Master's thesis

NTNU
Norwegian University of Science and Technology
Department of Computer Science

Theodor Astrup Wiik, Martin Nordby Johansen

Lamarckian Evolution of Pattern Producing Networks for Image Super Resolution

Master's thesis in Computer Science
Supervisor: Pauline Catriona Haddow
June 2021

NTNU
Norwegian University of
Science and Technology

Theodor Astrup Wiik, Martin Nordby Johansen

Lamarckian Evolution of Pattern Producing Networks for Image Super Resolution

Master's thesis in Computer Science
Supervisor: Pauline Catriona Haddow
June 2021

Norwegian University of Science and Technology
Department of Computer Science

Abstract

The field of single image super resolution has been a subject of research for many years. Interpolation has long been the industry standard. This method is flexible in terms of image input and output size. In the last decade, deep learning has gained popularity within image super resolution and has even outperformed interpolation. Such methods are often static in terms of their output size and thus have a limit to how large images they can input and output. This thesis explores the abilities of Compositional Pattern Producing Networks (CPPN) as a flexible method for single image super resolution similar to interpolation. As images come with varying characteristics and complexity, it is hard to design and optimize networks capable of producing detailed image recreations. The Lamarckian Evolution of Compositional Pattern Producing Networks algorithm (LE-CPPN), a combination of neural architecture search and backpropagation for CPPN optimization, was developed and tested. Experiments showed that the Lamarckian inheritance of weights significantly improves network performance compared to pure backpropagation training. CPPNs evolved by LE-CPPN, successfully produced detailed image recreations, and were further used to upsample images. The resulting upscaled images displayed interesting qualities, and the visual quality was satisfying. Regarding mathematical image similarity scores, the method scored lower than image interpolation.

Sammendrag

Feltet som innebærer å få høyere oppløsning på bilder har vært et emne for forskning i mange år. Matematisk interpolasjon har lenge vært en standard i industrien. Disse typer metoder er fleksible med tanke på bilders input- og outputstørrelse. Det siste tiåret har dyp læring fått større popularitet innenfor feltet, og har også oppnådd bedre resultater enn interpoleringsmetoder. Slike metoder er dog statiske med tanke på outputstørrelse og har derfor en grense for hvor store bilder de kan ta som input og output. Denne oppgaven utforsker muligheten til å bruke Compositional Pattern Producing Networks(CPPN) som en fleksibel metode for å få høyere oppløsning på enkeltbilder lignende interpolering. Siden bilder kommer med varierende karakteristikk og kompleksitet, ble det utviklet en metode for å optimalisere CPPN-er til hvert enkelt bilde. Lamarckian evolution of Compositional Pattern Producing Networks(LE-CPPN), en kombinasjon av nevralarkitektursøk og backpropagation ble utviklet og testet. Gridsøk og tilfeldig søk ble testet mot LE-CPPN, men klarte ikke å få like gode resultater. CPPN-er som utviklet seg med LE-CPPN klarte å produsere detaljerte gjenskapninger, og ble videre brukt for å få høyoppløselige bilder. Oppskaleringen av bilder med CPPN-er resulterte i bilder som fikk lavere matematisk score enn bilder som ble upsamplet med matematisk interpolasjon. Allikevel ble de visuelle egenskapene til bildene som ble produsert av CPPN tilfredsstillende.

Preface

The following master's thesis was conducted during the period of January 18th 2021 to June 2021. Parts of chapter 2 were based on work written for a pre-masters project in the fall of 2020. The work was supervised by Pauline Catriona Haddow, whom we wish to thank for introducing us to the world of Bio-Inspired Artificial Intelligence and providing invaluable feedback throughout the research process.

Theodor Astrup Wiik, Martin Nordby Johansen
Trondheim, June 9, 2021

Contents

List of Figures

List of Tables

Acronyms

ANN Artificial Neural Network. 7

CNN Convolutional Neural Network. 16

CPPN Compositional Pattern Producing Network. 9

GA Genetic Algorithm. 10

GAN Generative Adverserial Network. 17

HPO Hyperparameter Optimization. 20

LE-CPPN The Lamarckian Evolution of Compositional Pattern Producing Networks Algorithm. 22

LTU Linear Threshold Unit. 7

MLP Multi Layered Perceptron. 7

NAS Neural Architecture Search. 20

PSNR Peak Signal to Noise Ratio. 7

SSIM Structural Similarity Index Measure. 4

Chapter 1

Introduction

Ever since the digitalization of images, there's been a strong need for digital tools to rework and edit images. After 2012 the use of artificial neural networks enjoyed a surge in popularity after Krizhevsky et al. [17] utilized convolutional neural networks to successfully state-of-the-art performance on labeling images in the ImageNET contest. Since then there has been proposed a wide variety of AI models and architectures within the research field of image processing. Today, there are generative adversarial networks that can create seemingly realistic images of faces [21] from noise and variational autoencoders that efficiently encode and compress images.

Image super resolution is the concept of taking a low resolution image and increase the number of pixels to improve image quality and increase its dimensions. Classical mathematical approaches such as interpolation have for a long time dominated this domain. However, with the Revolution of AI in image processing neural networks have set new benchmarks in this domain, outperforming mathematical interpolation.

In 2009, Stanley et al. [35] introduced Compositional Pattern Producing Networks (CPPN). These are regular feedforward neural networks that map spatial coordinates to target values. Originally these were evolved and used to encode the weights of the neural networks, but the concept could also be utilized for creating images [33]. CPPNs have since then been used to create abstract art and even recreate small images from the MNIST dataset [9].

This thesis proposes a new method of utilizing CPPNs as a method

of achieving super resolution on images. A neuroevolutionary algorithm combining evolution and network training is proposed as a method of optimizing the performance and design of the CPPNs. The proposed model is tested and compared to mathematical interpolation and state-of-the-art AI achievements on standardized benchmarks.

The thesis is divided into chapters and will contain the following themes. The rest of chapter 1 will present the research goals and questions and the motivation behind them. Chapter 2 will introduce the background theory of the thesis. The basic concepts of Neural Networks, CPPNs, Genetic Algorithms, and image similarity measures will be presented. Chapter 3 will discuss state of the art within image super resolution, including models such as GANs, Autoencoders and CPPNs. Chapter 3 will also discuss relevant work within the field of neuroevolution and architectural search. Chapter 4 will present the proposed method in detail and explain the design choices that have been made. Chapter 5 presents the experimental plan, setup and results. Finally, Chapter 6 will discuss the findings of the experiments.

1.1 Goals and Research Questions

Research Goal *Explore the possibilities of using CPPNs for image super resolution.*

Research question 1 *How well will CPPN image recreation scale with increasing image complexity and size?*

Images come in varying complexities. Some can have many different colors and objects within the same frame, and some can be covered in a single color and pattern. When images become more complex they will also be harder to recreate as the CPPN needs to be more sensitive to small input changes. Image resolution can also impact the task of learning an image. With increasing image size, the image contains more information making the CPPN's task of learning the image increasingly hard. For a CPPN to upscale an image, the CPPN first needs to successfully recreate the original low resolution image.

Research question 2 *How can CPPNs best be designed and optimized for image recreation?*

Different images have different features and properties. One may be a portrait of a person, while another can be a landscape image. These differences may require CPPNs to have different network sizes, architectures and activation functions. Finding such hyperparameters to fit an image can be both a difficult and time consuming task. A genetic algorithm can automate this search through the hyperparameter space and converge on a hyperparameter configuration suitable for a given image.

Research Question 3 *What different qualities will CPPN super resolution display compared with those of traditional mathematical super resolution approaches?*

The most common way of resizing images is through pixel interpolation. These methods work very well but often come with unwanted effects. For example, keeping the square pixel pattern of low resolution images, or blurring details. CPPNs might display other, more or less desirable, qualities when used for super resolution.

1.2 Structured Literature Review Protocol

The literature review phase was initially guided by an interest in the field of neuroevolution. Learning about methods such as NEAT, HyperNEAT, and its extensions, sparked the interest in CPPNs and their abilities to create patterns across spatial dimensions. This paved the way for settling on a research topic and a thesis goal. Naturally, the following steps included researching the usage of CPPNs, image super resolution, and neural architecture search. Databases such as IEEE Xplore, Google Scholar, and Research gate were extensively searched for finding the papers relevant to the aforementioned fields. Papers With Code[1] provides a comprehensive and structured ranking of papers and their results. This list was thoroughly reviewed in finding state of the art techniques and performance within image super resolution and neural architecture search.

[1] paperswithcode.com; a website highlighting trending Machine Learning research and the code to implement it.

Chapter 2

Background Theory

2.1 Image Similarity Measures

It is easy for humans to compare two images and determine whether or not they are similar. However, this is a hard task for computers. In measuring the performance of a super resolution method having rigorous methods for measuring similarity is necessary. These measures are especially important when determining the degree to which a recreated image is adequately similar to the original image. This section will present two different image similarity measures; structural similarity index and peak signal to noise ratio.

2.1.1 Structural Similarity Index

Structural Similarity Index Measure (SSIM) was introduced in 2004 by Wang et al. [43]. The method offered an alternative of calculating image similarity based on the degradation of structural information, rather than pixel-wise errors. The motivation behind this method was to create a similarity measure that more closely resembles the way humans judge image similarity. SSIM uses three known properties of the human visual system: Luminance, Contrast and Structure. The similarity score is calculated as a combination of these three feature components and produces a value between -1 and 1. If two images have an SSIM of 1 they are completely identical, while -1 means they are very dissimilar.

Luminance of an image, denoted as μ_x is the mean value of its pixels and is calculated as shown in 2.1 where $_i$ represents the value of pixel i.

$$\mu_x = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{2.1}$$

The Contrast of an image, denoted as σ_x is the standard deviation of its pixels and is calculated as shown in 2.2 where x_i represents the value of pixel i.

$$\sigma_x = \left(\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu_x)^2\right)^{\frac{1}{2}} \tag{2.2}$$

The Structure of two images, denoted as σ_{xy} is the covariance of their corresponding pixels and is calculated as shown in 2.3.

$$\sigma_{xy} = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu_x)(y_i - \mu_y) \tag{2.3}$$

The Luminance, contrast and structure are then used to create a comparison of each feature between two images x and y. The Luminance Comparison component between images x and y is then calculated as in 2.4,

$$l(x,y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} \tag{2.4}$$

where c_1 is a constant included to maintain stability if the denominator equals 0. It is calculated from 2.5, where L denotes the dynamic range of pixels, typically 255 for standard images, and k_i is a constant.

$$C_i = (K_i L)^2 \tag{2.5}$$

The Contrast Comparison component between images x and y is then calculated as in 2.6.

$$c(x,y) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2} \tag{2.6}$$

The Structure Comparison component between images x and y is then calculated as in 2.7.

$$s(x, y) = \frac{\sigma_{xy} + C_3}{\sigma_x \sigma_y + C_3} \tag{2.7}$$

Finally, the three components are multiplied together as in 2.8 to give the structural similarity index value. $\alpha > 0$, $\beta > 0$ and $\gamma > 0$ and denote the relative importance of each component.

$$\text{SSIM}(x, y) = [l(x, y)]^{\alpha} \cdot [c(x, y)]^{\beta} \cdot [s(x, y)]^{\gamma} \tag{2.8}$$

Even though SSIM has been widely used as a similarity measure for images, it is highly sensitive to image noise. This is seen in figure 2.1. The two images to the right are very similar to the original, and have the same PSNR score. However, due to the increased variance, the middle image scores worse in terms of SSIM.

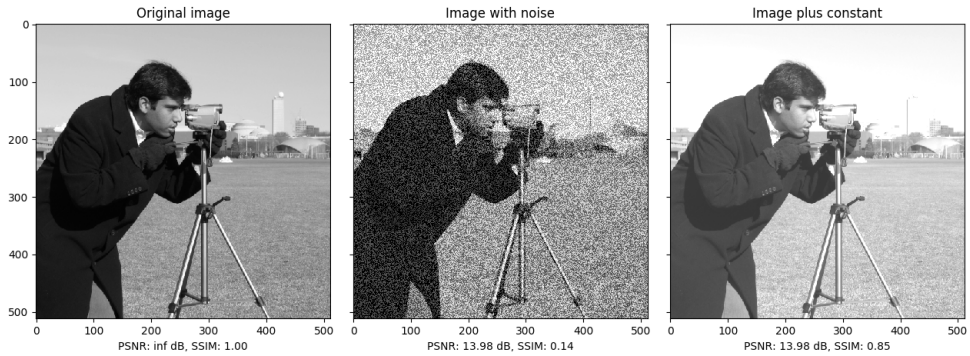


Figure 2.1: PSNR and SSIM of modified images to the original. The center image is the original plus random noise in the range of -0.2 to 0.2, and the image to the right is the original plus a constant of 0.2.

2.1.2 Peak Signal to Noise Ratio

Peak Signal to Noise Ratio (PSNRPSNR) is a measure of the ratio between the maximum strength of the original signal to the noise in the comparison image. The PSNR of two identical images would be ∞ as there is no noise present in the comparison image.

The equation for PSNR can be seen in equation 2.9

$$\text{PSNR} = 20 \cdot \log_{10}(\text{MAX}_I) - 10 \cdot \log_{10}(\text{MSE}) \tag{2.9}$$

where MAX_I is the maximum pixel value of the image, i.e 255 for an 8 bit pixel encoding, and

$$\text{MSE} = \frac{1}{mn} \sum_{i=1}^{n} \sum_{j=1}^{m} (I(j,k) - K(j,k))^2 \tag{2.10}$$

is the pixel wise mean squared error between the two images.

Figure 2.1 captures one of PSNRs weaknesses. The image to the right has lost less structural information than the one in the middle, yet the PSNR is the same in both images. This is because the PSNR only considers the MSE of each image, which in this case is the same.

2.2 Artificial Neural Networks

The first step towards artificial neural networks (ANN) came with the introduction of the perceptron [30]. A perceptron consists of a single linear threshold unit (LTU) that sums input values and applies the step function to output 1 if the sum is above a certain threshold and 0 otherwise. This architecture is illustrated in figure 2.2 which shows a perceptron with a 3-dimensional input layer, followed by a single LTU which sums its inputs and applies the step function to produce an output.

2.2.1 MLP

The Multi-Layer Perceptron (MLP) [8] is an ANN consisting of sequential layers of neurons where each layer is connected to the previous and the next. In such a network all layers between the input and output layer are called hidden layers. This architecture is displayed in figure 2.3. Each

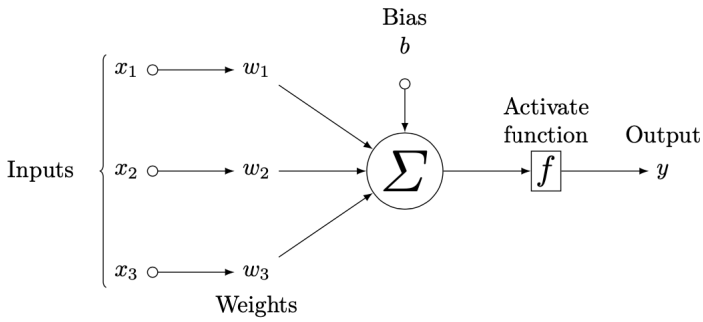


Figure 2.2: A three-dimensional input linear threshold unit

neuron in the hidden layers can be viewed as a single perceptron where the inputs are the outputs of the previous layers and its output is forwarded to each neuron in the next layer. However, whereas the perceptron simply uses a binary step function to decide its outputs, the neurons in an MLP can use a wide array of different functions. Usually referred to as activation functions, these functions output a real numbered value which is passed on to the next layer in the network. The purpose of the activation functions is to be able to capture nonlinear dependencies. With only linear activation functions an ANN would simply be multiple linear regression.

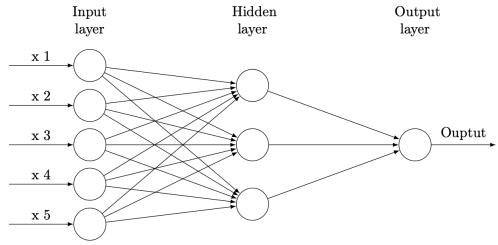


Figure 2.3: A multi layer perceptron

Optimization

The goal of the MLP is to approximate some function $\mathbf{y = f^*(x)}$ by learning the weight parameters θ, that best fit the mapping $y = f(x; \theta)$.

Traditionally, finding the optimal set of weights to approximate is done by the backpropagation algorithm [44]. Backpropagation requires a differentiable activation function as part of the algorithm is computing the gradients of each weight through the network. Weights can also be optimized through evolutionary algorithms by adjusting the weights through mutation and crossover enabling non-differentiable activation functions.

2.3 Compositional Pattern Producing Networks

Compositional Pattern Producing Networks (CPPN) is an umbrella term for feedforward ANNs that take spatial coordinates as input and produce a value for this specific point. The term was coined by Stanley et al. [35] for the Hyper-NEAT algorithm where such networks were evolved using the NEAT algorithm. These networks are often characterized by having many different activation functions across the network. When evolving CPPNs using the NEAT algorithm one can use any activation function since the weights are not optimized through backpropagation training but evolution.

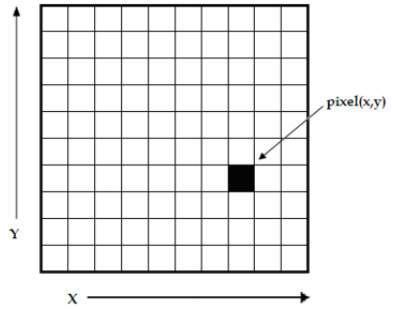


Figure 2.4: Pixel at coordinate $x = 8, y = 4$.

CPPNs can be used to create images by training the CPPN to approximate a mathematical function encoding of the image, *pixel at location* $(x, y) = f(x, y)$. The input layer of the CPPNs must be fed with coordinates representing the distance from the origin along the X and Y-axis. For an image of size N $\times$ M, each pair of coordinates from (0, 0) to (N, M) is propagated through the network and its output is mapped out on the

X- and Y-plane according to its corresponding coordinates. In figure 4.2 the black pixel is located in the 8th cell along the x-axis and the 4th cell along the y-axis. Thus, to get the value of that pixel from a CPPN one would feed the network with the coordinates (8, 4) and place the output in the corresponding cell.

2.4 Evolutionary Algorithms

Evolutionary algorithms draw inspiration from natural biology to obtain intelligent behavior. The main idea is to use natural selection and genetic variations to find the most optimized solutions to a given problem. Evolutionary algorithms use these ideas to simulate an environment with a natural selection pressure. Better solutions to a problem arise over generations of selection and changes to genetic material. This section will introduce three different evolutionary schemes that can be used when implementing evolutionary algorithms, before explaining the Genetic Algorithm(GA).

2.4.1 Evolutionary Schemes

Evolutionary schemes aim to set the rules and premise for how evolution will happen in an evolutionary algorithm. Darwinian evolution is the most commonly used, but it has been shown that the other schemes have worked better than Darwinian for specific problems [7]. For this reason, it is important to be aware of some different schemes that have been presented and know where they differ and where they are similar.

Darwinian Evolution

The most famous evolutionary scheme was presented by Charles Darwin in his 1859 book Origin of Species[5]. Darwin's theory of evolution by natural selection introduced a theory of how a population of individuals evolves. When a given population of an organism in an isolated area reproduces more than what their environment can support, only the individuals with the most favorable characteristics will survive and reproduce. This will turn into an internal tournament of *survival of the fittest* in which the fittest individuals survive. When only the fittest individuals in the population survive in the environment, the population will over time become

better adapted to their environment. For every new generation, there is a possibility of genetic mutation in parents' offspring. Changes in genetic material may cause the individual to form new characteristics that may better suit the environment, meaning that they are more likely to reproduce. When these changes occur over many generations, the newer populations will have developed characteristics that are significantly different from their predecessors and may have formed new species.

An important characteristic of Darwinian Evolution is that the individuals in a population are not actively trying to changes their characteristics, as their main goal is to reproduce. The changes in their genetic material only happen when new generations are formed, independently of the challenges their parents have faced. This means that the most important factor of evolution is selection, not the goal of developing new characteristics and abilities.

Lamarckian Evolution

Lamarck presented his evolutionary theory in 1809 [11], some decades before Darwin. He believed that the most important factor for evolution was the inheritance of obtained abilities and the struggle of an individual's predecessors. This means that the genetic material of an individual will change over its lifetime of learning and is passed on to its children. A common way to describe this concept is a Giraffe: a Giraffe will try to reach a tree's highest leaves, and over time it will stretch its neck. The giraffe's children will also have a stretched neck and may stretch it even more during its lifetime.

Baldwin Effect

The Baldwin Effect falls in between Darwinian and Lamarckian Evolution and was introduced by James Mark Baldwin in 1896[45]. The Baldwin effect describes, similarly to Lamarckian Evolution, that individuals are allowed to learn and change during their lifetime. However, where Lamarck allowed the genetic structure to change over the course of a lifetime, the Baldwin effect did not. The Baldwin effect relies on natural selection when reproducing. Moreover, the fitness of an individual is calculated with the learned behavior that is formed over its lifetime. The

difference from Lamarckian evolution is that the changed genetic material is not passed on to the offspring but rather the initial genetic material.

2.4.2 Genetic Algorithms

The genetic algorithm aims to simulate the evolutionary processes in a problem-specific space in which the goal is to find an optimal solution. These problems often have exponentially many solutions, making brute force search unrealistic. The selection pressure ensures that the genetic algorithm traverses the search space in the direction of better solutions. Thus the GA can find good solutions faster than a brute force method. The algorithm can be seen in figure 2.5.

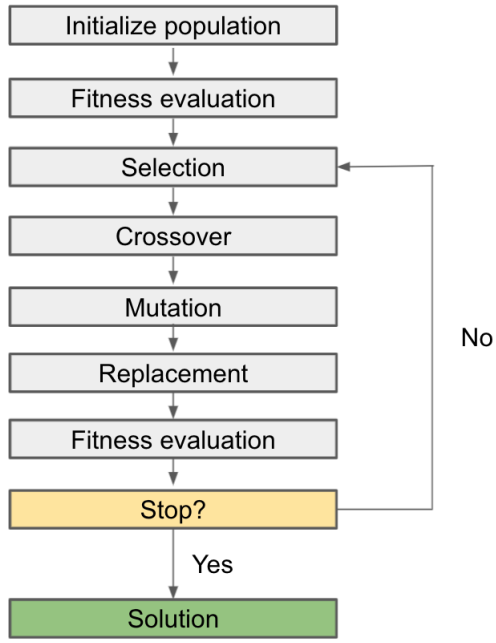


Figure 2.5: The steps of the Genetic Algorithm

The population in a GA consists of individual solutions to a problem. Each individual is represented by a set of variables called *chromosomes*,

and each variable is called a *gene*. When initializing the population, one usually assigns these variables randomly. Heuristic initialization may also be used, though such methods often take more time. Whenever a new population is created or initialized, each individual needs to be evaluated in accordance with a *fitness function*. The individuals with the highest fitness score are the ones with the best ability to solve the problem.

After this, the next step of the algorithm is the selection phase. This phase selects which individuals will become parents for the next generation. The selection method varies, but the general goal is to favour the individuals with the highest *fitness*.

When selecting individuals for reproduction, the main goal is to generate new combinations of genetic materials that have not been seen in previous generations. It is also important to choose different individuals, to keep the population diverse. Diversity in the population means that the population's individuals are genetically different. Population diversity is important in order to explore bigger areas of the problem space and prevents the algorithm from getting stuck in a local minimum or maximum. The selection algorithms used often vary, but tournament selection is a common approach. In tournament selection, a subset of the population is selected, and the individual with the highest fitness is selected with a probability p. If the best individual is not selected, the second-best individual is chosen with some other probability p until a candidate is chosen. The reason for having a stochastic selection method is to keep exploring the search space. If there is no exploration in the algorithm it can exploit a local minimum or maximum too quickly, and get stuck.

The individuals selected from the selection phase are then used to produce offspring, meaning that two individuals will combine their genetic material and form children. If two individuals, for example $[1, 1, 1, 1]$ and $[0, 0, 0, 0]$ are selected, offspring $[1, 1, 0, 0]$ may be formed. This type of crossover is called *single point crossover*. There are many different methods for crossover, but the overall goal is that the offspring produce solutions that still have all the necessary genes required for the problem, and that no problem constraints are violated.

After a new individual is produced through crossover, *mutation* might occur. This happens with a certain probability p, in which one or more genes' values change. With the example above, the third gene may mutate,

resulting in the offspring $[1, 1, 1, 0]$.

When the new offspring is created, the next step is to build the new population using the new individuals. Some methods use *elitism*, in which the best individuals of the previous generation are still preserved in the next generation. This is to preserve the best problem solutions across generations.

The process happening above is repeated during the evolutionary phase until some sort of condition is met. This can either be that a fitness threshold is reached, or if the implementer has put some sort of runtime limit to the algorithm.

When developing a genetic algorithm, it is important to try different parameters during experimentation. These include selection criterion and mutation rate, among others. It is important because they dictate the degree to which the system explores and exploits possible solutions. For example, when using a low degree of mutation, the algorithm will not explore many new solutions because there is little genetic change. This may cause the algorithm to get stuck in local minima.

2.4.3 Direct and Indirect Encoding

When using a genetic algorithm to evolve problem solutions, there are different ways of representing the problem. When the individual's chromosomes represent a direct solution to a problem, it is called a *direct encoding*. The genetic material of an individual is a direct mapping between the *genotype* and the *phenotype*. This is, however, not the only way to encode a solution. Sometimes the chromosomes are the ingredients needed to generate the solutions. This is called an *indirect encoding*. Indirect encodings allow the solutions to be bigger and more complex as it's only the genotype that is altered during mutation and crossover.

2.4.4 Genetic Algorithms with Lamarckian and Baldwinian Evolution

Lamarckian and Baldwinian evolution allow training during an individual's lifetime. During this training, either the phenotypic or the genotypic material can be altered. Whether or not these alterations are passed down to the offspring is what differentiates Baldwinian and Lamarckian

evolution. Baldwianian offspring will inherit the original unaltered genetic material from its parents, where Lamarckian inherits the altered material. In terms of evolving neural networks, one representation could be that the weights of the network constitute the phenotype and the architectural parameters of the genotype. The weights can be altered through back-propagation, and then either be passed down to the next generation or not.

Chapter 3

State of the art

3.1 Interpolation

The industry standard for image resizing is interpolation [24]. Interpolation calculates the value of a new pixel is a combination of its neighboring pixels. This method is considered extremely versatile since it can downscale and or upscale any image to any resolution. The problem, however, is that using interpolation for super resolution often retains artifacts of the low-resolution image such as checkerboard patterns and loss of smooth edges. These artifacts combat the entire point of super resolution as any human can easily see that the reproduced image resembles a low-resolution image.

3.2 Neural Network Based methods

After the success of Krizhevsky et al. [17], Convolutional Neural Networks (CNN) dominate the benchmarks of state of the art image processing problems. CNN's use of convolutional filters makes them especially good at feature extraction and representation. These abilities are what make them so effective for image processing. Chao et al. [3] introduced the first super resolution method using CNNs. The work introduced a deep CNN model which is trained to learn a mapping between low-resolution (LR) images and high resolution (HR) images. The model takes as input an LR image and outputs an HR image and its goal is to minimize the mean squared

error between the output and the original HR image. This approach needs high quantities of training images. To learn the mapping the model was trained on almost 400 000 pairs of LR and corresponding HR images. This method achieved a new state of the art for AI-based super resolution methods. Further work and improvements on such CNN-based mapping method mostly consist of designing better and deeper architectures to improve performance [6; 13; 14; 10; 51]. Lim et al. [22] achieved a significant performance gain with a proposed extra wide and deep architecture, showing that network architecture can significantly affect performance super resolution.

One drawback of this method is that the networks are prone to bias in their training data [40]. The Network can then learn mappings that do not generalize well across different images. The model can then produce unrealistic or unwanted details that are present in the training data but do not belong in the target image [4]. Also, CNN-based mapping networks have static input and output dimensions. This means that the model can only take input images of a given resolution and can only produce output images of a given resolution. Therefore one would need one separate model for scaling 100x100 images to 400x400 resolution, and another one for 100x100 to 600x600. This problem makes this model unsuited for day-to-day image scaling where flexibility of output size is needed.

Generative adversarial networks (GAN) [42] have gained interest in the past couple of years. GANs consist of two neural networks, one generative and one discriminative. This method trains the generative network to produce outputs that are then classified as right or wrong by a discriminator network. In terms of image super resolution, the generative network takes an LR image as input and produces an HR image. The discriminative network will then take this HR image as input and classify it as either a real HR image or a network generated HR image. Ledig et al. [19] first proposed using GANs for super resolution. What makes this method achieve superior results compared to the more basic CNN LR to HR mapping is that the generative network not only learns to minimize content loss i.e. pixel-wise mean squared error. Since the discriminative network learns to distinguish real HR images from reproduced/fake HR images the loss from the discriminative network represents features that typically distinguish real and fake HR images. This loss can be backpropagated all the way

to the generative network, providing a measure of perceptual loss to the generative model. The measure of perceptual loss makes the model able to generate details onto its HR output that otherwise are not present in the LR image. For example, a low resolution image has lost the detailed texture of grass, leaving only the color. A GAN can generate a grass-like texture, although not the same as in the original. This helps in making the HR images visually pleasing, which explains why this method has the highest Mean Opinion Score (visual quality evaluated by humans through surveys). Although the results can be visually pleasing, this often produces unwanted or unrealistic artifacts not actually present in the target image.

The introduction of attention mechanisms in deep learning introduced by Vaswani et al. [41] in 2017 created a new wave of novelty and innovations within the field of deep learning. Following this success, Zhang et al. [50] first introduced attention mechanisms to CNNs for super resolution achieving state of the art performance. The outputs of convolutional units in CNNs represent local feature activations and are unable to exploit spatial information outside of its local region [50]. Attention mechanisms, on the other hand, enables networks to learn the correlation between spatially independent features. By adding attention to a neural network it can more easily exploit interdependencies between features and learn what parts of an image are especially relevant.

One problem with these neural network-based super resolution methods is there no single model solves the case of general super resolution. As the resolution of the output image is dependent on the architecture of the model it can't be queried for an arbitrary resolution. It may be assumed that this is why interpolation methods are still the go-to for most photo editing software.

3.3 CPPN

Compositional Pattern Producing Networks were introduced as part of the HyperNEAT algorithm Stanley et al. [35]. In this algorithm, CPPNs are evolved to output a weight between two nodes in a target ANN. Effectively the CPPN encodes all connection weights in the target ANN and can be queried to reconstruct the weights by simply feeding the coordinates of the

Figure 3.1: Fox evolved by CPPN [Secretan et al.]

connection. The power of this encoding scheme was demonstrated when Fernando et al. [7] were able to reduce the number of parameters of a variational autoencoder network from 157684 to roughly 200 parameters.

Due to CPPN's ability to map spatial coordinates to values, they have gained popularity in creating images and abstract art [33] (see Figure 3.1). But, this ability is not restricted to abstract patterns. CPPNs can also be used for creating specific patterns or images. Fernando et al. [7] showed that CPPNs can successfully recreate small images of 28x28 pixels from the MNIST dataset. Their experiments showed a drastic increase in performance when using a combination of evolution and backpropagation training 2.4.1, compared to the simple NEAT algorithm. Further work on CPPN image recreation includes the Fourier-CPPNs introduced by Tesfaldet et al. [38]. Their work extended Mordvintsev et al. [25]'s implementation of trainable CPPNs and combined them with Fourier image analysis to successfully recreate missing textures within images. Both of these implementations used a predefined CPPN architecture instead of evolving one.

David Ha [9] used a CPPN as the generative network in a GAN model to create high resolution versions of small 28x28 images. His implementation included a noise-vector in addition to x and y coordinates in its inputs. The purpose of the noise vector is to work as a seed for the CPPN so that the network will produce a different image given the same coordinates but a different noise vector. A single CPPN was then able to recreate different images from different noise vectors. The ability of a single CPPN to create different images is a great advantage compared to CPPNs specialized at recreating a single image.

3.4 Hyperparameter Optimization and Neural Architecture Search

Hyperparameter optimization (HPO) is the task of choosing the correct hyperparameters for a neural network that is best suited for its specific task. This is a hard problem to solve as the number of network configurations grows exponentially when adding more configurations to the network. It is even harder to optimize when combined with neural architecture search (NAS), the problem of deciding the best possible architecture for a neural network.

NAS and HPO are problems that have been researched since the 1990s. Klein and Hutter [16] tried to create benchmarks and configurations that make it easier to evaluate different methods and compare them. Their research used four different regression datasets that are easily comparable. By testing 8 different HPO methods, they were able to show how the benchmarks could be used for future work when comparing methods. The tests showed that Bayesian optimization [12; 1; 34] methods perform well relatively quickly but do not obtain as high a performance as regularized evolution [28].

Wistuba et al. [47] state that it is not easy to choose a NAS for a given problem. They attribute this to the lack of baselines and benchmark datasets, which has made the comparison of different methods hard. Because different methods and research differ too much in terms of architectural search space, runtime, and data pre-processing, it is difficult to design a fair and standardized test. This is also reflected in Klein and Hutter [16] where the different methods only use the same amount of neural network layers. This makes the search space smaller and may be a disadvantage for methods that allow a larger search space. Wistuba et al. [47] do mention random search being a good baseline. In both Li and Talwalkar [20] and Sciuto et al. [31], random search performs at least as well as other established NAS methods, such as DART, ENAS, BayesNAS and NAO.

Real et al. [29] were one of the early successful attempts [47] at using a genetic algorithm to find CNN architectures for image classification. Their approach is very similar to the simple genetic algorithm with only a few modifications. Instead of performing crossover, individuals are simply

copied, mutated then trained between each generation. This algorithm allows the offspring to preserve its parents' weights, making it a variation of Lamarckian evolution. Real et al. [28] (2019) improved on this method and was able to find architectures that set new state of the art performance for image classification on the CIFAR-10 and ImageNet datasets.

Similar approaches of using evolutionary algorithms for NAS have been developed in other works [48; 23; 39]. These often generally follow the main steps of the genetic algorithm but vary in terms of choice of search space, mutation operators and selection functions.

Chapter 4

Method

The proposed CPPN optimization algorithm is a neuroevolution algorithm that evolves and trains CPPNs. This algorithm will henceforth be referenced as LE-CPPN (The Lamarckian Evolution of Compositional Pattern Producing Networks Algorithm). It was inspired by the success of Baldwinian and Lamarckian evolutionary versions of the NEAT algorithm [7], but modified to enable evolution of fully connected deep architectures. The goal of LE-CPPN is to evolve architectures and activation function configurations of the CPPNs. Each generation of CPPNs is trained to recreate a specific target image. In the end, the CPPN that received the best fitness when recreating the target image can then be used to recreate the image at an arbitrary resolution, by scaling the step length between inputs as described in 4.1.3.

This general overview of LE-CPPN is illustrated in Figure 4.1. The cycle repeats for a given number of generations after which the fittest individual is chosen as the final CPPN used for super resolution. The rest of this chapter is dedicated to explaining the components that make up the algorithm.

4.1 Compositional Pattern Producing Network Architecture

The compositional pattern producing networks evolved by LE-CPPN consists of an input layer of two nodes and an output layer of one or three

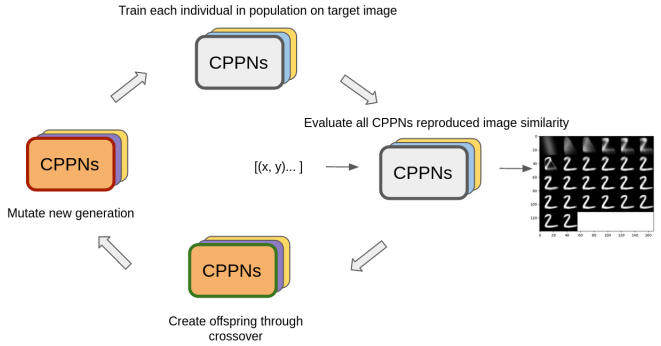


Figure 4.1: General overview of the hybrid evolution of CPPNs

nodes depending on whether the image is grayscale or RGB format. Each hidden layer, its activation function, and its size is chosen through evolution and trained by the evolutionary algorithm implemented for our model. Choosing to evolve the networks allows for an automatic search for activation functions and architectures best suited for each specific image. Since activation functions display different properties, some create repetition, some symmetry, and some negative values, different images can benefit from different activation functions [33].

In the CPPNs all nodes in a layer share the same activation function. This differs from the CPPNs traditionally evolved by the NEAT algorithm where each node in the network can have an arbitrary activation function independent from all other nodes. NEAT can evolve node-specific activation functions due to its direct encoding of nodes and connections [35]. On the other hand, sharing activation functions across all nodes in a layer allows for a more compact genome encoding and the algorithm to evolve larger networks.

The implication of this is that LE-CPPN can not optimize the activation functions at a node level to best fit an image, but only in a layer-wise fashion. Due to the complex nature of the problem at hand and the findings of Fernando et al. [7] discussed in 3.3 it was deemed essential to evolve deep and trainable networks. Thus this was a necessary trade-off.

Linear	x
Sigmoid	$\frac{1}{1+e^{-x}}$
ReLU	$\begin{cases} x, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases}$
Tanh	$\tanh x$
Sin	$\sin x$
Gaussian	e^{-x^2}

Table 4.1: Activation functions implemented for the CPPN.

4.1.1 Activation functions

The activation functions that were used for the CPPN are listed in table 4.1. Most are common neural network activation functions. However, both the sine function and the gaussian are not widely used in deep learning. These are included in the CPPNs as they have desirable features for image processing, where repetition and symmetry are common occurrences. The Gaussian function has been shown to capture symmetric features of a problem in a way others do not. Likewise, the sine function can capture repetition well[36].

4.1.2 Pre- and post-processing data

Before training, the pixel values of an image are normalized between 0 and 1. Also, the coordinates are normalized so that no pair is outside of the range (0,0) to (1,1). This means that for a 25x50 pixel image, pixels at coordinate (25, 50) and (15, 30) would be normalized to (1, 1) and (0.6, 0.6) respectively.

The input layer of the CPPNs is fed with coordinates representing the distance from the origin along the X and Y-axis. For each pair of coordinates from (0, 0) to (1, 1) given a step size N, the output of the network is mapped out on the X and Y plane according to its corresponding coordinates to create an image.

In figure 4.2 the black pixel is located in the 8th cell in the X Direction sell along the x-axis and the 4th cell along the y-axis. Thus, to get the value of that pixel from the CPPN one would feed the network with the

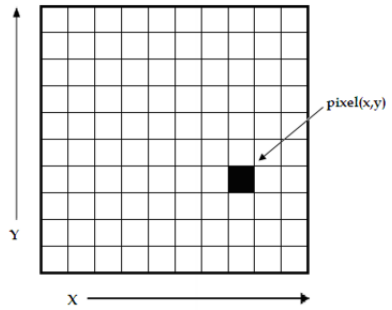


Figure 4.2: Image array

coordinates (8, 4) and place the output in the corresponding cell.

4.1.3 Super resolution

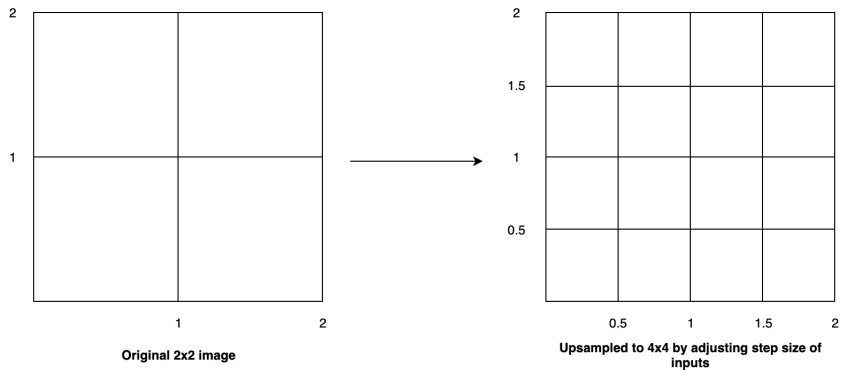


Figure 4.3: Scaling images by scaling step size

To achieve a higher resolution version of the image the CPPN has learned to recreate, one can simply decrease the step size between each pair of input coordinates. Say, in figure 4.3, a CPPN has learned to create pixel values for the coordinates of the original 2x2 image. Its original inputs would then be [(1, 1), (1, 2), (2, 1), (2, 2)] and the outputs would create a 2x2 image. Then by feeding the network twice as many coordinates along both the x and y axis the inputs of the network would

now be [(0.5, 0.5), (0.5, 1), (0.5, 1.5), (0.5, 2), (1, 0.5), (1, 1), (1, 1.5), (1. 2), (1.5, 0.5), (1.5, 1), (1.5, 1.5), (1.5, 2),(2, 0.5), (2, 1), (2, 1.5), (2, 2)]. The outputs from this increased input sampling would compose a 4x4 version of the original 2x2 image thus quadrupling the number of pixels. The step size between the pixel coordinate pairs can be arbitrarily small or large, thus the CPPN can produce any resolution.

4.2 Evolutionary Algorithm

The evolutionary algorithm that was used is a Lamarckian Genetic Algorithm, as explained in section 2.4.2, although two of the previously discussed evolutionary schemes, Baldwinian and Lamarckian evolution, are tested and compared to each other. The evolutionary process draws from Real et al. [29] discussed in 3.4 and Fernando et al. [7], but features some modifications.

The following subsections will present the evolutionary schemes' different genetic encoding, as well as the differences in their implementation. The genetic algorithm initialization, selection, crossover and mutation employed will be presented in the subsequent subsections.

4.2.1 Genome Encoding

As described in section 2.4.1, in Baldwinian and Lamarckian evolution all individuals have a lifetime of learning before being selected for reproduction. In the context of this research, this means their CPPN will be trained and evaluated before selection, differing from Darwinian evolution where there is no training of weights, only mutation. However, they are different when it comes to their offspring. In the Baldwinian model, the offspring will simply share the genetic material of their parents, the architecture and activation functions, but not their knowledge(network weights). In the Lamarckian model, the learned weights of the network are also passed down to the next generation instead of being randomly initialized.

LE-CPPN uses an indirect encoding that maps from an individual genome into its corresponding CPPN that will try to recreate the target image. The base genome, that is used by both the Baldwinian and the Lamarckian versions is comprised of two chromosomes: c_1 (layer sizes),

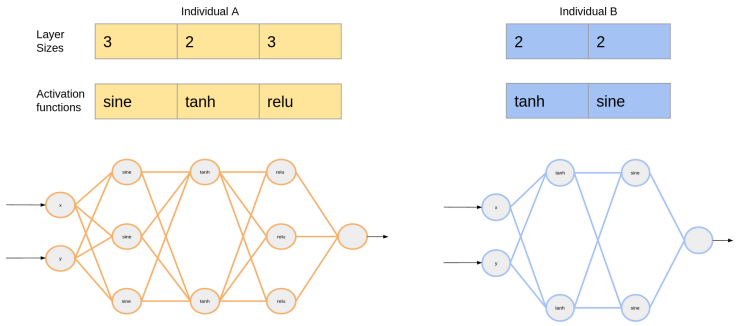


Figure 4.4: Genome encodings and corresponding CPPNs in LE-CPPN

and c_2 (activation functions). Chromosome c_1 is an array of positive integers that define the number of nodes in a specific hidden layer in the CPPN. The number of nodes allowed in a layer is defined by the hyperparameters n_min, n_max, both of which are positive integers. Chromosome c_2 contains an array of the activation functions that are used in the corresponding CPPN. Both c_1 and c_2 are of the same length, but they operate independently of one another, and one can mutate without the other doing so. In figure 4.4 a mapping from a genotype to its corresponding phenotype of two individuals is visualized. Each column in the table for the chromosomes represents a hidden layer in the corresponding CPPN.

4.2.2 Initial population

Each network in the genesis population is initialized with a simple architecture. The networks start with a single hidden layer with one of the defined activation functions selected randomly. Starting with 0 layers would result in a completely homogeneous genesis population. The number of nodes in the initial hidden layer is a randomly generated number, $n \in [n\_min, n\_max]$. The reason for having simple network architectures in the initial population is to allow network complexity to naturally emerge over generations of crossover and evolution. It is also important to avoid bias for certain parts of the search space. Having more complex initial networks may create a search bias for nonoptimal networks.

4.2.3 Evaluation

Before selection, the entire population is evaluated using the specified image similarity measure. Each individual draws an image with the same resolution as the target image and the similarity of the two are then used as the individual's fitness.

The individuals are evaluated after a fixed predefined number of CPPN training epochs. The fitness functions used for evaluation are Mean squared error and SSIM. The number of training epochs is a set parameter that holds for all individuals throughout the generations. Generally, more epochs will result in a better approximation of the CPPN's global maximum, but at the cost of computation and training time.

An important distinction to make is that the validation loss achieved during training is not used as a fitness measure and therefore disregarded in the selection process. As the validation loss is calculated from only a validation sample of size k of the training data, it is more accurate to evaluate on the entire image for the purpose of image recreation.

4.2.4 Selection

The genetic algorithm uses binary tournament selection. This method chooses two random candidates from the population. The individual with the best fitness of the two is selected for mating with a given probability p, or else the lesser fit individual is selected.

This method is run twice to select two parents who will then form offspring for the next population. The binary tournament selection scheme does not exert a very high selection pressure as an individual is as likely as any other to be chosen as a candidate. Only after the selection of the two candidates, the fitter individual has a higher probability of being chosen. To balance out the high rate of exploration, the algorithm can be configured to employ elitism by letting the N-best individuals survive from one generation to the next. These individuals' networks will not be trained further during their lifespan as this would give an unfair advantage.

4.2.5 Crossover

Contrary to the genetic algorithms for NAS discussed in **??**, LE-CPPN, performs crossover. After selection, single point crossover is performed on

two selected individuals. For the Baldwinian model, showcased in figure 4.5, two integers $n_1, n_2 \in [1, \text{size(functions) - 1}]$ are randomly generated for single point crossover. n_1 is the point at which the function layers will be combined, and n_2 is the point where the layer sizes will be combined. Which individual that comprises the first part of the offspring, is chosen randomly. If one individual has more hidden layers than the other, the remaining layers will be appended to the end of the resulting genome if and only if the longest individual has higher fitness. This is to favor smaller networks as long as they perform as well or better. Larger networks require more computation and take longer to train.

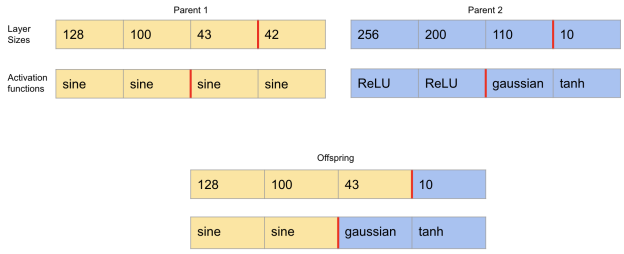


Figure 4.5: Baldwinian Crossover

The Lamarckian version performs crossover differently to the Baldwinian version, as shown in figure 4.6. A randomly generated number, n_1 is used to decide the crossover point. This is to preserve as many network weights as possible for the offspring to inherit. Only at the crossover point, the weights can not be preserved as the network structure is changed, and may no longer be compatible. The new weights created for this layer, are randomly initialized.

4.2.6 Mutation

The mutations that can occur are the same in both Baldwinian and Lamarckian evolution. Mutations will alter the genome with a probability $p \in [0, 1]$ such that new networks are generated for the new individuals. Although the changes to the genomes are similar, the resulting networks

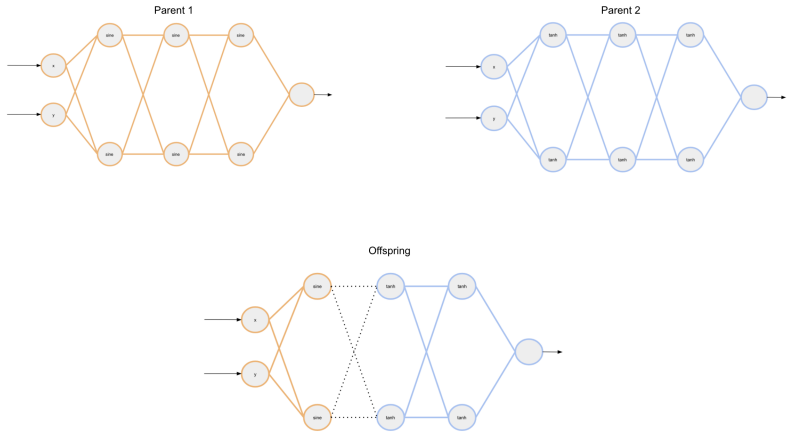


Figure 4.6: Lamarckian crossover. Stippled lines indicate initialization of new weights.

for Lamarckian and Baldwinian evolution will not be the same. The resulting CPPNs that emerge in Lamarckian evolution inherit the network weights of their parents, but when mutations occur, old weights can become incompatible with the new architecture. Similar to crossover, the layer at which a change happens, the layer's weights may need to be reset.

LE-CPPN uses the following mutation methods:

- **Add layer:** When adding a new layer, its size and activation function are selected randomly from the specified configurations seen in table 5.2. The layer index at which the new layer will lie is also chosen randomly. This mutation is important in order to explore deeper networks. More complex problems usually require deeper architectures.
- **Remove layer:** When removing a layer, a random layer in the genome is chosen and deleted. This is important in order to avoid too deep networks.
- **Mutate size:** When mutating the size, a random layer in the genome is chosen, and the number of nodes is changed to a randomly

assigned number. This is done in order to allow change in network structure without altering its activation functions. A genome may have the correct activation function, but the amount of nodes in the layer might not be optimal.

- **Mutate activation:** When mutating activation, a random layer in the genome is chosen, and its corresponding activation function will change to a different function within the specified set. When mutating activation function, the Lamarckian model will not change any network weights, as all layers are still compatible.

All the different mutations are allowed to happen when a new individual is created. Each mutation operator has the same probability of happening. This allows the CPPNs to change as much as possible and explore the search space. In addition, it is impartial to one mutation happening over another. This is especially important with regard to network size. If there is a different probability of adding a layer than removing one, the network will become increasingly more complex rather than converging towards an optimal size.

4.3 Hyperparameters

The hyperparameters that need to be defined for the model are divided into two categories, the LE-CPPN parameters and CPPN parameters, and can be seen in table 5.2. The CPPN epochs and batch sizes are not evolved and are the same for each individual. If these parameters were part of the genome, and subject to evolution, the competition would be unfair. Networks with higher epochs would be favored as they are allowed more training time. Another reason for not moving the hyperparameters to the genome level is to reduce the problem space. By adding more chromosomes, the problem's complexity increases as well, making the task more time-consuming.

LE-CPPN	Values
mutation rate	$p_1 \in [0, 1]$,
population size	$i \in \mathbb{N}$
generations	$g \in \mathbb{N}$
elitism number	$n \in \mathbb{N}$
parent selection probability	$p_2 \in [0, 1]$
evolution scheme	Lamarckian, Baldwinian
fitness function	MSE, SSIM
CPPN	
epochs	$e \in \mathbb{N}$
batch size	$b \in \mathbb{N}$
min layer size	$n_{min} <$ max layer size, $n_{min} \in \mathbb{N}$
max layer size	min layer size $< n_{max}, n_{max} \in \mathbb{N}$

Table 4.2: hyperparameters for LE-CPPN as well as each individual CPPN

Chapter 5

Experiments and Results

This chapter presents the experiments aimed to answer the questions posed in Section 1.1. Three experimental phases were conducted to answer the three research questions. The first section of this chapter will explain the preliminary testing executed during the implementation of the system, and how this guided choices for the final experiments. Further, the experimental plan and experimental setup will be presented, and finally the results.

5.1 Preliminary Testing

To evaluate the feasibility and scope to which the model was capable of recreating images, preliminary tests were conducted on a small set of images from the MNIST[18] dataset. MNIST is a collection of images of hand-drawn numbers, with a size of 28x28 pixels. The reason MNIST was used, was because the small size of the images would allow the model to finish relatively quickly. The hypothesis was that the CPPN would be able to recreate the images to a relatively high degree, acting as a green light for further research on more complex images.

The preliminary tests also helped gain insight into the model's hyperparameters, especially LE-CPPN's hyperparameters such as mutation rate and population size. It showed necessary to use a relatively high mutation rate to explore the problem space to a higher degree, while elitism would ensure not losing the best individuals.

One of the target images used for the preliminary tests can be seen in figure 5.1. The hyperparameters used are listed in table 5.1. The configuration was run on 10 images. Figure 5.2 shows the recreation of the best-performing individual from the first to the last generation. Here, one can see the emerging accuracy of the evolved CPPNs over each generation. The results showed that it takes LE-CPPN relatively few generations before a similar image is recreated, making 25 generations excessive. This can also be seen in figure 5.3 and 5.4.

mutation rate	0.3
population size	15
generations	25
elitism number	1
parent selection probability	0.7
evolution scheme	Baldwinian
fitness function	MSE
epochs	50
batch size	5
min layer size	4
max layer size	100

Table 5.1: Hyperparameters used for preliminary testing.

Figure 5.3 shows that little improvements were made after 10 to 15 generations. This is also reflected in the validation loss in Figure 5.4 where the training loss from generations 10, 15 20 and 25 follows the same trajectory and ends up with a similar validation loss. However, since the MNIST-images are so small, it was deemed necessary to keep the higher number of generations for more complex images.

The preliminary tests showed promise in recreating simple images and enforced the belief in successfully using CPPNs for more complex images. The hyperparameter tuning allowed us to find suitable parameters for the algorithm and almost all the parameters were reused for the final experiments. The CPPN parameters such as batch size and the number of training epochs were changed for later experiments as these significantly affect computation time with increasing image size.

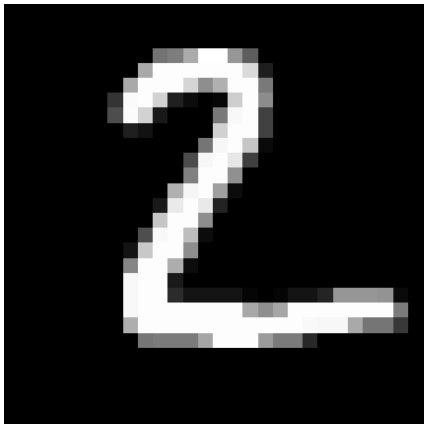


Figure 5.1: Target MNIST image.

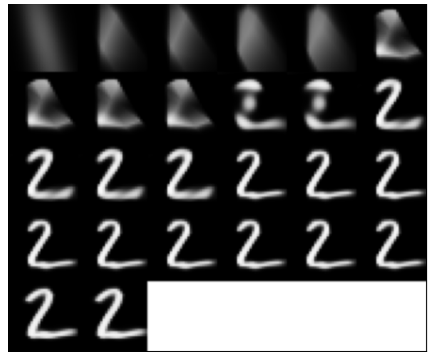


Figure 5.2: Best CPPN recreation from first to last generation on MNIST(left to right).

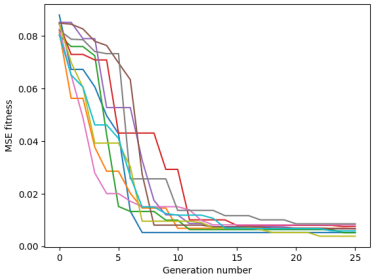


Figure 5.3: Fitness improvement over generations for each image.

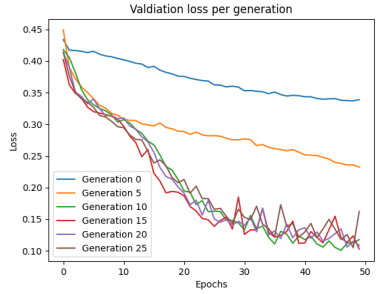


Figure 5.4: The validation loss over time of the best individual's CPPN of each generation from one run.

5.1.1 Dismissing Baldwinian Evolution

Both Baldwinian and Lamarckian evolution were used during the preliminary testing. It was initially meant to compare both the evolutionary methods during the ensuing experiments. However, after the preliminary tests, it was decided to only continue with Lamarckian evolution. In figure 5.5, a fitness comparison of Lamarckian and Baldwinian evolution run on

10 different MNIST images is visualized. This experiment was run with 200 epochs. The line is the average best fitness for all 10 images per generation, and the shaded area is the extreme values. Lamarckian improves much quicker than Baldwinian, and the spread between the worst and best individuals is generally smaller. Even though Baldwinian performs well, it is not able to obtain the same fitness as Lamarckian on a single image. The Lamarckian models also improve faster and have a smaller spread in the end results than what the Baldwinian model had. This is also reflected in the performance metrics, where Lamarckian had an average SSIM and PSNR of 0.9991 and 44.3057dB, and Baldwinian had 0.9707 and 28.2936dB. With a p-value of less than 0.05 for both metrics, it is also statistically significant that the Lamarckian model performs better than the Baldwinian model. The relatively high difference in PSNR compared to SSIM is due to the logarithmic nature of PSNR, making it sensitive to changes when MSE is close to zero. Two identical images will have a PSNR of infinity.

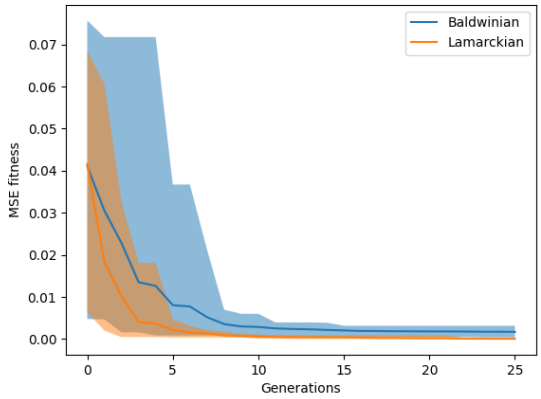


Figure 5.5: Average of best fitness per generation for all 10 images. Lamarckian and Baldwinian evolution.

These results are also seen in the models' resulting images. Both models recreate the MNIST images well, although Lamarckian is almost identical, capturing nearly all the details. This is most easily visualized in

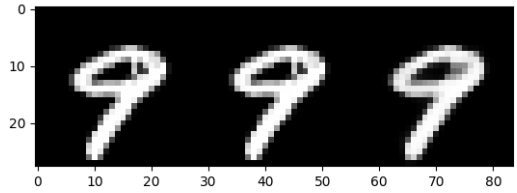


Figure 5.6: Comparison of MNIST(left), Lamarckian recreation(middle) and the Baldwinian recreation(right).

MNIST's 10th image, shown in figure 5.6, where the small line inside the 9's ring is not seen in the Baldwinian recreation but is easily seen in the Lamarckian.

5.2 Experimental Plan

The experiments conducted for this thesis aim to answer the different research questions posed in section 1.1. They are split into three phases, focusing on their respective question. The MNIST- Set5- and Set14-datasets [18; 2; 49] will be used for the different experiments. The MNIST dataset contains images of 28x28 resolution images, which is very low compared to standard images. These will serve as a benchmark for comparing CPPN recreation of larger images. Set14 Set5 contains images within a normal resolution range, with images from 100 to 768 pixels wide. All three datasets are used as standardized benchmarks for several computer vision tasks. Set14 and Set5 were chosen because they are the state of the art standard and super resolution results are easily accessible.

Phase 1: Feasibility

Phase 1 is linked to RQ1 and the feasibility of the proposed method. The method should be able to recreate images of any size in order to later produce upsampled versions. Larger images contain more information and should thus be harder for a network to learn. To gauge the feasibility of using CPPNs for super resolution, LE-CPPN needed to be tested on

images of varying complexity. Comparing the image similarity metrics from runs on both small and large images should provide insight into how size affects the recreational abilities of LE-CPPN.

RQ1 *How well will CPPN image recreation scale with increasing image complexity and size?*

Experiment goal: Test LE-CPPN on the MNIST- and the Set5-dataset. Analyze the ability to recreate images by comparing the SSIM and PSNR of the recreated images of each dataset.

Hypothesis: It is expected that the CPPN recreations will be able to recreate the general structure of the images, such as face outlines. However, it is expected that the CPPNs will not be able to capture the more complex details that make up higher resolution images.

Phase 2: Optimization

The structure and complexity are unique for all images. This can make it hard to create a general CPPN architecture that is suitable for every image. An essential step in the super resolution process is finding an optimal architecture for each image through NAS. The evolutionary approach proposed in this thesis (LE-CPPN) is tested against the standard baseline, random search, as well as grid search of predefined architectures.

RQ2 *How can CPPNs best be designed and optimized for image recreation?*

Experiment Goal: A grid search and random architecture search algorithm will be used to find optimal models for Set14. The results will then be compared to the results obtained by LE-CPPN.

Hypothesis: As LE-CPPN is a guided search, it is expected that it will find better architectures than grid and random search. It is also expected that even though CPPN architecture performs well on one image, the same architecture will not necessarily perform well on others.

Phase 3: Super Resolution and benchmark testing

In order to objectively evaluate the CPPN super resolution method, LE-CPPN will be tested against standard approaches and state of the art benchmarks.

RQ3 *What different qualities will CPPN super resolution display compared with those of traditional mathematical super resolution approaches?*

Experiment Goal: The models obtained by running LE-CPPN on Set14 and Set5 images scaled down by a factor of 2, 4 and 8 will be used to recreate the images at their original resolution. These results will be compared to bicubic interpolation and benchmarks from state of the art super resolution methods.

Hypothesis: If the image recreation abilities of the CPPN is adequate, it is expected that the CPPN upscaled images will be comparable to those upscaled by interpolation. Additionally, it is expected that the images upscaled by CPPN will have smoother looks and less pixelated patterns than those of images recreated by interpolation.

5.3 Experimental Setup

5.3.1 Hyperparameters

Some hyperparameters were shared for all the experiments across all three phases and can be seen in table 5.2. The distinctive hyperparameters of each phase will be listed in their respective sections. These hyperparameters are a result of the preliminary testing, in which different configurations were tested, but found not to perform better than those listed.

5.3.2 Phase 1 Setup

LE-CPPN is tested on a subset of 10 small-sized images from MNIST and normally sized images from Set5. The chosen images from MNIST are the first 10 images from the test set. For the Set5 experiment, the number of epochs was increased from 50 to 200. This was to give LE-CPPN a fair

mutation rate	0.3
elitism number	1
parent selection probability	0.7
evolution scheme	Lamarckian
fitness function	MSE
min layer size	4
max layer size	250
loss function	Binary Cross Entropy
optimizer	ADAM [15]

Table 5.2: hyperparameters used for preliminary testing.

amount of time to learn to recreate such large images, where the amount of data points is increased by a factor of about 153.

dataset	generations	population size	epochs	batch size
MNIST	15	15	200	5
Set5	15	15	200	100

Table 5.3: Experiment Phase 1 hyperparameters

5.3.3 Phase 2 Setup

In the phase 2 experiments, the grid and random search algorithm, as well as LE-CPPN, were run on all images from the Set14 dataset. Each image was scaled down by a factor of 4 to reduce the run time of all algorithms. All methods used 200 epochs and a batch size of 50. The grid search algorithm was configured to try all combinations in the cartesian product of the parameters listed in table 5.4. Random search was configured to try out 375 different network architectures so that the numbers matched the number of individuals evolved by LE-CPPN given the set number of generations and population size. The ranges from which the random search could choose depths, individual layer sizes, and activation functions are listed in table 5.5. Lastly, LE-CPPN was run with the parameters listed in table 5.6. The evolved CPPN architectures will be further analyzed to gain knowledge of the generality of image recreation.

Activation functions	Tanh	Linear	Sigmoid	Sine	Gaussian	ReLu	
Layer Sizes	25	50	75	100	150	200	250
Network Depths	1	2	3	4	5	6	7

Table 5.4: Phase 2: Grid search parameters

Activation functions	Tanh, Linear, Sigmoid, Sine, Gaussian, ReLu
Network depth	$0 < d \leq 10$
Layer Sizes	$4 \leq s \leq 250$

Table 5.5: Phase 2: Random search parameters

generations	population size	epochs	batch size
25	15	200	50

Table 5.6: Phase 2: Lamarckian evolution parameters

5.3.4 Phase 3 Setup

Super resolution experiments were conducted on images from different datasets and with different scaling factors. The images are first scaled down from the original sizes using the standard bilinear interpolation. In order to compare results with other state of the art methods, the standard downscaling factors of 2, 4, and 8 are used. From these downsampled images LE-CPPN is used to evolve CPPNs. These CPPNs are then used for upscaling the images to their original resolution. The upscaled images are compared to the original images and those upsampled by interpolation, to evaluate the accuracy of the super resolution methods. The experiment configurations are documented in tables 5.7 and 5.8. Due to the difference in image size resulting from varying scaling factors, different batch sizes were used for each scaling factor.

5.4 Phase 1: Feasibility Results

LE-CPPN was run 10 times on different images from MNIST and once for each of the 5 images of Set5. The best CPPN from each run was used to recreate the target image. The averages of the achieved structural

Dataset	Method	Scaling factor
Set5	Bicubic Interpolation	2
Set5	LE-CPPN	2
Set14	Bicubic Interpolation	4
Set14	LE-CPPN	4
Set5	Bicubic Interpolation	4
Set5	LE-CPPN	4
Set14	Bicubic Interpolation	8
Set14	LE-CPPN	8
Set5	Bicubic Interpolation	8
Set5	LE-CPPN	8

Table 5.7: Phase 3: Super resolution experiments

Scaling factor	Generations	Population size	Epochs	Batch size
2x	25	15	200	100
4x	25	15	200	50
8x	25	15	200	15

Table 5.8: Phase 3: Super resolution batch sizes

similarity index measure and peak signal to noise ratio from Phase 1 experiments can be seen in table 5.9. The LE-CPPN was able to achieve a slightly higher structural similarity index of 0.967 and 0.953 on the MNIST and Set5 images respectively. However, with a p-value of 0.2841, this result can not be considered statistically significant. This surprisingly small difference contradicts the hypothesis of the experiment. It seems that the extra training time LE-CPPN was given on Set5 was enough to make up for the increase in image size. Further, the achieved PSNR was higher for the Set5 images which means the mean error between the recreated and original images is lower for the Set5 images. Interestingly to note, is that the similarity measures SSIM and PSNR disagree on which dataset LE-CPPN performed the best on. This difference can be attributed to the varying image size. If the variance of the noise in the two recreated datasets is the same, an increase in pixels will affect the ratio and result in a higher PSNR. Thus it makes sense that the PSNR would be higher

for Set5 given similar noise variance.

Dataset	Mean SSIM	Mean PSNR	σ SSIM	σ PSNR
MNIST	0.967	27.654	0.0188	3.92
Set5	0.953	32.279	0.0301	2.196

Table 5.9: Phase 1: Image similarity scores

Figure 5.7: CPPN recreation of Set5 images. Originals followed by recreated

The SSIM on the MNIST dataset was better than for Set5, however by a smaller margin than expected. It seems that the increased number of training epochs was sufficient to enable the algorithm to create CPPNs capable of recreating detailed and complex patterns and colors.

In figure 5.7 the original images are shown with their recreated partners

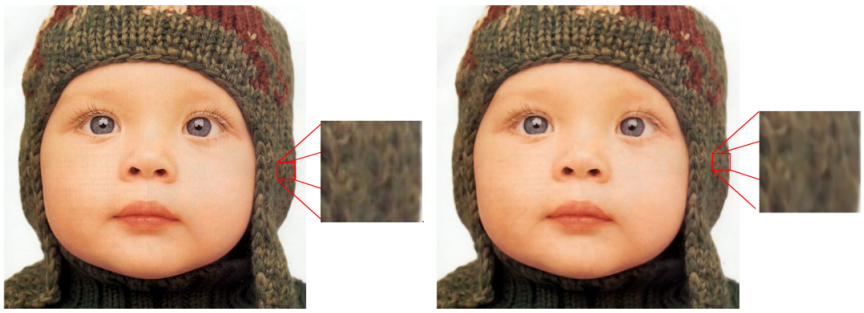


Figure 5.8: Textural difference, Original (left), Recreated (right)

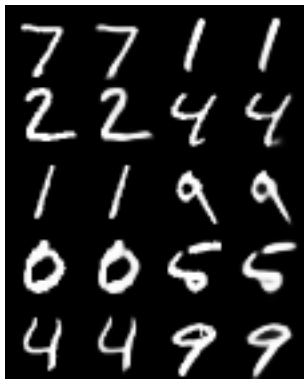


Figure 5.9: Mnist images to the left and its recreated counterpart to the right.

to the right. It is clear that the model has successfully recreated visually similar images. Only when looking at zoomed-in parts of the images, differences become apparent. In Figure 5.8 one can see the inaccuracies of the CPPNs recreation. The woven pattern of the child's hat is somewhat blurred compared to the original.

Figure 5.9 Shows the recreated MNIST images side by side with the originals. Despite achieving a higher SSIM on the MNIST recreations than the Set5 recreations, the differences are more apparent in MNIST. With fewer pixels per image, it is easier to see where the image pairs are visually different. With only 28x28 pixels, each pixel will have more effect on the

outcome of the image. This is apparent in the 5 and 9 images, where the resulting recreations have structural differences that are more noticeable than the Set5 recreations.

5.4.1 Convergence

Figure 5.10 shows the mean squared error for the best individual per generation for both MNIST and Set5. It is interesting that despite the difference in image size and complexity in the two datasets, LE-CPPN converges almost simultaneously.

Figure 5.11 displays the average network depth per generation. This plot shows how the best network architecture for the MNIST images stabilized around 3 hidden layers. For Set5 on the other hand, when given time to train for 200 epochs, the algorithm continued to evolve deeper networks, reaching an average depth of 5 by the end of the 15th generation. This demonstrates the evolutionary algorithm's ability to adapt the CPPN architectures to fit the complexity of an image.

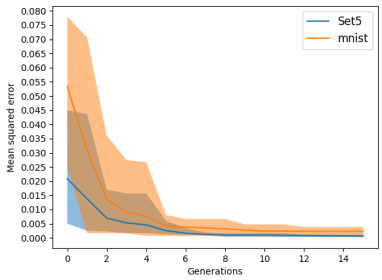


Figure 5.10: Fitness improvement over generations

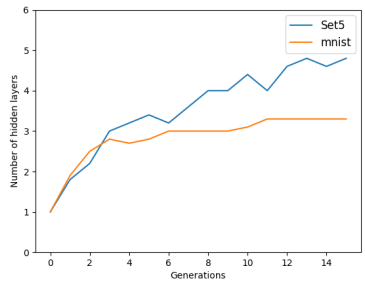


Figure 5.11: Average number of hidden layers per generation.

If one disregards the depth of the CPPNs, the run time of the LE-CPPN algorithm is linear with the number of pixels. This means that going from a 28x28 image to a 280x280 would increase the run time by a factor of 100. Consequently, the algorithm needed a lot more time to recreate Set5 than MNIST.

5.4.2 Conclusion

The findings of the phase 1 experiments contradict the hypothesis presented in the experiment plan (5.2). Given enough training, LE-CPPN successfully recreated complex and large images with results comparable to and even surpassing those achieved on small images. This shows that CPPNs have much potential for more complex tasks than MNIST. LE-CPPN also successfully adapted the size of the CPPNs to match the complexity of the given images, indicating that the method possibly can recreate even larger and more complex images with sufficient training, population size and the number of generations. The universal function approximation theorem [27] guarantees the approximation ability of neural networks for any function. Theoretically, a CPPN should then be able to recreate any image, regardless of size. These experiments have not found a limit to what images CPPNs can recreate, but have shown that sizes up to 800 pixels wide are certainly within the limit.

5.5 Phase 2: Optimization Results

The achieved structural similarity index measure and peak signal to noise ratio from Phase 2 experiments can be seen in table 5.10. From the table one can see that LE-CPPN scored considerably better than both grid and random search, and with a p-value less than 0.0001 can this finding can be considered statistically significant. This difference in performance is visible in the example in 5.12. LE-CPPN has learned more of the nuances and details of the original image, while the best CPPNs from random search and grid search are more blurred and less detailed. As suspected the task of learning a detailed image is so complex that pure backpropagation with random architectures can struggle to find a global optimum. This result was in line with the hypothesis of the experiment.

Dataset	Method	Mean SSIM	σ	Mean PSNR	σ
Set14	Grid Search	0.7607	0.0915	21.927	2.9097
Set14	Random Search	0.7737	0.0863	22.0625	2.9179
Set14	LE-CPPN	0.9297	0.0524	29.5334	4.5210

Table 5.10: Phase 2: Results

Figure 5.12: Method comparison: Images upscaled by a factor of 4 by their respective CPPNs.

5.5.1 Adaptability

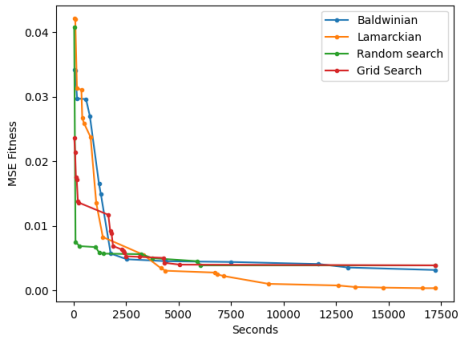


Figure 5.13: Fitness evaluation over time for each NAS method (trained on CPU Intel i7)

The major difference in performance can be credited to many factors. In total each algorithm tests around 400 network configurations. However, predefined architectures of the grid search and the randomness in random search make them unable to adapt their search to each image. LE-CPPN on the other hand will explore as many architectures, but adjust what kind of architectures to try out according to what architectures perform well. This is a great advantage when specializing an architecture for a specific image. This can be seen in Figure 5.13, where random and grid search find better solutions faster than LE-CPPN, but is outperformed relatively quickly. The slower start of LE-CPPN can be attributed to the simplicity

of its initial generations, which start with only one hidden layer. These simple networks will not be capable of capturing the images' structural complexity. Random search will have a random network depth no matter how far into the search it is, and can therefore find a deeper architecture relatively early, whereas LE-CPPN is dependent on mutation before that is possible.

5.5.2 The Lamarckian effect

It can be theorized that the crossover and mutation is a good method to explore new areas of the weight parameters' gradient landscape and thus avoid local minima that pure backpropagation training is unable to escape. When adding a layer to a network during mutation, the rest of the weights in the network remain the same, and the network is still close to the local minimum found during training. However, the addition of a new layer can alter the gradient landscape, and offer a new starting point for further training which leads to different minima. This is shown in table 5.11 where the SSIM score on image recreation for two identical CPPN architectures is shown. On the left column, the image recreation score for the CPPN evolved with Lamarckian evolution is shown, and an identical architecture with its weights randomly initialized and trained for 1000 epochs is shown in the left column. Interestingly, the evolved CPPNs perform significantly better than the ones without, even though every individual from evolution only trained for 200 epochs. This may indicate that the non evolved CPPNs get stuck in a local minima relatively early. It is known that a better weight configuration for the architecture exists, but they are not able to get there by using backpropagation only. These results also show how hard it is to improve when the gradients do not allow it. Both the baboon and bridge images have big differences in their performance, from 0.9276 to 0.5517 and 0.9703 to 0.5097 respectively. Baboon can be seen in figure 5.14, and while the LE-CPPN model bears a strong resemblance to its target image, the CPPN trained with only backpropagation only has some of the colors and general image structure correct. This shows the importance of network weights and how network optimization affects the result.

Image	Lamarckian evolved	Only backpropagation
monarch	**0.9601**	0.8852
flowers	**0.9790**	0.8338
bridge	**0.9703**	0.5097
ppt3	**0.9571**	0.7697
zebra	**0.8794**	0.6972
lenna	**0.9571**	0.8010
barbara	**0.8875**	0.8567
face	**0.9752**	0.8973
comic	**0.9545**	0.8153
pepper	**0.9612**	0.9127
man	**0.9083**	0.6343
coastguard	**0.8488**	0.6900
foreman	**0.9960**	0.9698
baboon	**0.9276**	0.5517

Table 5.11: Comparison of SSIM fitness on the evolved CPPN network architecture against the same architecture trained for 1000 epochs.

Downsampled Lamarckian evolved Only backpropagation

Figure 5.14: Baboon recreated with LE-CPPN and the same architecture, but only using backpropagation for training.

5.5.3 Architecture comparisons

To investigate if the architectures found by LE-CPPN would generalize to other images, for each architecture, 13 clones (182 in total) with randomly initialized weights were each trained on the other images in the set. The new networks were only trained with backpropagation and did not go

through the entire process of evolution. Each new network was trained on a single image for 500 epochs rather than 200. This was to make up for the head start CPPNs get from the Lamarckian weight inheritance in LE-CPPN.

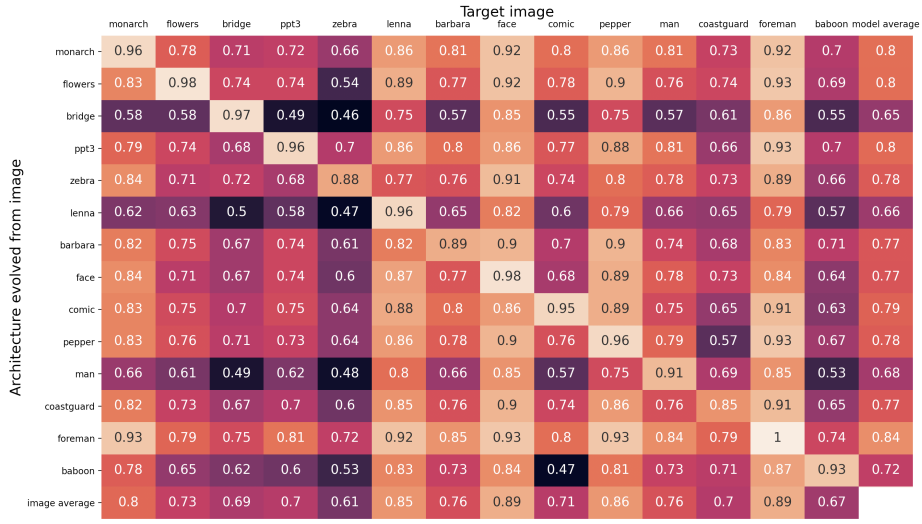


Figure 5.15: Image recreation SSIM by architectures found by LE-CPPN, but only trained on each individual image from Set14.

In figure 5.15, the architecture found by LE-CPPN for a specific target image is named along the y-axis. Along the x-axis is what image the architectural clone was trained on, and the heatmap shows the achieved SSIM for that image (lighter color indicates higher SSIM). In addition, the architectures' average across all the different images is shown in the last column, and the average score on the specific images is shown in the last row. Along the diagonal is the SSIM achieved by the original network evolved by LE-CPPN. Not surprisingly, from looking at this diagonal one can clearly see that the highest score obtained on each image, was with the model specifically evolved for that image.

Some images are easier to recreate than others. The lighter columns such as foreman or face achieved significantly better averages than the darker columns of zebra or baboon. However, the evolved scores of zebra

and baboon are not as low. This shows that it is possible to achieve a high score on difficult images. But defining a general CPPN architecture that works well for all images is difficult.

CPPN attributes: no. of	Pearson correlation coefficient
Trainable weights	0.3705
Hidden layers	-0.0110
Tanh layers	-0.1286
ReLU layers	0.0019
Sigmoid layers	-0.0439
Sine layers	-0.3886
Gaussian layers	0.4547
Linear layers	0.2032

Table 5.12: Cofficient of determination between CPPN attributes and average SSIM across all Set14 images

This becomes more apparent when comparing architectures, by observing whether some perform better than others based on their depth, amount of parameters and activation functions. In table 5.12, the correlation coefficient between average SSIM and different network parameters are shown. It seems that the number of trainable weights is more important than network depth, and that gaussian is a preferred activation function. Although there is found some correlation, the relationships between the variables and the SSIM are only weak, and it is difficult to conclude anything. This further shows the difficulty of using intuition and prior knowledge in designing CPPNs. It also highlights the practicality of having automated neural architecture search and optimization.

5.5.4 Conclusion

The findings of the phase 2 experiments enforce the hypothesis presented in the experiment plan (5.2). LE-CPPN successfully outperformed the grid search and random search algorithms with the given search parameters. Images are complex and of such a varying nature that an informed search such as the genetic algorithm is necessary for optimizing CPPN architectures for individual images. The Lamarckian inheritance of weights

seems to improve CPPN performance greatly compared to pure training. Also, the heatmap in figure 5.15 showed how the evolutionary process aids the CPPNs in performing well for their specific image, and how these architectures do not necessarily generalize across images. These findings indicate the necessity of specialized CPPNs.

5.6 Phase 3: Super resolution Results

Dataset/Scaling factor	Method	Mean SSIM	Mean PSNR	σ SSIM/PSNR
Set5 2x	LE-CPPN	0.9105	27.7551	0.0297/2.7643
Set5 2x	Bicubic	0.9554	32.2755	0.0283/3.4079
Set14 4x	LE-CPPN	0.6931	20.8578	0.1189/2.6195
Set14 4x	Bicubic	0.7882	23.7101	0.1/2.8846
Set5 4x	LE-CPPN	0.6332	18.6935	0.0878/3.0747
Set5 4x	Bicubic	0.87	26.54	0.047/3.399
Set14 8x	LE-CPPN	0.5713	17.711	0.1293/2.2803
Set14 8x	Bicubic	0.659	20.2709	0.1247/2.7231
Set5 8x	LE-CPPN	0.6211	18.5542	0.1105/3.215
Set5 8x	Bicubic	0.72	021.816	0.0841/3.6162

Table 5.13: Phase 3 results: Similarity between upscaled images and originals

The image similarity scores for each combination of dataset and scaling factor are presented in table 5.13. LE-CPPN consistently scores lower than Bicubic Interpolation on both SSIM and PSNR. For all pairings, the results can be considered statistically significant except for Set14 8x, where the p-value was 0.0792 (> 0.05)and thus not statistically significant. As the scaling factor decrease the achieved SSIM and PSNR improve significantly. This is largely because the amount of information lost in an image scaled down by a factor of 2 is a quarter of the information lost in one scaled down by a factor of 4. Thus there is much less new information that needs to be inferred by the super resolution step.

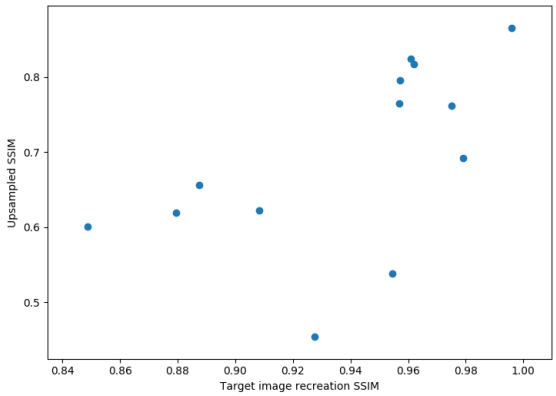


Figure 5.16: Scatter plot showing relation between image recreation SSIM and super resolution SSIM

The scatter plot in figure 5.16, shows the relation between the recreated image SSIM and the upscaled image SSIM. Surprisingly, it is hard to see a clear pattern showing that high SSIM on the recreation of the training image leads to high SSIM on the upsampled image. The Pearson correlation coefficient between recreation SSIM and upsampled SSIM is 0.5831, which is only considered to be a moderate positive correlation. Since all image recreation scores are fairly high (> 0.85, see Figure 5.11), it might be that the low resolution target image itself is the problem. In the downsampling process, too much information was lost to regain the same image through CPPN upsampling.

5.6.1 Deep Learning State of the Art comparison

Tab 5.14 compares results from other state of the art deep learning methods. SRResNet [19], CAR [37] and SRGAN [19] each achieved the best ranked results in terms of SSIM, PSNR and mean opinion score (MOS, obtained through a human survey of visual quality) respectively. Of these methods only SRResNet can beat the performance of bicubic interpolation in terms of SSIM, however, they all achieve a higher PSNR. As discussed in 3.2 these methods are trained to create a mapping from low resolution

to high resolution images. As a result, these networks can create new information where textures were lost in downsampling. This is exemplified in Figure 5.17 where the super resolution GAN by Ledig et al. [19] has inferred patterns onto the girl's scarf without them being present in the original. LE-CPPN and Bicubic interpolation on the other hand are not trained in any way on high resolution images, making them unable to infer such textures. For LE-CPPN to successfully recreate textures they must be present in the low resolution image. This disability seems to be a major drawback of CPPNs and Interpolation. Still, a drawback with the GAN and the other methods presented is that the models are limited in the range of their input and output sizes. The models produced by LE-CPPN could if needed produce any resolution. This is a great advantage when it comes to practical use because the combinations of upsampling needed and original image size can be infinitely many. Additionally, GANs and CNN mapping approaches need large amounts of training data with a varied collection of images. On the contrary, LE-CPPN only needs the low resolution target image to train and perform the upsampling.

Dataset	LE-CPPN	SRResNet [19]	CAR [37]	SRGAN [19]
Set14 4x	0.6931/20.8578	**0.8184/28.49**	0.7326/26.39	0.7397/26.02
Set5 2x	0.9105/27.7551	n/a	0.9658/38.94	n/a

Table 5.14: State of the Art comparison (SSIM/PSNR)

Original Bicubic 4x LE-CPPN 4x SRGAN 4x [19]

Figure 5.17: State of The Art image comparison

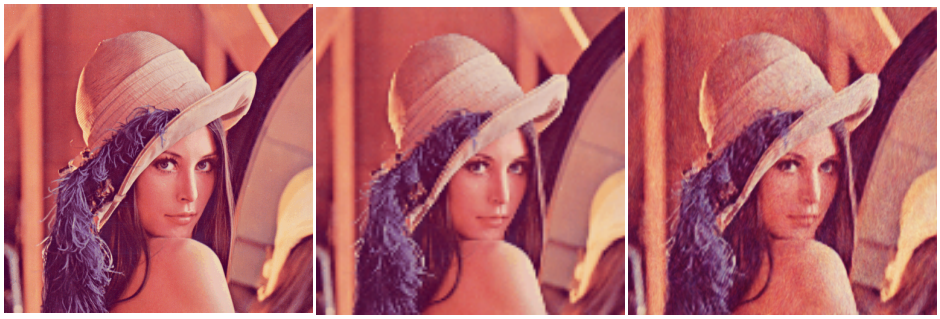
5.6.2 Visual Comparison to Interpolation

Figures 5.21, 5.22 and 5.23 (found on pages 58-60) compares the recreations of the set14 images at a scaling factor of 4. By inspecting the images side-by-side, it is more difficult to pinpoint a superior method. SSIM and PSNR are only the mathematical approaches to calculate image similarity and are, in the end, less important than human perception. Figure 5.18 is an example of where LE-CPPN has produced an arguably more pleasing upsampled image than interpolation. The LE-CPPN image has stronger colors, clearer edges and is arguably visually more similar to the original, despite the SSIM and PSNR being lower. There are also some images where LE-CPPN has not created a very good approximation, such as the baboon image and zebra. It is hard to know exactly why the CPPNs struggle to upsample these images, but the common properties of those images are that they contain fine textures, such as the baboon's fur and the grass in the zebra image. These properties seem to be hard to capture.

Figure 5.18: Super resolution example: Pepper

Edges/Curves

As expected LE-CPPN recreated images do not contain the same pixelated patterns often found in interpolated images. This is especially apparent when looking at edges in the upsampled images. Images upsampled with



Original HR Bicubic 4x LE-CPPN 4x

Figure 5.19: Super resolution example: Lenna

LE-CPPN contain smoother and more distinct lines than the interpolated counterparts. This is exemplified in figure 5.19. The shoulder of the woman in the rightmost image is free of any stair-like distortions. A theory of why, is that CPPNs can easily converge to represent linear and polynomial functions, and so can easily approximate the mathematical function of edges and curved lines.

Artifacts

Common artifacts of bicubic interpolation include stair-like patterns, blurring, and loss of detailed texture. Such artifacts are generally not wanted, and getting rid of these is important to gain a realistically looking upsampled image. LE-CPPN does not produce similar artifacts but clearly shows distinctive trademarks in certain photos. In the upsampled images one can see repeating patterns and effects created by the CPPN. The images often contain wave-like distortions and overly smooth edges. These effects combined tend to make the images appear painted and unrealistic. These artifacts can be seen in the left image of figure 5.20. The model has struggled to recreate the grass texture from the original image, and a wave-like distortion is prevalent in the image. Another form of the artifact that is apparent in some of the LE-CPPN images is noise. In figure 5.20 this noise resembles the noise that can occur when taking pictures with a high ISO setting [Nikon]. Since SSIM is particularly sensitive to noise, this probably attributes to giving very similar LE-CPPN images a lower

SSIM (as explained in section 2.8).

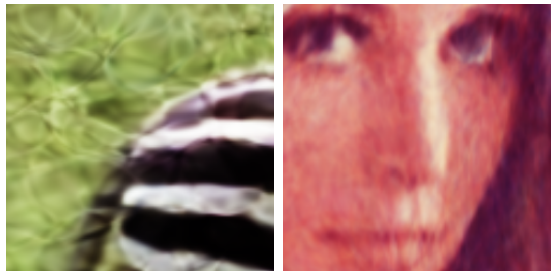


Figure 5.20: Artefacts found in LE-CPPN SR images

5.6.3 Conclusion

The experiments conducted in phase 3 showed that CPPNs have the potential for super resolution. Although LE-CPPN scored lower than interpolation on the image similarity metrics, the visual results show that the method is able to produce pleasing images, some even more so than their interpolated counterparts. The strong suit of CPPN super resolution seems to be curves and straight lines, as these tend to come out less pixelated. However, the method brings its own disadvantages. The training of CPPNs is a lot more time-consuming than interpolating images. In addition, the artifacts of this method can be more pronounced than those found in interpolation, especially in images where the textural details have been lost in the downsampling. An important advantage of using a CPPN for super resolution is the flexibility to which one wants to upsample an image. Once a model for a specific image is found, one can generate a new image of arbitrary resolution.

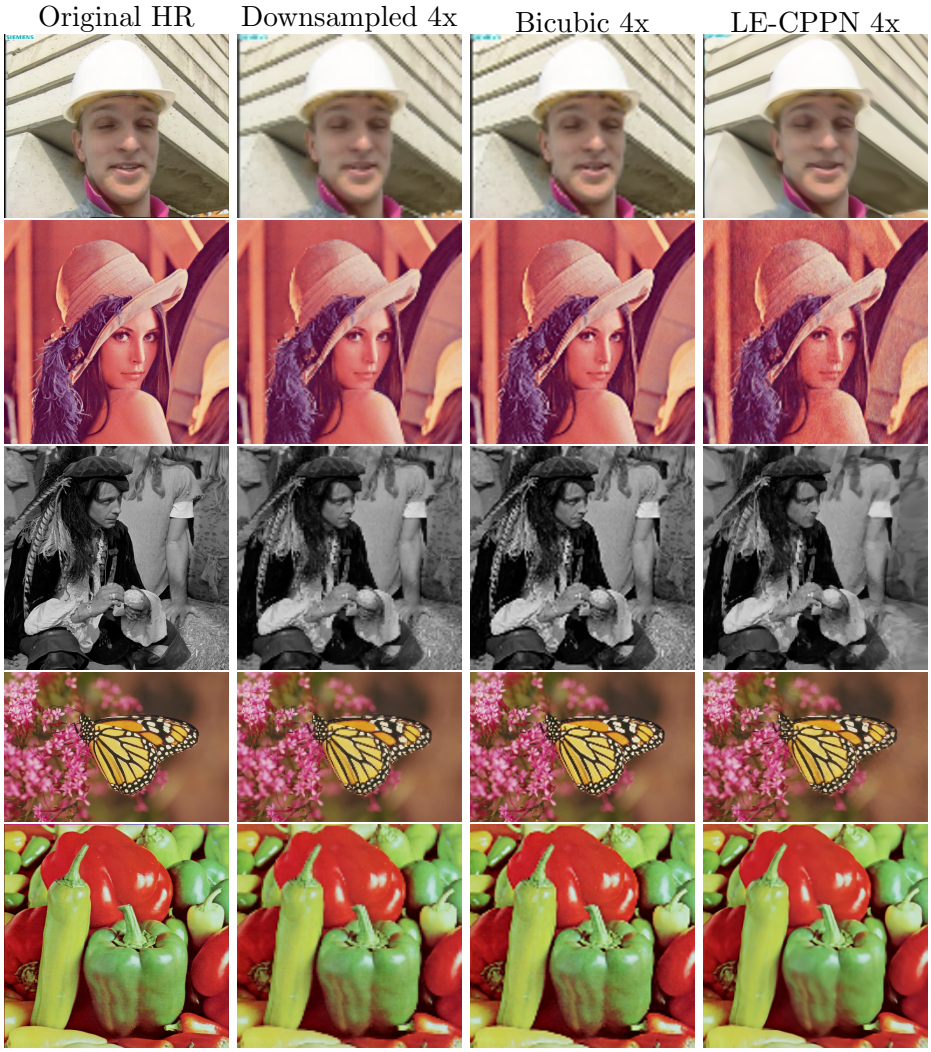


Figure 5.21: Image 1 to 5 of Set14

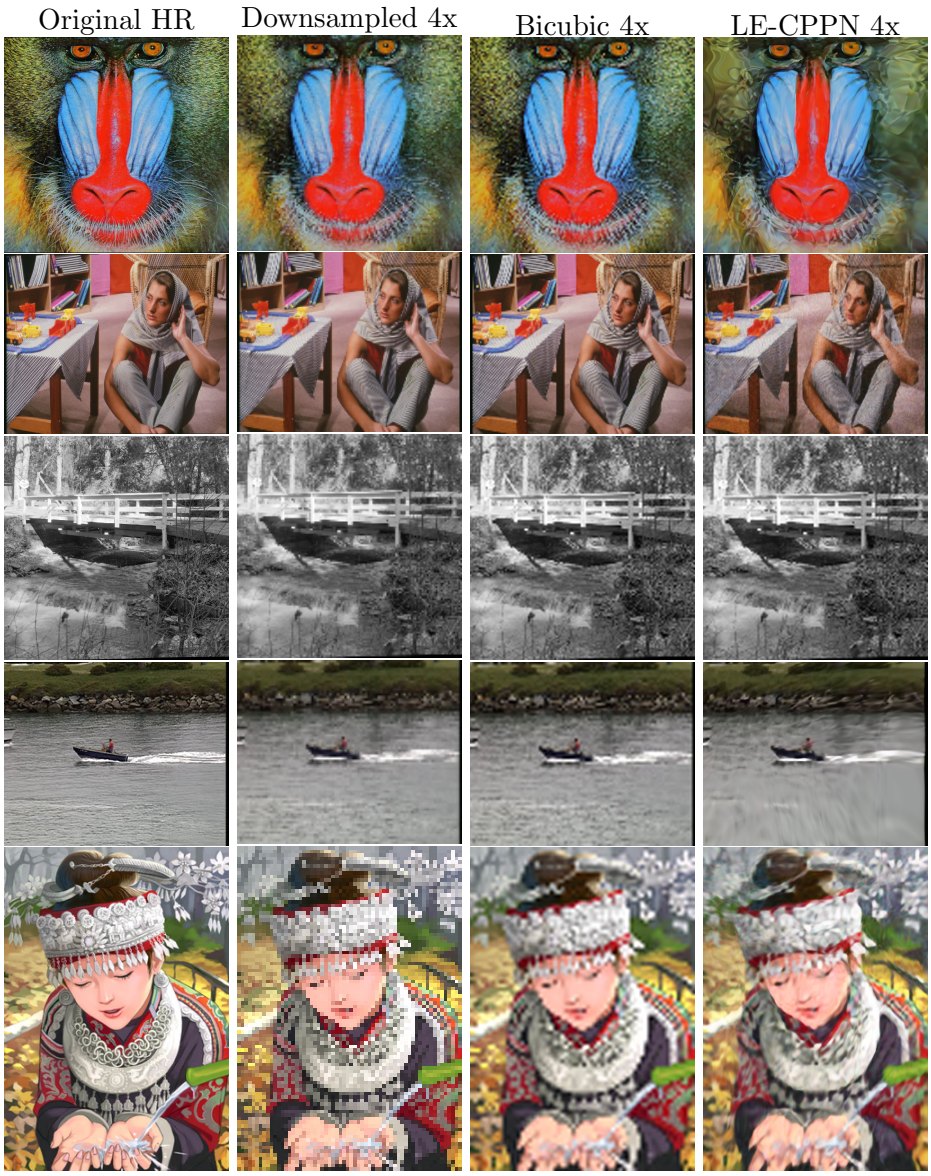


Figure 5.22: Image 6 to 10 of Set14

Figure 5.23: Image 11 to 14 of Set14

5.7 Additional Experiments

An interesting result of exploring the CPPNs abilities came from feeding them coordinates outside of the intervals they were trained on. This gives an indication of what the CPPN would generalize to the outside of the training space. Each CPPN was fed with coordinates from the interval [-2, 3] but they had only been trained on the interval [0, 1]. Examples of the resulting images can be seen in figure 5.24. It seems the CPPNs have correctly generalized the colors of each target image to the rest of the coordinate space. The patterns on the other hand, bear little resemblance to the original images. Regardless, the patterns in each separate image are still very dissimilar to each other. The straight lines of the flower image contrast the blurry elliptic spots around the baboon. One might theorize that they are somewhat derived from the training set or that they are produced by the combinations of activation functions. For example, the CPPN of the baboon image contains only trigonometric and hyperbolic activation functions (Sine and Tanh). This could be the cause of the elliptical patterns. The CPPN of the flowers contains one layer of the ReLU activation (which is a piecewise linear function), followed by four layers of Gaussian functions. This architecture could be the cause of the straight lines found around the flowers. The black-box nature of ANNs makes it difficult to conclude. Further, these properties of extending the inputs to the network beyond what the CPPN is trained on can be used to create interesting artificial artwork, which has garnered a lot of interest in recent years.

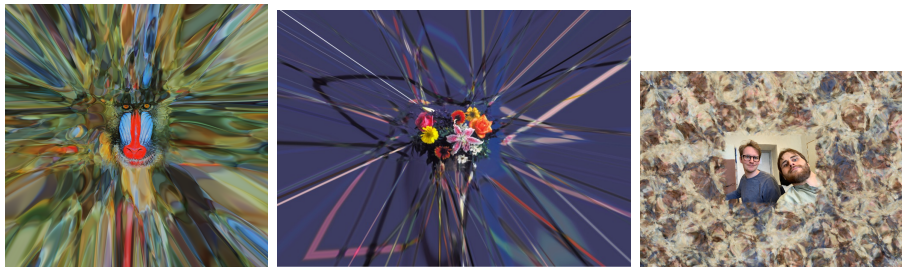


Figure 5.24: Outside the target frame

Chapter 6

Conclusion

The goal of this thesis was to explore the potential of using CPPNs for image super resolution. In order to reach the goal, three research questions were posed which in turn led to three experimental phases. The results from each phase influenced the direction of the project and led to an interesting insight into the potential, optimization and super resolution abilities of CPPNs. The resulting answers from the project are summed up below.

Research question 1 *How well will CPPN image recreation scale with increasing image complexity and size?*

Research question 2 *How can CPPNs best be designed and optimized for image recreation?*

Early testing showed that a simple trial and error for architecture and hyperparameter optimization achieved poor results and so the optimization of CPPNs seemed to be a challenging part. To successfully recreate larger images, the CPPNs had to achieve high recreational SSIM and PSNR scores. To improve on trial and error and random search a Lamarckian evolutionary algorithm (LE-CPPN) was developed. This method successfully outperformed other architectural search methods and showed promise in being an efficient CPPN optimization method in general. The Lamarckian evolutionary scheme proved to be important. When using the evolved architectures, they performed worse with only backpropagation than when

evolved by LE-CPPN. The CPPNs evolved from LE-CPPN were able to recreate standard sized images to the extent that they were practically indistinguishable from the originals. This showed that with good optimization techniques CPPNs can recreate images of large sizes. It was also showed that it is difficult to find a general model that can recreate every image. Henceforth, using a search method seems necessary to produce CPPNs capable of recreating unique images.

Research question 3 *What different qualities will CPPN super resolution display compared with those of traditional mathematical super resolution approaches?*

The results were varying and consistently obtained lower image similarity scores than interpolation. The method also scored lower than state of the art deep learning methods. The resulting images from LE-CPPN displayed desirable qualities such as clearly defined curves and edges. Visually speaking the images were pleasing even compared to interpolation. An important advantage of using a CPPN for super resolution is its flexibility to create any image resolution from a single model, something other deep learning methods can not.

Goal: Explore the possibilities of using CPPNs for image super resolution.

From answering the research questions came insight into the CPPN super resolution abilities. It proved to be a possible method of upsampling images, but the SSIM and PSNR scores were lower than other state of the art methods. When compared to bicubic interpolation, its super resolution ability was varied but comparable. It was interesting to see that CPPNs, with the right optimization technique, could recreate large images almost as accurately as small ones. Also, experiments showed that CPPN super resolution succeeds in preserving curves and edges during upsampling, something which is often lost in interpolation. Altogether, CPPN image super resolution proved to yield viable results when upsampling images. The method is flexible but difficult to optimize and needs further study to gauge its full potential.

6.1 Future work

During the project there surfaced several questions that could be potential routes to take for further exploration of the methods presented in this thesis.

- Ha [9] incorporated a random noise vector as input to the network functioning as a seed for enabling a single CPPN to create different images. A similar technique could be incorporated in the super resolution method so that one would not need a single CPPN for each image, but rather have a one-to-many relationship. The challenging part of this method would probably be that it can potentially be detrimental to the overall image recreational accuracy of the CPPN.

- In this research, all the networks that were generated through LE-CPPN were simple feedforward networks. However, there are many types of networks that have performed better than feedforward networks for different tasks. Convolutional Neural Networks(CNNs) are proven to perform well on image recognition tasks, and Long Short Term Memory(LSTM) networks have obtained good results for temporal data[46]. An interesting task would be to extend the Lamarckian Evolutionary scheme for neural architecture search on more sophisticated networks.

- Another way to improve upon LE-CPPN could be to change the chromosome variations for the genetic algorithm. For example, adding a chromosome for batch size could be beneficial because choosing the correct batch size is not always intuitive. It could also be possible to add more constraints with regard to search space.

- LE-CPPN has been successful in learning to recreate images with high accuracy. It would be interesting to evaluate how LE-CPPN works on problems that need to find a target function that generalizes well outside of its training data.

- Downsampling of images is another common image processing task. In scaling down images avoiding aliasing and information loss can be challenging. Using CPPNs for downscaling of images rather than super resolution could perhaps be an alternative to interpolation.

6.2 Contributions

This thesis has presented a Lamarckian evolution algorithm (LE-CPPN) for optimizing CPPNs to recreate images and showed that it successfully outperforms the random search baseline as well as a standard grid search method. It was shown that this algorithm was able to produce CPPNs capable of learning and upsampling detailed images of varying sizes and complexities, though not to a higher degree than other state of the art methods. The super resolution abilities of CPPNs were explored and it was shown that visually pleasing images can be obtained through this method. In addition, Lamarckian evolution proved to be very useful when optimizing CPPNs. A CPPN architecture that was evolved using LE-CPPN performed significantly better than an identical architecture only trained with backpropagation. Using the Lamarckian inheritance of weights can therefore potentially be useful on problems different from image recreation and super resolution.

Bibliography

[1] Bergstra, J., Bardenet, R., Bengio, Y., and Kégl, B. (2011). Algorithms for hyper-parameter optimization. In *25th annual conference on neural information processing systems (NIPS 2011)*, volume 24. Neural Information Processing Systems Foundation.

[2] Bevilacqua, M., Roumy, A., Guillemot, C., and line Alberi Morel, M. (2012). Low-complexity single-image super-resolution based on nonnegative neighbor embedding. In *Proceedings of the British Machine Vision Conference*, pages 135.1–135.10. BMVA Press.

[3] Chao, D., Chen, C. L., Kaiming, H., and Xiaoou, T. (2015). Image super-resolution using deep convolutional networks.

[4] Dai, T., Cai, J., Zhang, Y., Xia, S.-T., and Zhang, L. (2019). Second-order attention network for single image super-resolution. pages 11057–11066.

[5] Darwin, C. (1936). *The origin of species*. Everyman's library. Dent.

[6] Dong, C., Loy, C. C., He, K., and Tang, X. (2014). Learning a deep convolutional network for image super-resolution. pages 184–199.

[7] Fernando, C., Banarse, D., Reynolds, M., Besse, F., Pfau, D., Jaderberg, M., Lanctot, M., and Wierstra, D. (2016). Convolution by evolution: Differentiable pattern producing networks.

[8] Fukushima, K. (2007). Neocognitron. *Scholarpedia*, 2:1717.

[9] Ha, D. (2016). Generating large images from latent vectors. *blog.otoro.net*.

[10] Haris, M., Shakhnarovich, G., and Ukita, N. (2018). Deep back-projection networks for super-resolution.

[11] Holzinger, K., Palade, V., Rabadan, R., and Holzinger, A. (2014). *Darwin or Lamarck? Future Challenges in Evolutionary Algorithms for Knowledge Discoveryand Data Mining*, pages 35–56.

[12] Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*, pages 507–523. Springer.

[13] Kim, J., Lee, J. K., and Lee, K. M. (2016a). Accurate image super-resolution using very deep convolutional networks.

[14] Kim, J., Lee, J. K., and Lee, K. M. (2016b). Deeply-recursive convolutional network for image super-resolution.

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

[16] Klein, A. and Hutter, F. (2019). Tabular benchmarks for joint architecture and hyperparameter optimization.

[17] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. pages 1097–1105.

[18] LeCun, Y. and Cortes, C. (2010). MNIST handwritten digit database.

[19] Ledig, C., Theis, L., Huszar, F., Caballero, J., Cunningham, A., Acosta, A., Aitken, A., Tejani, A., Totz, J., Wang, Z., and Shi, W. (2017). Photo-realistic single image super-resolution using a generative adversarial network.

[20] Li, L. and Talwalkar, A. (2019). Random search and reproducibility for neural architecture search. *CoRR*, abs/1902.07638.

[21] li, Z. and Luo, Y. (2017). Generate identity-preserving faces by generative adversarial networks.

[22] Lim, B., Son, S., Kim, H., Nah, S., and Lee, K. M. (2017). Enhanced deep residual networks for single image super-resolution.

[23] Liu, H., Simonyan, K., Vinyals, O., Fernando, C., and Kavukcuoglu, K. (2018). Hierarchical representations for efficient architecture search.

[24] Liu, J., Gan, Z., and Zhu, X. (2013). Directional bicubic interpolation.

[25] Mordvintsev, A., Pezzotti, N., Schubert, L., and Olah, C. (2018).

[Nikon] Nikon, U.

[27] Nishijima, T. (2021). Universal approximation theorem for neural networks.

[28] Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. (2019). Regularized evolution for image classifier architecture search.

[29] Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y. L., Tan, J., Le, Q. V., and Kurakin, A. (2017). Large-scale evolution of image classifiers. 70:2902–2911.

[30] Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408.

[31] Sciuto, C., Yu, K., Jaggi, M., Musat, C., and Salzmann, M. (2019). Evaluating the search phase of neural architecture search. *CoRR*, abs/1902.08142.

[Secretan et al.] Secretan, J., Beato, N., D'Ambrosio, D., Rodriguez, A., Campbell, A., and Stanley, K. *Picbreeder*.

[33] Secretan, J., Beato, N., D'Ambrosio, D., Rodriguez, A., Campbell, A., and Stanley, K. (2008). Picbreeder: Evolving pictures collaboratively online. *Proceedings of Computer Human Interaction Conference*, pages 1759–1768.

[34] Springenberg, J. T., Klein, A., Falkner, S., and Hutter, F. (2016). Bayesian optimization with robust bayesian neural networks. *Advances in neural information processing systems*, 29:4134–4142.

[35] Stanley, K., D'Ambrosio, D., and Gauci, J. (2009). A hypercube-based encoding for evolving large-scale neural networks. *Artificial life*, 15:185–212.

[36] Stanley, K. O. (2007). Compositional pattern producing networks: A novel abstraction of development. *Genetic programming and evolvable machines*, 8(2):131–162.

[37] Sun, W. and Chen, Z. (2020). Learned image downscaling for upscaling using content adaptive resampler. *IEEE Transactions on Image Processing*, 29:4027–4040.

[38] Tesfaldet, M., Snelgrove, X., and Vázquez, D. (2019). Fourier-cppns for image synthesis.

[39] Thomas Elsken, Jan Hendrik Metzen, F. H. (2018). Simple and efficient architecture search for convolutional neural networks.

[40] Tommasi, T., Patricia, N., Caputo, B., and Tuytelaars, T. (2015). A deeper look at dataset bias.

[41] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. *CoRR*, abs/1706.03762.

[42] Wang, S. (2017). Generative adversarial networks (gan): A gentle introduction [updated].

[43] Wang, Z., Bovik, A., Sheikh, H., and Simoncelli, E. (2004). Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612.

[44] Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560.

[45] Whitley, D., Gordon, V., and Mathias, K. (1998). Lamarckian evolution, the baldwin effect and function optimization.

[46] Wiik, T., Johansen, H. D., Pettersen, S.-A., Baptista, I., Kupka, T., Johansen, D., Riegler, M., and Halvorsen, P. (2019). Predicting

peek readiness-to-train of soccer players using long short-term memory recurrent neural networks. In *2019 International Conference on Content-Based Multimedia Indexing (CBMI)*, pages 1–6. IEEE.

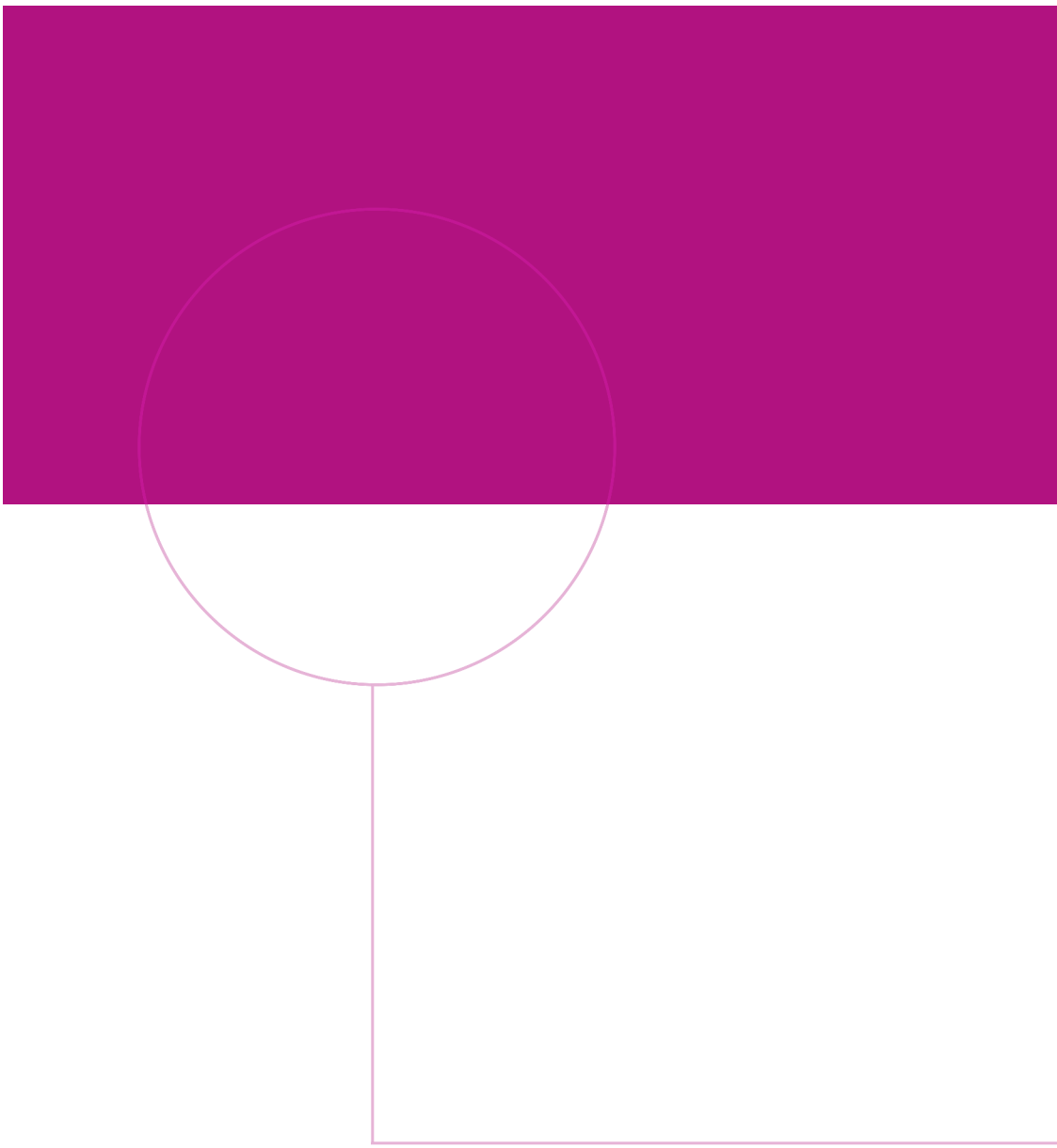
[47] Wistuba, M., Rawat, A., and Pedapati, T. (2019). A survey on neural architecture search. *CoRR*, abs/1905.01392.

[48] Xie, L. and Yuille, A. (2017). Genetic cnn. pages 1388–1397.

[49] Zeyde, R., Elad, M., and Protter, M. (2012). On single image scale-up using sparse-representations. In Boissonnat, J.-D., Chenin, P., Cohen, A., Gout, C., Lyche, T., Mazure, M.-L., and Schumaker, L., editors, *Curves and Surfaces*, pages 711–730, Berlin, Heidelberg. Springer Berlin Heidelberg.

[50] Zhang, Y., Li, K., Li, K., Wang, L., Zhong, B., and Fu, Y. (2018a). Image super-resolution using very deep residual channel attention networks. *CoRR*, abs/1807.02758.

[51] Zhang, Y., Tian, Y., Kong, Y., Zhong, B., and Fu, Y. (2018b). Residual dense network for image super-resolution.



NTNU

Norwegian University of
Science and Technology