NOVA IMS Information Management School

MAA

Mestrado em Métodos Analíticos Avançados
Master Program in Advanced Analytics

Deep Semantic Learning Machine: A Convolutional Network Construction Algorithm

Lukas Früchtnicht

Dissertation presented as partial requirement for obtaining the degree

Master of Science in Advanced Analytics and Data Science

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

Deep Semantic Learning Machine:
A Convolutional Network Construction Algorithm

This document was created using the (pdf)LaTeX processor, based on the "novathesis" template[1], developed at the Dep. Informática of FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

This thesis was prepared under the supervision of:

Ivo Carlos Pereira Gonçalves

Assistant Professor

at University of Coimbra

and

Mauro Castelli

Associate Professor

at Nova IMS

Abstract

The Semantic Learning Machine (SLM), an algorithm that evolves the topology of feed-forward neural networks (NN), has shown remarkable results in generalization and computing time. It has the benefits of searching the space of different NN architectures under a unimodal fitness landscape in any supervised learning problem. Recent research used the SLM at the end of a Convolutional Neural Network (CNN) instead of fully connected layers outperforming state-of-the-art CNNs. It was proposed to extend the SLM to explore the possibility of optimizing the convolution layers - evolving the full CNN topology. This thesis introduces an operator to optimize the convolution layers, extending the SLM to the Deep Semantic Learning Machine. Initial results, computed using the mnist dataset, show that the algorithm does work but are of limited interpretability. Real-life practicability remains to be improved due to high memory and computational requirements.

Keywords: Deep Semantic Learning Machine, Semantic Learning Machine, Geometric Semantic Genetic Programming, Machine Learning, Convolutional Neural Networks

Resumo

Semantic Learning Machine (SLM), um algoritmo que evolui a topologia de redes neurais feed-forward (NN), tem mostrado resultados notáveis em generalização e tempo de computação. Tem benefícios de pesquisar o espaço de diferentes arquiteturas NN sob um cenário de aptidão unimodal em qualquer problema de aprendizagem supervisionada. Investigação recente recorre ao uso de SLM no final de uma redes neurais convolucional (CNN) em vez de camadas totalmente conectadas, superando CNNs de última geração. Foi proposto estender o SLM para explorar a possibilidade de otimizar as camadas de convolução - evoluindo a totalmente a topologia CNN. A presente tese apresenta um operador para otimizar as camadas de convolução, estendendo o SLM para a Deep Semantic Learning Machine. Os resultados iniciais, calculados usando o conjunto de dados mnist, mostram que o algoritmo funciona, mas revelam uma interpretabilidade limitada. A aplicabilidade em cenários reais precisa ainda de melhorias devido aos altos requisitos de memória e computação.

Palavras-chave: Deep Semantic Learning Machine, Semantic Learning Machine, Aprendizagem Automática, Redes Neurais Convolucional, Programação Genética Geométrica Semântica

Contents

List of Figures

Acronyms

AI Artificial Intelligence

CE Cross entropy

CNN Convolutional Neural Network

CP Convolution Part

CP-layers Convolution and pooling layers

DSLM Deep Semantic Learning Machine

EDV Error Deviation Variation

GA Genetic Algorithm

GP Genetic Programming

GSGP Geometric Semantic Genetic Programming

GSM-NN Geometric Semantic Mutation Neural Network

GSM-CNN Geometric Semantic Mutation Convolutional Neural Network

ILSVCR ImageNet Large Scale Visual Recognition Challenge

ls learning step

MRI Magnetic Resonance Imaging

ms mutation step

NCP Non-Convolution Part

NN Artificial Neural Network

ReLU Rectified Linear Units

SLM Semantic Learning Machine

SSHC Semantic Stochastic Hill Climber

Tahn Hyperbolic Tangent

TIE Training Improvement Effectiveness

Chapter 1

Introduction

Artificial Intelligence (AI) is a term used in the center of discussions about the future in today's world. The research field of AI is vast and highly interrelated among other domains than computer science. It already impacts many people's everyday lives through various touchpoints: weather forecast in meteorology, online advertising, health, transportation, or finance. More and more companies are getting ready to change their businesses to rely on data-driven decisions, and as a result, whole business models deteriorate, and new ones appear.

For many people finding meaning in images, and art was thought to be proprietary to humankind and has been an exciting field of research in recent years and years to come. The main challenges are object segmentation and classification in, e.g., self-driving cars or magnetic resonance imaging, image colorization, super-resolution, and image synthesis. While the current approach of training Convolutional Neural Networks (CNNs) by finding optimal weights has shown great successes, the search for an adequate architecture still lies in the researcher's hands. This thesis examines Geometric Semantic Genetic Programming (GSGP) as a methodology to efficiently evolve CNNs, potentially laying ground work for better solutions to the challenges described above.

In previous works, GSGP has been extended to a neuroevolution algorithm, the so-called Semantic Learning Machine (SLM), to evolve multi-layer feed-forward neural networks. This approach has shown outstanding performance in various tests. Recently, the question if the approach can be extended to optimize the architecture of convolutional layers in CNNs has been raised. This would allow extending the SLM to optimize the whole topology of a CNN. This thesis proposes an operator that optimizes the architecture convolution layers (and pooling layers) based on GSGP and SLM principles. An extended variant of the SLM, the Deep Semantic Learning Machine (DSLM), is proposed using that operator. To evaluate the optimization of convolution layers with the proposed operator, the DSLM is confronted with recognizing handwritten digits and compared with the SLM and traditional CNNs.

This thesis's remainder is structured in the following way: The next chapter gives broad

theoretical background in optimization (section 2.1) and Machine Learning (section 2.2). Subsequently, the concept of both Artificial Neural Networks (NN) (in section 2.3) and CNNs (in section 2.4) are explained in more detail. Further, Genetic Programming (GP) and GSGP are derived in section 2.6 from Genetic Algorithms (GA) which are explained in section 2.5. Afterward, a detailed review of the Semantic Learning Machine (SLM) is given in section 3, and the Deep Semantic Learning Machine (DSLM) is proposed in section (4). Experimental results on the proposed algorithm are shown in section 5 and limitations of this thesis and possible future work are set out in section 6. In the end, a conclusion is drawn.

Chapter 2

Theoretical Background

This section gives the reader theoretical background to understand the idea and challenges of the DSLM algorithm. Readers who come from machine learning, primarily the computer vision domain, or feel familiar with a topic may skip the first subsections as the basic concepts are introduced here. However, it is vital to understand the concepts of GSGP laid out in subsection 2.6 to understand how the SLM and DSLM work.

2.1 Optimization

Optimization deals with finding the best solution from a set of all feasible solutions according to specific criteria. The concept of optimization can be found in many everyday situations: A supermarket wants to have the optimal stock of products to offer every customer what he/she might buy, have low inventor cost, and make sure that no product goes off. When driving a car, a person wants to take to optimal route to the destination. What is considered *optimal* can depend on more factors than travel time. Other vital factors could be if a toll route is used or not, or the average fuel consumed during the journey. It is essential to consider all criteria to find an optimal solution and that an optimal solution can be a *minimum* (e.g., time spend) or *maximum* (e.g., money earned). An optimization problem models a problem in general terms (i.e., routing), and only the problem instance is solved by an optimization algorithm (i.e., the route from A to B at time t).

An optimization problem can be represented in the following way:

Given a function $f : X \rightarrow \mathbb{R}$, sought is an element $x^* \in X$ such that $f(x^*) \leq f(x) \forall x \in X$ (for minimization problems) or such that $f(x^*) \geq f(x) \forall x \in X$ (for maximization problems).

The domain X of f is called the *search space*, containing all possible solutions, while each element of X is called *candidate solution* or **feasible solution**. Solutions that are similar to one another are called neighbors. The *neighborhood* is a vital concept in optimization and essentially is a mapping that assigns a set of (neighboring) solutions $y \in X$ for each $x \in X$: $N(x) : X \rightarrow 2^X$, where 2^X constitutes all possible subsets of X.

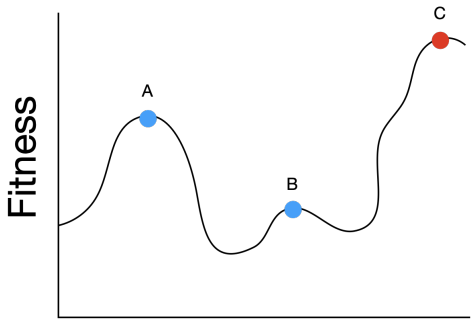

Figure 2.1: Fitness landscape with locally optimal solutions A and B and the global optimum C

The function f is called the *objective, fitness,* or *loss function*. It assigns a value to each candidate solution. A plot of (X, f, d) of an optimization problem instance, a so called *fitness landscape* [50], can help to visualize the difference of *locally* and *globally* optimal solutions as shown in figure 2.1. Here, d is a distance measure (e.g., Manhattan distance or Euclidean distance) of two solutions. It is only wise to consult a fitness landscape if the distance between solutions has a meaning, i.e., a metric search space is in place.

A solution is a locally optimal solution (minimization) concerning its neighborhood N if: $f(x') \leq f(x) \forall x \in N(x')$. A solution is a globally optimal solution (minimization) if: $f(x^*) \leq f(x) \forall x \in X$. If a solution's fitness is *directly proportionate* to the distance to a globally optimal solution, no locally optimal solutions exist, an essential concept for this thesis's remainder. Such a fitness landscape is called a *uni-modal fitness landscape*.

Members of the search space X are often restricted to have specific criteria to represent essential constraints in the optimization problem. For example, each solution must be a positive integer number in inventory planning as half an egg is impossible to store, and a negative egg does not exist. These constraints are usually modeled as qualities such as $h(x) = 0$, inequalities $g(x) \leq 0$ or variable bounds $x_{min} \leq x \leq x_{max}$.

A simple example of an optimization problem is the binary knapsack problem. Here, The problem is to maximize the sum of the values of the items in a knapsack so that the sum of the weights is less than or equal to the capacity of the backpack. It it can be expressed as:

$$\text{maximize} \sum_{i=1}^{n} v_i x_i$$

$$\text{subject to} \sum_{i=1}^{n} w_i x_i \leq W \text{ and } x_i \in \{0, 1\}$$

A set of n items, each with a weight of w_i and a value v_i is given with the maximal capacity of W. The value of x_i indicating if the item i is in the knapsack or not.

For real-life problems, the search space is too exhaustive to plot a fitness landscape and then choose a globally optimal solution. Every possible solution must be evaluated using the fitness function (exhaustive search) to draw a fitness landscape. Many different approaches on how to retrieve the optimal or close to the optimal solution have developed. It should be mentioned that Wolpert and Macready showed that there does not exist a superior algorithm for all different kinds of optimization problems. An over-performance of one algorithm for some problems will result in under-performance for the remaining problems, known as the *no free lunch theorem* [49]. For the remainder of this chapter and this thesis, only heuristics and meta-heuristics are further examined as they are the basis for the following chapters. Further, the field of problem complexity is omitted as it is not essential to the following chapters. Interested readers are encouraged to read about this topic in textbooks.

Heuristic methods determine solutions to an optimization problem practically but are not guaranteed to reach an optimal solution. However, they are usually less computationally intensive than exact methods and are often applied to complex problems to which no exact method is known. While heuristics are usually designed problem-specific, meta-heuristics are strategic problem-solving frameworks that can be applied to various problems [10]. Most of them rely on the concept of neighborhood search.

The most straightforward approach iteratively explores the search space: If a solution in the neighborhood is better than the current best solution, update the current best solution until no better solution is in the neighborhood. For a maximization problem, this method reaches the next local optimum in the fitness landscape. It is thus called **hill-climbing**. However, it is not guaranteed to reach a globally optimal solution. For the sake of completeness, some other meta-heuristics are briefly mentioned here, but a complete explanation would go beyond establishing a theoretical background and is not necessary for understanding the next chapters.

Simulated Annealing [26], inspired from thermodynamic processes of cooling materials slowly to reach more stable crystals, is similar to hill-climbing but also accepts less optimal solutions with decreasing probability over time:

$$P(C, x, x') = \begin{cases} 1, & \text{if } f(x') \geq f(x) \\ \exp(f(x) - f(x')/c), & \text{otherwise} \end{cases}$$

Where C is the control parameter, which should decrease towards zero but never reaching it. Simulated Annealing is proofed, under certain conditions, to converges to a globally optimal solution [19].

Particle Swarm Optimization is a biologically inspired meta-heuristic that mimics observations of swarming animals such as birds of fish [25]. Many particles explore the search space simultaneously. Each particle tends towards the best-known solution but is also influenced by

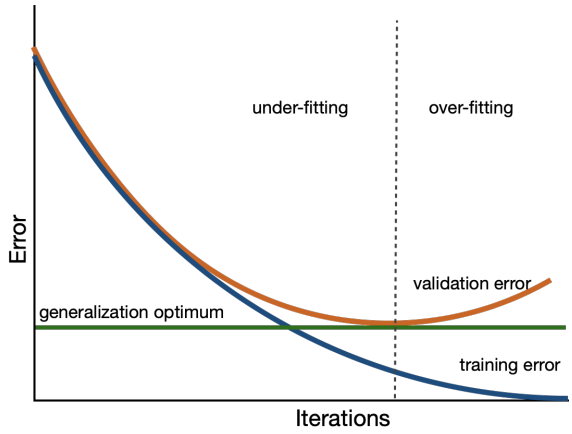

Figure 2.2: Over-fitting and under-fitting: Error on training and test data

random movement.

Another biologically inspired meta-heuristic are Genetic Algorithms (GAs), which are inspired by Charles Darwin's theory of evolution [13]. Here, many individuals explore the search space and undergo selection and variation to form better individuals and hence better solutions. Section 2.5 deals with GAs in depth.

2.2 Machine Learning

The term machine learning is generally described by algorithms that improve themselves through experience. Machine learning can be sub-grouped into supervised learning and unsupervised learning. In unsupervised learning, an algorithm tries to find hidden relationships within the data. In supervised learning, an algorithm sees a set of labeled training data with the goal to learn a function that maps the data to the labels. As an additional requirement, the function must also have the same property when tested on data of the same kind other than the training data, known as unseen data, test data, or validation data. Supervised machine learning problems can be further divided into regression problems, estimating a quantity, and classification problems to create a mapping to a discrete variable, i.e., discriminate among different classes.

If the learned function or model of the training data performs well on unseen data, it is *generalizing*. However, experiments show that this is not always the case. The algorithm might not have learned enough from the training data and is then said to be *under-fitting* the data, or it might have memorized the data exactly, and thus the learned function does not hold a general truth. Figure 2.2 shows the typical relationship between the error on training data and the error on the test data of an iterative supervised machine learning algorithm over different iterations. The more iterations, the better the training data is modeled. However, this is only true for a certain degree regarding the error on the unseen data and thus the generalization. From a certain point onward, the model will start to model the random noise contained in the training data and

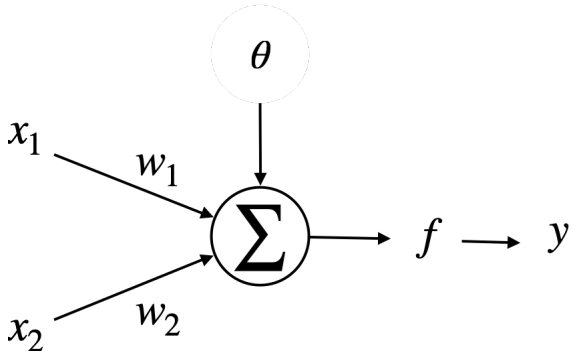

Figure 2.3: Simple Percepton with two inputs

is *over-fitting*. The error in the unseen data increases and the model develops a high variance. The theoretical optimal generalization is highly influenced by the available training data [17]. Only by pure chance, an algorithm could come up with a model that exceeds the knowledge hidden in the training data. A desirable algorithm will produce a model that minimizes the training error while not sacrificing performance on unseen data.

All learning algorithms have means by which they evolve the model over time, e.g., how a possible function is created and thus representing the only way the algorithm can improve the model's performance. These concepts are known as the inductive bias of a model [21]. The problem of finding the best optimal learning algorithm is also known as the bias-variance trade-off. Different algorithms with different inductive biases have evolved. The next chapter gives an overview of NNs and CNNs. Later, the concept of GSGP is derived from GP and GA. While there are many more algorithms that are possible to study in the context of machine learning, these lay the foundations for the proposed DSLM in section 4.

2.3 Neural Networks

NN are biologically inspired machine learning algorithms that try to mimic the function of the human brain. A brain is composed of many neurons that send out signals after receiving certain stimuli. The same structures can be found in NN. The simplest form of a NN was invented as early as 1958 by F. Rosenblatt and is called Perceptron [40].

Only a single neuron composes the Perceptron NN shown in figure 2.3. It exhibits a *feed-forward* property as information from the training data only flows in one direction, from the input to the output. Here, inputs denoted as x_n, weighted connections w_n, a bias θ, and an activation function f calculate an output y as following:

$$y = f\Big(\sum_{i=1}^{n} x_i w_i + \theta\Big)$$

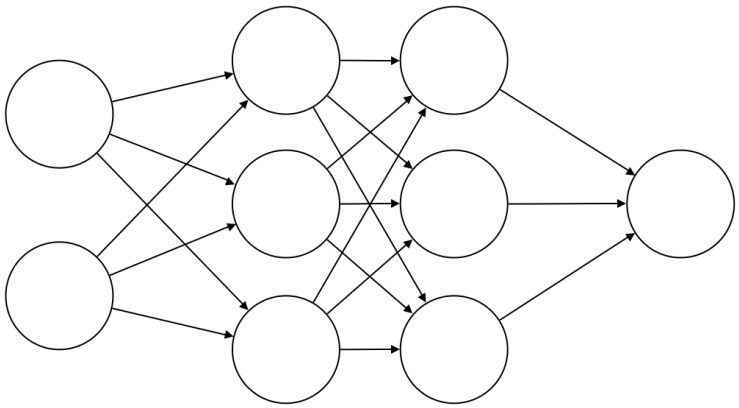

Figure 2.4: Feed Forward Neural Network with 2 input neurons, 2 hidden layers and 1 output neuron

A possible activation function f could be the threshold function:

$$f(s) = \begin{cases} 1, & \text{if } s \geq 0 \\ -1, & \text{otherwise} \end{cases}$$

The goal is to learn the *weights* such that for each input X, a correct y is calculated and can be achieved with the perceptron learning rule, which can further be investigated in [40]. Another way of looking at the Perceptron is that each input and weight are pairs of regressors and parameters of a linear regression equation $y = \beta_n x_n + \theta$. A single neuron perceptron can only solve linearly separable problems as the decision boundary is only formed by straight lines.

The figure 2.4 shows a multi-layer feed-forward neural network with biases omitted for readability, which is referred to as NN for the remainder of this thesis. A *layer* refers to a set of neurons that are not connected to each other. Layers in feed-forward networks receive incoming connections from layers closer to the input layer and have outgoing connections to layers closer to the output layer. Layers between the input and output layer are called hidden layers. With the Adaline [48] learning rule first used the *error* obtained on the training data set to update the weights, and in 1985 the modern form of the well-known gradient descent back-propagation algorithm was introduced [30]. As its name suggests, the algorithm propagates errors from the output neuron back to the hidden neurons to calculate a gradient. For a detailed explanation of the back-propagation algorithm and other weight optimizes, please see [18].

It should be mentioned that apart from the threshold activation function, which is rarely used nowadays, many other activation functions can be found in NNs. Sigmoidal functions such as the logistic function $\sigma(x) = \frac{1}{1+e^{-x}}$ or the hyperbolic tangent (Tahn) $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ have been one of the most common in the earlier days while recently the Rectified Linear Units function $f(x) = max(0, x)$ (ReLU) is used widely. Moreover, it is common to use the softmax function $\sigma(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}$ at the output layers of a NN for classification tasks as the output is easily interpreted as a class probability by humans. The swich function $f(x) = x \cdot \sigma(x)$ with $\sigma(x)$

Figure 2.5: People imagined by a Generative Adversarial Network

as the logistic function, proposed by Google Brain [38], has interesting properties that allow for changing more weights than if using ReLU, which resulted in better performance for many tasks. Many other activation functions exist.

2.4 Convolutional Neural Networks

Convolutional Neural Networks are multi-layer feed-forward neural networks with specialized layers that are used to capture *spatial* relationships within the input data they receive. In 1988 the back-propagation algorithm was first introduced for one-dimensional CNNs [46] and later extended to two-dimensional CNNs [31]. The two-dimensional CNNs are widely used for imaginary analysis as features in an image highly depend on spatial arrangement. They are used in various domains and influence many people's everyday life. Many modern smartphones rely on face identification for authorization (such as Face ID [23]) using CNNs. Moreover, advances in medical applications such as magnetic resonance, computed tomography, and microscopy imaging have been made [12] and this field of research is still growing. Also, CNNs are used for detecting patterns in various time series domains, e.g., myocardial infarctions can be detected using a person's electrocardiogram signals with a CNN [2]. Perhaps one of the fascinating uses of CNNs is the generation of fake images as shown in figure 2.5 [47] and the editing capabilities they bring to photo and video editing software [3]. The remainder of the section introduces the convolution operation and other concepts that come with it, and later a shot discussion about the development of different architectures is held.

In its most general form, the convolution operation can be described as combining two functions on a real value input. E.g., to get a smooth version of noisy measurements of some time dependent data $x(t)$ a weighting function $w(a)$, where a is the age of the measurement, can be applied for all data points:

$$s(t) = \int x(a)w(t-a)da$$

which is called convolution and expressed as:

$$s(t) = (x * w)(t)$$

Where the input function x is known as the *input*, the weighting function w as the *kernel* or *filter* and the resulting smooth version s is also known as the *feature map*. In machine learning, the

input data usually is discretized such that data is provided only in certain time intervals. Thus, t can be assumed to take only integer values, and the above can be rewritten as:

$$s(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a)$$

Given that data in machine learning is usually represented as a multi-dimensional array or tensor. Further, it is usually assumed that the input and the kernel have a value of zero. Except for the set of points, a value is stored, making it a summation over a finite number of the respective array entries. When talking about image data, the convolution usually happens over the horizontal and vertical dimensions of an image I with a two-dimensional kernel K to get a smoother estimate of that respective image:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n)$$

Convolution is commutative, and can be rewritten as:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n)$$

with the kernel being flipped relative to the input. Many libraries implement cross-correlation instead of convolution (e.g., Tensorflow [1]).It is the same operation, without flipping the kernel K. The only reason to flip the kernel is to gain the commutative property which is not needed for CNNs. For this thesis convolution and cross-correlation are treated equally as the operation without flipping the kernel. An example of the convolution operation of input and kernel and the resulting feature map is shown in figure 2.6.

Taking this concept to a layer of a neural network (in this example to a CNN for image processing), each neuron of a convolution layer is no longer connected to all previous inputs but only a particular region of the input tensor, the receptive field. This is the case because the kernels are usually much smaller than the input tensor to detect small features of the input.

Moreover, all neurons applying the same kernel share the same weights connecting them to different regions of the input. This has beneficial effects to machine learning as the sparse connections require less memory to store values and calculating the output of one neuron takes less steps. Given m inputs and n outputs the matrix multiplication usually requires $O(m \times n)$ run time. If each neuron can only have k inputs it reduces the run time to only $O(k \times n)$, while in practise k is several numbers of orders of magnitude smaller than m. While the parameter sharing of weights does not reduce the forward propagation through the network, much fewer parameters have to be learned compared to fully connected layers.

Another important aspect when talking about convolution layers is how the kernel behaves at the input edges. Applying a 2x2 kernel to a 4x4 input results in a 3x3 output and shrinks the input for any subsequent layers (see figure 2.7. The concept of *padding* prevents this. The

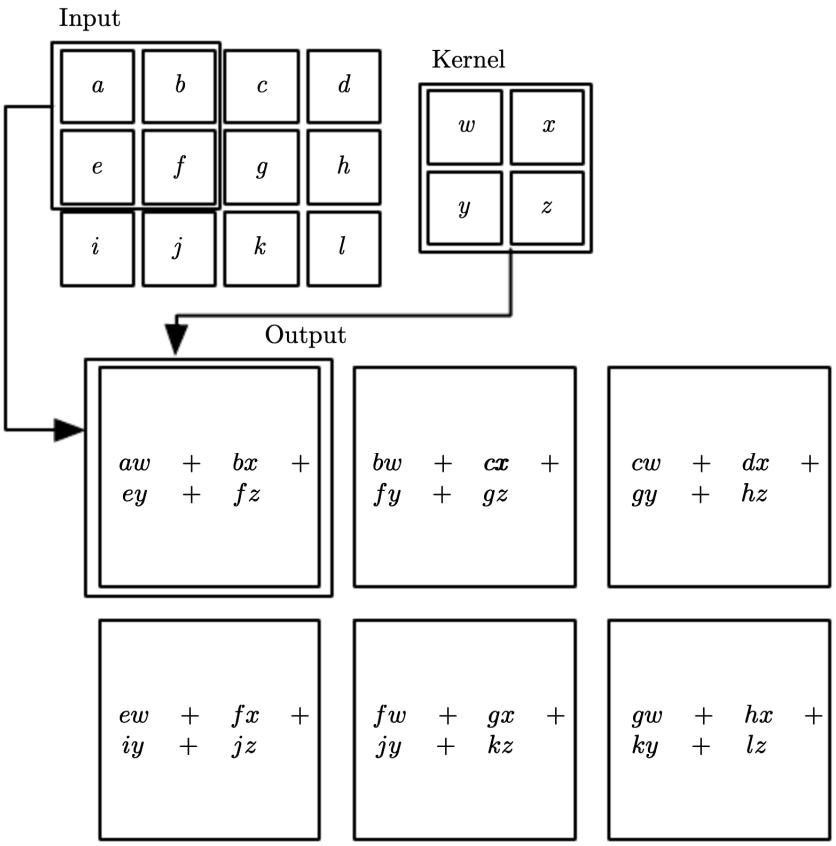

Figure 2.6: Convolution operation [18]

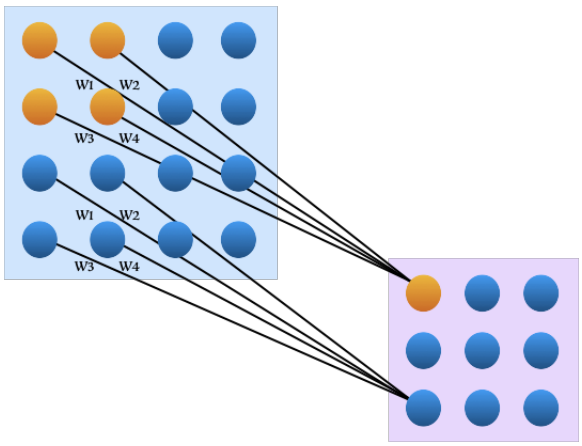

Figure 2.7: Convolution layer

input is treated as if enclosed by zeros to ensure the output size does not decrease. The previous example, using only the original inputs to calculate the output, is called *valid* padding. A scenario where enough zeros are added to ensure that the output size stays the same is known as *same* padding. In the example shown in figure 2.7 the input would grow to the size of 6x6with zeros at the edges. Thus the output after applying the 2x2 kernel would still be of size 4x4. Same padding allows creating CNNs of arbitrary depth.

How, i.e., with what pace, the kernel shifts over the data also affects the output dimensions. In the previous example, the kernel moves from each input directly to the next one, known as a *stride* of one. However, it is possible to leave out steps such that the kernel moves more than one step at a time resulting in a smaller output feature map and less computational effort, as only half as many values are computed. The stride can be specified for every dimension of the input tensor. For example, that only every second step in the horizontal axis and every step in the vertical axis is considered.

Pooling, also called sub-sampling or down-sampling, is a strategy applied to make the output of convolution operation invariant to small input changes. It replaces the output of a neuron with a summary statistic of its neighborhood. The neighborhood is a react-angular space around the neuron. Applying a pooling layer to an output feature map reports statistics such as the maximum, minimum, average, or L^2 norm instead of the actual output. Because summary statistics are reported over a whole neighborhood, they can be reported forneurons apart, reducing the input of the subsequent layer k neurons apart, reducing the input of the subsequent layer k times, thus improving the CNN's computational efficiency.

Having established the essential peculiarities of the convolution and pooling operations in CNNs, the general architecture of CNNs incorporates more. Just as for neurons in the NN, the entire feature map resulting from a convolution operation can be, and usually is, subject to an activation function. Moreover, a convolution layer is often composed of more than one kernel or filter. The kernels in the same convolution layer have the same size to produce feature maps of the same dimensions. Before the output layer with their task-specific activation functions, CNNs usually have some fully connected layers as in NN. The part of the convolution and pooling layers could be seen as a *feature detector*, whereas the fully-connected layers serve as a discriminator among the detected features in the data. Some textbooks refer to the combination of the convolution operation, the activation, and the pooling as one convolution layer (complex layer terminology). For this thesis, each is regarded separately, and a convolution layer only refers to the convolution operation (simple layer terminology). An exculpatory architecture of a CNN is shown in figure 2.8.

As mentioned before, CNNs are mostly used for computer vision tasks. The data found here is multi-dimensional. A color image has two dimensions of height and with but also comes

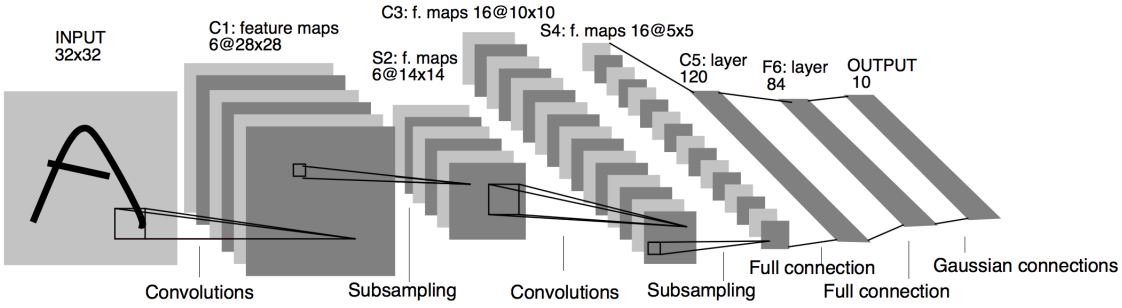

Figure 2.8: LeNet-5 [32]

with three different channels for each pixel to encode the color. Colored video has a third dimension of time. For this thesis, and in many libraries, the convolution over two-dimensional image data is defined as $2D$ - Convolution, which could sound counter-intuitive as the kernel actually has three dimensions, including the one for the channels. The convolution over video or volumetric data is defined as $3D$ - Convolution.

After the LeNet [31], many different CNN architectures constantly improved on what is possible and the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) emerged as the main challenge for researchers. One of the architectures to mention when dealing with CNNs are the AlexNet [28] (winner of the ILSVRC challenge in 2012) which popularized CNNs in computer vision, the ZF Net [51], the GoogLeNet [42], the VGGNet [41], and the ResNet [22]. The development of the CNNs shows two interesting things: Over the years, the CNNs got deeper (as particular research suggested as well [42]), with more stacked layers preceding the feed-forward layers. While the LeNet-5 picture only has two convolution layers the VGGNet already has 19, and the ResNet increased to 152 convolution layers. The researchers of the ResNet introduced *skip-connections* to deal with the problem of vanishing gradients, which occur during back-propagation of errors for deep neural networks (for a detailed explanation please see [22], the peculiarities of the back-propagation algorithm are not part of this thesis), which allowed them to stack that many convolution layers. In most studies, the kernel usually had a size of 3x3 or 5x5.

2.5 Genetic Algorithms

Genetic Algorithms (GAs) are evolutionary optimization algorithms inspired by the theory of evolution of species by Charles Darwin [13]. They rely on the principles of reproduction, adaption, inheritance, variation, and competition. GAs start with a set (*population*) of random initial solutions called *individuals*. They are represented as fixed-length strings. Through a *selection process*, some individuals are selected to undergo a variation phase where the effects of *crossover* (two solutions exchanging parts of their string representation) and *mutation* (parts of the string representation is modified) change the individuals. Solutions resulting from the variation phase are called *off-springs*. All off-springs form a new generation of the population,

who serve as parents for the next generation. This evolutionary process continues until an optimal solution is found or the maximum number of iterations is reached. Many different variants from the basic one described above exist.

For a more in-depth understanding, the next paragraph gives a short review of different selection methods, followed by a short overview of variation operators. At last, some examples of real-world applications are given.

Individuals with higher fitness must be more likely to be selected in the selection process. For all individuals, the probability of being selected must be bigger than zero, and selection can be made with repetition. Many different selection algorithms exist, and only three common examples are presented here. The fitness proportionate selection schemes select individuals with proportionate probability to their fitness. The selection pressure is highly influenced by the diversity of fitness values of the population, which can result in a problem of premature convergence [14, 20]. Ranking selections [4] rank the individuals according to their fitness, and the selection probability is set (inversely, for minimization problems) according to their rank in a linear or exponential relation. In Tournament selections, n individuals are randomly selected, and the best individual of that tournament is chosen for the variation phase. The selection pressure is dependent on the tournament size n where less fit individuals have a higher probability of being selected when n is small [14]. Tournament selection has the advantage that not all individuals' fitnesses have to be evaluated. This is of great advantage, as fitness evaluation is computationally expensive [11, 14, 20]. In general terms, high selection pressure allows for exploitation of the search space of solutions with the risk of getting stuck at local optima. On the other hand, low selection pressure allows for a high exploration of the search space but might be inefficient as fitness plays a less important role. Finding an appropriate balance in exploration and exploitation is crucial for evolutionary computation methods such as GAs. Selection can be further extended by the concept of elitism, which always keeps the best individual, without applying variation operators, for the next generation, thus not losing the current best solution. However, this implies knowing the fitness value of each individual.

While the selection of individuals depends on an individual's fitness, its *phenotype*, crossover, and mutation change the individual's string representation, i.e., its *genotype*. The most simple form of crossover is the one-point crossover. Here two individuals swap their string information from a certain point onward. The most simple mutation is the point mutation, where, with some probability, each character of the string representation can be replaced with a random one. Crossover is usually referred to as the conservation operator as no new information is added to the population's gene pool (genes are only swapped). In contrast, mutation is known as the innovation operator introducing potentially new information into the gene pool. In metric search spaces, *geometric variation operators* can be defined. The geometric crossover is defined as an operator that can produce only off-springs that are part of the same line segment between the two parents [37]. A line segment of a metric space (S, d) is the set

$[x; y] = \{z \in S | d(x, z) + d(z, y) = d(x, y)\}$ where $x, y \in S$ are called the extremes of the segment. A geometric mutation operator is defined as an operator that produces mutated individuals who are a member of a set of points surrounding the parent within a distance threshold d [36], which is why it is also referred to as *ball* or *box mutation*. If a uni-modal fitness landscape (see section 2.1) is given, specific properties arise from these operators, which are essential for the remainder of the thesis. The geometric crossover can not produce off-springs that have worse fitness than the worst of its two parents. Moreover, the geometric semantic mutation can only produce off-springs whose fitness derives as much as $[-d, +d]$ compared to the parent.

In the past decades, various optimization problems were solved by means of genetic algorithms. For instance in loan decision [35], portfolio optimisation [8], allocation of goods in shop shelves [7] as well as machine learning hyper-parameter optimisation [9, 34, 39]. They also represent the basis for Genetic Programming, which is presented in the next section.

2.6 Genetic Programming

Genetic Programming (GP) describes a set of machine learning techniques that belong to Evolutionary algorithms as GA and are introduced by Koza and Poli in 1992 [27]. GP relies on the same principles of evolution as GA and can be seen as an extension to them, with the difference that not fixed-length strings are evolved, but computer programs. This section first introduces general concepts of GP and later gives an overview of recent developments, especially regarding GSGP.

The computer programs in GP are usually expressed as syntax trees with *nodes* indicating the instruction to execute, and *links* which indicate the arguments to take as an input for the instructions. *Internal* nodes are functions, while *terminal* nodes are usually variables or constants. An exemplary program could be:

$$max(x * x, x + 3 * y)$$

which translates into a tree as seen in figure 2.9. It is also common to express programs in prefix notation such as Lisp S-expressions, where the function always precedes its arguments as here the correspondence between the tree is easy to see. The example above becomes:

$$(max(*xx)(+x(*3y)))$$

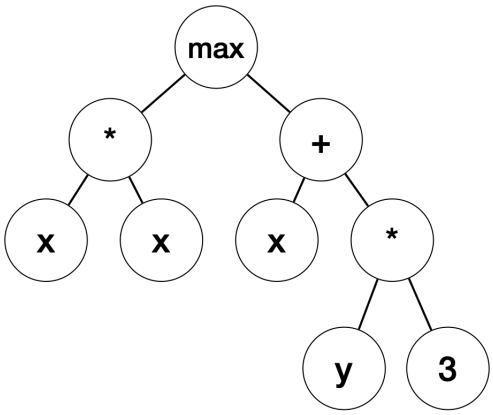

Figure 2.9: GP individual

The GP algorithm can be described as:

1. Generate an initial random population.
2. Repeat until the stopping criterion is reached:
   2.1 Select individuals who should act as parents
   2.2 Apply variation operators to generate off-springs
   2.3 Off-springs become the next generation's population.
3. Return the best individual.

While beforehand the set of functions and terminals, the stopping criteria have to be set. The selection, mutation, and crossover methods must be defined, and the initialization method and the fitness function. The latter usually is the error of the individuals executed over the whole training data set. Common error measures for Regression tasks are the Mean Squared Error or Mean Absolute Error. The most well-known measure for classification tasks is Accuracy, while the Cross-Entropy (CE) Loss is often more desirable to optimize as it gives more thorough information considering class imbalances and false positives as well. Selection works the same way as for GA; please see 2.5.

The set of functions and terminals should be combined as freely as possible. Thus it is important that all combinations of them are valid, which can be done using only one type or taking care of type checking though the cause of evolution. Moreover, evaluation safety must also be assured, e.g., introduce a safe division operator that hinders a division by zero. The set of functions and terminals is very problem-specific and thus needs fine-tuning for each task at hand. In the next paragraphs, different ways of building a GP individual, different criteria on how to stop the evolutionary process, as well as variation operators, and the latest achievements in GSPG are laid out.

The two basic ways of building a GP individual are the grow and the full method, which build the tree until a certain depth. As the name suggests, the grow method randomly samples terminals and functions until the maximum depth is reached or all nodes are terminals, resulting

in trees of heterogeneous depth. The full method implies that all trees have the same depth, the maximum depth, as only functions as sampled until it reaches adding terminal only at the very end. While these two could initialize the whole population, one could also initialize, with varying maximal depth, half the population with the full and the other one with the grow method called ramped half and half. Yet another initialization method, Evolutionary Demes Despeciation Algorithm, used subpopulations - or demes - which are initialized using the ramped half and half method [43].

In most cases, the evolutionary process stops if the maximum generation (iterations) is reached. For the specialty forms of GP, the GSGP and SLM (which will be introduced shortly), two stopping criteria that involve the development of the error, have been proposed and show competitive generalization results [16]. The Error Deviation Variation (EDV) measures the percentage of models, which perform better than the current model and have a lower sample standard deviation of absolute error than the current best model. This allows measuring if the training error reduction happens less uniformly, which might indicate a start of over-fitting. The search is terminated only when a significant majority of models improve training performance at the expense of larger error deviations. Another proposed criterion proposed is the Training Improvement Effectiveness (TIE), which measures the effectiveness of a semantic variation operator as a percentage of which it produces a superior neighbor. As the TIE drops below a certain threshold, it is hypothesized that as training error improvements are harder to find, they possibly come with a bigger trade-off to generalization.

Variation operators follow the same ideas as in GA but work on different representations. The simplest crossover in GP, the standard crossover, exchanges parts of two parents' sub-trees and creates two off-springs. Random nodes from the two parents are chosen as mutation points. The sub-trees of these mutation points are exchanged, and two off-springs are created as shown in figure 2.10. On the other hand, the standard mutation creates only one off-spring from one parent. Here, a mutation point is randomly selected from a parent, and the sub-tree is replaced by a random tree 2.11.

Whilst more mutation and crossover methods could be referenced, this thesis will have a particular look at the Geometric Semantic Mutation and the Geometric Semantic Crossover proposed by Moraglio et al., which introduced the era of Geometric Semantic Genetic Programming (GSGP) [36]. To understand GSGP, it is essential tp first introduce the concept of the *semantics* of a computer program. GSGP is seen as a supervised learning technique with inputs and corresponding outputs and the goal to learn the underlying function. The semantics of a program are the outputs of an individual over the data set, and each individual can be represented as a point in that space. The semantic distance is then defined as the distance, using the fitness metric, between the corresponding output vectors. This semantic distance serves as an indirect *genotypic distance* as it measures the distance of the error resulting from the genotype-phenotype mapping of the function over the data set. The semantic distance is only a pseudo metric of the

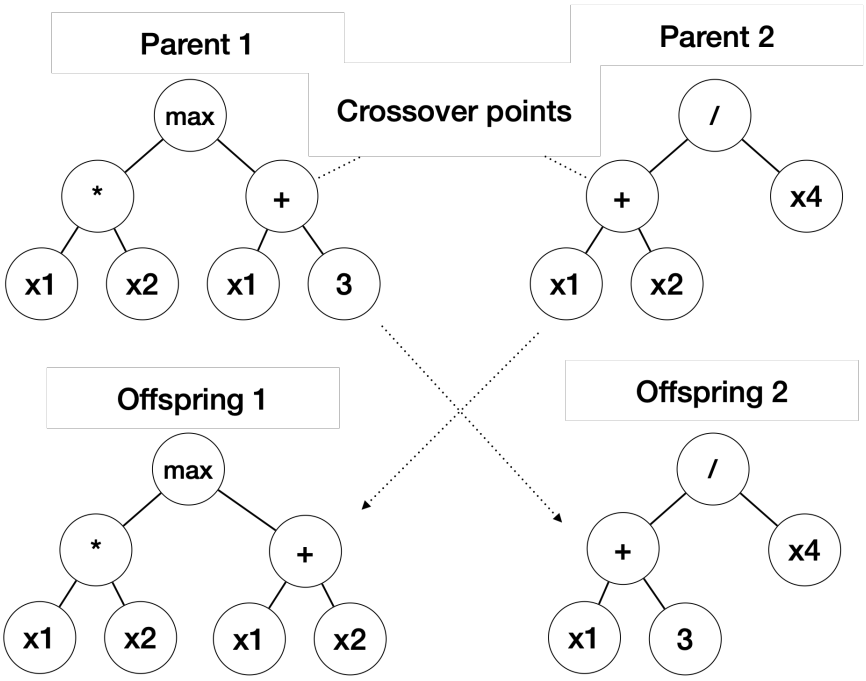

Figure 2.10: Standard Crossover GP

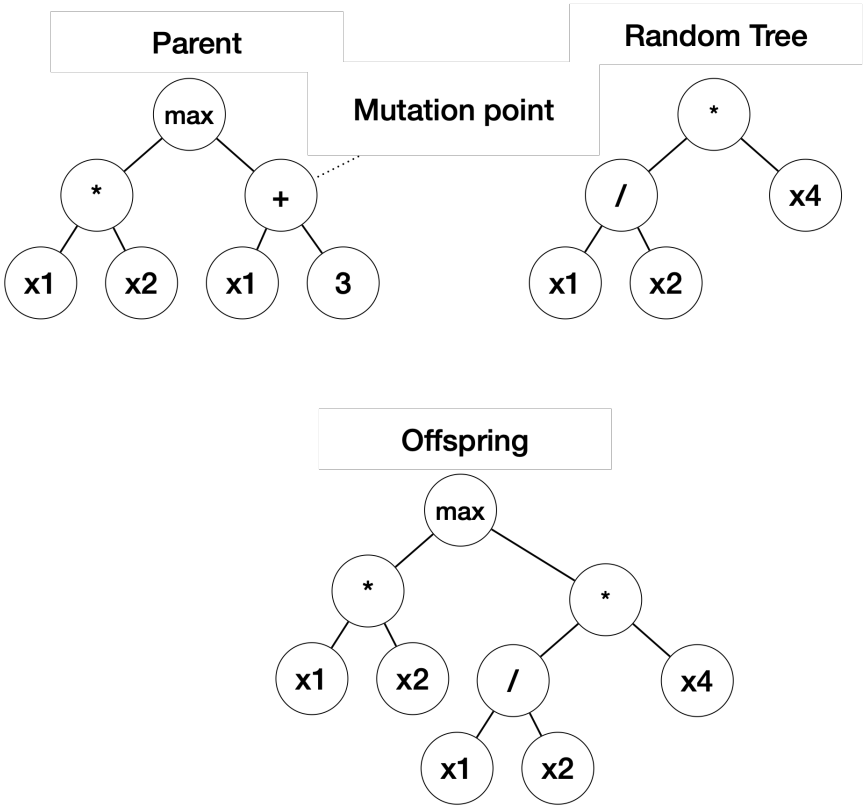

Figure 2.11: Standard Mutation GP

distance between two functions. Two different functions may lead to the same error resulting in a semantic distance of zero, violating the identity of indiscernibles principle of metrics. The semantic fitness of an individual is the distance to the optimum, which is known in supervised learning and is represented by the target vector y of the training set. Thus, the fitness of an individual is i to the distance of the global optimum resulting in a uni-modal semantic fitness landscape (see section2.1). This uni-modal fitness landscape is favorable in terms of search efficiency as no local optima exist. Further, Moraglio et al. [36] proposed semantic aware variation operators that make it possible to reach the semantic neighbors of an individual. The geometric semantic crossover and the geometric semantic mutation follow the same idea as described in section 2.5, such that for the crossover, the off-springs belong to the set of the line segment between the to parents in the semantic space and the geometric semantic mutation creates off-springs which lie within a certain *radius* of the parent in the semantic space.

Geometric Semantic Crossover: Given two parent functions $T_1, T_2 : \mathbb{R}^n \rightarrow \mathbb{R}$, the geometric semantic crossover returns the real function $T_{XO} = (T_1 * T_R) + ((1T_R) * T_2)$, where T_R is a random real function whose output values range in the interval $[0, 1]$.

Geometric Semantic Mutation: Given a parent function $T : \mathbb{R}^n \rightarrow \mathbb{R}$, the geometric semantic mutation with mutation step ms returns the real function $T_M = T + ms * (T_{R1} - T_{R2})$, where T_{R1} and T_{R2} are random real functions.

The main drawback of these operators is that the off-springs grow in size as the generations increase. The Geometric Semantic Crossover is causing *exponential* growth, while the mutation causes individuals' *linear* growth, making them hard to evaluate after some generations. Vanneschi et al. [44] proposed a new implementation of these operators, which work without actually building the new tree as a whole and then evaluate the data set. They propose to store semantics of the created trees in memory as they do not change, which allows them only to evaluate parts of the new trees. This allowed them to test GSGP for the first time on real-world multi-dimensional data sets and showed competitive results for training and unseen data [44]. For the sake of completeness, it should be noted that their implementation of the mutation operator always had a logistic function as a root node of the random trees, which implies that the perturbation effect of the mutation ranges in $[-ms, ms]$, albeit not explicitly mentioned it is shown in their GSGP library [6]. In extensive experiments, Gonçalves shows [17] that the bounding of the random trees dramatically impacts the generalization ability using various mutation steps and different bounding functions. Moreover, it is shown that the mutation operator is *vital* to drive the evolutionary process [17, 44]. In various tests, the Semantic Stochastic Hill Climber (SSHC) [36], which uses only the geometric semantic mutation operator to produce a neighborhood equal to the population size of other GSGP variations, is the fastest to adapt to training data. Vanneschi et al. [44] argue that it is the geometric properties of the variation operators which are responsible that GSGP shows such good results on unseen data. They propose that the properties hold independently of the data and thus also hold on unseen data. Moreover, Gonçalves argues

[17] that the bounding of the random trees is the reason why GSGP generalizes that well. When the generalization plateau is reached, and no further improvements can be made using the data at hand, the bounding accomplishes that the adversarial and beneficial mutations balance each other out as no arbitrarily large jumps in semantics could result from a single mutation.

As the bounded geometric semantic mutation showed to be the driving force for good generalization, Gonçalves also explored how an adaptive or better optimal mutation step would benefit the generalization ability of SSHC. Rewriting the mutation to the linear system

$$RI * ms = (t - P)$$

where RI is the vector resulting from subtracting the two random trees, t is the target vector of known values, and P the vector of the semantics of the parent, the mutation step ms can be computed using the Moore-Penrose pseudoinverse for each mutation. He also proposes another version with an extra variable weighting the parent. In experiments, it is shown [17] that the adaptive mutation operators fit the training data very fast and without over-fitting in the first generations but tend to overfit after some generations.

Recently, Vanneschi et al. [45] proposed another semantic mutation operator called random vector mutation which enables GP to memorize training data and generate predictions in a similar fashion as the K nearest neighbor algorithm. This approach overcomes the issues of individual growth, but relies on the training data to generate predictions by means of memorization instead of learning.

Chapter 3

Semantic Learning Machine

The Semantic Learning Machine [15, 17] (SLM) is an algorithm to evolve NNs (see 2.3) and relies on the principles of GSGP (see 2.6) and is effectively a geometric semantic hill climber in the search space of NNs. In this chapter, the theoretical basics of the SLM are explained, and subsequently, the latest research achievements are presented.

Generalizing the Geometric Semantic Mutation as a linearly independent combination of two computational elements, it can be applied to NNs and is now abbreviated as GSM-NN. In the context of NNs an already existing NN is combined with a randomly generated NN. The first proposal [15] only considers a single hidden layer per network, no bias, and the identity activation function at the output layer such that the similarities to GSGP are easy to follow and shown in figure 3.1. Using only a single neuron for the randomly generated NN, the single weight to the output neuron called *learning step* (ls) is equivalent to the *mutation step* ms in GSGP.

Using this GSM-NN mutation operator, the algorithm searches of the space of NNs in a unimodal fitness landscape **without the use of back-propagation**. The algorithm starts with a set of random networks of which the best is selected. Then neighbors are created using the GSM-NN operator, and the best one is chosen (just like in hill-climbing). This process continues until the stopping criterion is reached. The SLM algorithm can be summarized in the following steps:

1 Generate N random NNs.
2 Select the best performing NN (B) with regard to the chosen loss function.
3 Repeat until a stopping criterion is reached:
    3.1 Apply the GSM-NN operator to B N times to generate N neighboring NNs.
    3.2 Choose the best among the N neighbors with regard to the loss function as the next B, provided it is better than the current B.
4 Return B.

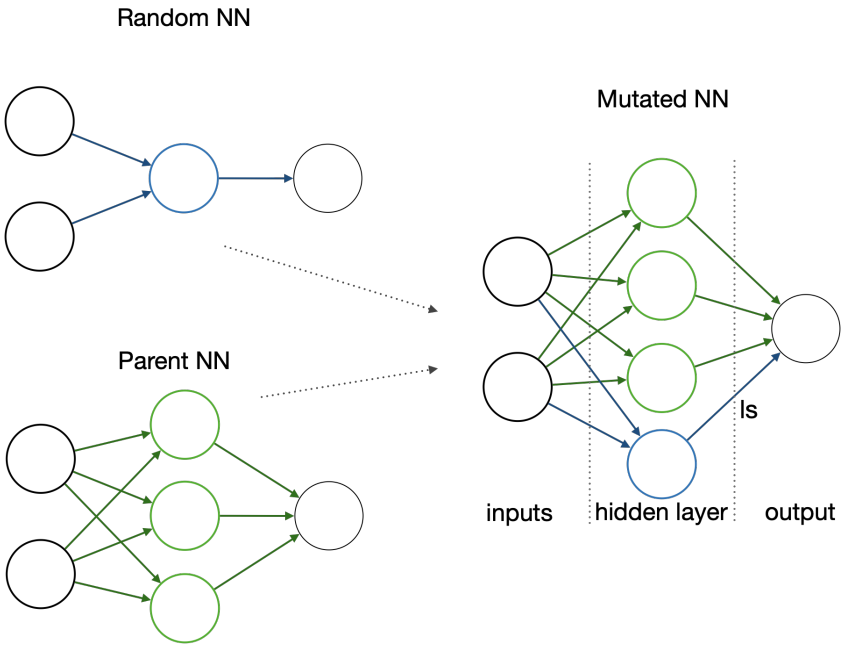

Figure 3.1: Application of GSM-NN

The initial random NNs can be generated very freely with any number of hidden layers and neurons using any activation function and connection weights. Gonçalves shows multiple ways the initial GSM-NN operator could be extended to work for several hidden layers [17]. The work of Jagusch et al. [24] adds one neuron to each hidden layer and connects to a random subset of neurons from the previous layer. While Lapa et al. [29] add multiple new neurons to each hidden layer. Each neuron can select freely from which neuron of the previous layer connects are received. The mutated network is no longer fully-connected and its sparseness can be controlled by defining how many input connections a new neuron will receive. It should be noted that this formulation can imply more than linear growth of individuals if the number of connections grows arbitrarily large. Further, outgoing connections must only connect to *new* neurons which are the result of the mutation, and only connect to the output neuron(s) to ensure that the mutation actually is a linearly independent combination of two computational elements.

The weights to the output neuron(s) can be computed optimally for the initial random and the mutation. In the case of a linear activation function and a linear loss function the weights can be computed using the Moore-Penrose pseudoinverse as described at the end of section 2.6 and by [15, 17].
The stopping criterion can be the set as the maximum number of iterations or using the TIE or EDV criteria (see section 2.6) or any combination of them.

In experiments, it is shown that the SLM exhibits the same characteristic with good performance and only slight over-fitting after generations as standard SSHC with a bounded mutation. Moreover, using an optimal computed learning step, the SLM outperforms the SSHC with statistical

significance on all experiments [17]. The concept of sparseness, setting a random subset of weights of the randomly generated network to zero, is explored and showed a positive impact on the training error [17]. Moreover, it is shown that the EDV stopping criterion is very helpful for generalization if the optimal learning step is computed [17]. In further comparisons with other supervised machine learning algorithms, the SLM showed highly competitive results in terms of generalization error and computational time required [17, 24]. Lapa et al. [29] use the SLM as a classifier instead of traditional fully connected layers in a CNN and outperform a state-of-the-art CNN architecture. Further, they propose to use the technique of the geometric semantic mutation to evolve the convolution layers of the CNN, and not only the fully connected layers at the end. This suggestion is taken up in the next section, and the Deep Semantic Learning Machine to evolve Convolutional Neural Network typologies is presented.

Chapter 4

Deep Semantic Learning Machine

The Deep Semantic Learning Machine (DSLM) is designed to evolve CNN *topology* with the geometric semantic properties as GSGP or the SLM. This chapter first sets a high-level overview of how the algorithm is intended to work and sets out new notations for different parts of the CNN, making it easier to extend the SLM to the DSLM. In the next step, the challenges of random initialization of CNNs are addressed, and an initialization technique is proposed. Subsequently, possible mutation operators are discussed, and at last, a method to compute the optimal weights if non-linear loss functions (or activation functions at the output layers) are used is presented in brief.

Generally speaking, the DSLM algorithm follows the same general logic as the SLM as a Hill Climber for CNNs in a uni-modal fitness landscape:

1 Create N random CNNs as individuals.
2 Choose the best CNN (B) with regard to a loss function
3 Repeat until stopping criteria is reached:
    3.1 Generate N neighbors of B by applying geometric semantic mutation operator.
    3.2 Replace B with the best neighbor, provided it is better than the current B.
4 Return the best individual

The part of a CNN without convolution or pooling layers is now referred to as the Non-Convolution Part (NCP), the part with convolution and pooling layers (CP-layers), which serves as a feature detector, now referred to as the Convolution Part (CP). The initialization and mutation of the NCP is essentially covered by the SLM. The initialization of the CP brings challenges that are not present in the SLM or traditional GSGP and the geometric semantic mutation of the CP has to be defined.

In GP and GSGP it is the goal to create random individuals; however, it is impossible to freely combine different CP-layers. The output feature map of a CP-layer must be compatible with the subsequent CP-layers settings (i.e., kernel size, padding, etc.). Ensuring that only *valid*

combinations are made during the construction of the random CP is thus essential. This can be achieved by setting a *limit* on the dimensions of the output feature map. If that limit is reached, no further CP-layers shall connect to it.

The second aspect to consider is the trend that CNNs got deeper and deeper with many stacked layers, which is not particularly useful as the algorithm is designed to store intermediary outputs in memory as proposed for GSGP [44]. CNNs typically work with image data, which already has high memory requirements; thus, storing all intermediary outputs can become challenging depending on the size of the data set and the depth of the network (typically, hard tasks will require much data and many stacked layers). This issue will not be dealt with in this thesis, but the general concept of the DSLM is set out and tested initially.

There are many possible ways on how to create a random CP. It should consist of data-specific CP-layers, which means that for image data, $2D$-Convolution layers that are channel aware and $2D$-Pooling layers should be used. They should be combined as freely as possible until a certain depth is reached and the NCP begins. To ensure that only valid connections of CP-layers are made the dimensions of the output feature map of each CP-layer **must be tracked**. Further, extending the idea of skip connections, the CP should be allowed to have a certain width, meaning that one output feature map can be connected to **multiple** pooling and/or convolution layers such that two or more simultaneous streams of operations exist and interact with each other if dimensions allow for it. The proposed DSLM uses the Tensorflow Keras Functional API [1] for the CP as many pre-build high-performance layers can be used. For a detailed description of each component, the reader is referred to the library documentation.

While there are many other possible ways to create a random CP using the method visualized in the figure 4.1 allows to create simple CPs without skip connections, CPs with skip connections as the ResNet [22] and also simultaneous streams of operations as described above. In the pseudo code, a random Conv2D layer refers to a Conv2D layer with a random kernel size, random padding, and stride setting, random amount of filters, a random kernel initialization method, and a random activation function, which are all subject to a user defined range of values. A random Pooling2D layer can be of AveragePooling2D or MaxPooling2D with varying strides, pool size, and padding. Also, each layer has its own depth parameter which is one more than the previous layer's depth, but Concatenation layers have the same depth as they only concatenate tensors and no pooling or convolution operation is performed.

Building the NCP follows the same logic as building the random NN as proposed for the SLM (see section 3), with multiple hidden layers containing multiple neurons and using the CP's output as input values.

Further, a geometric semantic mutation operator must be defined for CNNs and shall be called GSM-CNN. While there are also many possible ways of defining such a variation operator, the

```
Result: A random CP
Add a Inputlayer as the initial layer to the CP;
while all layers without subsequent connections are not of type Flatten do
    if max width of the CP is reached then
        Choose a random layer (I) not of type Flatten and without subsequent layers to
         serve as input for the next layer;
    else
        Choose a random layer (I) not of type Flatten to serve as input for the next layer;
    end
    if I's output dimensions are above a predefined threshold, or I's depth is equal to max
     depth then
        add a Flatten layer subsequent to I;
    else
        if any other CP layer's output dimensions matches I's output dimension then
            Either randomly sample from those layers and add a Concatenate layer to them
             and I and deduct 1 from the current CP width for each layer other than I
             without subsequent connections. Or add a random Conv2D or Pooling2D
             layer subsequent to I.
        else
            Add a random Conv2D or Pooling2D layer subsequent to I.;
        end
    end
end
```

Figure 4.1: Pseudo code: Building a random CP

most simple approach is to create a new random CNN and combine it linearly independently at the output neuron(s). The new random CP can be created the same way as described by figure 4.1, starting at the second if statement using the Inputlayer as the initial I and and resetting the current depth of the CP. Previously computed feature maps can also serve as additional inputs through a Concatenation layer, the same way a new neuron can connect to all neurons of the previous layer in the GSM-NN mutation. However, for the CP, the horizontal and vertical dimensions of all output feature maps, which should serve as a combined input for a CP-layer, must match (indicated by the dotted line).

Another approach to mutating the cp could be to start the growth of the new CP not from the input layer but from a randomly selected CP-layer. This could be useful as it would allow the CP to grow in depth, which might generate more detailed features. However, this approach is not tested here.
Subsequent to mutating the CP, the NCP is mutated the same way as if using the GSM-NN operator used by Lapla et al. [29] with the difference that the number of inputs to choose from is extended by the last neurons of the new random CP. Otherwise, the NCP could not benefit from any newly created features of the CP. The concept of the GSM-CNN operator is visualized in figure 4.2, for simplicity leaving out the concatenation layer. All researches have shown significant benefits of computing the optimal learning step. Thus the GSM-CNN should always compute the optimal learning step.

Summarizing the GSM-CNN operator:

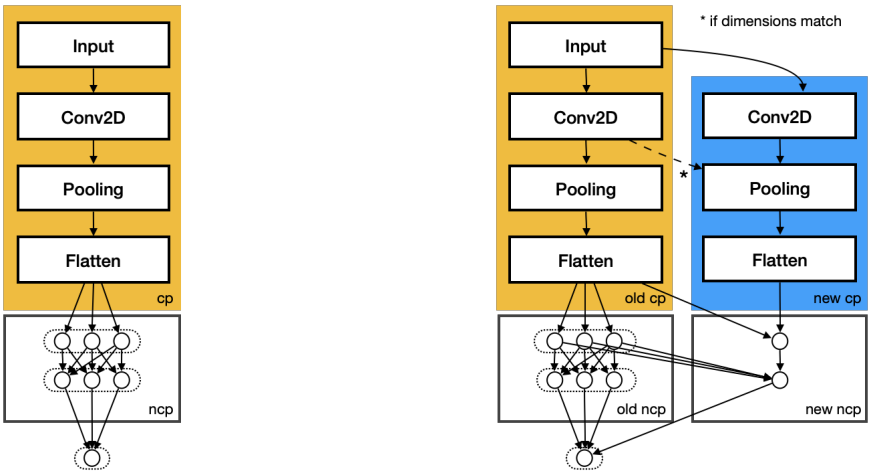

Figure 4.2: GSM-CNN example

1. Grow an additional random CP from the input layer.
2. Include the outputs of the additional CP as input neurons to any new NCP.
3. Create a new NCP using the GSM-NN operator.
4. Compute the optimal learning step.

Using this GSM-CNN mutation operator to evolve CNNs, the DSLM resembles a search of various CNN architectures under the same uni-modal fitness landscape as for the SLM or GSGP. There is no need to optimize weights or kernels using the traditional back-propagation algorithm.

Considering the experimental classification task in section 5 with 10 different class labels, the setup of using only one neuron without activation function as the output should be changed. Instead, 10 output neurons using the softmax function as activation should be used. This allows calculating the CE loss, a well-suited metric for classification tasks. This, however, also changes the way an optimal learning step can be computed; due to the non-linearity, the Moore-Penrose pseudoinverse method can not be applied. Figure 4.3 shows the weights from the new hidden neurons to the output neurons with softmax activation function, which are subject to optimization, leaving out the bias weights for simplification. The goal here is to find the weights from the new NCP to the output neurons, which minimize the CE loss of the whole data set, computed using the softmax activation function while keeping the inputs of the old NCP fixed. A possible approach to solve such large-scale non-linear optimization problems is using the L-BFGS-B algorithm [5] which is not further explained in this thesis. Using the L-BFGS-B optimization technique allows to always generate better off-springs as in the worst-case scenario, the mutation will have no impact on the individual's semantics as the new weights can have a value of zero. Using the gradient descent algorithm could be another possible solution to finding these optimal weights (not training all weights).

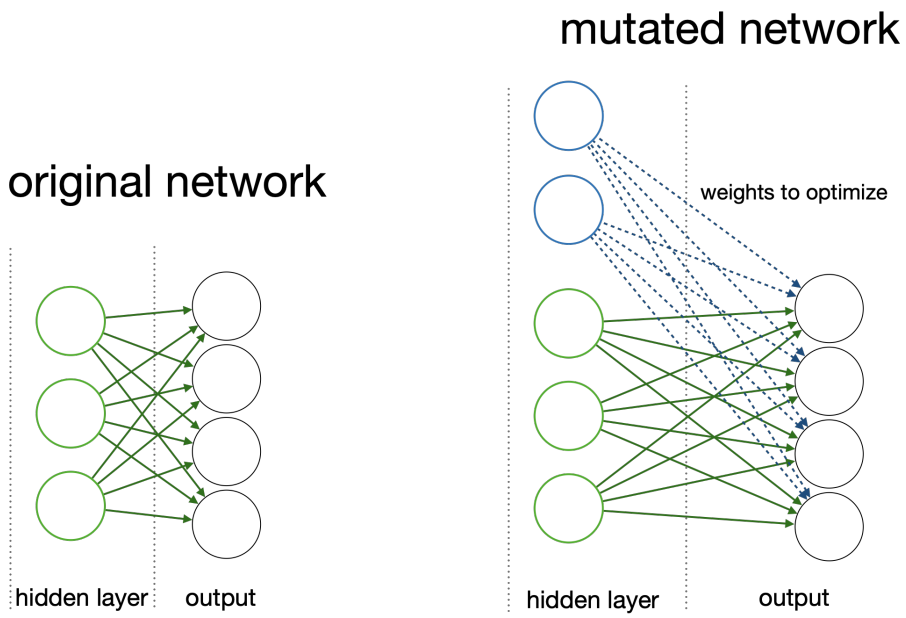

Figure 4.3: Multi-class classification output connection

Chapter 5

Experiments

To obtain first indications if the proposed DSLM algorithm does work, it will be tested on a bench-mark data set and compared to the SLM algorithm and a traditional CNN architecture in terms of training fitness and generalization ability.

5.1 Dataset

As this research paper represents initial work, no real-world data sets will be tested, especially considering the problems which might occur due to individual growth. The algorithms will be tested on the mnist data set [33] (examples in figure 5.1) which contains 60,000 train and 10,000 test grayscale images of handwritten digits $[0; 9]$ of size 28x28. The data set is balanced, meaning each class is represented equally. This data set, albeit only of scientific relevance with no use case for the real world, is the biggest data set on which the SLM and DSLM have been tested on (to the best knowledge of the author) and represent a challenge by itself. The GSM-CNN and GSM-NN operators imply (at least) linear growth of the individuals, which makes evaluation quite challenging after a few generations due to memory and computational constraints.

5.2 Experimental Settings

For the experiments, the following parameters are used to build the CP and the NCP for the DSLM and the SLM. In the CP:

activation functions: ReLU, Tahn and Linear
strides: 1,1 and 2,2
padding: valid and same
number of filters: [1,5]
kernel size: 3,3 and 5,5
pooling size: 3,3 and 5,5

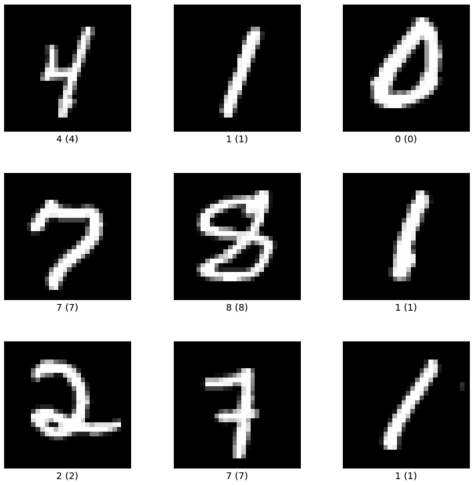

Figure 5.1: MNIST handwritten digits examples

The threshold of the dimensions of the output feature map from which only Flatten layers can be added is set to 5x5.
For the NCP, ReLU is always used as an activation function, and each neuron and bias weight is constrained by 0.1. Sparseness is set to a value within 0.75 and 1, and the optimized learning step is always computed, and, if not stated otherwise, 100 neurons per hidden layer are used.

For the DSLM the effect of different parameters of the maximal depth of the CP, the maximal width of the CP, different values for the number of hidden layers in the NCP is tested simultaneously to a different number of new neurons per mutation per hidden layer. Two different values for the number of neurons in the last hidden layer, two different neighborhood sizes, and allowing to apply only the GSM-NN mutation, i.e., only mutation the NCP, are tested and presented in the next section. If not otherwise stated, the base values for the DSLM and SLM tests are:

max depth CP: 3 (only DSLM)
max width CP: 1 (only DSLM)
hidden layers: 2
new neurons per hidden layer: 10
neurons in last hidden layer: 25
NCP mutation as standalone allowed: yes (True) (only DSLM)
neighborhood size: 2

These initial values are chosen to use as little memory and computational resources as possible. For the classification task, the CE is used as a loss for training and validation (only every second generation) over ten generations using five different seeds. Please note that not all generations contain values for all five seeds, as in some runs, memory issues occurred, or the individual evaluation took too long. Such cases are marked with asterisks (*).

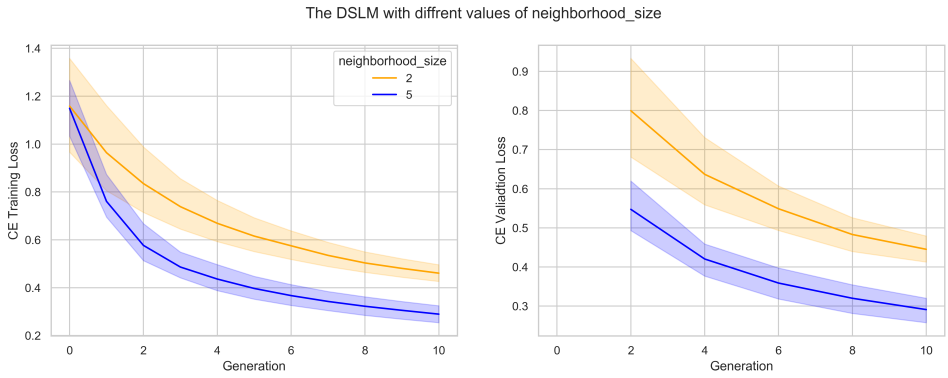

Figure 5.2: DSLM neighborhood size

The CNN architecture used for a benchmark is derived from the LeNet-5 presented in section 2.4. However, the input dimensions are changed to 28x28x1 to fit the mnist data set dimensions. Adam is used as a weight optimizer. For comparison reasons, the batch size is set to equal the data set size such that each epoch only contains one batch. Evolving the SLM or DSLM for ten generations with a neighborhood size of five implies that 55 evaluations of the training data happen. Thus the number of epochs of training is limited to 55 for the CNN as well. In another test, a batch size of 265 images is chosen, which allows updating the network's weights more frequently on a subset of the whole data set while keeping the number of epochs fixed.

5.3 Experimental Results

At first, the results of testing various DLSM parameters are presented. Subsequently, the SLM with different parameters is presented, and a comparison is made. At last, they are compared to the two CNNs trained with Adam as described above. The dark line represents the mean value computed using the five different seeds, and the shaded area marks the and 95% confidence interval.

A bigger neighborhood means that the chance of finding a better neighbor increases as more neighbors are explored and should allow a faster convergence. The results of figure 5.2 indicate the same without any noticeable over-fitting on validation data. However, the subsequent test will still be carried out using a neighborhood of size two as it is less computationally expensive and does not limit the evolution of a single CNN by any means.

Allowing to evolve only the NCP of the CNN by only applying the GSM-NN operator by chance is tested. During the cause of network evaluation, the new features generated by the mutated CP might improve the discriminability among different classes at a decreasing rate. However, further evolving the NCP, which discriminates given the CP's features, could be of

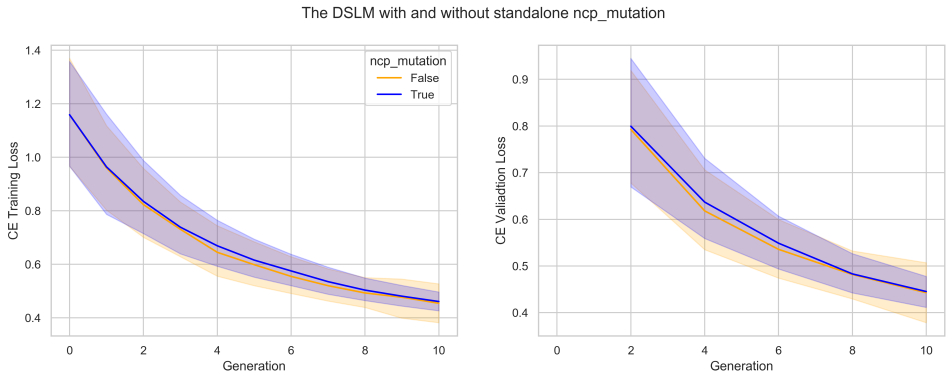

Figure 5.3: DSLM ncp standalone *

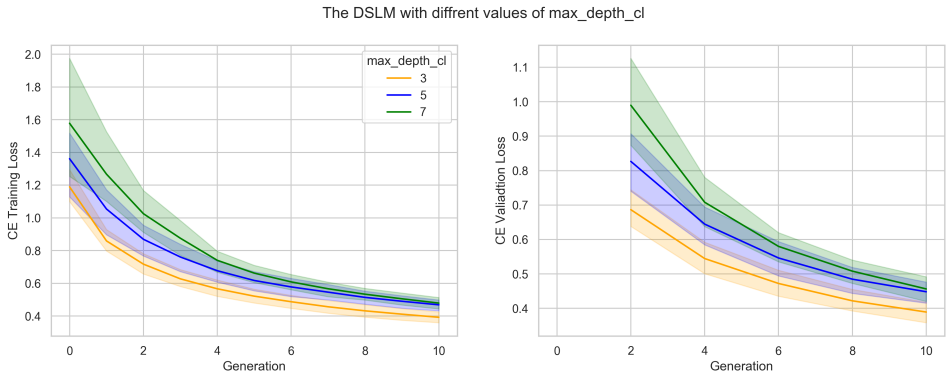

Figure 5.4: DSLM max depth cp values

higher importance in the later generations. The results shown in figure 5.3 show almost an equal reduction in mean CE loss on both training and validation data. However, the 95% confidence area is substantially smaller for the later generations if the GSM-NN operator can be applied by chance instead of always applying the GSM-CNN operator. On the other hand, they are always applying the GSM-CNN operator results in some lost data points for the later generations. Thus, concluding that the higher confidence results from applying the GSM-NN operator could be misleading. To create more reliable results in the subsequent tests, with fewer missing data points, the GSM-NN operator is used by chance to evolve the CNN in the subsequent tests.

The values of 3, 5, and 7 as max depth parameters for the CP tests if more stacked CP-layers allow generating more detailed features, resulting in better performance of the DSLM. The results as shown in figure 5.4 indicate that, for the mnist data set, the maximal depth of 3 for the CP yields the best performance in terms of CE loss after ten generations without noticeable over-fitting. This could indicate that for this data set, the detail level of features is less critical, and thus, the value of 3 is chosen to be continued in the subsequent tests.

Likewise, the max width parameter is tested with values of 1, 2, and 3, whilst one represents a simple CNN architecture (with possible skip connections). The others allow creating separate

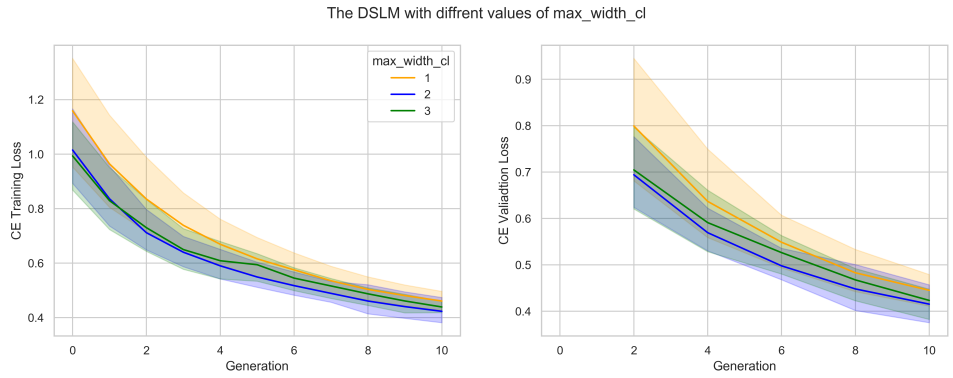

Figure 5.5: DSLM max with cp values *

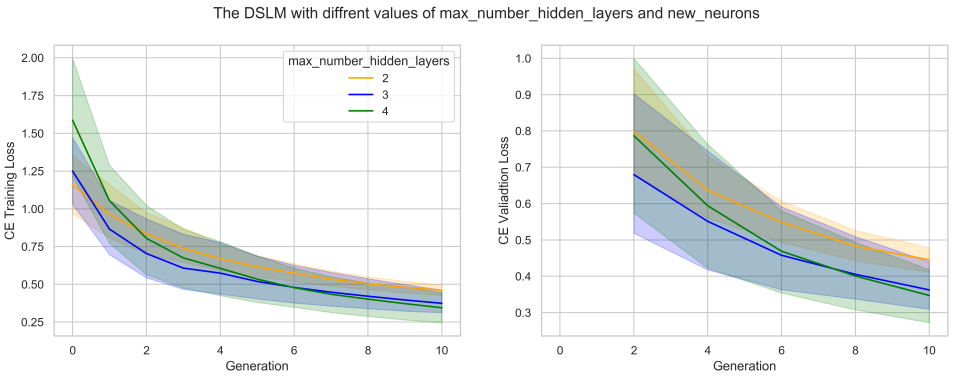

Figure 5.6: DSLM number of hidden layers and number of new neurons

streams of operations in the CP, allowing to create more diverse features. The results of figure 5.5 show that a higher width yields better results for the mnist data set on both training and validation loss. However, the later generations do not represent five different runs for the reasons described above. The value of 1 as the max width parameter is kept to generate more reliable results for the subsequent experiments. However, the idea of separate streams of computation to generate more diverse features is not proven to be wrong.

In figure 5.6 the results of using different parameters regarding the complexity of the NCP are shown. As mentioned above, the NCP is mostly responsible for discriminating the features computed by the CP. Here the value pairs of (2,10), (3,25), and (4,50) are tested for the number of hidden layers and the number of new neurons added per layer by the mutation operator. More layers and more new neurons per layer yield better performance in CE loss on training and on validation data and do not show any over-fitting in the first ten generations. Thus for the next test, the values of (4,50) are kept.

The impact of the layer size of the last hidden layer for creating the initial random NCP is evaluated, using a neighborhood size of 5 and (4,50) for the number of hidden layers and the number of new neurons per hidden layer. The weights from the neurons in the last hidden

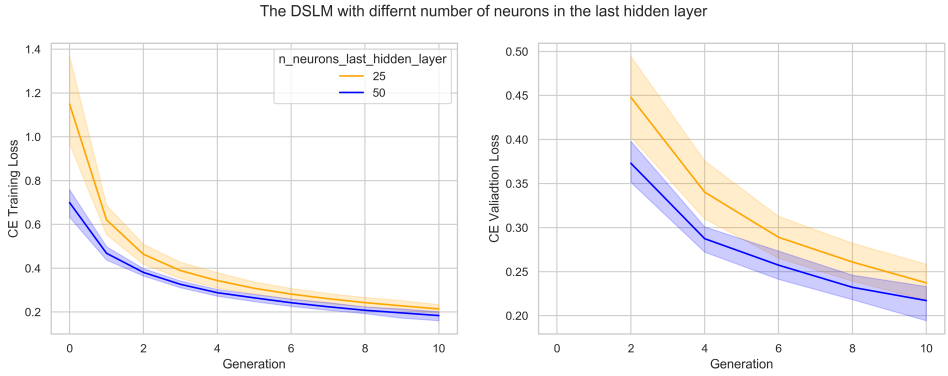

Figure 5.7: DSLM neurons last hidden layer *

layer of the mutated part of the networks to the ten output neurons are computed using the L-BFGS-B algorithm described in section 4. The more neurons are present in the last hidden layer, the more degrees of freedom are present to compute the optimal step and thus, should lead to a lower CE loss in the first generations. The results indicate that the loss is indeed much lower in the first generations and also stays lower in the subsequent generations. In total, 8 data points from generation 4 onward are missing for the run, with only 25 hidden neurons in the last layer. However, the confidence interval of 95% does not seem to change much during generations. One can notice a slight increase in the confidence interval from generation 8 to 10 for 50 hidden neurons in the last layer and less reduction of validation loss compared to 25 hidden neurons in the last layer. This phenomenon could be of interest for future studies but is beyond this thesis. The validation loss is still continuously reduced, and no over-fitting is happening.

The SLM is tested with the same number of different hidden layers and new neurons per layer and two and five as neighborhood sizes. The different number of neurons in the last hidden layer of initially building the random NN. The SLM exhibits the same behaviors as the DSLM for all tested parameters, but the number of hidden neurons in the last layer upon initialization. The results can be found in appendix A. The best setup resulting from these tests uses a neighborhood size of 5, 4 hidden layers with 50 neurons added to each layer per mutation and 25 neurons in the last hidden layer.

The major difference between the SLM and the DSLM is that the DSLM has the chance to create meaningful features of the input image through the random CP. On the other hand, the number of features will grow linearly by applying the GSM-CNN, bearing the risk of the curse of dimensionality for the NCP. This risk is met by allowing the GSM-NN operator to be applied by chance instead of always applying the GSM-CNN operator. The interesting question now is if the features produced by the CP contain enough information, even for the toy example of handwritten digits, to allow the NCP to discriminate better among classes than the SLM on the raw image data. The results in figure 5.8 compare the best-found parameters from the tests above. The DSLM uses a max depth of CP of 3, max width of the CP of 1 and allowing the GSM-NN

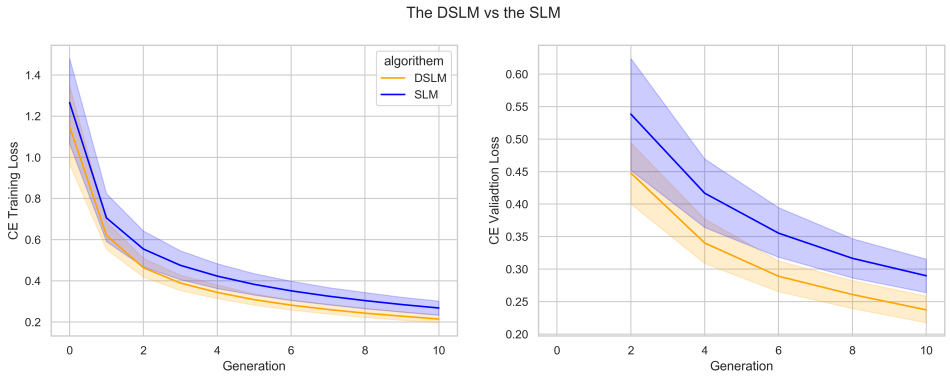

Figure 5.8: DSLM vs. SLM

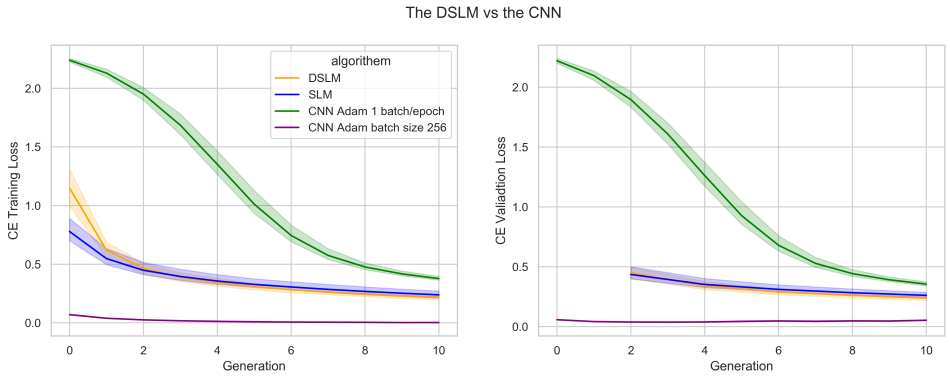

Figure 5.9: DSLM vs. SLM vs. CNN

mutation to be applied by chance. Both algorithms use 4 hidden layers, adding 50 neurons per layer in any mutation and having 50 neurons in the last hidden layer for the DSLM and 25 for the SLM. A neighborhood size of 5 is used for both. The results indicate that both the SLM and the DSLM are able to reduce the CE loss for training and validation data significantly and without over-fitting within the tested ten generations. The DSLM is able to slightly outperform the SLM for both training and validation loss. This contradicts the cause of dimensionality resulting from too many features and encourages the idea that the random CP can indeed produce features that are useful for discrimination. However, this test does not allow to conclude anything with statistical significance and thus calls for more extensive research regarding the DSLM. Also it should be noted that the SLM is not specifically designed to work on two dimensional data.

Comparing the results of the DSLM and SLM with the traditional way of training CNNs is a difficult task and is arguably influenced by the unusual batch size. Also, no different values for the various hyperparameters are tested. Figure 5.9 shows the training and validation losses of the best found set up of the DSLM and SLM and the CNN as described in the section above. For better comparison, only every fifth epoch is shown stating at epoch 5 to resemble the amount of fitness evaluations at each generation. The line plot shows that, especially in the first generations, the DSLM and SLM converge much faster than the CNN trained using the adam optimizer and

one batch/epoch. However, in the later generations, the performance of the DSLM and SLM is almost met. Using batches of 256 images the weights of the CNN are changed more than 200 times more often while the whole data set is evaluated the same number of times. Here the CNN with Adam outperforms the SLM and the DSLM clearly, even considering the first epoch only.

Chapter 6

Limitations and Future Work

The results presented in the section before indicate that the random CP of the DSLM algorithm could be able to produce features on which the NCP part can discriminate better than the SLM on the original data. However, the tests obtained in the previous sections are not suitable for statistical analysis to compare the performance in a scientific way. Further tests using more datasets need to be carried out. Thus, this work can only be seen as a proof of concept of the DSLM without clear benchmarks to other algorithms. This is because the image data set tested, albeit being only a toy example, clearly shows the negative impact of (at least) linear growth of the individuals making the current approach of evaluating individuals over the whole data set and keeping each intermediary semantics in memory not efficient for larger real-world data sets. Storing the intermediary semantics to the hard drive is not the solution to the problem; this would slow down evolution even further. Making use of the semantic stopping criteria can be beneficial to reduce the computational requirements and finding the best generalizing model. Before exploring different strategies on other possible formulations of the GSM-CNN operator, different hyperparameters, and even bigger data sets, the performance issues in bigger data sets should be addressed.

The use of batching seems like a promising way of addressing computational and memory issues that come with the growth of individuals and could potentially speed up the convergence. Batching is especially promising, considering that the geometric properties of the variation operator are argued to also hold on unseen data [44]. Further, the whole evolution process is highly parallelizable as each created neighbor can be evaluated separately; thus, also the use of big data technologies and distributed computing frameworks could be explored to address the performance issues. Considering the size of individuals after the evolutionary process and the resulting long evaluation time of even a single data instance, the DSLM algorithm's models will not be suitable for tasks where computing time is crucial such as real-time annotations or similar. In domains where computing time is not that crucial, but the quality of the prediction is of most importance, i.e., in radiomics, the DSLM models could excel, provided they are proven to be superior to traditional CNNs.

Chapter 7

Conclusion

This thesis had the objective to propose a way to optimize the CP-layers of a CNN with a geometric semantic mutation operator to be able to extend the SLM to the DSLM. In the beginning, the theoretical background on NNs and CNNs is established to understand how they work and what problems are to be dealt with when proposing a geometric semantic mutation operator for the CP of CNNs. Moreover, the motivation of using a geometric semantic mutation operator is given: A supervised learning problem where the error is measured as a distance to the known target vector the search space of functions resembles a unimodal fitness landscape. Applying the geometric semantic mutation operator allows to effectively explore this search space by means of creating semantic neighbors of the function. A way to modify the CP-layers is shown, and extending the GSM-NN, the GSM-CNN is proposed. Using the same hill-climbing strategy, the SLM is extended to the DSLM using the GSM-CNN operator to optimize the whole topology of a CNN. The DSLM is tested with the goal to recognize handwritten digits. Here the DSLM shows promising results, albeit of no statistical significance. Moreover, the downsides of using the semantic operator as described by Vanneschi et al.[44] become very clear with image data and CNNs as storing the semantics intermediary imposes high memory requirements through the linear growth of individuals. Also, the gain in computational complexity is prominently adverse, as, for some individuals, the evaluation time took extremely long, making the algorithm, as of now, not useful in practice. In the end, future research questions are raised, with the idea of using batching to make the algorithm more practical. The future relevance of the DSLM still has to be established.

Bibliography

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[2] U. R. Acharya, H. Fujita, S. L. Oh, Y. Hagiwara, J. H. Tan, and M. Adam. "Application of deep convolutional neural network for automated detection of myocardial infarction using ECG signals." In: *Information Sciences* 415-416 (2017), pp. 190 –198. ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2017.06.027. URL: http://www.sciencedirect.com/science/article/pii/S0020025517308009.

[3] Adobe. *Photoshop: Now the world's most advanced AI application for creatives, year = 2020, url = https://blog.adobe.com/en/publish/2020/10/20/photoshop-the-worlds-most-advanced-ai-application-for-creatives.htmlgs.oj8hr9, urldate = 2020-12-27*.

[4] J. E. Baker. "Adaptive selection methods for genetic algorithms." In: *Proceedings of an International Conference on Genetic Algorithms and their applications*. Hillsdale, New Jersey. 1985, pp. 101–111.

[5] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu. "A limited memory algorithm for bound constrained optimization." In: *SIAM Journal on scientific computing* 16.5 (1995), pp. 1190–1208.

[6] M. Castelli, S. Silva, and L. Vanneschi. "A C++ framework for geometric semantic genetic programming." In: *Genetic Programming and Evolvable Machines* 16 (Mar. 2014). DOI: 10.1007/s10710-014-9218-0.

[7] M. Castelli and L. Vanneschi. "Genetic algorithm with variable neighborhood search for the optimal allocation of goods in shop shelves." In: *Operations Research Letters* 42.5 (2014), pp. 355–360.

[8] T.-J. Chang, S.-C. Yang, and K.-J. Chang. "Portfolio optimization problems in different risk measures using genetic algorithm." In: *Expert Systems with applications* 36.7 (2009), pp. 10529–10537.

[9] C. Di Francescomarino, M. Dumas, M. Federici, C. Ghidini, F. M. Maggi, W. Rizzi, and L. Simonetto. "Genetic algorithms for hyperparameter optimization in predictive business process monitoring." In: *Information Systems* 74 (2018), pp. 67–83.

[10] H. A. Eiselt and C.-L. Sandblom. "Heuristic Algorithms." In: *Integer Programming and Network Models*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 229–258. ISBN: 978-3-662-04197-0. DOI: 10.1007/978-3-662-04197-0_11. URL: https://doi.org/10.1007/978-3-662-04197-0_11.

[11] Y. Fang and J. li. "A Review of Tournament Selection in Genetic Programming." In: Oct. 2010, pp. 181–192. ISBN: 978-3-642-16492-7. DOI: 10.1007/978-3-642-16493-4_19.

[12] M. GmbH. *MindPeak sets standards*. 2020. URL: https://www.mindpeak.ai/2020/05/10/together-with-din-deutsche-institute-fÃr-normung-e-v-mindpeak-sets-standards/ (visited on 12/27/2020).

[13] D. E. Goldberg. "Genetic algorithms in search, optimization and machine learning." In: *Reading: Addison-Wesley, 1989* (1989).

[14] D. E. Goldberg and K. Deb. "A comparative analysis of selection schemes used in genetic algorithms." In: *Foundations of genetic algorithms*. Vol. 1. Elsevier, 1991, pp. 69–93.

[15] I. Gonçalves, S. Silva, and C. M. Fonseca. "Semantic learning machine: a feedforward neural network construction algorithm inspired by geometric semantic genetic programming." In: *Portuguese Conference on Artificial Intelligence*. Springer. 2015, pp. 280–285.

[16] I. Gonçalves, S. Silva, C. M. Fonseca, and M. Castelli. "Unsure when to stop? ask your semantic neighbors." In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2017, pp. 929–936.

[17] I. C. P. Gonçalves. "An exploration of generalization and overfitting in genetic programming: standard and geometric semantic approaches." Doctoral dissertation. Universidade de Coimbra, 2017.

[18] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[19] B. Hajek. "COOLING SCHEDULES FOR OPTIMAL ANNEALING." English (US). In: *Mathematics of Operations Research* 13.2 (1988). Copyright: Copyright 2017 Elsevier B.V., All rights reserved., pp. 311–329. ISSN: 0364-765X. DOI: 10.1287/moor.13.2.311.

[20] P. J. Hancock. "An empirical comparison of selection methods in evolutionary algorithms." In: *AISB workshop on evolutionary computing*. Springer. 1994, pp. 80–94.

[21] D. Haussler. "Quantifying inductive bias: AI learning algorithms and Valiant's learning framework." In: *Artificial intelligence* 36.2 (1988), pp. 177–221.

[22] K. He, X. Zhang, S. Ren, and J. Sun. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[23] A. Inc. *An On-device Deep Neural Network for Face Detection*. 2017. URL: https://machinelearning.apple.com/research/face-detection (visited on 12/27/2020).

[24] J.-B. Jagusch, I. Gonçalves, and M. Castelli. "Neuroevolution under unimodal error landscapes: an exploration of the semantic learning machine algorithm." In: *Proceedings of the genetic and evolutionary computation conference companion*. 2018, pp. 159–160.

[25] J. Kennedy and R. Eberhart. "Particle swarm optimization." In: *Proceedings of ICNN'95 - International Conference on Neural Networks*. Vol. 4. 1995, 1942–1948 vol.4. DOI: 10.1109/ICNN.1995.488968.

[26] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. "Optimization by simulated annealing." In: *science* 220.4598 (1983), pp. 671–680.

[27] J. R. Koza and J. R. Koza. *Genetic programming: on the programming of computers by means of natural selection*. Vol. 1. MIT press, 1992.

[28] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks." In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger. Vol. 25. Curran Associates, Inc., 2012, pp. 1097–1105. URL: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[29] P. Lapa, I. Gonçalves, L. Rundo, and M. Castelli. "Enhancing classification performance of convolutional neural networks for prostate cancer detection on magnetic resonance images: A study with the semantic learning machine." In: *Proceedings of the genetic and evolutionary computation conference companion*. 2019, pp. 381–382.

[30] Y. LeCun. "Une procedure d'apprentissage ponr reseau a seuil asymetrique." In: *Proceedings of Cognitiva 85* (1985), pp. 599–604.

[31] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. "Backpropagation applied to handwritten zip code recognition." In: *Neural computation* 1.4 (1989), pp. 541–551.

[32] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition." In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[33] Y. LeCun, C. Cortes, and C. Burges. "MNIST handwritten digit database." In: *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist* 2 (2010).

[34] S. Lessmann, R. Stahlbock, and S. F. Crone. "Optimizing hyperparameters of support vector machines by genetic algorithms." In: *IC-AI*. 2005, pp. 74–82.

[35] N. Metawa, M. K. Hassan, and M. Elhoseny. "Genetic algorithm based model for optimizing bank lending decisions." In: *Expert Systems with Applications* 80 (2017), pp. 75–82.

[36] A. Moraglio, K. Krawiec, and C. G. Johnson. "Geometric semantic genetic programming." In: *International Conference on Parallel Problem Solving from Nature*. Springer. 2012, pp. 21–31.

[37] A. Moraglio and R. Poli. "Product Geometric Crossover." In: *Parallel Problem Solving from Nature - PPSN IX*. Ed. by T. P. Runarsson, H.-G. Beyer, E. Burke, J. J. Merelo-Guervós, L. D. Whitley, and X. Yao. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 1018–1027. ISBN: 978-3-540-38991-0.

[38] P. Ramachandran, B. Zoph, and Q. V. Le. *Searching for Activation Functions*. 2017. arXiv: 1710.05941 [cs.NE].

[39] A. Reiling, W. Mitchell, S. Westberg, E. Balster, and T. Taha. "CNN Optimization with a Genetic Algorithm." In: *2019 IEEE National Aerospace and Electronics Conference (NAECON)*. IEEE. 2019, pp. 340–344.

[40] F. Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), p. 386.

[41] K. Simonyan and A. Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition." In: *International Conference on Learning Representations*. 2015.

[42] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. *Going Deeper with Convolutions*. 2014. arXiv: 1409.4842 [cs.CV].

[43] L. Vanneschi, I. Bakurov, and M. Castelli. "An initialization technique for geometric semantic GP based on demes evolution and despeciation." In: *2017 IEEE Congress on Evolutionary Computation (CEC)*. IEEE. 2017, pp. 113–120.

[44] L. Vanneschi, M. Castelli, L. Manzoni, and S. Silva. "A new implementation of geometric semantic GP and its application to problems in pharmacokinetics." In: *European Conference on Genetic Programming*. Springer. 2013, pp. 205–216.

[45] L. Vanneschi, M. Castelli, L. Manzoni, S. Silva, and L. Trujillo. "Is k Nearest Neighbours Regression Better Than GP?" In: *Genetic Programming*. Ed. by T. Hu, N. Lourenço, E. Medvet, and F. Divina. Cham: Springer International Publishing, 2020, pp. 244–261. ISBN: 978-3-030-44094-7.

[46] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang. "Phoneme recognition using time-delay neural networks." In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 37.3 (1989), pp. 328–339. DOI: 10.1109/29.21701.

[47] P. Wang. *thispersondoesnotexist.com, year = 2020, url = https://thispersondoesnotexist.com, urldate = 2020-12-27*.

[48] B. Widrow and M. E. Hoff. *Adaptive switching circuits*. Tech. rep. Stanford Univ Ca Stanford Electronics Labs, 1960.

[49] D. H. Wolpert and W. G. Macready. "No free lunch theorems for optimization." In: *IEEE Transactions on Evolutionary Computation* 1.1 (1997), pp. 67–82. DOI: 10.1109/4235.585893.

[50] S. Wright. *The roles of mutation, inbreeding, crossbreeding, and selection in evolution*. Vol. 1. na, 1932.

[51] M. D. Zeiler and R. Fergus. *Visualizing and Understanding Convolutional Networks*. 2013. arXiv: 1311.2901 [cs.CV].

Appendix A

SLM Test results

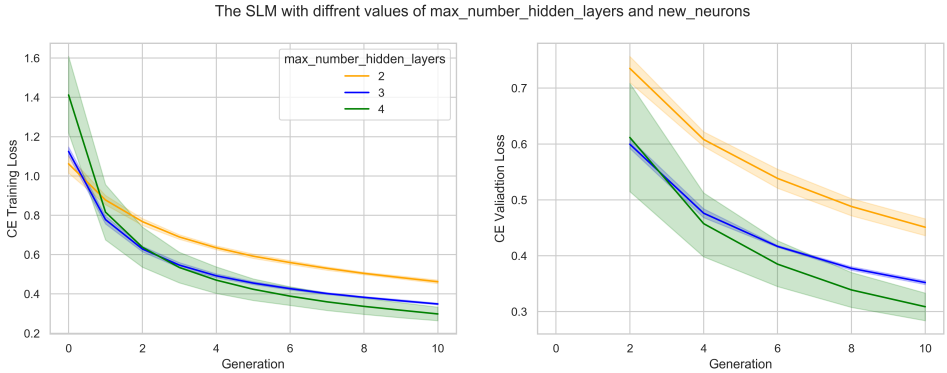

Figure A.1: Different number hidden layers and added neurons for the SLM. Here 10, 25 and 50 neurons are tested for the number of added neurons too.

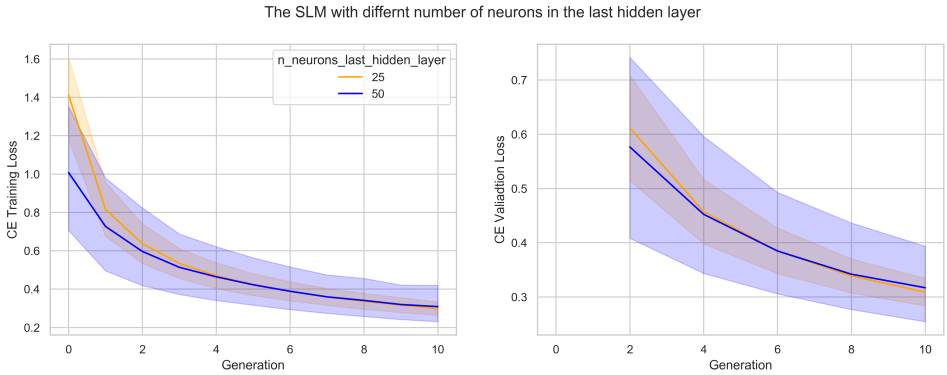

Figure A.2: SLM with 25 and 50 neurons in the last layer upon initialization.

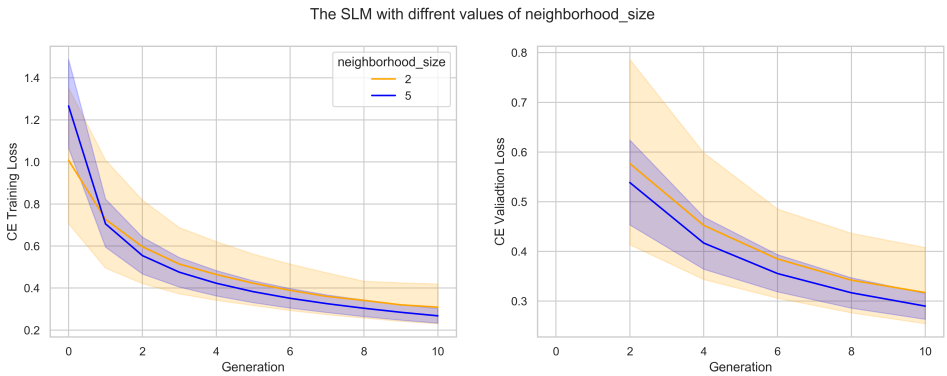

Figure A.3: SLM with a neighborhood size of two and five.