This is not close to the circular best results we had in single target MMMS (Section 4.5) since we removed half of the activations. In spite of that, these solutions vaguely resemble a red circular shape. The solutions that are in the middle of the chart, which means they are equal distance from the target content images, show both characteristics. They have vertical stripes in the side and red blob in the middle.

5.6.3 Future Work

Although the best solutions are promising, more intensive work on this is required for further improvements. As previously mentioned, we used a very simple MSE formula as fitness. Multi objective fitness [47], and more accurate activation choice might improve the results.



(a) Vertical and horizontal stripes



(b) Circle and vertical stripes

Figure 5.11: Best results of 10 runs. Standardized fitness plots are averaged over 10 runs.

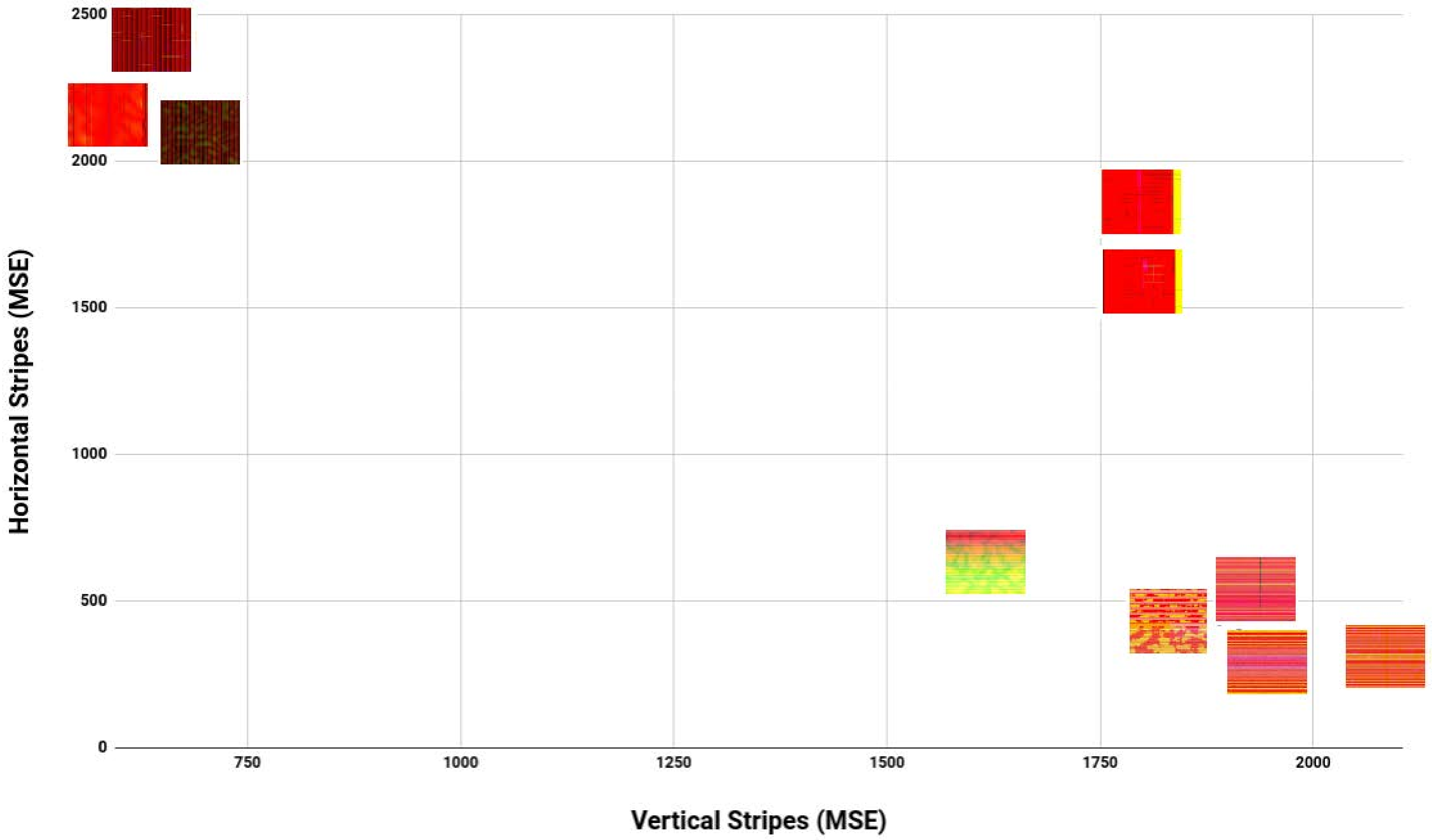


Figure 5.12: MMMS combination of horizontal and vertical stripes. Lower MSE in any axis means they are close to the corresponding target image in that axis.

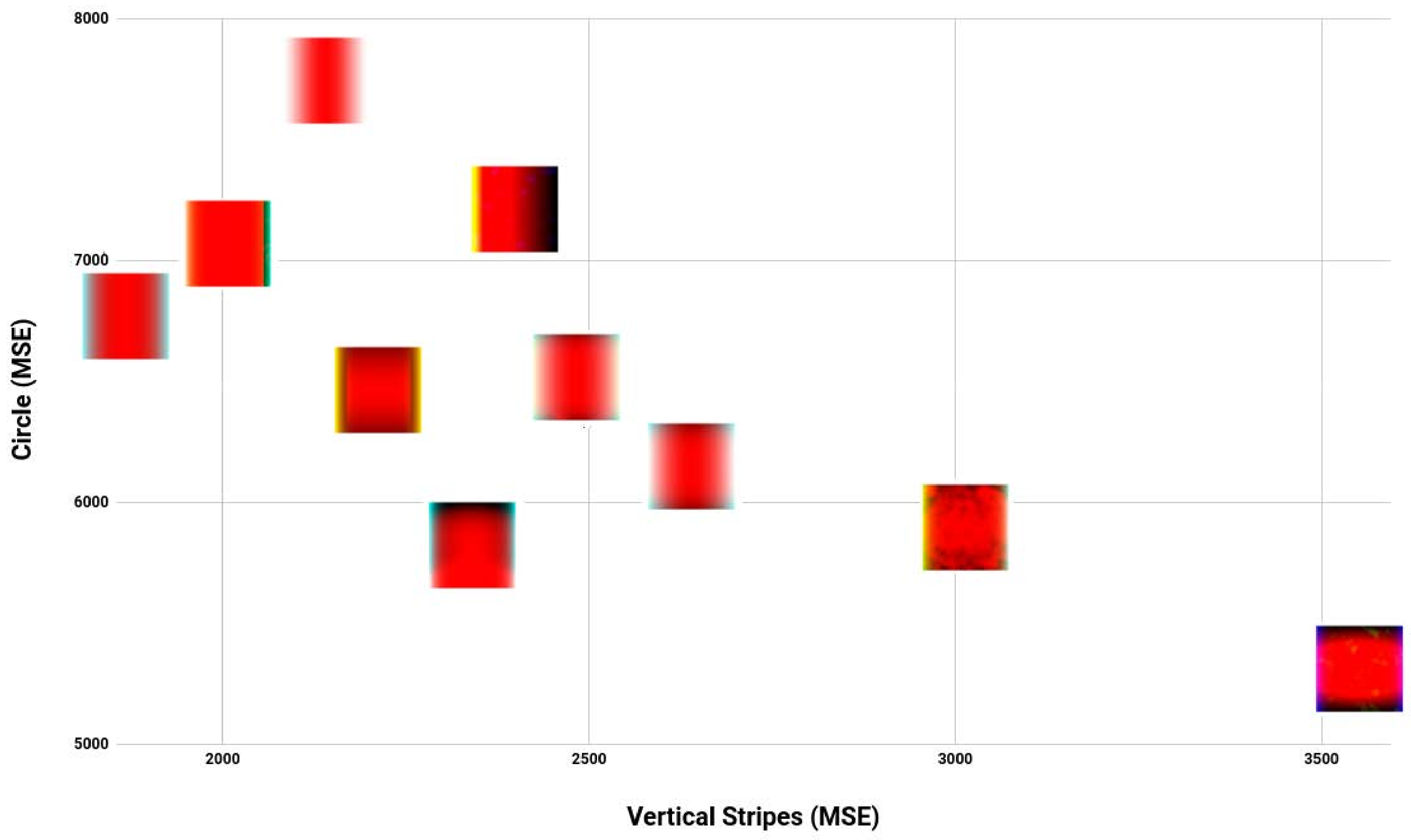


Figure 5.13: MMMS combination of the vertical stripes and the circle. Lower MSE in any axis means they are close to the corresponding target image in that axis.

Chapter 6

Fully Connected Layer

In this chapter, we will discuss the use of the fully connected layer as a guide to our evolutionary GP system instead of using only the convolutional layers. In all the previous experiments, we utilized the group of convolutional layers in the deep network, which is a strong prior to the total network. We only utilized the convolutional layers as a GP fitness earlier in Section 5.5 (MMMS) to guide our evolutionary system and did not use the trained fully connected layers. The fully connected layer or FC layer is a major component of any deep neural network. Here, we want to investigate the effect of the fully connected layer as our GP fitness. This fully connected layer is a more abstract representation of any given image than the convolution layers.

6.1 Background

The fully connected layer is a traditional deep MLP used for classification purposes. Every neuron has full connections to all the neurons in the previous layer. This is why it is called "fully connected". This can be thought of as a general purpose regular neural network. Their activations are computed with matrix multiplication and a bias offset. The final output dimension of the FC layer depends on the specific architecture used. In our case, we used the pre-trained VGG-19 architecture. This architecture classifies 1000 objects so the fully connected layer's final output contains 1000 channels (one for each class). Given an input image, the final output layer will produce numbers that represents the probability of the image being a certain class. The fully connected layer uses a softmax activation function in the output layer. It spreads the 1000 score values from the final FC layer from 0 to 1 and thus provides the probability of the each class. The sum of the output probabilities from the fully connected layer is 1. In summary, the convolutional layers are providing a meaningful, low-dimensional, and somewhat invariant feature space and the fully-connected layer is learning a function in that space.

6.1.1 Fully Connected Layer System Overview

As mentioned in the previous chapters, the convolutional and pooling layers represent high-level features of the target image. The goal of the fully connected layer is to utilize these features for image classification based on the training dataset. In our case, we used the VGG-19 model which was trained on the ILSVRC-2012 dataset containing images of 1000 classes [4]. The model was trained on 1.3 million images from 1000 classes. The major difference between the architecture we used in Chapter 5 and this one is we are now including the final FC layer. It has a total of 123,638,760 additional parameters compared to the convolutional layers, including all weights and biases. The output from the convolutional layers represents high-level features in the data, and the fully-connected layer is a method of learning non-linear combinations of these features.

6.1.2 Using Fully Connected Layer As a Guide to Evolution

Our goal is to utilize the FC layer to guide the GP evolutionary process so that we might evolve images with similar visual features of the class we specify. For example, if we specify the class which represents "peacock" in the output layer and configure GP to score high on that class, we might get an image with similar characteristics to a peacock. This has been done in the past in a different context [41]. The motivation was, given that DNNs are able to classify objects with remarkable accuracy, to determine the differences between computer and human vision and how can they fool the DNN. They used a different evolutionary technique to generate images and configure the system to get high confidence score to each dataset class. Their intention was to fool the DNN. For example, in [54], it was revealed that changing an image in a way imperceptible to humans can cause a DNN to label the image as something else entirely with almost 99.99% confidence. This even happens with complete white noise totally unrecognizable to human eyes, where the DNN believes with absolute certainty that the noise represents familiar objects. The work in [41] used EA (evolutionary algorithm) to produce such images. Their system produced intriguing results which were not visually close to the images of the given class from a human perspective, but the confidence level of the class was high. As a result, some images had an interesting artistic look and feel. We will perform similar experiments using our GP system.

6.2 System Design and Initialization

The trained deep CNN architecture will be the same as in Chapter 4 i.e. VGG-19, except for two major differences: 1) it has a fully connected part with a 1000x1 vector in its output layer passing though softmax function to assign the probability of the class scores; 2) instead of providing the target content image, we will provide only the class number we are interested in. For example, in the 1000x1 output matrix, class #84 represents a peacock. We will provide this class number so that GP fitness maximizes a high score in #84. In other words, GP will maximize the class score in #84 and therefore have low scores elsewhere. This way the evolved image might have a visually similar pattern of a peacock. Our goal is to obtain such an image generated by the GP, which scores very high in that class. The hyperparameters we will be using are default VGG-19 hyperparameters. In Chapter 5 we modified many hyperparameters (pooling layers, window size, strides etc.) to reduce the size of the activation matrix in the convolution layer. However, those modifications are not needed since we will be using the fairly compact final output vector which has only 1000 dimensions. The block diagram of the total system is illustrated in Figure 6.1.

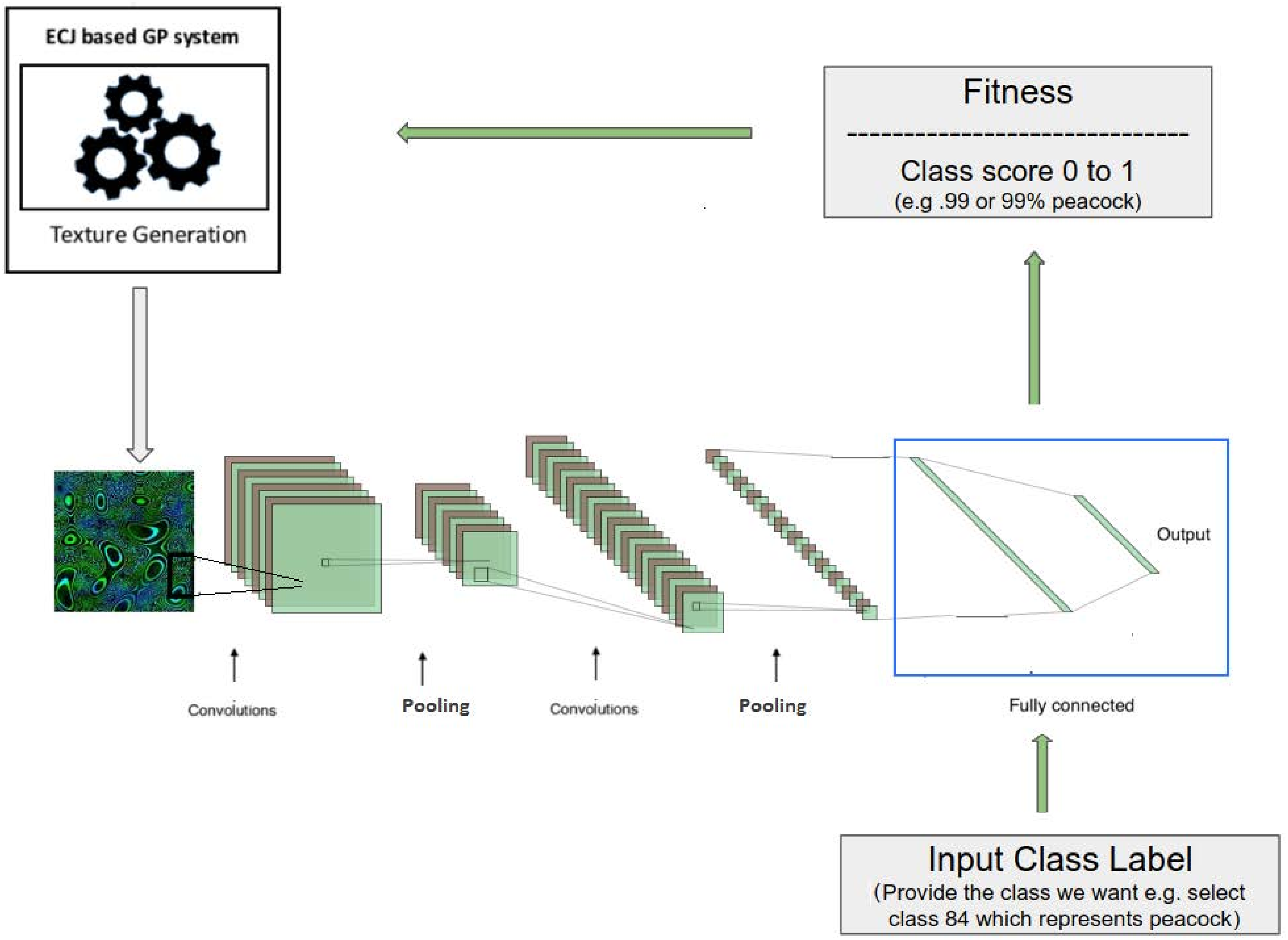


Figure 6.1: System design of fully connected layer with GP. Green box is the new FC layer output we included. The FC layer output is a 1000x1 vector.

6.2.1 Fitness

In the fully connected layer system, we will only specify the class number we want GP to maximize. If the class number we are interested in is n, the fitness can be expressed by

$$Fitness = 1 - Class\ Score_n \quad (1 \leq n \leq 1000)$$

First, GP produces a texture. The texture goes through the network including the final output. We then select the class we are interested in (for example peacock is class #84) and measure the score in that class. The score is between 0.0 and 1.0. A score of 1.0 means the deep CNN is 100% certain that the image is of that class. We measure the difference of that score from 1.0 and return the difference to GP as fitness. If GP produces the class score of 0.33 in class # 84 then we return 1.00 - 0.33 = 0.67 as the fitness. GP continues to minimize this value which will increase the score in that specific class.

6.3 Experiment Details and Analysis

We begin by specifying the class we want. We will run multiple experiments with various classes. We know that GP may not produce some of the class image, since it will be extremely hard for GP to get a high score for some specific and complex classes. For example, in the ILSVRC-2012 dataset out of those 1000 classes, there is a handful of classes denoting different breeds of dogs. It will be very difficult for GP to represent a dog with our current texture language. However, our goal is to evolve images having visual characteristics of entities in the target class. We want patterns of given class exhibiting in the GP produced images and score very high score on that class.

Before moving on to our experimental results we need to clarify the concept of the "confidence score" in the fully connected output layer. In deep CNN terminology, the confidence score (a value between 0.0 and 1.0) means how confident the deep CNN is about the class identity of the given image. We are going to use this score as the GP fitness.

For experimental purposes, we selected the classes which we think GP has a chance to produce a more significant score. We have an intuition that certain repeated texture patterns might be easier for GP to produce. GP's noise based texture language is good at producing repeated textures . In Table 6.1, we list the classes used including the class numbers in the final output layer of the ILSVRC-2012 dataset [4]. We also include the real life images of corresponding classes and their confidence scores from our deep CNN module. We can later compare our GP generated image scores using this table for reference.

Class Description	Example Image	Class Label	Deep CNN Score
Honeycomb		**599**	94.96%
Orange		**950**	97.3%
Peacock		**84**	100%
Pomegranate		**957**	98.6%
Pinwheel		**723**	99.91%
Banana		**954**	99.6%
Tennis Ball		**852**	99.7%

Table 6.1: Class description and class number defined by the ILSVRC 2012 dataset. The confidence scores from deep CNN (VGG-19) are listed for reference.

Our GP engine uses the parameters and texture language listed in Tables 5.1 and 5.2. All the experiments run for 25 generations and we perform 8 runs per class. We are reducing the number of runs since the fully connected layers has more parameters than CNN layers, and therefore is more computationally expensive. The overall time required for one run is more than double of the previous experiments.

The results produced by the GP vary and are often promising (see Figure 6.2 through Figure 6.4). For example, in all the cases, though they were not producing the exact same images of the provided class, the confidence level we were getting is often very high. In other words, the deep CNN classified evolved images as correct ones. Although subjective, some results captured recognizable features of the target classes. All of our best results did not retain any shapes of those original class objects, but still managed to get a high confidence score from deep CNN. For example in the case of the peacock class (Figure 6.2a) we can definitely see it is exhibiting the texture and color of a peacock feather with a prominent dark green texture. One point to note is that the GP language is considerably adept at producing this type of texture. As a result, we are repeatedly evolving a peacock feather texture and the deep CNN is highly confident that this is a peacock. Aspects of both shape (concentric rings) and color (dark green) are present in the evolved images.

It is fascinating to see how the deep CNN classifies objects. The exact shapes of objects may not be required for object classification. In most cases, the deep CNN is looking for certain distinguishing patterns and colors in that image regardless of their shape. The deep CNN is a complex network of non-linear parts and the pattern or image characteristics it tries to find in the given image depends on the training. The DNN will focus on specific characteristics that differentiates a class from all the others. These characteristics can be unusual and unexpected. Color often plays a dominant role. In all the experiments our best results are accurately developing the color properties of those target objects. In case of peacock (Figure 6.2a), orange (Figure 6.3b), pomegranate (Figure 6.3a) and banana (Figure 6.4b) GP is getting the correct colors. The results of the honeycomb class are equally interesting. In Figure 6.2b notice the confidence level is quite high. The GP evolved images are getting both shape of the and color of the honeycomb. The evolved best results are amber grids with a circuit-like pattern. They even have dark shades around the edges of those grids just the real life hexagonal cell of honeycomb. Notice the worst result is completely out of place comparing to the other results due to the lack of amber color.

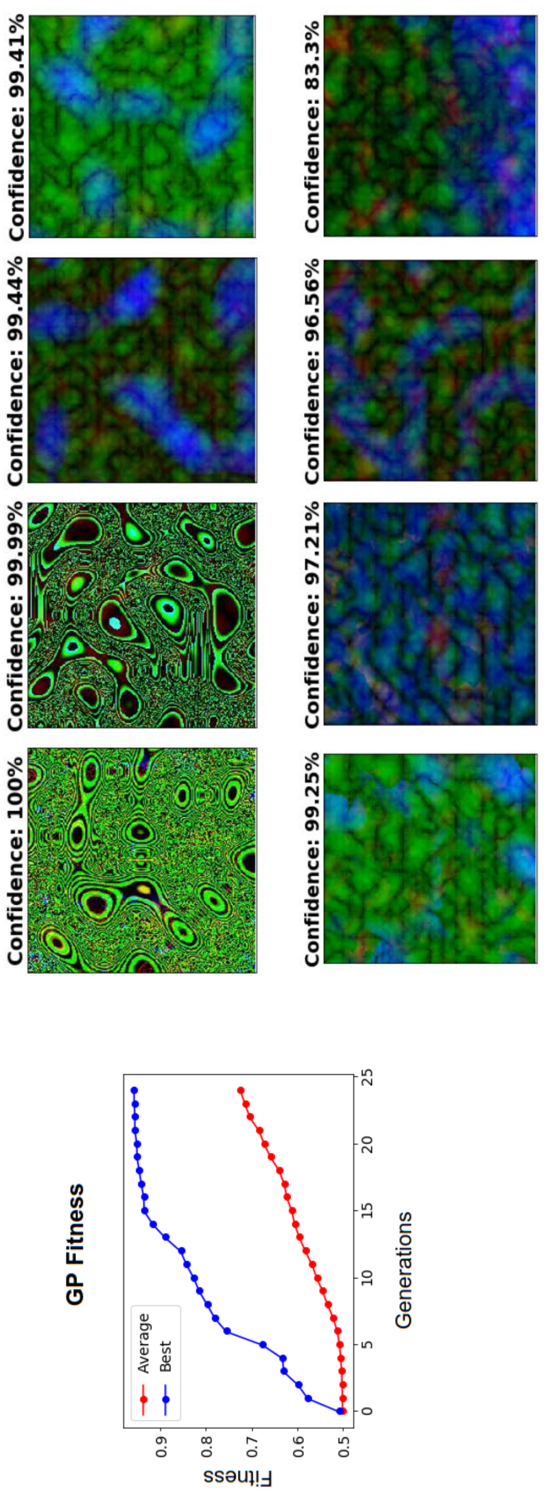
The best results of pomegranate class (Figure 6.3a) arguably might look very close to pomegranate seeds even to a human observer. The first 4 results similar to a bowl of pomegranate fruit seeds and they have high confidence values. Notice how the last result does not look like a pomegranate since it has green shades. As a result, the confidence score is low. We can conclude that the evolved images are successfully capturing both shape and colors of pomegranate seeds.

The orange experiment (Figure 6.3b) is also another good example of color dominance in the trained deep CNN. In all the runs, the best results capture aspects the orange's outer skin texture and color. In the case of the pinwheel (Figure 6.4a), the result may not be as obvious as in other experiments. Nonetheless, the confidence level of those images is high, and we can notice pinwheel shape within the solution images. This shape property differs from other examples since pinwheels are not color specific. Pinwheels usually have multiple solid colors in their wheel blades. Evolved images do not conform to any specific color. Rather, they focus on the pinwheel blade-shaped features, and solid areas of color.

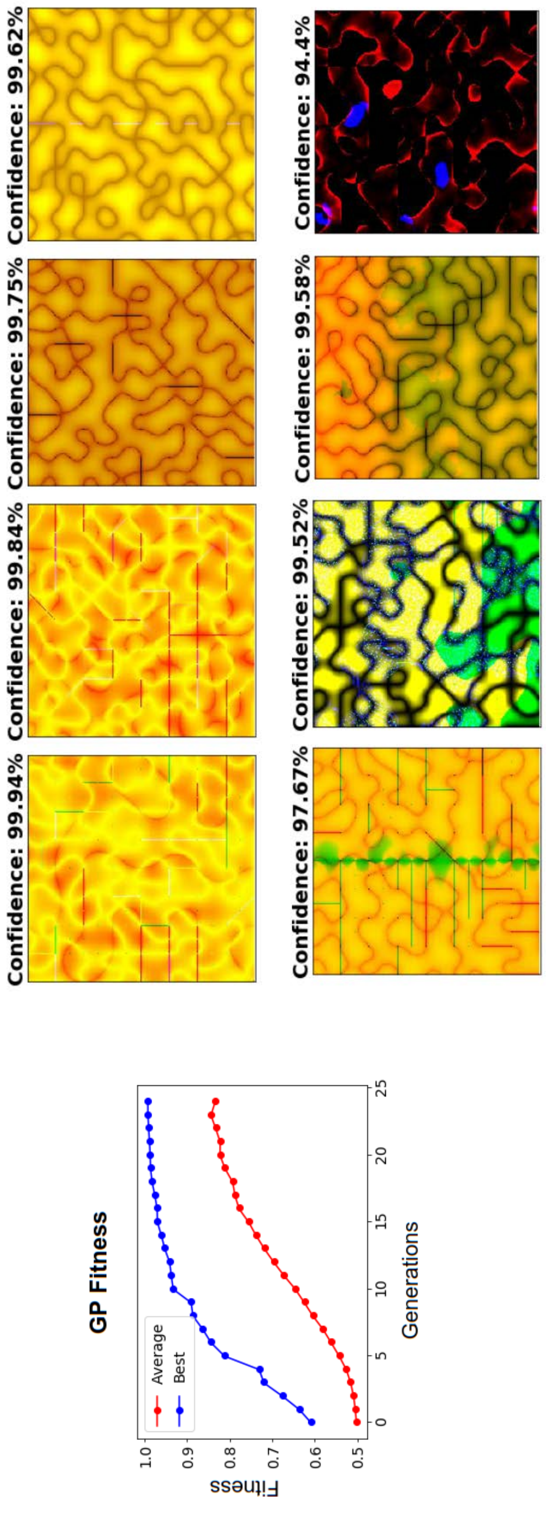
The Banana (Figure 6.4b) experiment is an unsuccessful experiment. GP had trouble producing banana like images. The best results have a very low confidence score comparing to the other experiments. Some of the results are exhibiting banana like characteristics such as yellow shape with dark around the edges, which are present in the real life image of banana as well.

Another rather unsatisfactory experiment is the tennis ball (Figure 6.5). All the best results with high confidence score have green dots on the larger background images. It is possible that all training images of tennis ball in ILSVRC dataset have the tennis ball inside a flat tennis court rather than a centered and focused picture of a tennis ball. Notice how a single green dot can fool the deep CNN as a tennis ball. Also, a patch of pure lime green color gives a 12% match to a tennis ball.

Our results reflect the nature of the trained deep CNN. We can say that by looking at our results which achieved very high confidence score. In addition, work done by [41] also suggests similar notion. All the training examples of orange, peacock, pomegranate, and banana have fixed object-specific colors. The peacock training examples are always dark green, and orange and pomegranate are orange and red respectively. But the pinwheel object is color independent and its classification is strictly based on shapes and patterns rather than color properties.

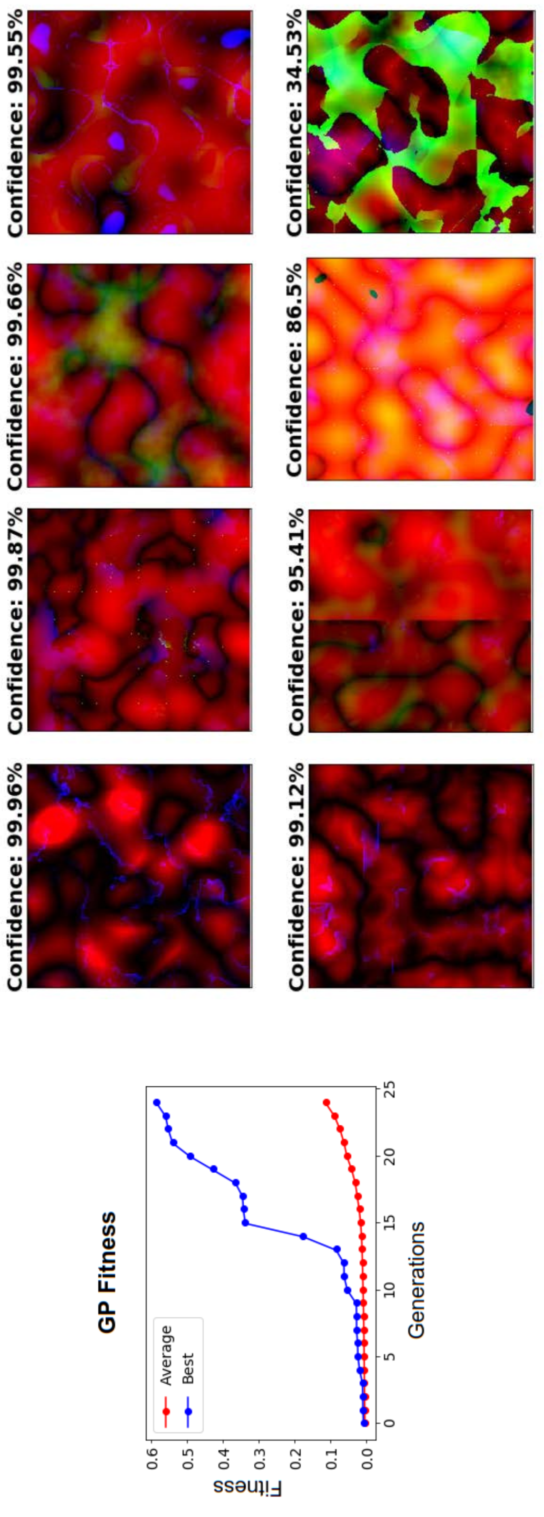


(a) Target Class: Peacock

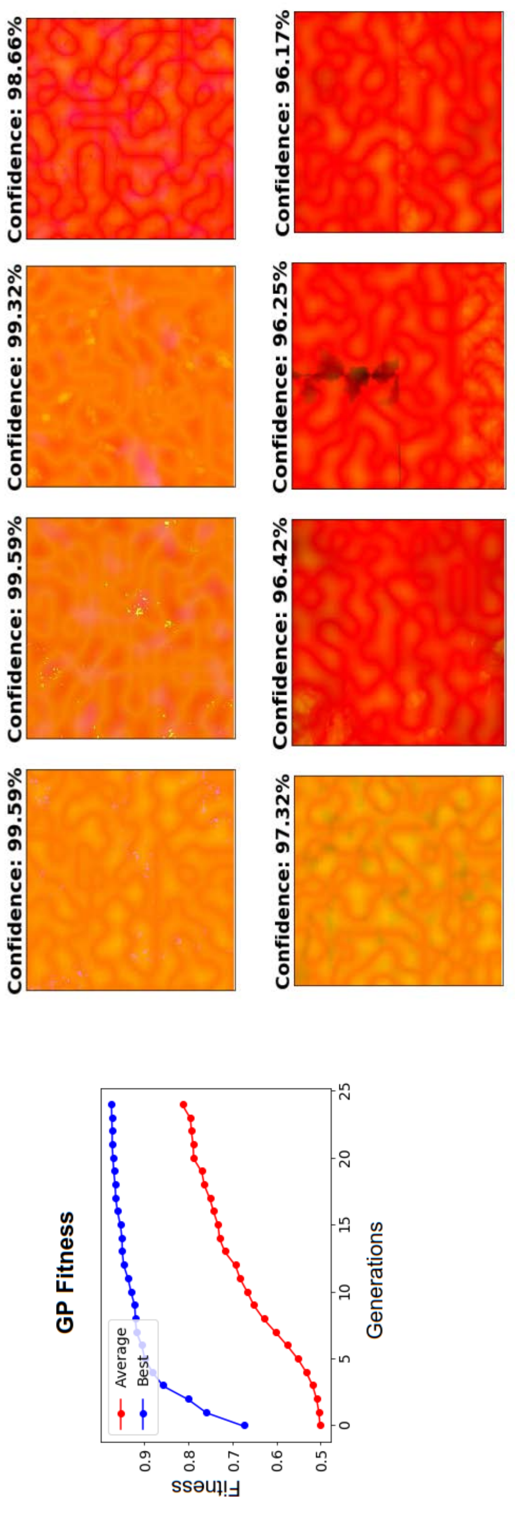


(b) Target Class: Honeycomb

Figure 6.2: Best results of 8 runs. Standardized fitness plots are averaged over 8 runs.

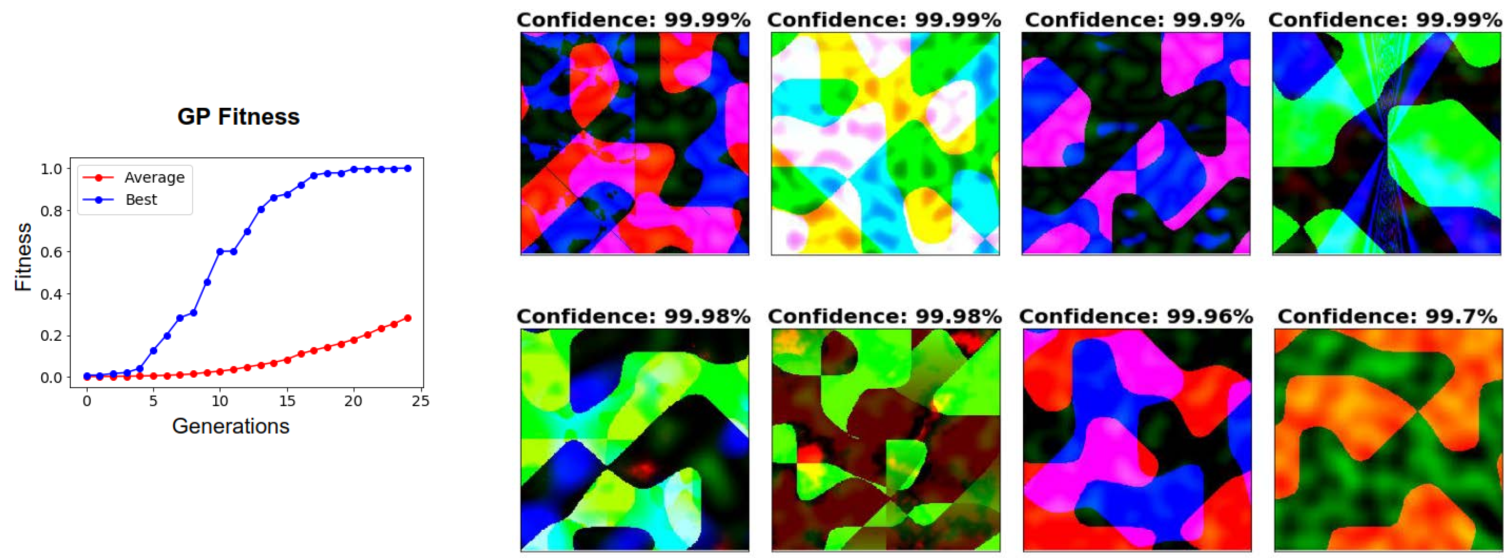


(a) Target Class: Pomegranate

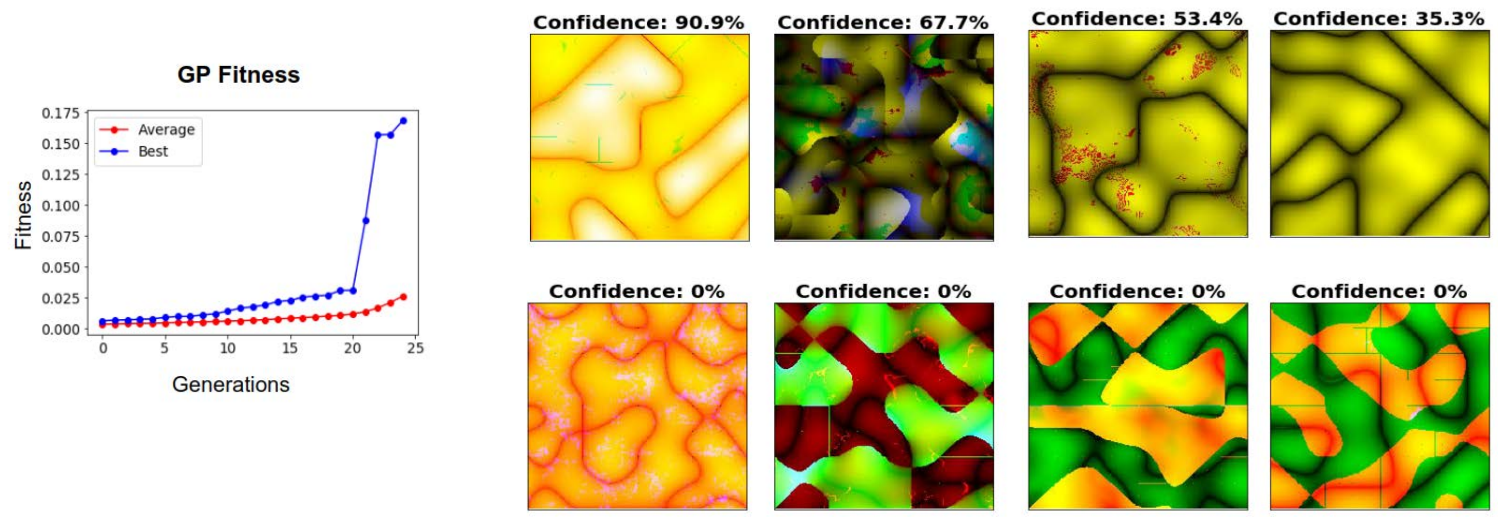


(b) Target Class: Orange

Figure 6.3: Best results of 8 runs. Standardized fitness plots are averaged over 8 runs.

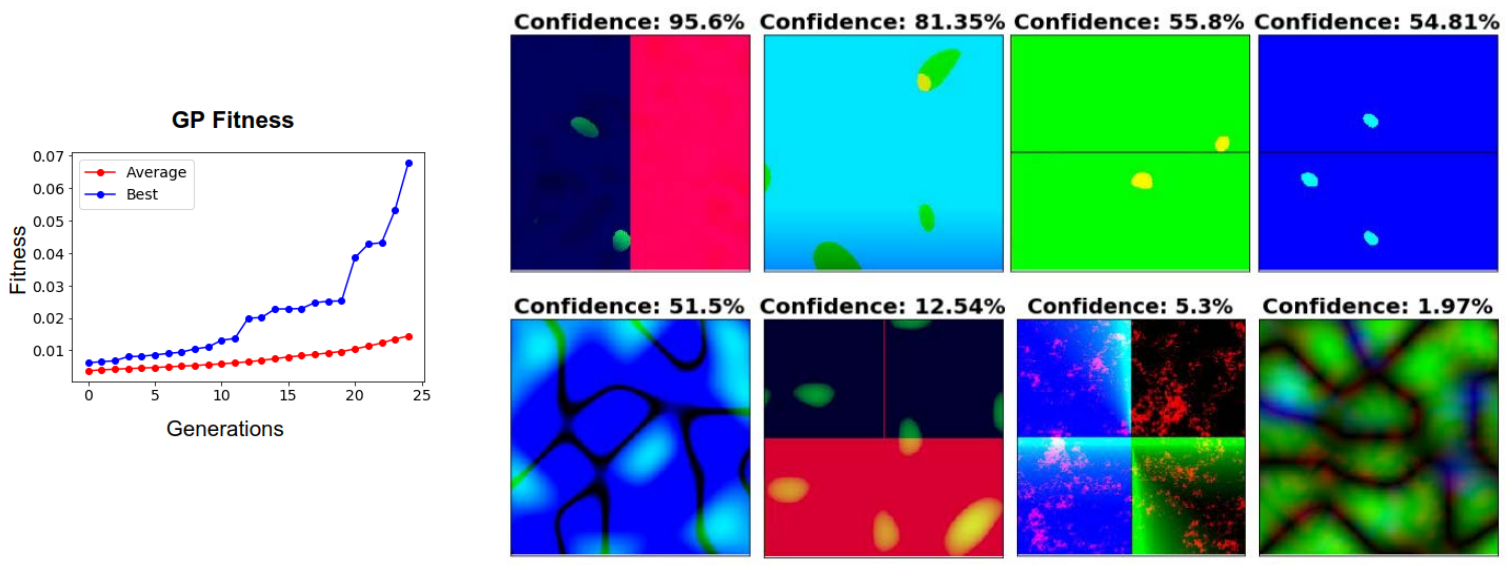


(a) Target Class: Pinwheel



(b) Target Class: Banana

Figure 6.4: Best results of 8 runs. Standardized fitness plots are averaged over 8 runs.



(a) Target Class: Tennis Ball

Figure 6.5: Best results of 8 runs. Standardized fitness plots are averaged over 8 runs.

6.4 Target Combinations Using the FC Layer

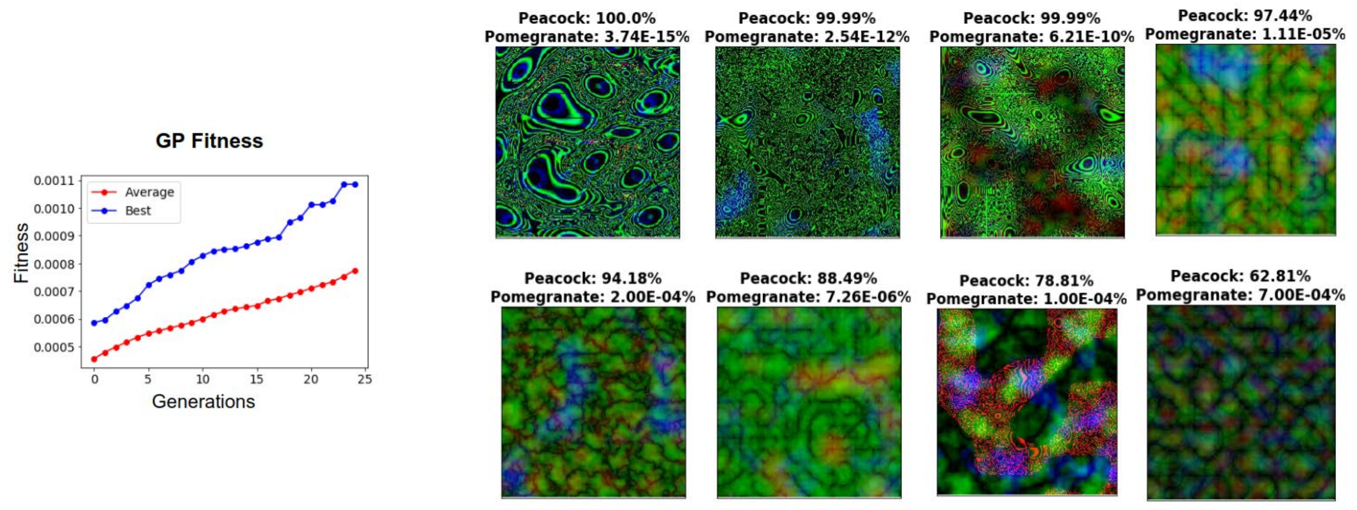
We are now interested in experimenting with multiple classes. This will allow us to perform target combinations. Instead of specifying one single target class, we are going to specify two classes and observe how GP performs. We already performed similar experiments in Section 5.6 using CNN activation matrices. Here, the GP parameters are the same as in Tables 5.1 and 5.2. We are running the GP evolution for 25 generations. If n_1 and n_2 is the class number of the first and second target respectively, the total fitness of this system is:

$$Fitness = (1 - Class\ Score_{n_1}) + (1 - Class\ Score_{n_2}) \quad (1 \leq n \leq 1000)$$

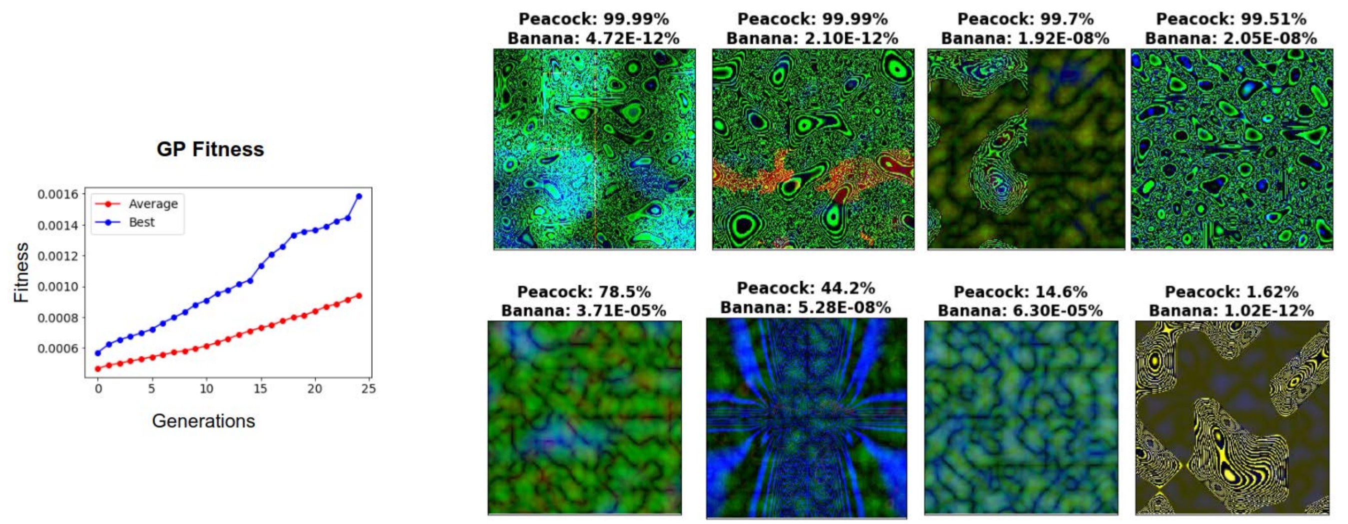
Note that we do not weight the terms. The fitness is the sum of the scores from both of the classes.

Target combinations using the FC layer is not as impressive as in Section 5.6. Results are completely one-sided i.e. highest confidence score is always from only one of the two classes. However, from the fully connected layers point of view, this is reasonable in many ways. First, the point of having a trained large complex non-linear network is to drive the high class score in one single class so that deep CNN can classify images accurately. Specifying two classes is “abusing” the system. Second, all the training example images are from single class objects. The deep CNN likely has never seen an image which is a mixture of two different classes. For example, we experimented with peacock and banana. These are totally two different objects from two different classes. It will be hard to get a good score in both of the classes since such an object (or training image) does not exist. In terms of GP, it is just a pixel level texture change, but for the deep CNN, this is totally unknown object.

In the first experiment, we will specify the peacock and pomegranate class as the target classes by selecting both the class #84 and the class #957 (see Table 6.1 for class reference). In Figure 6.6a we can see that GP is producing images with dominantly peacock properties. But it is interesting to see some red blob showing in some of the best results. This might be the contribution of the pomegranate class in the fitness. Although the confidence score of the pomegranate class is very low. Similar to the experiments in Section 5.6, here GP is also minimizing the error in one specific class, in this case, peacock. A different fitness strategy might improve these results but this is beyond the scope of the thesis. In the case of the peacock and banana, the same bias towards the peacock can be seen (Figure 6.6b). But again we can see some yellow blobs in the texture occasionally appear, which might come from the banana class.



(a) Target Class: Peacock and Pomegranate.



(b) Target Class: Peacock and Banana.

Figure 6.6: Best results of 8 runs. Standardized fitness plots are averaged over 8 runs.

6.5 Conclusion

Our evolved images often come with high certainty values from the deep CNN and often share some key visual characteristics of the target classes. But results did not evolve all the characteristics of the classes i.e. we would not mistake evolved images as the objects from those classes. This is because the deep CNN has a different model of the classes than what a human observer will have. However, this can be potentially useful as a high-level artistic tool for evolutionary process.

Most of the images used to train the ILSVR 2012 are complex real-world photos of species, animals and objects. Our evolved results are biased and limited due to the simple texture language. If we could have a robust GP system which could produce complex images, we might evolve more interesting images closer to the target classes. The whole idea of using a fully connected layer is to provide GP a sophisticated fitness strategy so that GP can conform to specific characteristics. The fully connected layer provides a high-level guide. But still, as we have seen from the experiments, this is successfully guiding the GP towards specific image attributes found in the target class, which is promising. For multi-class image combinations, we might need a more involved and complex fitness strategy to be able to balance between the two target classes e.g multi-objective analysis. This certainly requires more work in the future.

Chapter 7

Basic CNN

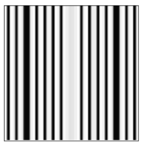
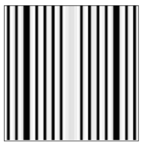
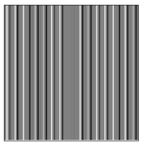
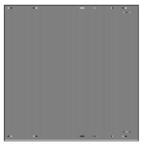
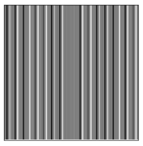
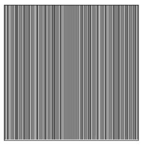
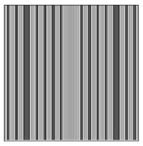
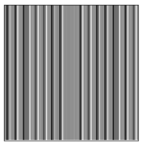
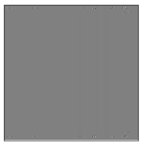
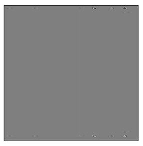
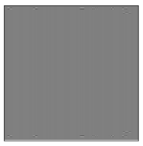
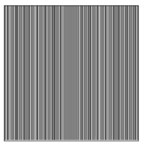
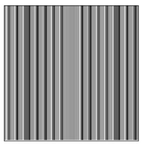
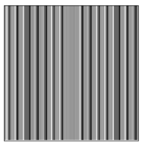
7.1 Introduction

Based on the premise of how CNN learns to capture high-level information of any given image, we propose a similar, but a vastly simplified network. This chapter uses a basic CNN for GP fitness. This basic CNN uses selected hand-picked filters in its layers. This network is untrained and quite shallow. The goal is to run the GP with the basic CNN so that GP can exploit the information captured in the higher layers of the basic CNN. We will apply this network in a manner similar to the experiments in Chapter 5. This will give us insights such as whether or not training is really necessary.

From the background study, we know that all popular CNN architectures are very deep and contain millions of learnable weights. During the training phase, those learnable weights get adjusted by backpropagation. That includes all the weight values of image kernels or filters in the convolutional layers. Filters in those layers are responsible for detecting certain properties of the image. For example, the first layer detects simple properties like edges, a blob of color, etc. The second layer detects contour and corners, and so on. One important factor to mention here is that those detector filter weights are determined during the training phase and they are not fixed throughout the whole training process. That means during training, weights associated with filters get adjusted and this helps the filters to capture different features from the input image. They are initialized with random weights at first. As the training continues they adjust their weights until they are capable of capturing meaningful properties. Vigorous training with millions of images, thousands of classification labels and many hidden layers makes those filters extremely capable feature detectors.

Putting aside CNN for a while, from a pure image processing point of view an image kernel or convolutional kernel refers to a small matrix of fixed dimensions, say 3x3 or 5x5, which operates in the neighborhood of the same dimensions with the input image [27]. It can be used to make image processing effects like blurring, embossing, line detection, edge detection by doing convolution between the kernel and any given image. These images kernels matrices are fixed values matrices.

Table 7.1: Image kernels and result after convolution (greyscale)

Kernel Matrix	Image Result	Kernel Matrix	Image Result
Blur $1/9 \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$		**Gaussian Blur** $1/16 \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	
Left Sobel $\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$		**Top Sobel** $\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$	
Right Sobel $\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$		**Outline Edge** $\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8.0 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharp $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$		**Embross** $\begin{bmatrix} -2 & -1 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix}$	
Positive 45 $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$		**Negative 45** $\begin{bmatrix} -2 & -1 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix}$	
Horizontal $\begin{bmatrix} -1 & -1 & -1 \\ 2 & 2 & 2 \\ -1 & -1 & -1 \end{bmatrix}$		**Vertical** $\begin{bmatrix} -2 & -1 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix}$	
Random Float $\begin{bmatrix} 0.83 & 2.53 & -1.01 \\ -0.10 & -0.28 & 0.11 \\ 0.96 & 0.14 & -1.61 \end{bmatrix}$		**Random Integer** $\begin{bmatrix} -4 & 3 & 4 \\ 0 & 1 & -2 \\ -4 & -3 & 2 \end{bmatrix}$	

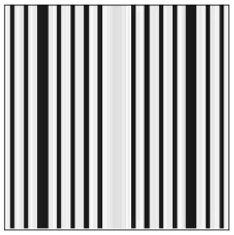


Figure 7.1: Input image

Table 7.1 illustrates commonly used image kernel matrices and their result after applying to the input image in Figure 7.1

Using the two concepts, convolution and the use of multi-layer network, we designed a simple untrained network of fixed convolutional filters. We will stick to a very basic network. We handpicked 14 convolutional filters ranging from blur filter to specific line detector filters. Inspired by the VGG [50] used in Chapter 4, we stacked multiple layers of convolution filters on top of each other. We also leveraged the idea of activations and dimensionality reduction by using ReLU and pooling layer (Max-pooling) respectively. We tried to tune our hyperparameters similar to the VGG network. That means our image stride and kernel size is the same as those used in the VGG network. However, a minor difference is we choose not to add zero padding around the border of the input layer because some line detector filters are detecting an edge on the boundary areas.

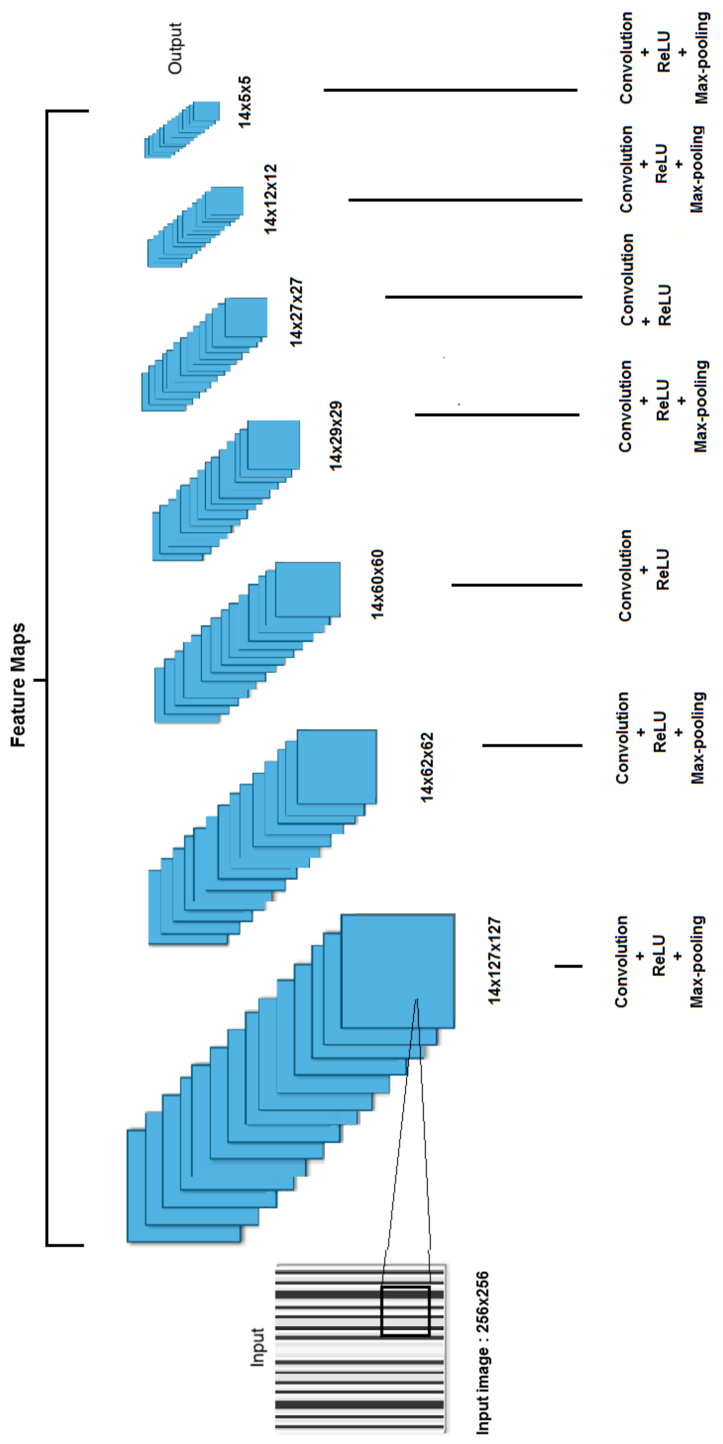

Input
Input image : 256x256
Feature Maps
14x127x127
14x62x62
14x60x60
14x29x29
14x27x27
14x12x12
14x5x5
Output
Convolution + ReLU + Max-pooling
Convolution + ReLU + Max-pooling
Convolution + ReLU
Convolution + ReLU + Max-pooling
Convolution + ReLU
Convolution + ReLU + Max-pooling
Convolution + ReLU + Max-pooling

Figure 7.2: Basic CNN architecture

Our input layer takes a 256x256 input image illustrated in the Figure 7.1. Then it goes through the first layer of convolution filters. Each of those 14 kernels applied to the input image individually will give 14 distinct feature maps after the convolution process. Then it passes through ReLU and Max-pooling. After that, like any other convolution networks, the next layer of convolutional filters (same as the previous layer) will perform convolution individually on those resultant feature maps from the previous layer. This process is repeated several times. This altering Convolution + ReLU + Pooling process continues until the final output layer. The architecture of the basic CNN is illustrated in Fig 7.2

Note that unlike the VGG network or other popular architectures, the number of feature maps in our network is of fixed size. Other networks usually have different numbers of feature maps in different layers. In terms of the deepness of the architecture, we choose to implement 7 layers. The stacking arrangement is inspired by VGG [50]. From Figure 7.2, we can see there is no pooling layer between 3 and 4. Also, there is no pooling layer between 5 and 6. The final output contains 14 feature maps of the dimension 5x5.

7.2 GP Integration

We will use the same data flow architecture from the previous CNN experiments (see Chapter 4 Figure 4.1). First, we feed a content image target to the basic CNN. It goes through all the layers until it reaches the final output. Next, the GP generated individual texture will go through the same network and we will calculate the MSE between the GP individual and content target from the final output layer. The MSE will be used as the GP fitness. All the GP parameters and texture language are listed in Section 5.1 in Tables 5.1 and 5.2.

7.3 Experiments and Result Analysis

7.3.1 Greyscale

We start by using single channel greyscale image in the input layer. The input layer matrix will be constructed as a 256x256x1 matrix. After passing through all the layers the final output is a 14x5x5 matrix. Figure 7.3 illustrates what 14 convolutional filter captures in its first layer. It is evident that different filters are capturing their specific properties. If the input image is a vertical stripe, then top Sobel, outline and vertical line detector filters are actually responding to those pixels. On the other hand, positive 45, negative 45 and horizontal line detector filters are not responding to any pixel values from the input image.

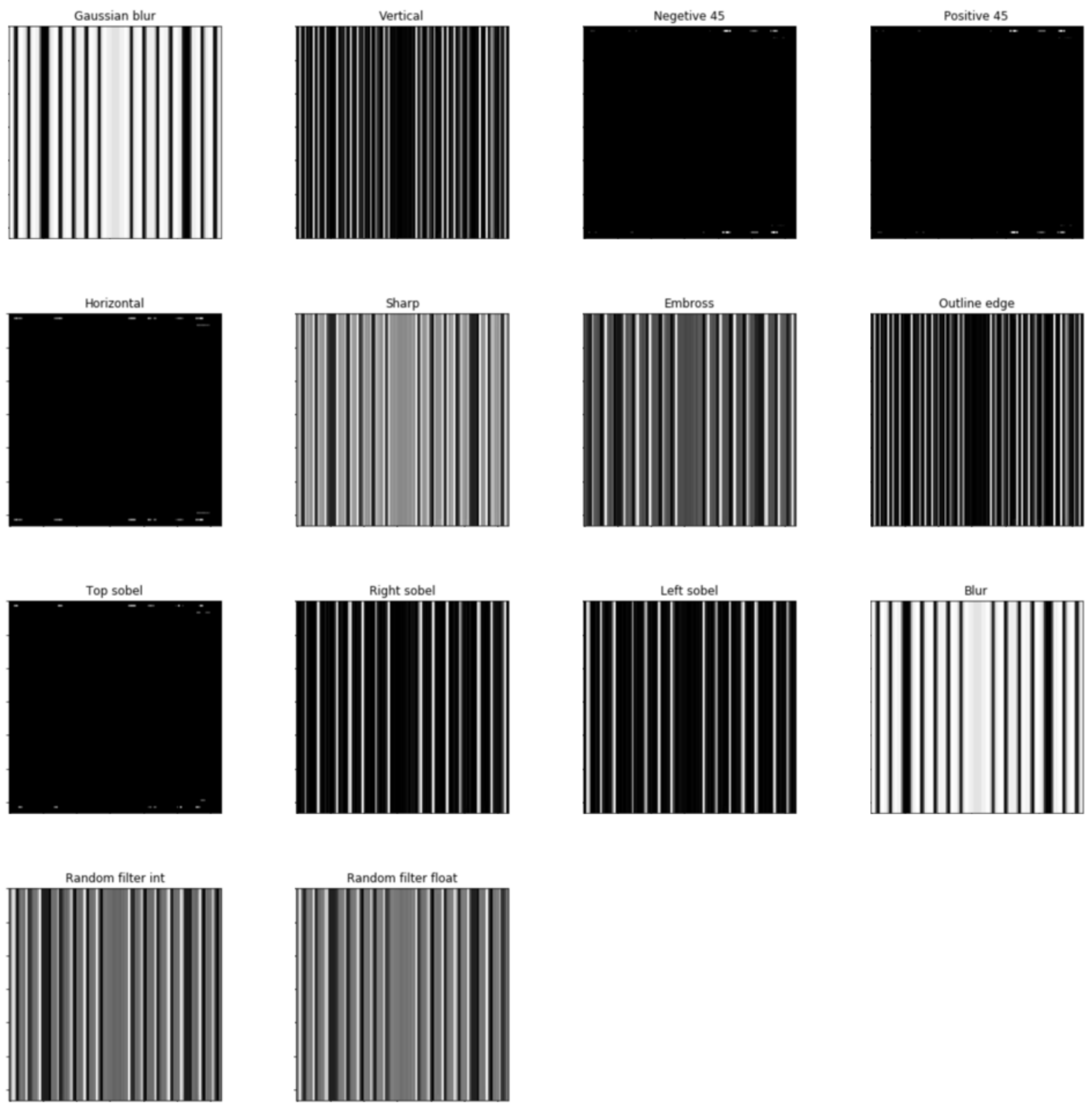
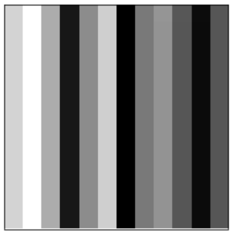
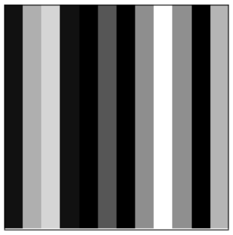
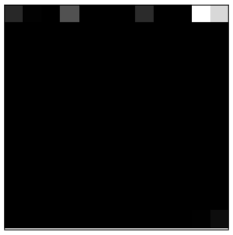


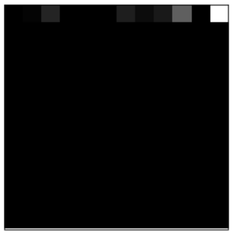
Figure 7.3: 14 feature maps from the first layer after convolution + ReLU + Max pooling

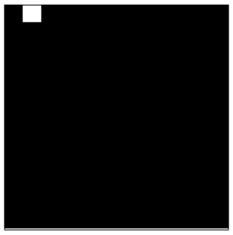
As we move from lower layers to higher layers, higher layer output reflects the same characteristics from the first layer, i.e., it is encoding the properties specific to the input image in very small dimensions of matrices. If we try to visualize this, we can convert the 2D activation maps into greyscale images by scaling the values between 0 to 255. Figure 7.4 is such conversion of second last layer (layer 6). The reason for showing the visualization of the 6th layer is that the final layer feature maps are too small to correctly visualize as an image. Notice how Figure 7.4d, 7.4e , 7.4c and 7.4i are almost black which indicates there is no activation.

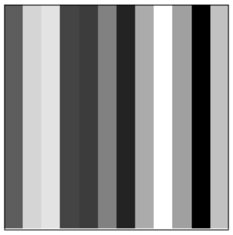

(a) Gaussian Blur

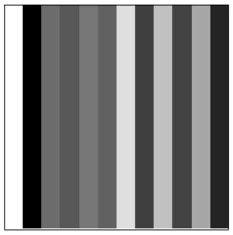

(b) Vertical

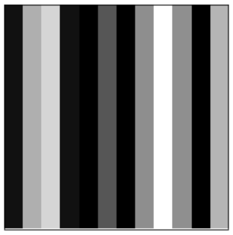

(c) Negative 45

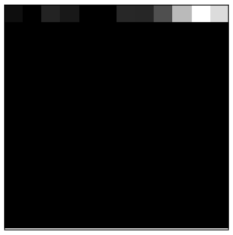

(d) Positive 45

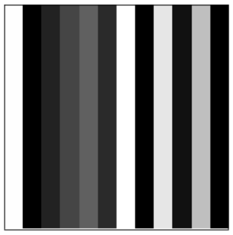

(e) Horizontal

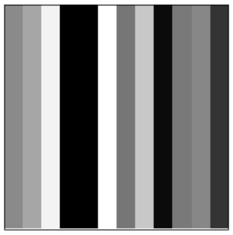

(f) Sharp

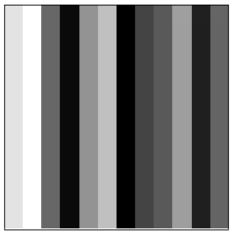

(g) Embross

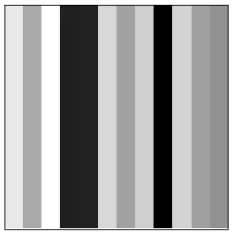

(h) Outline Edge


(i) Top Sobel


(j) Right Sobel


(k) Left Sobel


(l) Blur


(m) Random Integer

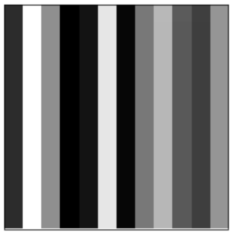

(n) Random Float

Figure 7.4: Visualization of activations of 14 feature maps from 6th layer

After all the pooling and ReLU, in the final layer, the feature maps are reduced to a very small patch of a 5x5 matrix. If we try to inspect the level of activation of the 14 features from the Table 7.2 we can see that the left Sobel, vertical filters are heavily activated since vertical line properties are present in the input image. On the other hand, positive and negative 45, horizontal and top Sobel is very lightly activated. The horizontal activation is almost zero. This proves that the final output encoded by our network is consistent with the input image characteristics.

Table 7.2: Mean activations of the 14 feature maps in the final layer(Layer 7).

Filter Name	**Mean Activation**
Embross	16652246.66
Left Sobel	16117603.37
Random filter (int)	15418029.52
Vertical	14257100.00
Outline Edge	14257000.00
Random filter(float)	13683788.20
Right Sobel	13253261.43
Sharp	12112331.20
Blur	7834109.10
Gaussian Blur	7707683.70
Top Sobel	3596.65
Positive 45	1287.54
Negative 45	189.46
Horizontal	0.20

With that in mind, we can say that the network is acting as a simple feature detector. We used this output layer as a GP fitness and let GP run for 30 generations. Figure 7.5 illustrates best results from 10 GP runs. Note that like previous pre-trained VGG network we did not reduce the number of feature maps in our first experiment. From Figure 7.5, it is evident that simple characteristics of the striped image are captured by our simple network is directing the evolutionary process in a specific way so that those characteristics are occurring in those GP produced resultant images. Horizontal and vertical lines are distinctly captured.

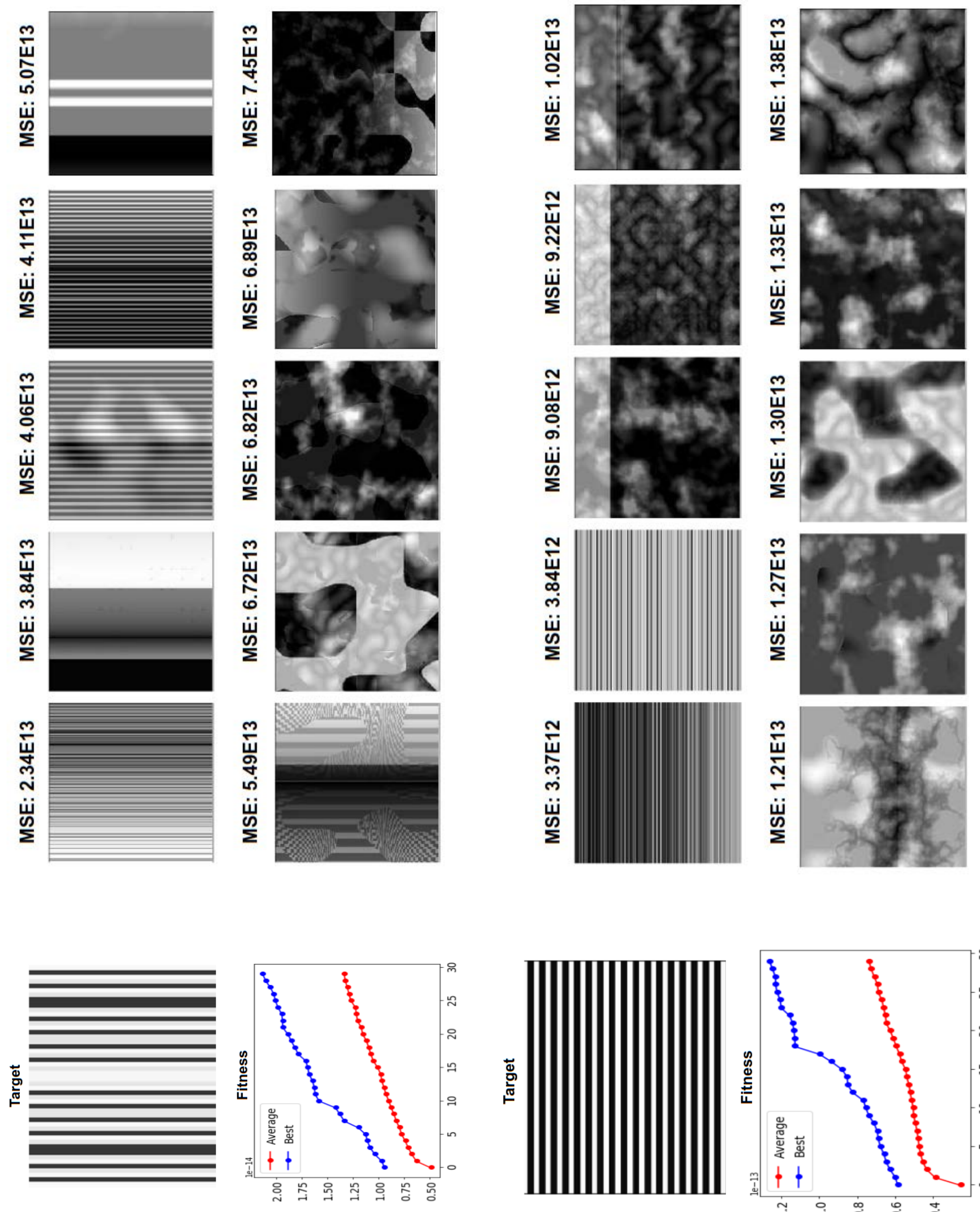


Figure 7.5: Best results of 10 runs. Standardized fitness plots are averaged over 10 runs.

Although simple images like horizontal and vertical stripes were successfully captured and encoded by the network, for a more involved image like a circle, results are less convincing (Figure 7.6). It is safe to say that those 14 filters and feature maps are unable to capture complex features like curves. Also, max pool greatly reduces the dimensions and causes invariance to small translations in our greatly simplified network. It is possible that the final layer output is unable to detect continues curved nature of input image.

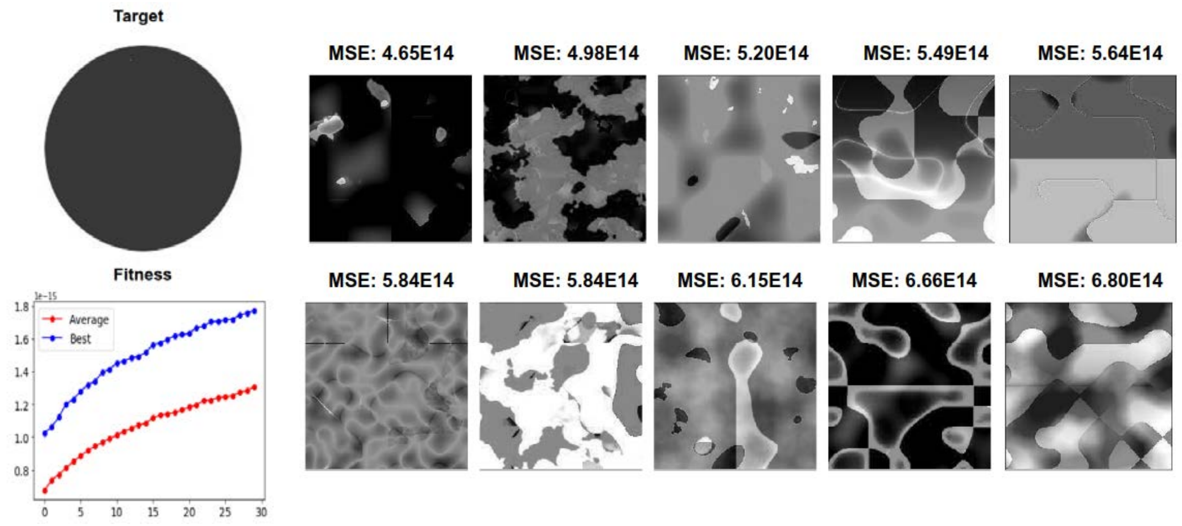
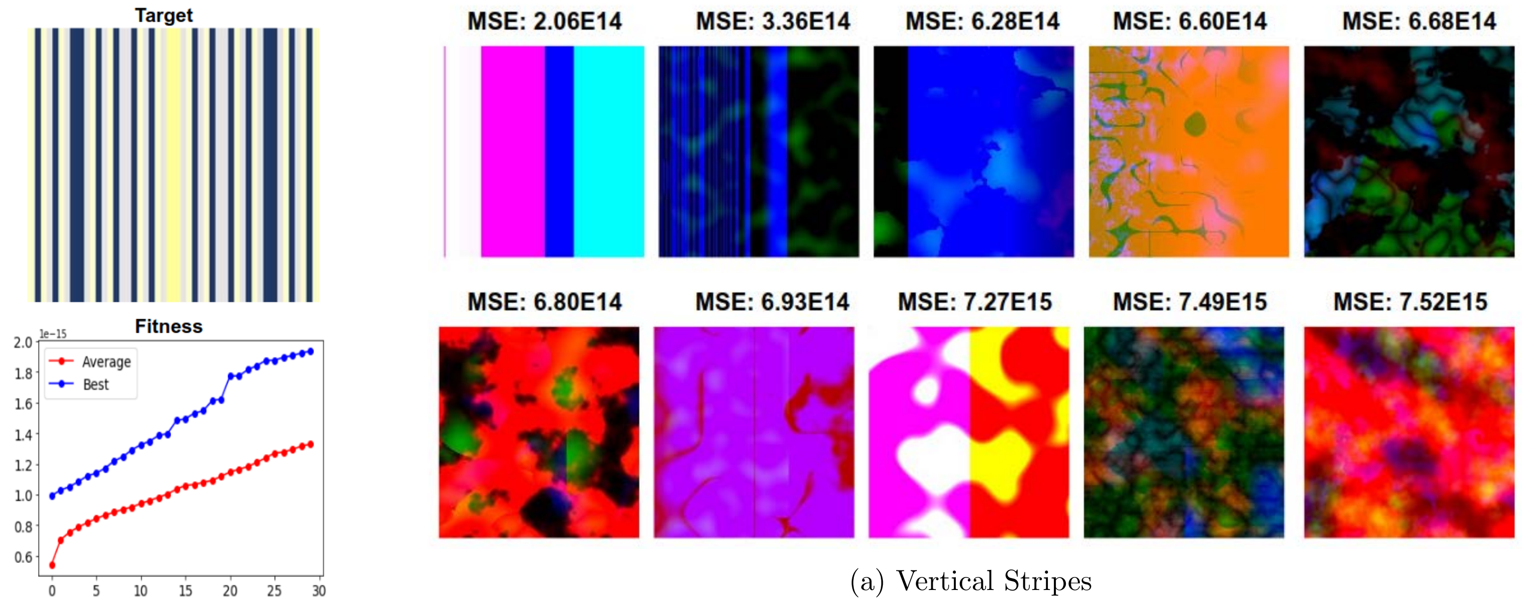


Figure 7.6: Best results of 10 runs. Standardized fitness plots are averaged over 10 runs.

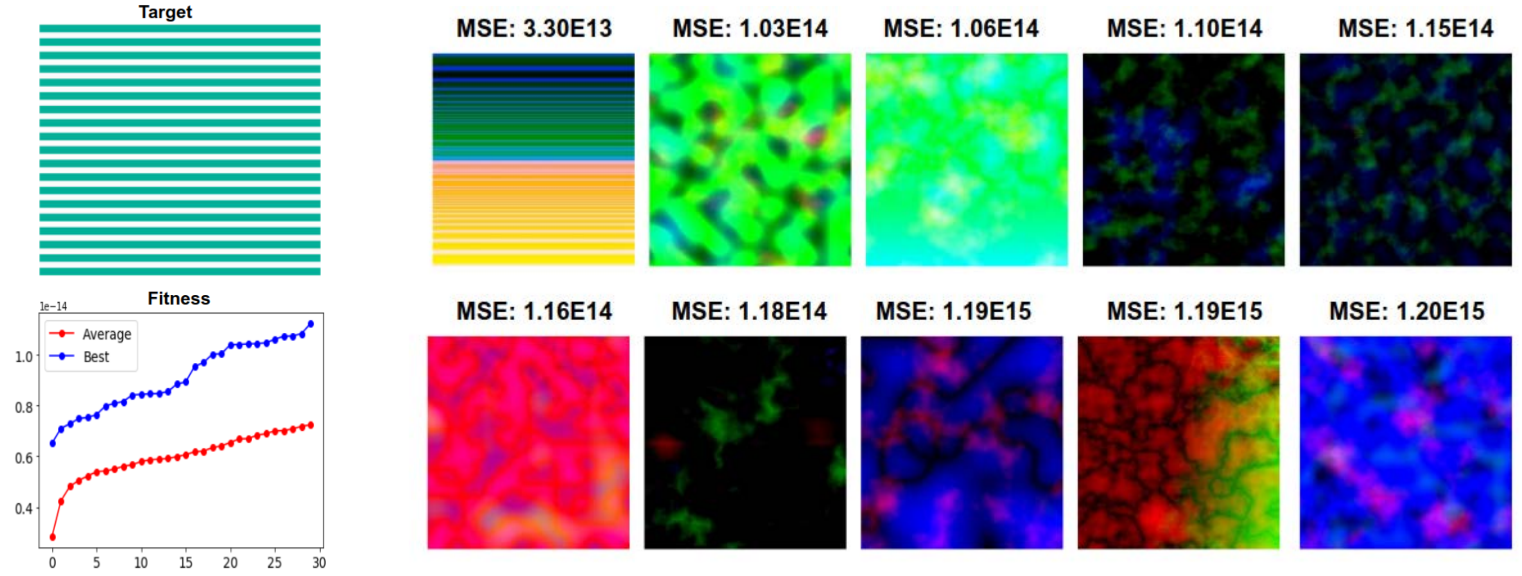
7.3.2 RGB Image

This experiment is a slightly modified version from the greyscale one described in Section 7.3.1. Instead of applying the convolution to a single channel greyscale image, we simply took a 3 channel RGB image as an input and performed multi-channel convolution from the very first layer. All the other configurations are identical to the previous greyscale experiment. The results are shown in Figure 7.7 and 7.8. The same observations from the previous section are still valid here. Note that the only difference between greyscale input and RGB input is the number of input channels. All the other configurations are the same as previous greyscale experiment in Section 7.3.1. The final target matrix dimension is 14x5x5 as before. After the first multi-channel convolution with the filters and the RGB input image, the internal hidden layer dimensions become similar to the Figure 7.3 and Figure 7.4.

From Figure 7.7a and Figure 7.7b, it is evident that the best results (lower MSE) from the vertical and horizontal experiment, are capturing the main characteristics of the content image i.e. the characteristics of the stripes. This is possible due to the presence of the line detector filters in our filter set. In the case of the circle (Figure 7.8a) content image, results are less promising. This is due to the lack of representation of a circular shape by our fixed filter set. In the case of the rectangular content image, the first two best results have rectangular shapes in the middle (Figure 7.8b). Since we have multiple line detector filters, capturing rectangular characteristics is possible in a limited way.

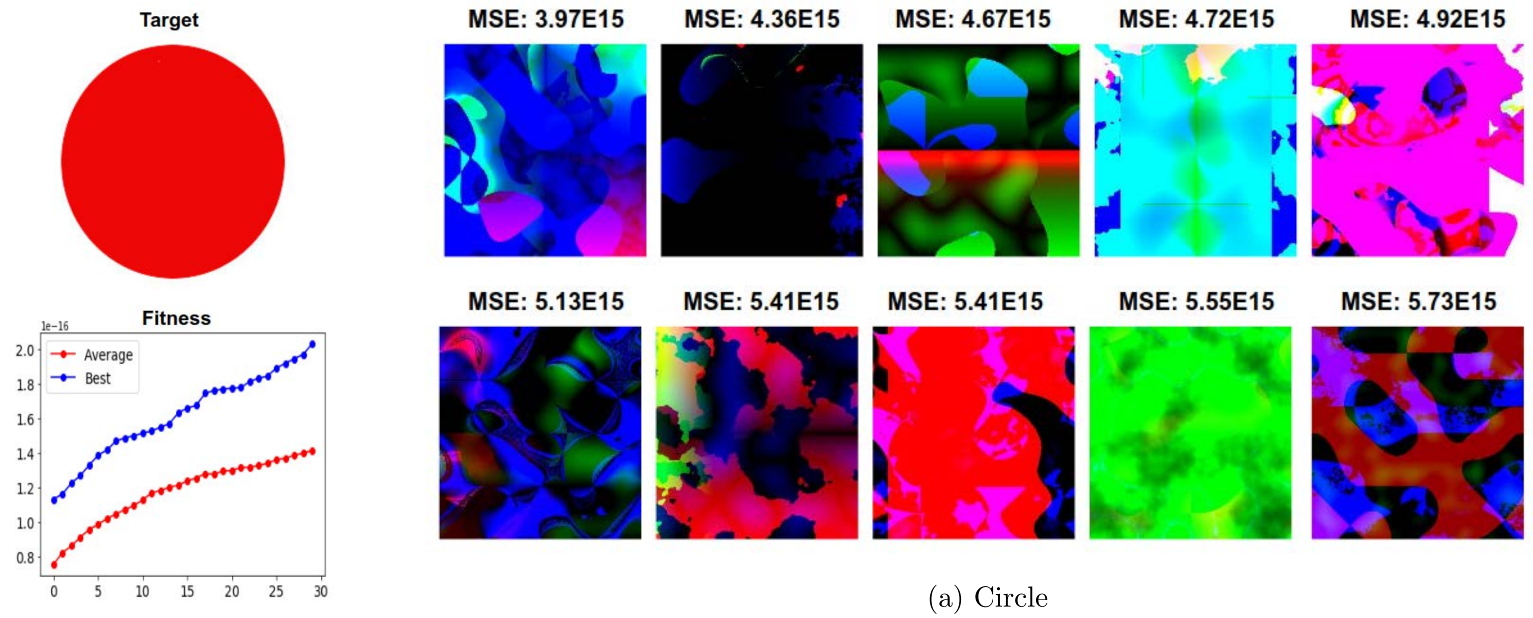


(a) Vertical Stripes

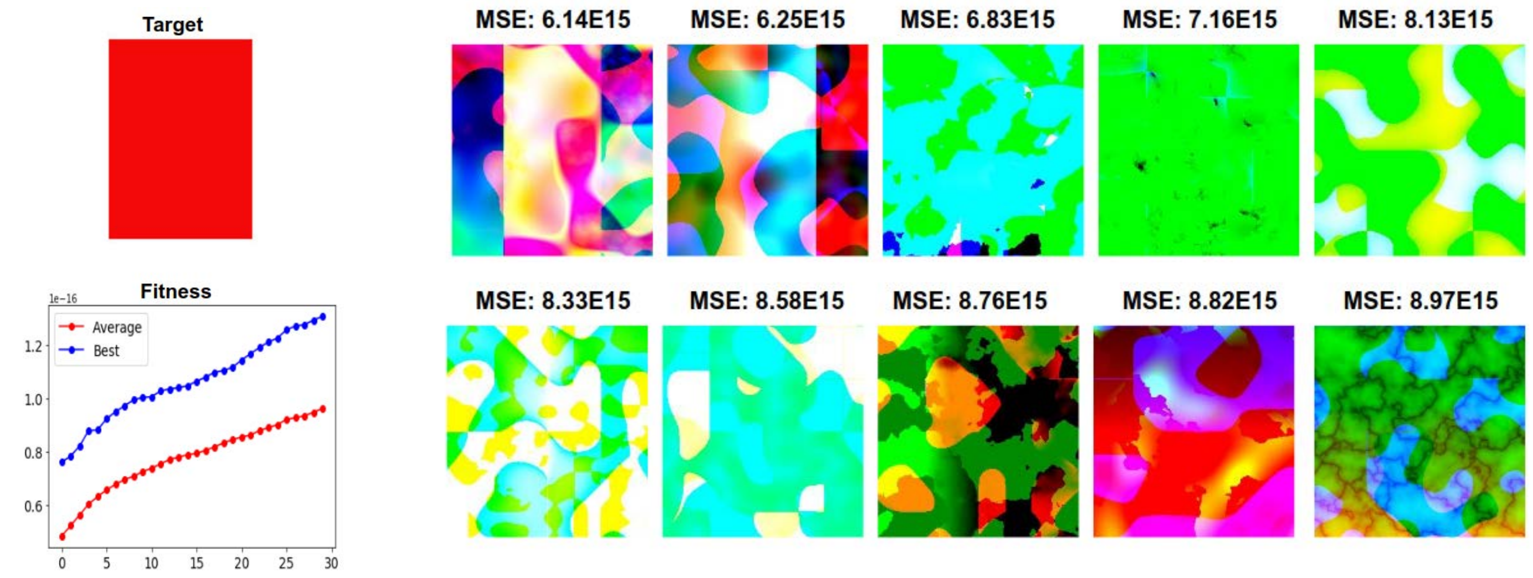


(b) Horizontal Stripes

Figure 7.7: Best results of 10 runs. Standardized fitness plots are averaged over 10 runs.



(a) Circle



(b) Rectangle

Figure 7.8: Best results of 10 runs. Standardized fitness plots are averaged over 10 runs.

Chapter 8

Conclusion

8.1 Summary

A deep CNN trained on millions of images forms a very high-level abstract representation of any given target image in its higher layers [22]. Our main purpose was to utilize this high-level information to guide the evolutionary process. In order to do so, we implemented a system which contained a robust GP system and a pre-trained deep CNN model. Using the high-level overview of the content image was challenging due to the tremendous amount of information which was too overwhelming for GP fitness. Also, the inner working of a deep CNN model is almost a "black box" and it is difficult to know what components will be the most useful for GP to use. With a careful investigation of the convolution layers and some fine-tuning of the components, we minimized the dimension of the convolutional layer significantly without losing content information. After that, we proposed a statistical pre-processing strategy called MMMS to select a number of relevant activation matrices to use as a fitness to the GP. It is a simple heuristic to identify relevant activation maps for specific target objects. Using the reduced number of activation matrices gave us promising results, i.e., evolved images retained main characteristics like shapes and patterns of the content image. After the initial success with MMMS, we ventured into multi-image combinations. This was more difficult. We suspect that the image combinations used directly cannot correlate with the training that was done with the deep CNN. The trained deep CNN probably never saw any image which is a combination of two classes.

Additionally, we investigate the idea of using a more high-level and abstract content information as the fitness to the GP. For that, we used the deep CNN's final output layer class scores which is generally used for image classification as the GP fitness. We performed the necessary experiments, and we had moderately good results for some specific cases. The best results produced by the GP scored very high for some specific classes objects. For example, when we specified the target class as peacock the result produced by the GP scored 99% confidence score by the deep CNN, and it definitely shared "peacock" features like feather colour and textured pattern of the feather. It shows what is sufficient for deep CNN classification. The deep CNN classifies an image by looking at the unique features. For example, to classify a peacock, it is sufficient to look for the green feather pattern rather than a complete peacock. In case of a tennis ball, it is sufficient to find lime colored texture without a shape of a ball to classify it. With all that said, we can safely assume that there is a difference between what pre-trained VGG needs for classification (images characteristics) versus what are the user's expectations of the evolved images. A trained deep CNN only looks for specific colour and pattern to classify a peacock whereas a user looks for a bird.

Finally, we implemented our own CNN without training to see if raw untrained activation matrices are of use. Instead of using a massive pre-trained deep CNN, we introduced our own shallow network which only contained 16 filters in each layer. We got slightly promising results only in basic striped images. This strongly suggests that training of network is important.

Based on all the experiment results, we can conclude that information encoded in the deep CNN model can be used as a strategy to guide evolutionary art. This tool has promise as a guide for image evolution.

8.2 Future Work

Our approach can be improved and further explored in many ways. First of all, the GP language we used was inherently inflexible due to the restricted nature of a simple texture language. The resulting images tend to produce images with a repeated textured pattern rather than retaining complex shapes. Noise also added some bias in the nature of resulting images. Using a more flexible and complex GP texture language could further improve results for certain content images.

Regarding the deep CNN module, multiple strategies could be undertaken for improving overall performance. The biggest problem with the trained deep CNN model is the lack of understanding of the inner workings of the hidden layers. The trained deep CNN model is a "black box", and the MMMS strategy tried to heuristically deconstruct the CNN to find the most important activations. We used variations of different object images paired into two groups. This might be improved if we could use more images of similar objects and paired them into multiple groups. Then, finding the common activations would be more accurate. It will be interesting to see if we can shuffle the images in all the groups and again apply the MMMS to find the common activations. This might narrow down the number of common activations further.

The statistical measure we considered could be improved further. We used the mean of the minimum matrix. More sophisticated statistical information may give us valuable insights into the activations filtration process.

This may be an indication of the importance of training. For target combinations, one could try more complex fitness strategies instead of using the simple sum of the MSE of the two images. Multi-objective optimization with ranking (e.g Pareto) could improve the results [47]. This is also applicable to the multi-class fully connected experiments as well. We saw that in the case of the fully connected multi-class experiments the results were poor. That was mostly due to the nature of the training of the deep CNN and the use of our oversimplified fitness strategy. Multi-objective fitness might help in this case. But again this can get complicated too. If it is the inherent nature of the encoding of the images, then the network will not have distributional statistics associated with the combination. This definitely needs more investigation in the future. Novelty search might improve the results also.

The basic CNN we implemented was quite shallow and, in every layer, we used the same sets of filters. Rigorous experiments with different arrangements of filters in different layers may further improve the results. Another approach is to perform own custom training with own intended styles of images to influence the CNN. For example, one could train the CNN with human expectations of a particular set of objects and train the CNN in a way that human expectation reflects the CNN classification. This could result in dramatically improved results, and maybe the best strategy for evolutionary art.

Finally, our main goal was to use a deep CNN model to guide the evolutionary process. We used the VGG-19 model as our deep CNN model. Currently there are many new DNN models published [48, 30, 53]. Using these new models might change the results significantly.

Bibliography

[1] Artificial neuron. https://commons.wikimedia.org/wiki/File:ArtificialNeuronModel_english.png. Accessed: 2018-06-06.

[2] Connection of input and convolution layer. https://commons.wikimedia.org/wiki/File:Conv_layer.png. Accessed: 2018-06-06.

[3] Convolutional operation. http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution. Accessed: 2017-09-16.

[4] ILSVRC 2012 dataset. http://image-net.org/challenges/LSVRC/2012/browse-synsets. Accessed: 2018-06-21.

[5] An intuitive explanation of convolutional neural networks. https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/. Accessed: 2017-09-16.

[6] Kenneth Stanley. Evolution of a spaceship. http://nn.cs.utexas.edu/demos/spaceship_evolution/rocket.html. Accessed: 2018-05-31.

[7] Large Scale Visual Recognition Challenge (ILSVRC). http://www.image-net.org/challenges/LSVRC/. Accessed: 2018-0-16.

[8] Maxpooling operation. https://en.wikipedia.org/wiki/File:Max_pooling.png. Accessed: 2018-06-16.

[9] Shumeet Baluja, Dean Pomerleau, and Todd Jochem. Towards automated artificial evolution for computer-generated images. *Connection Science*, 6(2-3):325–354, 1994.

[10] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. *Computer vision–ECCV 2006*, pages 404–417, 2006.

[11] Yoshua Bengio. Deep learning of representations for unsupervised and transfer learning. In Isabelle Guyon, Gideon Dror, Vincent Lemaire, Graham Taylor, and Daniel Silver, editors, *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, volume 27 of *Proceedings of Machine Learning Research*, pages 17–36, Bellevue, Washington, USA, 2012. PMLR.

[12] Peter Bentley and David Corne. An introduction to creative evolutionary systems. In Peter Bentley and David Corne, editors, *Creative Evolutionary Systems*, The Morgan Kaufmann Series in Artificial Intelligence, pages 1 – 75. Morgan Kaufmann, San Francisco, 2002.

[13] Philip Bontrager, Wending Lin, Julian Togelius, and Sebastian Risi. Deep interactive evolution. In *Computational Intelligence in Music, Sound, Art and Design: Proceedings of the 7th International Conference on EvoMUSART 2018*, pages 267–282. Springer, Cham, 2018.

[14] Soumith Chintala. Pytorch documentation. `http://pytorch.org/docs/master/`. Accessed: 2017-09-16.

[15] Soumith Chintala. PyTorch: Tensors and Dynamic neural networks in Python with strong GPU acceleration. `http://pytorch.org/`. Accessed: 2017-09-16.

[16] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2005*, volume 1, pages 886–893. IEEE, 2005.

[17] Richard Dawkins. *The Blind Watchmaker*. Norton & Company, Inc, 1986.

[18] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255. IEEE, 2009.

[19] Li. Deng and Dong Yu. *Deep Learning: Methods and Applications*. Foundations and trends in signal processing. Now Publishers, 2014.

[20] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *International Conference on Machine Learning*, pages 647–655, 2014.

[21] David Ebert, F. Kenton Musgrave, Darwyn Peachey, Ken Perlin, and Steven Worley. *Texturing & Modeling, A Procedural Approach.* AP Professional, USA, July, 1998.

[22] Leon Gatys, Alexandar Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Jun 2016.

[23] Leon Gatys, Alexander Ecker, and Matthias Bethge. Texture synthesis using convolutional neural networks. In *Advances in Neural Information Processing Systems 28*, 2015.

[24] Leon Gatys, Alexander Ecker, Matthias Bethge, Aaron Hertzmann, and Eli Shechtman. Controlling perceptual factors in neural style transfer. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Jul 2017.

[25] Michael Gircys. Image evolution using 2d power spectra. Master's thesis, Brock University, Dept. of Computer Science, 2018.

[26] David Goldberg. *Genetic Algorithms in Search Optimization and Machine Learning.* Addison Wesley, 1989.

[27] Rafael Gonzalez and Richard Woods. *Digital Image Processing.* Prentice Hall, Upper Saddle River, New Jersey, USA, 2002.

[28] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* MIT Press, 2016.

[29] Jeanine Graf and Wolfgang Banzhaf. Interactive evolution of images. In *Evolutionary Programming IV: Proceedings of the Fourth Annual Conference on Evolutionary Programming*, pages 53–65. MIT Press, 1995.

[30] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[31] Alaa Eldin M Ibrahim. *Genshade: An Evolutionary Approach to Automatic and Interactive Procedural Texture Generation.* PhD thesis, Texas A&M University, 1998. AAI9915243.

[32] John Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, MA, USA, 1992.

[33] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems*, volume 1, pages 1097–1105. Curran Associates Inc., USA, 2012.

[34] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[35] Chaun Li and Micheal Wand. Combining markov random fields and convolutional neural networks for image synthesis. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2479–2486, June 2016.

[36] Jing Liao, Yuan Yao, Lu Yuan, Gang Hua, and Sing Bing Kang. Visual attribute transfer through deep image analogy. *ACM Trans. Graph.*, 36(4):120:1–120:15, July 2017.

[37] David Lowe. Object recognition from local scale-invariant features. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 2, pages 1150–1157. IEEE, 1999.

[38] Sean Luke. ECJ: A Java-based evolutionary computation research system, version: 23. `https://cs.gmu.edu/~eclab/projects/ecj/`. Accessed: 2017-09-16.

[39] Penousal Machado and Amílcar Cardoso. Computing aesthetics. In Flavio M. de Oliveira, editor, *Advances in Artificial Intelligence*, pages 219–229. Springer-Verlag Berlin Heidelberg, 1998.

[40] Penousal Machado and Amílcar Cardoso. NEvAr — the assessment of an evolutionary art tool. In Geraint Wiggins, editor, *AISB'00 Symposium on Creative and Cultural Aspects and Applications of AI and Cognitive Science*, Birmingham, UK, 2000.

[41] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 427–436, June 2015.

[42] Ken Perlin. An image synthesizer. In *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*, pages 287–296, New York, NY, USA, 1985. ACM.

[43] Riccardo Poli, William Langdon, and Nicholas McPhee. *A field guide to genetic programming.* Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008. (With contributions by J. R. Koza).

[44] Juan Romero and Penousal Machado. *The Art of Artificial Evolution: A Handbook on Evolutionary Art and Music.* Springer Publishing Company, Incorporated, 2014.

[45] Steven Rooke. Eons of genetically evolved algorithmic images. In Peter Bentley and David Corne, editors, *Creative Evolutionary Systems*, The Morgan Kaufmann Series in Artificial Intelligence, pages 351 – 365. Morgan Kaufmann, San Francisco, 2002.

[46] Brian Ross, William Ralph, and Hai Zong. Evolutionary image synthesis using a model of aesthetics. In Gary G. Yen, Lipo Wang, Piero Bonissone, and Simon M. Lucas, editors, *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, pages 3832–3839, Vancouver, 6-21 July 2006. IEEE Press.

[47] Brian Ross and Han Zhu. Procedural texture evolution using multi-objective optimization. *New Generation Computing*, 22(3):271–293, Sep 2004.

[48] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 3856–3866. Curran Associates, Inc., 2017.

[49] Jimmy *et al.* Secretan. Picbreeder: A case study in collaborative evolutionary exploration of design space. *Evolutionary Computation*, 19(3):373–403, 2011.

[50] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[51] Karl Sims. Artificial evolution for computer graphics. *Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, 25(4):319–328, July 1991.

[52] Karl Sims. Interactive evolution of equations for procedural models. *The Visual Computer*, 9(8):466–476, Aug 1993.

[53] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[54] Christian Szegedy *et al.* Intriguing properties of neural networks. *CoRR*, abs/1312.6199, 2013.

[55] Stephen Todd and William Latham. *Mutator: a subjective human interface for evolution of computer sculptures.* IBM United Kingdom Scientific Centre, 1991.

[56] Stephen Todd and William Latham. *Evolutionary art and computers.* Academic Press, 1992.

[57] Andrea Wiens and Brian Ross. Gentropy: Evolutionary 2d texture generation. In Darrell Whitley, editor, *Late Breaking Papers at the 2000 Genetic and Evolutionary Computation Conference*, pages 418–424, Las Vegas, Nevada, USA, 8 July 2000.

[58] Jason Yosinski, Jeff Clune, Anh Mai Nguyen, Thomas J. Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. *CoRR*, abs/1506.06579, 2015.

[59] Matthew Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013.