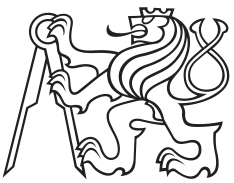
Master's Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

HyperNEAT and Novelty Search for Image Recognition

Bc. Tomáš Kocmánek

May 2015

Supervisor: Ing. Drchal Jan Ph.D.

České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra počítačů

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: **Bc. Tomáš Kocmánek**

Studijní program: Otevřená informatika
Obor: Umělá inteligence

Název tématu: **Spojení algoritmů HyperNEAT a Novelty search pro rozpoznávání obrázků**

Pokyny pro vypracování:

Prozkoumejte možnosti automatizovaného hledání příznaků (feature vectors) z obrazových dat pomocí hyperkubického nepřímého kódování neuronových sítí (metody vycházející z algoritmu HyperNEAT). Pro zvýšení diverzity výsledných příznaků zkombinujte váš algoritmus s přístupem Novelty search. V použitých architekturách výsledných neuronových sítí se můžete inspirovat architekturami používanými pro hluboké neuronové sítě (deep neural networks), např. konvolučními sítěmi. Testujte na databázi psaných číslic MNIST.

Seznam odborné literatury:

Lehman, J., & Stanley, K. O. (2011). Abandoning objectives: evolution through the search for novelty alone. Evolutionary Computation, 19(2), 189-223. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/20868264

D'Ambrosio, D. B., & Stanley, K. O. (2007). A Novel Generative Encoding for Exploiting Neural Network Sensor and Output Geometry. In Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation - GECCO '07 (pp. 974-981). New York, NY, USA: ACM Press. doi:10.1145/1276958.1277155

Koutník, J., Schmidhuber, J., & Gomez, F. (2014). Evolving Deep Unsupervised Convolutional Networks for Vision-Based Reinforcement Learning GECCO '14.

http://yann.lecun.com/exdb/mnist/

Vedoucí: Ing. Jan Drchal, Ph.D.

Platnost zadání: do konce letního semestru 2015/2016

doc. Ing. Filip Železný, Ph.D.
vedoucí katedry

L.S.

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 26. 3. 2015

Acknowledgement / Declaration

I would like to take this opportunity to thank my advisor Jan Drchal for his support and guidance and I thank my family for their support.

I appreciate Petr Olšák, for creating the template for ČVUT theses.

Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum, provided under the programme Projects of Large Infrastructure for Research, Development, and Innovations (LM2010005), is greatly appreciated.

I declare that I worked out the presented thesis independently and I quoted all the used sources of information in accord with Methodical instructions about ethical principles for writing an academic thesis.

.....

Abstrakt / Abstract

Tato práce je počáteční pokus o prozkoumání použití Novelty Search v NeuroEvolučním algoritmu HyperNEAT na doméně rozpoznávání obrázků. Doteď tato kombinace byla použita pouze na problémech spojených s navigací v bludišti nebo pohybem robota, ale ještě nikdy nebyla použita v oblasti obrazových dat. HyperNEAT používá nepřímé kódování v evoluci umělých neuronových sítí. V této práci představíme tři nové přístupy v detekci feature v obrázcích, kde první z nich má nadějné výsledky. Výsledkem této práce je, že Novelty Search může hrát důležitou roli v evoluci feature vektorů rozpoznávající obrázky.

Klíčová slova: HyperNEAT, Novelty Search, Deep Neural Networks, rozpoznávání obrázků, databáze MNIST

This thesis is an initial attempt to investigate the use of Novelty Search in the NeuroEvolutionary algorithm called HyperNEAT on the subject of image recognition. Up until now this combination was used mainly for maze navigation or robot movement, but never in image recognition. The HyperNEAT uses an indirect encoding in evolution of Artificial Neural Networks. We present three novel approaches in feature detection, where the first of them shows promising results. It was concluded that the Novelty Search can play a significant role in the evolution of feature vectors in the image recognition domain.

Keywords: HyperNEAT, Novelty Search, Deep Neural Networks, Image Recognition, MNIST database

Contents /

1 Introduction 1
1.1 Background 1
1.2 Problem Definition 3
1.3 Related works 3
2 Methods 5
2.1 MNIST database 5
2.2 Artificial Neural Network 6
2.2.1 Resilient Propagation 7
2.3 Novelty Search 8
2.4 NeuroEvolution of Augmenting Topologies 9
2.5 Hypercube-based NEAT 12
2.6 Deep Neural Networks 14
3 Research 16
3.1 Direct Approach 16
3.2 Evolution of detectors 18
3.3 Using the Novelty Archive 20
3.4 Selecting detectors for the second level 20
3.4.1 Random Selection 20
3.4.2 Maximal Difference 20
3.4.3 Smallest Entropy 21
3.4.4 Maximal Distance 22
3.5 Grid based Multi-level Hierarchy 22
3.5.1 Grid with rows depending on the class 23
3.5.2 Grid with both rows and columns depending on the class 24
3.6 Complex Detectors 25
3.6.1 Architecture with complex detectors 26
3.6.2 Behavior of complex detectors 27
3.6.3 Minimal criteria of detectors behavior 28
4 Implementation 30
4.1 Architecture 30
4.2 HyperNEAT and Artificial Neural Network 30
4.3 Detector storage 31
4.4 Handling testing labels 32
4.5 10th fold Validation 32
4.6 Selecting subset of images 33
5 Experiments and Results 34
5.1 Comparison of Detector Selection Methods 34
5.2 Detectors from the storage versus newly generated 35
5.3 Structure of detectors 36
5.4 Simple ANN classificator 37
5.5 Comparing results with related work 37
5.6 Grid based Multi-Level Architecture 38
5.7 Architecture with complex detectors 39
6 Discussion 41
6.1 Success of direct approach 41
6.2 Why Grid based architecture was a failure? 41
6.3 Problems with complex detectors 42
6.4 Complexity of the approaches 44
6.5 Confusion matrix 45
7 Summary 46
7.1 Future Research 46
7.2 Conclusion 47
References 48
A Parameters of HyperNEAT 51
B Abbreviations 52
C CD content 53

Tables / Figures

3.1. Example of a detector behavior 19
5.1. The experiment comparing detector selection methods 35
5.2. Comparison of detectors selected from the storage and newly generated 35
5.3. Impact of the number of used detectors in the classification .. 36
5.4. The grid based architecture performance 39
5.5. The error rates of architecture with the complex detectors. 40
6.1. The confusion matrix 45

2.1. Images from the MNIST database 5
2.2. Illustration of the Artificial Neural Network 6
2.3. Experiments with theNovelty search on the maze 8
2.4. A genotype to phenotype mapping example 9
2.5. Example of mutation in the NEAT 10
2.6. Example of the crossover in the NEAT algorithm 11
2.7. A CPPN describes connectivity 12
2.8. Experiment with rescaling the substrate 13
2.9. Illustration of the Deep Neural Network 14
3.1. The illustration of the detector 16
3.2. Complete structure used for image recognition 17
3.3. The grid based architecture ... 22
3.4. The grid with rows depending on the class 24
3.5. The grid with rows and columns depending on the class 24
3.6. The illustration of a bigger grid 25
3.7. The illustration of the structure of complex detector 26
3.8. The illustration of the tree-like architecture 26
3.9. The averaged images from the MNIST database 27
3.10. The transformation from averaged images into the behavior matrix 28
3.11. Examples of detectors with nearly zero sum of the behavior 29
4.1. Logo of the Encog framework . 31

4.2. 10th fold validation 33
5.1. Experiment with a simple ANN classificator..... 37
6.1. The transformation of the image with the grid architecture 42
6.2. Outputs of complex detectors . 43

Chapter 1
Introduction

In this work we present a method for image recognition, that to the best of our knowledge is a unique and new approach that has not been experimented before. We are combining NeuroEvolution method HyperNEAT with Novelty Search. The performance of our method suggests that it is promising and could be more improved. In the experiments we show that our approach is much better than the use of HyperNEAT alone based on the experiments from the work [1] (2013).

The image recognition task is one of the most researched tasks in the domain of Machine Learning. It is also because the computer vision is an important field in the real life application. So far computers are only trying to reach a performance comparable with humans. There are also many competitions where individuals or teams can compete in real life problems and even win hundreds of thousands dollars if they create state-of-the-art concept for a particular domain. One of such competitions is Kaggle [1]), where you can find competitions in the domain of Machine Learning. On Kaggle are often image recognition problems, like detecting diabetes from high-quality images of eyes retina, classifying plankton from deep sea images or Kinect gesture recognition.

In this work we are using the MNIST database of images. The state-of-the-art classificators have an error rate under 0.5% in this database. The goal of this work is not to improve the state-of-the-art results, but to investigate the use of the HyperNEAT and Novelty Search in the image recognition task.

In this chapter we describe the background of the domain, the problem definition and related works. In the second chapter Methods we describe the image database that we are using, Artificial Neural Networks, Novelty Search, HyperNEAT and Deep Neural Networks. In the third chapter Research we describe the proposed methods and the problems we had with them. In the fourth chapter the implementation part of our research is described. The fifth chapter is concerned with the experiments. The last two chapters contain discussion, future work and a summary of the work.

The bibliography, a detailed settings of used methods and a list of attached binary files can be found in the appendices.

1.1 Background

The artificial neural networks (ANN) have long held promise in the fields of machine learning and artificial intelligence. If only for its inspiration in the biological neural networks, like the central nervous systems of animals, in particular the brain. ANNs are used to estimate or approximate functions that can depend on a large number of

[1]) http://kaggle.com/

inputs that are generally unknown. Additionally, neural networks can exhibit some useful properties. They can be applied to many kinds of tasks, be fault tolerant, and generalize well over previously unseen input. There are also problems connected with them like over-fitting, difficulty to train deep networks or time complexity of the training phase.

We also know that biological neural networks were produced via an evolutionary process. The evolution inspired algorithms are called evolutionary algorithms (EA), a subset is called genetic algorithms. An EA uses mechanisms inspired by biological evolution, such as reproduction, mutation, recombination and selection. Evolution of the population then takes place after the repeated application of the above operators until it finds the solution or a solution with sufficiently small error. The EA requires only a scalar evolution signal called *fitness* and often performs well over many task domains. There are many domains where the EA comes up with solution that the humans did not expect, but which has better performance then human-created solutions.

The combination of ANN and EA is called NeuroEvolution (NE), where an evolution evolves a population of ANN. There are many neuroevolutionary algorithms. One common distinction is whether algorithms evolve only the strength of the connection weights for a fixed network topology or if they evolve both the topology of the network and its weights.

Another distinction is whether each element of the evolved network is directly encoded as a separate gene, called a *direct encoding*, or whether there is gene reuse through which one gene may encode many network elements, called an *indirect encoding*. In the direct encoding schemes the genotype directly maps to the phenotype. Therefore every neuron, connection and weight in the neural network is specified directly and explicitly in the genotype. In contrast, in the indirect encoding schemes the genotype specifies indirectly how that network should be generated. The NE has enjoyed success in many kinds of tasks. Most notably it has been successfully applied to tasks where the exact target outputs or policy are unknown, and only a scalar evaluation signal is available. The NE methods typically have been able to produce only networks processing a small number of inputs and outputs (low-dimensional input and output spaces) and had problems with processing directly hundreds or thousands of inputs or outputs.

In 2007, a method which tries to overcome this limitation called Hypercube-based Neuro-Evolution of Augmenting Topologies [1] was developed. It is a neuroevolution method that uses an indirect encoding, called hypercube weight pattern encoding. The indirect encoding makes it easier for the evolutionary algorithm to generate large-scale neural networks by exploiting the geometric regularities in a potentially high-dimensional input space.

The Novelty Search was first introduced in 2011 by Lehman and Stanley. The main idea is that during the evolution we forget the goal of the program and instead we evolve population of individuals, which have distinct behavior from each other. This helps us to overcome the local maxima and deceptive traps in the search space. In some problems and scenarios, ignoring the goal is better way of reaching it than by following the goal criteria.

1.2 Problem Definition

The pattern recognition or image recognition is a branch of the machine learning that focuses on the recognition of patterns and regularities in the image data. This field is closely connected with the computer vision, where the task is to create an algorithm that can recognize objects, describe different scenery or react on the image input the same way as humans would. One of the tasks in the image recognition is to classify the images into one of the predefined classes with the highest probability.

A theme in the development of image recognition has been to duplicate the abilities of a human vision by electronically perceiving and understanding an image. The image understanding can be seen as analyzing the symbolic information from the image data using models constructed with the aid of geometry, physics, statistics, and learning theory. It is a deeply studied field with significant advancement in the last decade. There are many ways to approach image recognition ranging from a linear classifiers, k-nearest neighbors, through non-linear classifier up to artificial neural network or convolutional networks.

There are many domains in the image recognition task. From a simple position detection in the image, through image classification, up to labeling of image scenery. In this work we are working with a classification of hand-written digits, because it is an interesting domain that was deeply studied and therefore there are many results to compare. For this purpose we are using the MNIST database of hand-written digits.

1.3 Related works

Although there are many different approaches in the image recognition we list in this section mainly related works with the indirectly encoded artificial neural networks, which is the approach used in this work.

The HyperNEAT was investigated in several works as a part of image recognition task [2], [1] and [3]. But usually on easy pattern recognition tasks like recognition of position of single object in the image, founding the biggest object or recognition of a direction of 'U' like object. In those domains it showed good results. The improved version of HyperNEAT called HyperNEAT-LEO introduced in [4] shows significant improvement in the Retina Problem task over other mentioned methods.

In the work [5] (2013) Verbancsics and Harguess explored the use of HyperNEAT as a feature extractor in the image recognition domain of numbers. They investigated the use of HyperNEAT in the Deep Learning architectures. They found out that HyperNEAT itself struggles to find ANNs that perform well in the image classification. However they demonstrated an effective ability of HyperNEAT to act as a feature extractor and they combined it with classical ANN trained by the backpropagation. We are going to investigate the use of HyperNEAT even further and combine it with Novelty Search. This combination is to our knowledge a novel research that nobody have experimented with, therefore we compare our results with their work, since they have most similar approach and they also used the MNIST database as a testing environment.

The deep neural network (DNN) is a leading architecture in the image recognition with the best performance. It is an artificial neural network with multiple hidden layers of units between the input and output layers. In the work [6] and [7] they used

evolution of deep convolutional network to evolve compressors and recurrent neural network (RNN) controllers that drive a race car in the TORCS racing simulator using only the visual input. Another use of convolutional networks is in the work [8] where they demonstrated power of this architecture on the MNIST database.

The Novelty detection is the identification of new or unknown data that a machine learning system has not been trained with and that was not previously aware of. Novelty detection is an important addition to supervised classification algorithms. A trained classifier should be able to distinguish an unknown input for which an attempt at classification would not be known. It was investigated in the domain of image recognition and proved to have important impact on the performance. In the work [9] they used an image receptive fields neural network with various novelty criteria to the task of multi-object localization in various backgrounds.

Novelty detection is not the same as a Novelty Search. We are not aware of any work where the authors used Novelty Search in the image recognition domain. It is because the Novelty Search was introduced for the evolution algorithms, but EA alone are not good candidates for an image recognition.

Chapter 2
Methods

In this chapter we describe the used algorithms and the database used in our research. In the first section we define the MNIST database, the second section is concerned with the artificial neural networks and Resilient training method, the next section is section about Novelty Search, and the last two sections are on NEAT and HyperNEAT algorithms.

2.1 MNIST database

The MNIST database of handwritten digits [1]) contains a training set of 60 000 images and a testing set of 10 000 images. It is a subset of the NIST database. The original images (with 20x20 pixels) from the NIST database have been size-normalized and centered in a fixed-size 28x28 pixels image.

Half of the images in the MNIST database was collected among Census Bureau employees and the second half was collected among high-school students. The training set contains images from approximately 250 writers. The testing images were created by different writers than writers of numbers in the training set.

The error rate is defined as a percentage of incorrectly classified images.

Many methods have been tested on this database which makes it great for the comparison. Official scoreboard contains the best results for many different methods. It contains error rates ranging from 12% for linear classifiers to 0.23% for a committee of 35 convolution networks [10]. The comparison of performance of many methods used on this database can be found in the paper [8] written by the author of the MNIST database.

Figure 2.1. Images from the MNIST database.

[1]) http://yann.lecun.com/exdb/mnist/

2.2 Artificial Neural Network

In the machine learning, the Artificial Neural Networks (ANN) are a family of statistical algorithms inspired by the central nervous network of animals in the nature. They are used to approximately estimate functions based on the samples. The history of ANN reaches up to the 1958 when F. Rosenblatt defined the first perceptron, an algorithm for a pattern recognition, using simple addition and subtraction. The research stagnated after the publication of M. Minsky and S. Papert in 1969. They have discovered two key issues connected with the ANN. The first issue was that a single-layer neural networks were incapable of processing the XOR circuit. The second significant issue was that computers were not sophisticated enough to effectively handle the long run time required by the large neural networks. The ANN were gradually overtaken in popularity in machine learning by the support vector machines and other simpler methods such as a linear classifiers. They have experienced second boom in the late 2000s with the advent of deep learning.

The ANN is represented as a system of interconnected *perceptrons* or *neurons*. The neurons are layered in several layers. The ANN usually contain one input layer, where each neuron gets one value from the input space, zero or more hidden layers and one output layer as illustrated in the figure 2.2.

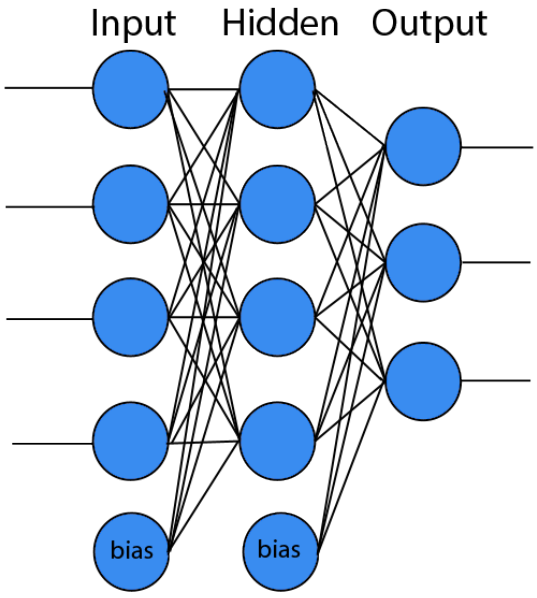


Figure 2.2. The ANN is a system of interconnected neurons, similar as in the human brain. Each circular node represent neuron and each line represent connection. In each layer can be found a bias neuron, which is always active.

The neuron is a mathematical function which receives one of more inputs and sums them into the output. Summing of the nodes are weighted and determining the weights are the main task of training neurons. The output of the kth neuron is computed via following function:

$$y_k = \varphi(\sum_{i=0}^{n} w_{ki} x_i)$$

where φ is transfer function, x_i is ith input of neuron and w_{ki} is weight for input i. The purpose of the transfer function is to bound the output in some way. We define

some properties of a transfer function, usually we want to bound the output to the interval $[0, 1]$. Most commonly used transfer function is the Sigmoid function, which has several properties. For example the fact that it has a pair of horizontal asymptotes and non-negative first derivatives. The function is defined as follows:

$$S(x) = \frac{1}{1 + e^{-x}}$$

The weights in the ANN are determined by training the network with samples for a known correct output. One of the first algorithms for training the ANN was *Backpropagation*, an abbreviation for backward propagation of errors. This method calculates the gradient of a loss function with respects to all the weights in the network. The gradient is fed to the optimization method which in turn uses it to update the weights of the neurons, in an attempt to minimize the loss function.

2.2.1 Resilient Propagation

In this work we use a training technique of the ANN in the final classification method as introduced in [11] called Resilient propagation, which overcomes disadvantages of pure gradient-descent method used in backpropagation. It also uses partial derivative over all samples but only accounts its sign. Moreover it updates weights independently. First we define partial derivative and then weight-step rule of Resilient propagation.

For the gradient descent method in backpropagation method we need to calculate the derivative of the squared error function with respect to the weights of the network. Assuming one output neuron the squared error function is:

$$E = \frac{1}{2}(r - y)^2$$

where E is a squared error of a neuron, r is an expected output and y is an actual output of a neuron. The change of the weight in the backpropagation is as follows:

$$\Delta w_{ij} = -\alpha \frac{\partial E}{\partial w_{ij}}$$

where α is the learning rate. In the Resilient propagation we do not change the weights according to the global value of the steepest descent but instead each weight has an individually evolving update-value Δ_{ij} which is updated based on the sign of the gradient. Therefore the weight step is based only on the sign of the gradient and the update-value of a particular weight. Whenever the gradient is higher than zero we add Δ_{ij} to the weight and whenever it is lower than zero we subtract Δ_{ij}. However, there is one exception. If the sign of the gradient changes from the previous step, i.e. the last step was too large and missed the optimum, then the previous update of the weight is reverted. In that case to avoid double punishment of the update-value we forbid its adaptation in the succeeding step.

Last mechanism in the Resilient propagation is the evolution of update-value Δ_{ij} e.g. the learning rate of the particular weight. The change of the update-value depends on the signs of the previous and current gradient.

$$s_{ij} = \frac{\partial E}{\partial w_{ij}}(t-1)\frac{\partial E}{\partial w_{ij}}(t)$$

Whenever s_{ij} is higher than zero, we multiply the update-value by 1.2, whenever it is lower than zero we multiply it by 0.5 (both values were selected empirically in original work [11]).

2.3 Novelty Search

Novelty search (NS) is a unique approach to search without any objectives. It was proposed by Lehman and Stanley in [12]. It rewards behaving differently rather than converging to some ultimate goal as it is in the traditional approach of searching. Therefore in the search for novelty there is no pressure to do better but only to do something new.

While it may sound strange, in some problems and scenarios, ignoring the goal is the right way how to reach the goal. This can be explained that sometimes intermediate steps of how to reach a goal are completely opposite from the goal itself. Those scenarios are called *traps*, where blindly following the goal always get you in the blind spot. The Novelty Search was studied mainly in the maze navigation task [13], biped locomotion [12] or [14], grammar evolution on the Santa-Fe trail [15] or body-brain co-evolution [16].

Novelty search is best suited for deceptive tasks where we can intuitively define behavior of the individuals. For example it can be a maze where we can take final position of the agent as his behavior. At first agents bump into the walls, then search finds out how to avoid walls then how to move in the maze etc. In the figure 2.3 you can see the difference when the search is trying to find the solution that is the closest to the goal and when it only looks for a new behavior.

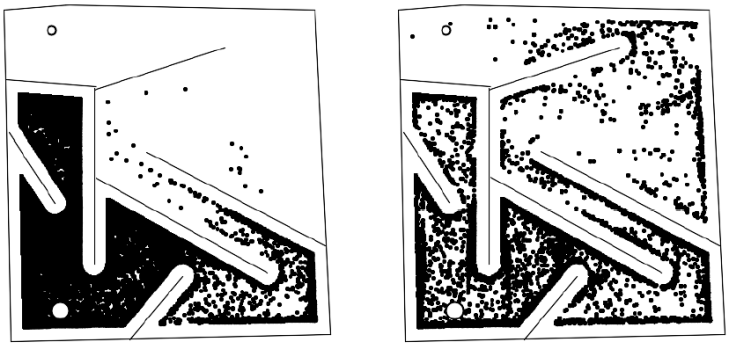


Figure 2.3. Experiment with the Novelty search. On the left there are results of search powered by the fitness. On the right are the results of search powered by the search for novelty. Source: [12]

Novelty search can be easily implemented to any existing evolutionary algorithm. There are three important changes in comparison to evolutionary algorithm. The first is that we need to implement behavior of individuals. This metric is usually domain dependent and should aim to cover all interesting behaviors. The second is to change fitness metric, which indicate progress towards the goal with novelty metric. This metric is usually the average distance of behaviors to the k-nearest neighbors. The last change is the implementation of the archive. The archive remembers individuals which were significantly different in the time of their discovery and use them to avoid backtracking in the search.

When we are running the novelty search, during evaluation of individuals it calculates the distance of a new individual from all individuals in the population and individuals in the archive. This calculates novelty of an individual in comparison to the current population and in comparison to the past novel individuals. Next the novelty is calculated from those distances based on the k-nearest neighbors. If the individual receives a high novelty value it is saved to the archive. The rest in the algorithm is the same as the original evolutionary algorithm.

In the work [17] Lehman and Stanley proposed an improvement of the Novelty Search. They added minimal criteria value which each individual has to reach to be used in the breeding and left in the population. This threshold assures that the individual satisfy at least a strict minimum to be useful. Their improvement accelerated the evolution.

The improvement was done by adjusting the computation of distance between two behaviors. Whenever an individual does not reach the threshold, the NS receives distance zero when it evaluates its distance with the others. By this it thinks that the individual is same as all individuals in the population and therefor it is shortly removed from the population.

2.4 NeuroEvolution of Augmenting Topologies

The NeuroEvolution of Augmenting Topologies (NEAT) is a genetic algorithm for the evolution of artificial neural networks developed by Ken Stanley in 2002 [18]. This algorithm does not need a human experimenter to define the structure of the ANN it rather evolves it by itself. During the evolution it alters weights of connections between the neurons and the structure of the network in an attempt to find the fittest individual as well as maintain diversity in the population. Each neuron outputs value of the sigmoid function over its inputs as in the usual ANN. The algorithm applies three main techniques to achieve its goal: it tracks genes with *innovation number* to maintain the compatibility between different topologies, it develops topologies incrementally from simple structures to the complex ones and it preserves the speciation (evolution of different species) in order to give the time to complex structures to evolve.

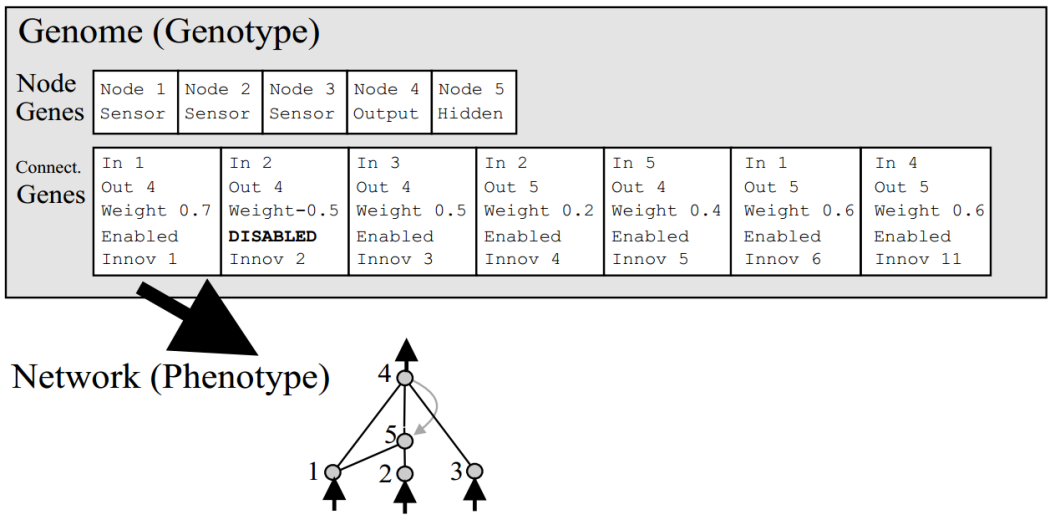


Figure 2.4. A genotype to phenotype mapping example. A genome contain 5 neurons and 7 connections. One of connections is disabled and one is recurrent. Source: [18]

The NEAT genetic encoding enables to easily align two genomes during the crossover. Genomes are linear representation of the ANN 2.4. Each genome contains list of node genes and connection genes. The list of node genes identifies each node as either input, hidden or output neuron. Each connection gene specifies the connection between in-node and out-node, its weight, if it is enabled and its innovation number (in some literature called *historical marking*) which allows aligning corresponding genes in different topologies.

The algorithm works as an ordinary evolutionary algorithm. Therefore we first describe the mutation of genes, then the crossover and at last the improvements over the simple evolution.

The mutation can change both the connection weight or the network structure. Changing the weight of the connection is done by changing the weight value in the gene. The structural change can occur in one of the two ways as depicted in the figure 2.5. It either adds a new connection or a new node. In former case a new gene with new innovation number is added into the genome. It has random weight and connects two random neurons that were not connected by any other connection. In the add node mutation an existing connection is split in half and a new node is added on its place. The split connection is disabled and two new connections are added into the genome. The first new connection, leading into the new node, receives a weight 1 and the second connection receives a weight same as was the weight of disabled connection. This minimizes the effect of mutation and ensures that the behavior of the network changes only slightly after the mutation.

Through the mutation the topologies of individuals in the population will get gradually larger and more complex.

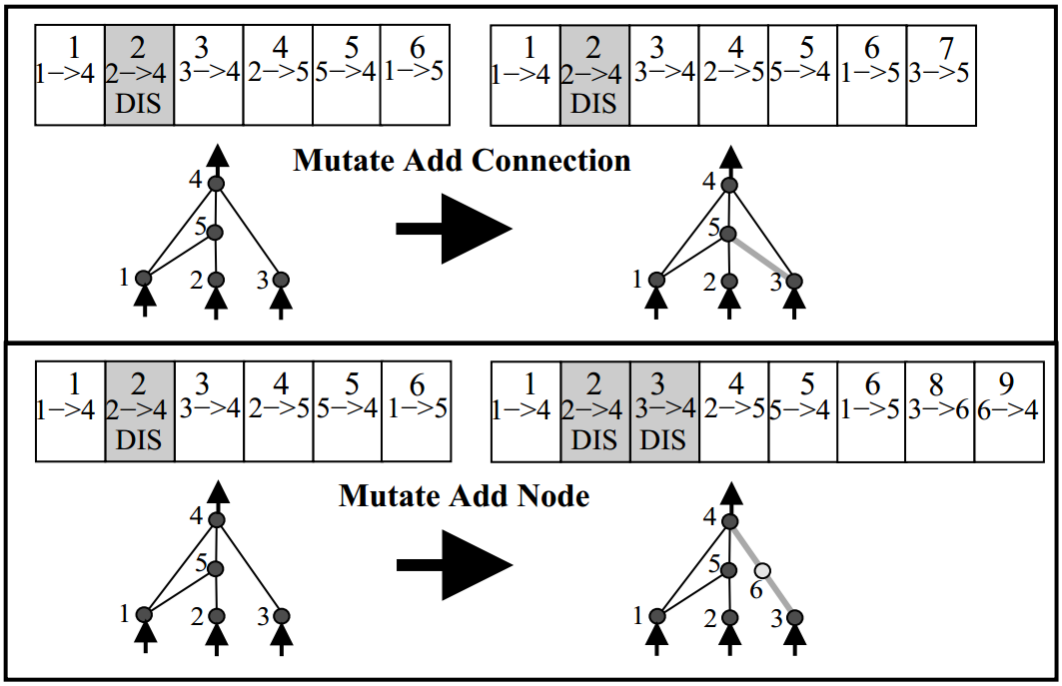


Figure 2.5. The two types of structural mutation in NEAT. Both types, adding a connection and adding a node, are illustrated with the connection genes of a network shown above their phenotypes. Source: [18]

Except of the mutation, the crossover is another important phase of the evolution which ensures that the fitter individuals transfer their genome to the descendant more often. For the selection of parents the tournament is used, but the roulette rule could be used too. During the tournament X individuals are selected and two fittest are selected as parents.

The innovation number tells us which genes match up with which genes and therefore can be crossovered. Whenever new connection we assign a globally new innovation number and assigning it to the new connection. Since the innovation number does not change over the course of evolution it represents a chronology of the appearance of every gene in the system over the evolution.

During the crossover both genomes from parents are aligned based on the innovation numbers of genes. Genes that do not match are either disjoint or excess, depending on whether their innovation numbers are in the range of innovation numbers of the second parent. Those genes represent the structure which is not present in the second parent.

During the composition of the offspring in the crossover each gene is randomly selected from one of the parents. Except of excess and disjoint genes which are always transferred into the offspring from the more fit parent. An example is presented in figure 2.6.

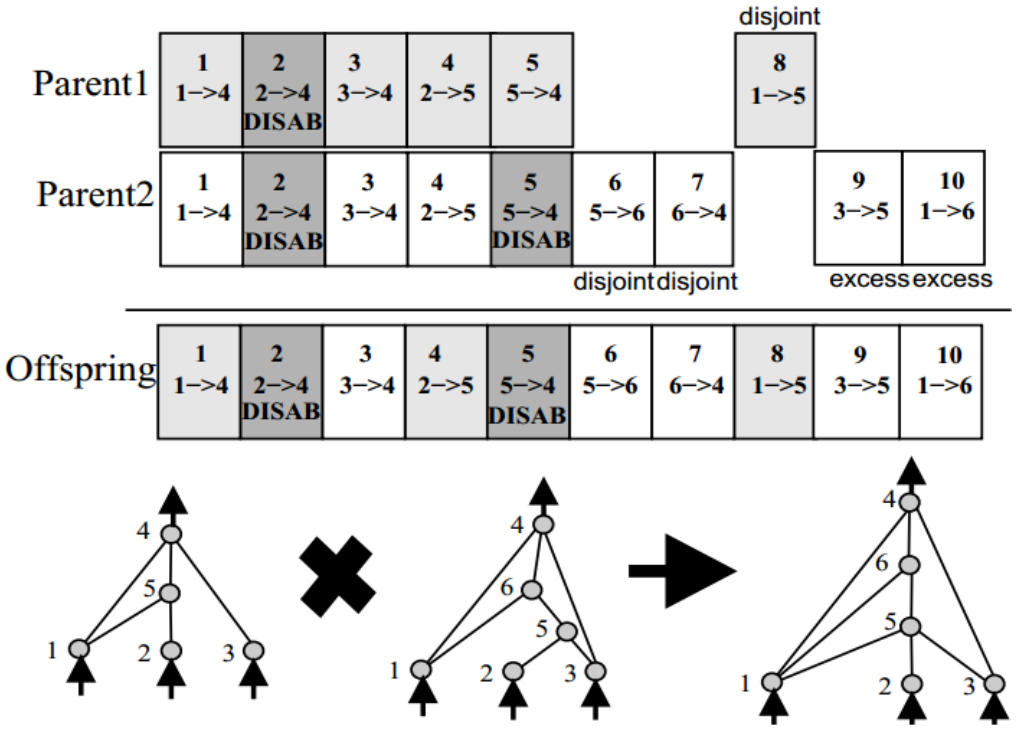


Figure 2.6. Example of the crossover in the NEAT algorithm. Matching up genomes for different network topologies using the innovation numbers. In this example the same fitness for both parents is assumed, therefore offspring gets excess and disjoint genes from both parents. Source: [18]

The innovation number also protects crossover from the competing conventions problem. When we can evolve two distinct genomes that both encodes the same function only they have permuted genomes.

The last part of the NEAT is protecting innovation through species. Creating species can help new individuals in the process of adjusting by competing with only individuals in their niche instead of with the whole population, where less innovated individuals can already by adjusted. The idea is to divide population into species with similar topologies. For this task the innovation numbers are used. The more excess and disjoint nodes two parent has the less compatible they are and the less evolutionary history they share. When two parents have too many nodes, we evaluate them as two different species and deny their breeding.

2.5 Hypercube-based NEAT

Hypercube-based NEAT (HyperNEAT) is a generative indirect encoding that evolves the artificial neural networks with the principles of the widely used NEAT algorithm. It was presented by Gauci and Stanley in the paper [1] (2007). It is a novel technique for evolving large-scale neural networks utilizing the geometric regularities of the task domain. The effectiveness of the geometry-based learning in the HyperNEAT has been demonstrated in the multiple domains and representations, such as the checkers [19], the multi-agent predator prey and the room-cleaning problem [20], a game of Go [21], and the RoboCup Keepaway [22].

For evolving large-scale ANNs that exploit geometric motifs we need a powerful and an efficient encoding. For this purpose the HyperNEAT uses encoding called the connective Compositional Pattern Producing Networks (connective CPPNs). The CPPN is actually an ANN where each node instead of outputting value of a sigmoid function over the sum of its inputs, it can contain any geometrical function. The available CPPN activation functions are an absolute value, a bipolar sigmoid, a Gaussian, a linear, a sine, and a step function. An example of such network can be seen in the right part of the figure 2.7.

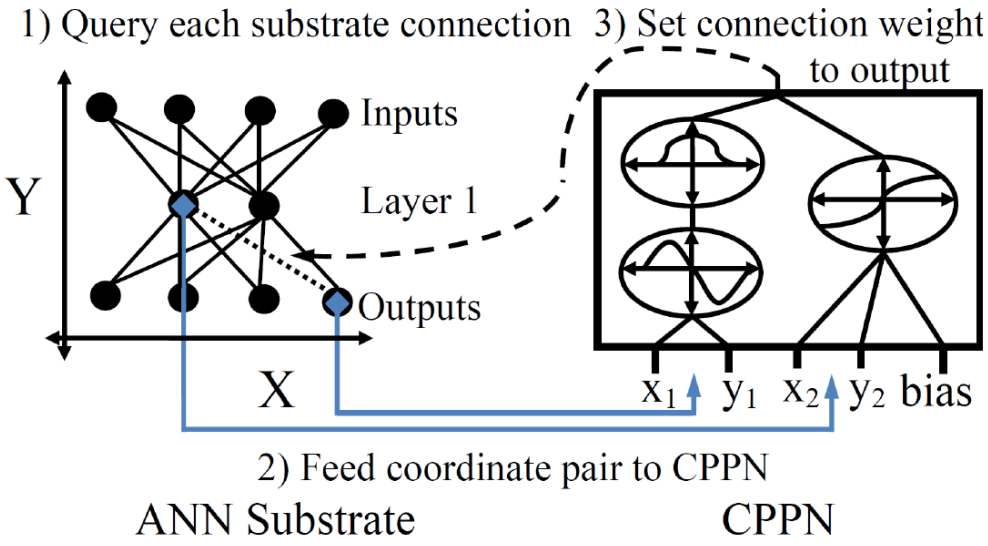


Figure 2.7. A CPPN describes connectivity. The ANN substrate has assigned coordinates. Step 1: Every connection between the neurons in the substrate is queried by the CPPN to determine its weight. Step 2: For each such query, the CPPN inputs the coordinates of the two neurons, which are highlighted in the substrate. Step 3: The weight between them is the output by the CPPN. Source: [4]

The CPPNs are functions of geometry that output connectivity patterns whose nodes are situated in the n dimensions, where n is the number of dimensions in a Cartesian space. Consider a CPPN that takes four inputs labeled $x1$, $y1$, $x2$, and $y2$. This point in a four-dimensional space can also denote the connection between the two-dimensional points $(x1, y1)$ and $(x2, y2)$. The output of the CPPN for that input represents the weight of a connection between the neurons, see figure 2.7. In this way we can compute weights between all neurons in any ANN without restrictions on the size of that network. The ANN which weight patterns are produced by the CPPN in this way is called *substrate* so that they can be verbally distinguished from the CPPN itself.

The CPPN is evolved by the NEAT algorithm with two adjustments. The first step is that when a new node is generated it gets one of the available activation functions at random instead of the sigmoid function in the NEAT algorithm. The second step is that the fitness value of the CPPN in the NEAT evolution is determined with respect to the correctness of the output of the substrate generated by the particular CPPN.

We can define 3D artificial neural networks as well. Where each layer is a grid of neurons where all of them has the same z coordinate and where layers differs by the z coordinate. In this work we are using this improved 3D version, because the algorithm can more easily discover regularities or anomalies in the images.

One of the most important features of the HyperNEAT is the ability to rescale the substrate. We can for example double the number of neurons in the substrate and assign them coordinates in the same range as in the smaller substrate and then due to the geometrical patterns which the CPPN exploits it has a similar performance with the bigger grid as it had in the original smaller grid. This can be used by training the HyperNEAT first over small set of the images, then rescale substrate and start using bigger images and so on. This way we can train the network to recognize more complex structures. Another use could be during the image recognition where instead of rescaling images to fit to the input of network we can rescale size of the ANN to recognize image in its original size. You can see impact of rescaling in the experiment from the original HyperNEAT paper [1] in the figure 2.8.

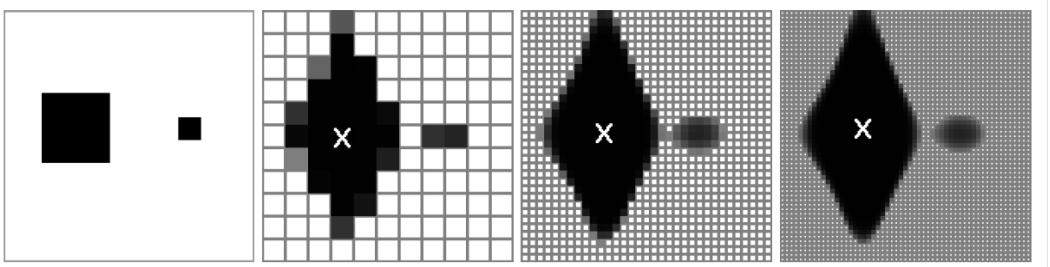


Figure 2.8. Activation Patterns of the same CPPN with different rescaling. The placements of the objects for this example are shown in (a). Each square represents a neuron in the target field. A darker color signifies higher activation and a white X denotes the point of the highest activation, which is correctly at the center of the large object in (a). Source: [1]

2.6 Deep Neural Networks

The Artificial Neural Networks with few hidden layers as described above are called shallow. With more hidden layers we theoretically can get better results. But training such networks becomes significantly more complicated with more layers.

The Deep Neural Networks (DNN) are defined as an ANN with several layers. Usually a network with a 7 or more layers is considered as a DNN. In the DNN we have to use non-linear activation function in the neurons of hidden layers. Because in the case that we would use a linear functions in multiple hidden layers, their composition would be another linear function and therefore it could be represented by only one hidden layer.

There are two main problems with DNN. First is a higher possibility of over-fitting. It is because of the added layers of abstraction, which allow them to model rare dependencies in the training data. There are several methods how to avoid over-fitting like a decay, a sparsity or a dropout regularization. In the dropout, some number of units are randomly omitted from the hidden layers during the training. This helps to break the rare dependencies that can occur in the training data.

The second problem connected with the DNN is the high complexity of training such network. Backpropagation and gradient descent have been the preferred method for training these structures due to the ease of implementation, but with each added layer the training time grows rapidly. That is why we cannot add hundreds of layers and expect significantly better results.

We can separate the DNN into several networks which are learned gradually. It can be represented as each networks output is connected into an input of adjacent ANN. We will call those separated neural networks *levels* to verbally distinguish them from layers in the ANN.

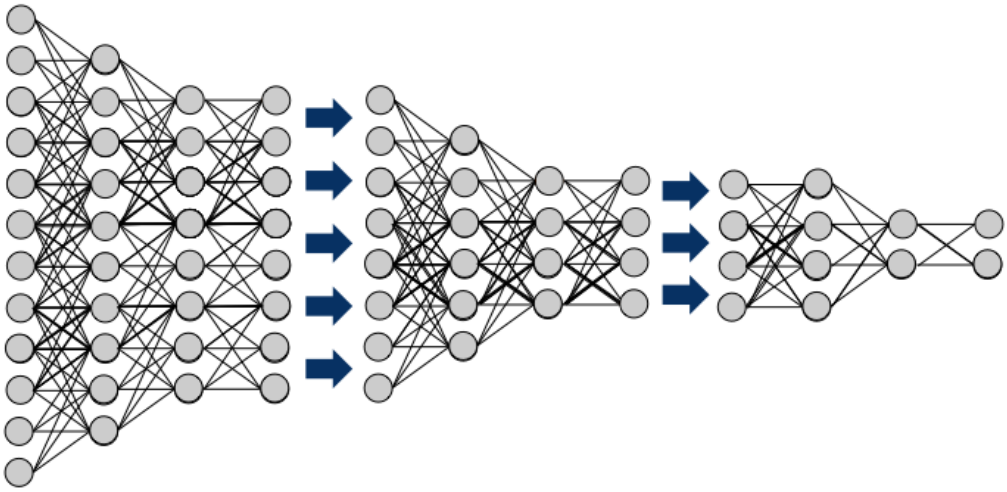


Figure 2.9. Illustration of Deep Neural Network approach with several levels. It is composed of 3 levels where each is an ANN with 2 hidden layers. Source: http://theanalyticsstore.ie/deep-learning/

Similar approach is used in the Convolutional neural network (CNN). The CNN consists of multiple layers of small neuron collections which look at small portions of the input image, called receptive fields. The results of these collections are then tiled so that they overlap to obtain a better representation of the original image. This

is repeated for every layer, where several different levels alternate. The CNN may include pooling layers, which combine the outputs of neuron clusters. They consist of various combinations of convolutional layers and fully connected layers, with pointwise nonlinearity applied at the end of or after each layer.

Chapter 3
Research

I have developed a novel approach for the image classification, that so far nobody have investigated. In this chapter I describe the research of this topic, how my methods work, what were some problems and how I have overcome them. The research has three parts. In the first part we investigate a simple substrates with one output. The second part combines the substrates into bigger approach and tries to use them in a grid based architecture. The third part examines possibilities of complex detectors with more than one output.

3.1 Direct Approach

The first part of this work deals with a two level approach. Where first level is a set of substrates evolved by the HyperNEAT with the Novelty Search and the second level is a basic ANN trained with Resilient propagation. The second layer has the output of substrates evolved in the first level as the input. The task of the first level is to find as much diverse features in the images as possible and then the second level performs the classification of the image over the found features.

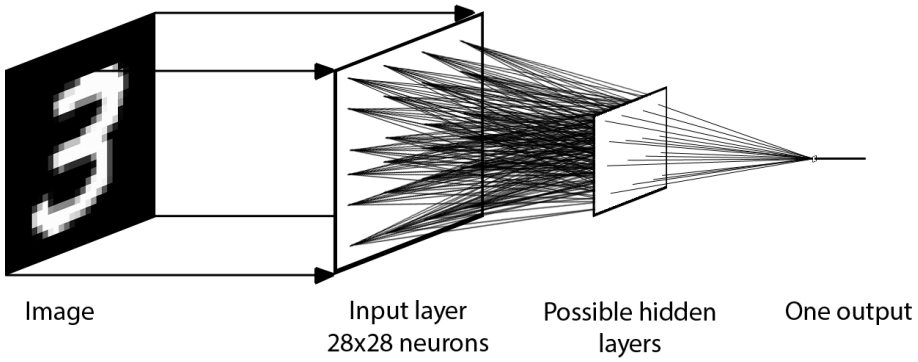


Figure 3.1. The illustration of the detector. Each pixel of the image has corresponding input neuron. There can be zero or more hidden layers and all layers are fully connected to adjacent ones. The output is only one value.

In the first level the HyperNEAT evolves population of substrates (fully connected Artificial Neural Networks), which process image and output some value. We define the structure of those substrates to output only one value. Their only task will be to detect *anything* in the original image. We call them *feature detectors* or simply detectors, since they only detect *something* in the image and outputs value showing how much the *something* was detected in the image. We do not care what they detect. It can be dot in the corner of image or complete 8-like shape. In most cases it will not be anything that human would care about or even a useless detail detected in all images

no matter of their class. But even those detectors are important for the evolution and help evolving better and useful detectors.

A detector is an ANN with two dimensional layers. It has all neurons fully connected to all neurons in the following layer. The images in the MNIST database has 28x28 pixels. Therefore each detector has a two-dimensional input layer of size 28x28 neurons. The number of hidden layers will depend on the experiments, but in the simple case we do not use any hidden layers. The output layer contains only one neuron, which is important due to the nature of the detectors. Later in this work we will modify detectors and introduce more complex output layer with more neurons.

The mathematical definition of a simple detector without any hidden layer is as follows:

$$detector(P) = sigmoid\left(b + \sum_{i=1}^{w}\sum_{j=1}^{h} P_{i,j} * W_{i,j}\right)$$

where P is a matrix of pixel values from the original image, W is a matrix of weights, b is a bias value and w, h are width and height of the image. In this case of the ANN without any hidden layers the definition is basically extended definition of a neuron over two-dimensional input.

The detector is a simple structure as shown in figure 3.1 and can be interpreted as a transformation from image space into the output space, which in this case are real numbers between 0 and 1. This number can be represented as a ratio of how much a feature was detected in the image.

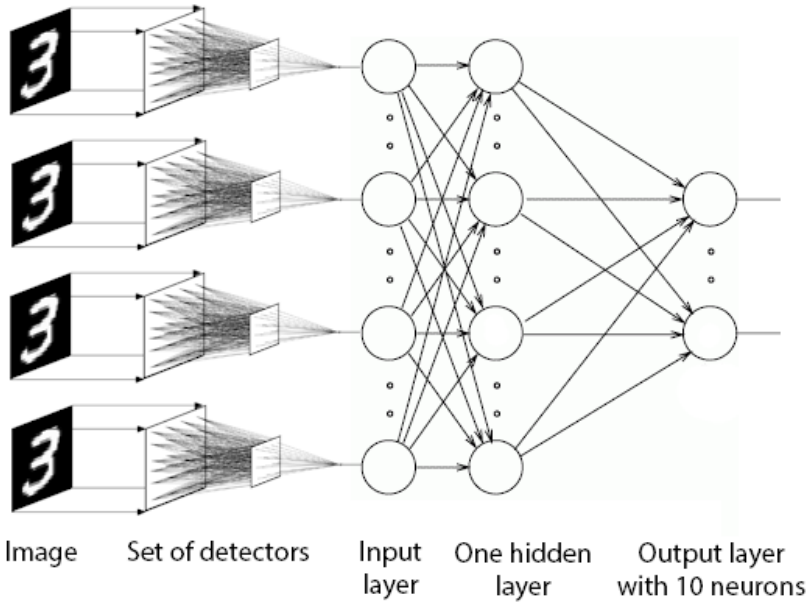


Figure 3.2. Complete structure used for image recognition. First each detector transform image into one value output, then following ANN classify image based on outputs of detectors.

Main idea behind detectors it that we do not want detectors to solve any classification task. We need them to find a new feature in the image that were not detected by other detectors. We need a set of detectors with as diverse outputs as possible. Therefore even

a detector that is stimulated on 5 out of 10 classes is useful, i.e. it detects *something* in 5 classes.

The classification of the image lays in the second level of the ANN. This ANN processes all outputs of the set of detectors and classifies the original image. The ANN contains input layer with as many neurons as the number of detectors, one hidden layer and 10 neurons as the outputs, illustrated in the figure 3.2. We could use more hidden layers with different sizes, but we decided to use only one hidden layer with 50 neurons. We do not want to study the performance of the second layer but the performance of the detectors.

The final classification of the image is based on the position of the output neuron with highest activation value in the second level ANN. There is 10 output neurons and each corresponds to one class of number i.e. 0-9.

3.2 Evolution of detectors

The evolution of detectors is an important part of this work. It ensures creation of many detectors from which we can select the subset of the best detectors for the second classification level.

First let us consider using of a HyperNEAT without the Novelty search. To generate detectors via a simple HyperNEAT we need to define a fitness metric. In the work [5] they defined the detectors with 10 output values and as a fitness they used correctness of classification of the detectors. But in our work we have only one output in the detector. Therefore we could define the fitness function as a metric of how correctly it distinguish any class from the rest. Therefore we would aim to have a set of detectors where each detector would be specialized for one of the classes. Then we would not need the second level for classification and could classify based on the majority of activated detectors of one class.

This would rob us of many detectors that for example can distinguish numbers 1 and 7 from the rest of numbers. Another problem is that in the set of detectors most of them would recognize the number 1 which is the simplest, but it could happen that none of the detectors could recognize number 5, because of its complicated shape. We could define another fitness metric that would reward also detectors which can recognize any set of digits from the rest. But then there would be another set of detectors, that the fitness would not take into account which can for example can partly recognize some classes etc.

As you can see there are many problems with a typical evolutionary approach, that would be difficult to solve with a fitness function. Therefore we will not care about some quality of detectors and only focus on the diversity of detectors. I. e. during the evolution we do not care if detector detects black dot in the corner of image or perfectly number 4, we only need it to detect something novel that other detectors do not. This approach is called Novelty Search.

We define the novelty of detector in the following way. To define novelty, we need to first define a *behavior* of detector. We propose a behavior definition as a vector with 10 real numbers, each can acquire a value between 0 and 1. Each value in the behavior corresponds to one of digit classes and represent a probability of recognition for particular class by the detector.

The behavior is computed as follows. First we process the training images through the detector and get one output value for each image. We round all values to 0 or 1. This will give us boolean function if a feature in the image was recognized by the detector or not. Note that when we use the outputs in the second level we use them without rounding, the rounding is only for the purpose of calculation of the behavior.

When we have the rounded values we sum them in the original class. This way we get 10 numbers each for one of digit classes representing how many images were recognized by the detector. The last step is to divide each value by the total number of images of that class. This will give us a prior probability of recognition of class by the detector. Example of behavior is shown in the table 3.1.

Class	0	1	2	3	4	5	6	7	8	9
Behavior	0.1	0.93	0.22	0.41	0.62	0.09	0.34	0.78	0.23	0.51

Table 3.1. Example of detector's behavior that detects vertical line in the image. You can notice that numbers which contain vertical part in the middle have higher probability of being recognized by the detector.

The mathematical definition of behavior is the following:

$$\forall c \in C : behavior[c] = \frac{\sum_{p \in P_c} round(detector(p))}{|P_c|}$$

where C is a set of classes, in our case digits 0 to 10, P_c is set of images for the class c.

Now we need to define the distance metric, which is used in the Novelty Search to define novelty of detector. For this purpose we use Manhattan distance of behaviors between two detector behaviors, and divide this metric by 10 in order to have a normalized value. This shows us how differently the detector behave from the others. The farther apart behaviors are the more novel they are considered.

The novelty of detector, which is used instead of fitness, is computed as an average distance to the k-nearest neighbors (in our experiments we use the 10 nearest neighbors). The higher the novelty of detector is the more novel the detector is considered and the more it is prioritized over less novel detectors in the evolution.

The Novelty Search is used in the way as was described in the previous chapter. We are evolving the population of detectors and whenever we need to select two individuals for the breeding we use the tournament rule. Therefore select several individuals, from them select the two with highest novelty and breed them. For the process of evolution we are using the HyperNEAT, therefore the crossover and mutation are handled by the HyperNEAT.

Our novelty metric covers all interesting behaviors of detectors. But some are less useful than others. For example a detector which has 50% probability of recognition of some classes is less useful than detector with those probabilities closer to 0% or to 100%. It is because when it has crisp behavior over some classes we can rely on it more than when it detects the class only sometimes. But that does not mean we should remove all detectors that are not crisp, they still can be important in combination with other detectors.

3.3 Using the Novelty Archive

During the Novelty Search the algorithm stores detectors which are novel in some generation in the *archive*. The original reason for the use of archive was to avoid backtracking during the search. Since if some novel individual was created the search would explore its neighbor. In case that it found a solution the search would stop. In case that the search would not found the solution even in its neighbor we would not want to search there again in the future, therefore adding it to the archive partly prevent searching in that area in the future.

But in our approach we are not searching for one solution and we want the novel individuals from all generations. That is why we give the archive second purpose. In the archive are individuals which had a high novelty in some generation and it is highly possible that they are not in the final population. Therefore before selecting the set of detectors used for the final classification level, we join the last population with the individuals in the archive. This will give us more distinct population.

The archive can contain hundreds or thousands of detectors at the end of evolution, because it is not bounded to any maximal size, where most of them will have similar behavior. We need to define a metric to select only a subset of detectors for the final classification level. This metric is described in following section.

3.4 Selecting detectors for the second level

The set of detectors from the last population and archive can easily get up to thousands of detectors during the evolution. Similar detectors can get into the archive, some detectors are useless, for example these which detect everything or nothing. Those detectors will not improve the classification by the second level. Also due to the time complexity of the training of the succeeding ANN we cannot use them all. Those are some of the reasons why we need to use some method to select only a subset of detectors.

3.4.1 Random Selection

Suppose that we have several thousands of detectors in the population and we want to select only subset of 100 detectors. A trivial method would be to select them at random. They are already novel enough, since they make it into the archive or the last population. But this method can select a lot of similar detectors which do not improve the final classification that much. Therefore we define some more informative methods.

3.4.2 Maximal Difference

We would like to use more informative method than random selection. We know the behavior of each detector and we do not want useless detectors such as detectors which detect all classes, none class or each class on 50%. Therefore we define following method for detector selection called *Maximal Difference*. This method first computes for all detectors the difference between their most recognized class and second most recognized in the behavior, i.e. the difference between highest value in behavior and the second highest. Then it select the detectors which has the highest difference. In other words it selects those detectors which recognize one class more often than the rest of classes.

If we would select only the first n detectors with the highest difference, we would get unequal representation of recognized classes. For example it could select only detectors detecting the number 1, because it is easier to recognize from other classes, therefore it would have more detectors with the highest difference. To avoid this problem we improve this method to select the same number of detectors detecting each class independently. Therefore even if some class has poor performance for recognizing it from the rest it gets at least some attention.

3.4.3 Smallest Entropy

Previous method assumes that detectors are already good enough by themselves, but we do not put any such restriction on them during the evolution and so the previous method does not use the full potential of detectors. We do not want only detectors that can recognize only one class from the rest, we also want detectors that can separate different subsets of digits from the remaining.

We can define useful detectors as the ones which do not contain in the behavior any probability around 50%. The more crisps they are the better for the recognition process. The more their probabilities are closer to the 50% the less reliable they are.

In order to use more useful detectors we define method which use the entropy of behavior to select detectors. The entropy is defined as follows:

$$H(B) = -\sum_{i=0}^{9}(B(i)log(B(i)))$$

where B is the behavior of detector. The entropy is an interesting metric since it has smallest values around 0 or 1 and has highest value for 0.5. Therefore the smaller an entropy is the more crisps and more useful the detector becomes. We define method that selects detectors with as small entropy as possible and we call it the *Smallest Entropy*.

The Smallest Entropy method has one issue. When selecting detectors it usually select several similar detectors because all of them has the smallest entropy in the population. We need to solve this issue by defining metric that forbid selecting similar detectors.

We solve this problem by defining the *certificate* of behavior. It is a string of length 10 containing only zeros and ones. We use it to define which set of classes is already detected by selected detectors to avoid selecting similar ones.

The certificate is calculated by rounding all values in the behavior. For example the behavior from the figure 3.1 would have certificate `0100100101`. The certificate serve as an identification of which classes are recognized with the detector.

During the selection of detectors we are selecting detectors with the smallest entropy and whose certificate in not in the set of selected detectors. In other words when we have already selected detector with one certificate we do not need another detector recognizing same classes only with higher entropy, i.e. with the less crisp behavior. In case that there would be no detector remaining to select, the method selects the rest of detectors at random. This can happen because there is only 1024 different certificates and generating detectors does not always cover all possible certificates due to the fact that some classes are more difficult to differentiate than others. But during

the experiments we found out that usually only up to 5 detectors were selected at random.

3.4.4 Maximal Distance

Previous methods do not use information about distance between the detectors. They only care about the behavior of detectors but not about actual coverage of the behavior space by selected detectors. We can exploit this information and get the most diverse set of detectors.

We define method *Maximal Distance* and define it as follows. At first it selects initial detector at random from the whole population. Then it gradually adds detector whose distance to the closest selected detector is maximal. In other words, it gradually selects detectors which are the farthest from the selected detectors.

This will select most diverse set of detectors. Moreover this method selects the farthest detectors first therefore it usually starts with selecting the ones with the more crisps behavior. Therefore it ignores vague detectors with probability of recognition for some classes of 0.5. But conversely from previous methods it later selects even the less crisps detectors if they are far enough from the rest.

3.5 Grid based Multi-level Hierarchy

So far the direct approach was defined, where the original images are directly transformed into the one value outputs. This approach seems to be trivial and its extension could improve the results.

Inspired by the deep networks we try to discover more than two level approach. As shown in the experiment section, the direct approach has good results, therefore in this section we try to extend it into the approach with more levels.

The output of each detector is a value between 0 and 1, it can be represented as a black and white color. Therefore if we would select a set of detectors, ordered their outputs into a grid, we could consider that grid as a new image. The new images could be then used again for the evolution of new detectors and repeat this process several times before final classification. The final classification is handled in the same manner as in the direct approach.

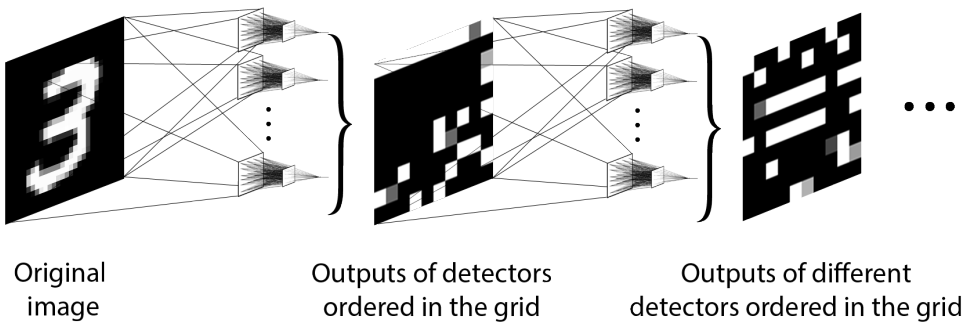


Figure 3.3. We can use the detectors to transform image into a new image and repeat this process several times before final classification.

The transformation made by a single level can be defined as follows:

$$\forall a = 1, ..., w', b = 1, ..., h' : p'_{a,b} = detector_{a,b}(p)$$

where w', h' are the width and height of transformed image, p is an original image and the p' is the transformed image. For example if we define width and height of transformed image equal to 3, we need to select 9 detectors and the transformed image space would be equal to the following:

$$p' = \begin{pmatrix} detector_{1,1}(p) & detector_{1,2}(p) & detector_{1,3}(p) \\ detector_{2,1}(p) & detector_{2,2}(p) & detector_{2,3}(p) \\ detector_{3,1}(p) & detector_{3,2}(p) & detector_{3,3}(p) \end{pmatrix}$$

This way we can transform all images from the original image space P into the transformed image space P'.

In following sections we describe two ways of creating a grid, each of them use only one additional level. Therefore it transforms an image into a new space, on the transformed images it generates new detectors and then it tries to classify it. In other words it adds one level to the direct approach.

Both methods utilize formerly described detectors, which they select from the population and reorder them into a new two-dimensional grid. Then outputs of detectors in the grid are considered as a new image and new detectors are generated over those images. Last step is to use the second detectors and classify the original images (note that we could repeat the process of creating new grids again).

However, we will see from the results of experiments that both approaches have worse performance than the direct approach. In the discussion after the experiments we try to explain the reasons of this failure.

3.5.1 Grid with rows depending on the class

For the first approach we define simple ordering of detectors in the grid. It generates a grid where on each row detectors can recognize one particular class more than others.

This method uses detectors selected with the Highest Difference method, i.e. the detectors which can recognize one particular class much more then the second most recognizable class.

The grid consists of 10x10 detectors, where detectors on each row are selected based on their preferred class. For the row i we select detectors which have the highest probability of detection a class i and as small as possible probability of detection for the second most detected class.

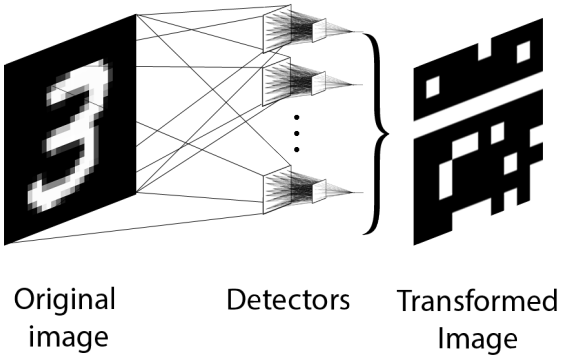


Figure 3.4. The illustration of the approach creating grid with rows depending on the class. It takes 100 detectors and order their outputs into the grid. You can notice that all detectors in the row that correspond to the number three are activated in the transformed image.

Whenever we have all detectors aligned in the grid, we transform all images from the database to a new space and consider them as new images. We could again repeat the process with the grid, but we use those new images to train the detectors which are fed into the final ANN for classification.

We try to improve this approach and experiment with bigger grid which use 20x20 detectors. In the second grid only change is that each two rows contain detectors recognizing the same class. Therefore detectors from first two rows recognize digit 0, detectors from next two lines recognize digit 1, etc.

3.5.2 Grid with both rows and columns depending on the class

Previous approach lack any ordering inside of each row. We can try to explore the possibility of ordering in the rows in this approach. During the selection of detectors we take into account also second most detected class based on the column in the grid. Therefore for the row i and column j we select detector with the highest probability of detection for class i and the second highest for class j.

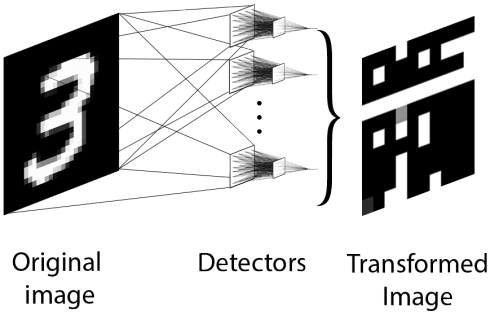


Figure 3.5. The illustration of the approach creating grid with rows and columns depending on the class. It takes 100 detectors and order their outputs into a grid. You can notice that all detectors int the row corresponding to number three and most of the detectors from the column corresponding to number 3 are activated in the transformed image.

Detectors selected into the grid are the ones with the highest difference between the first and the third most recognized class. In the following figure you can see how the detectors are ordered and that most of the detectors are activated on the corresponding row and also the corresponding column.

We also experiment with a bigger grid of 20x20 detectors. In the similar manner that we have increased the grid in the first approach, we increase the grid in this approach. Therefore instead of one detector in the grid that recognize class i most often and j second most often, we have 4 detectors with this attribute. It is illustrated in the figure 3.6.

Figure 3.6. The illustration of a bigger grid. Digit before row defines most recognized class for the detectors in that row. Digit on top of the column defines second most recognized class for each detector in that column. This is a transformed image of digit 6.

3.6 Complex Detectors

In the Direct approach and the Grid based approach we are using detectors with only one output. This structure of detectors provide a simple way of defining the behavior, but during the evaluation of the image lots of information is lost because of that simplicity in the output. It could be argued about the impact of this reduction, because in the experiment section it is shown that this simple structure has really promising results. But we try to investigate more complex detectors and using the Novelty Search on more levels.

In this third part of our work we describe method of sequentially scaling down an image until we get one output detectors. Their output will be fed as in previous approaches into the final ANN trained by Resilient propagation. For example the first level of detectors would transform images of 28x28 pixels into 16x16, then the second layer would transform those into 8x8, the third level would transform them into 4x4 and the last level of detectors would transform it to a single value.

The structure of the detector is similar to the structure of detectors from the direct approach only that the input and the output layer has different sizes. In the figure 3.7 you can see the structure for the first level detectors transforming original images of 28x28 into images of 14x14 pixels.

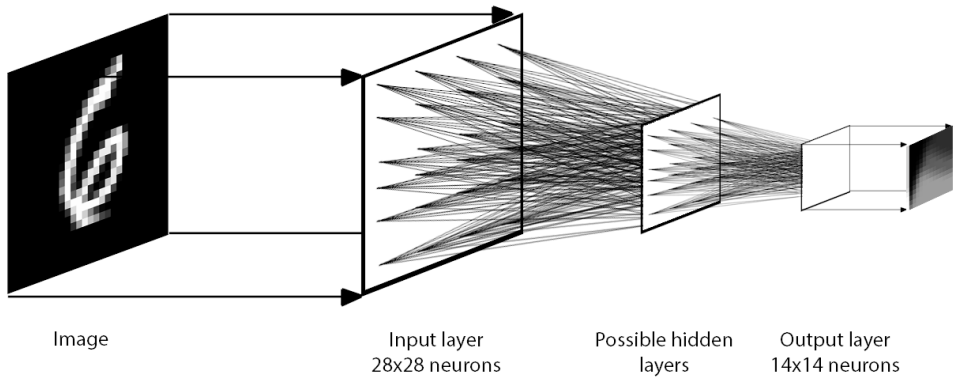


Figure 3.7. The complex detector transforming MNIST images into 14x14 pixels image. It has input layer of 28x28 neurons, possible hidden layers and output layer of 14x14 neurons.

3.6.1 Architecture with complex detectors

We have defined the structure of detectors. In this section we describe how we are using the complex detectors in the classification process.

The architecture has a tree-like structure, where first we evolve set of detectors from initial images. Then each of the detectors transform all training images into its own transformed space. On each of the transformed spaces we evolve new detectors and transform that space even further. We continue this process until we evolve detectors with one value output. You can see the illustration in the figure 3.8.

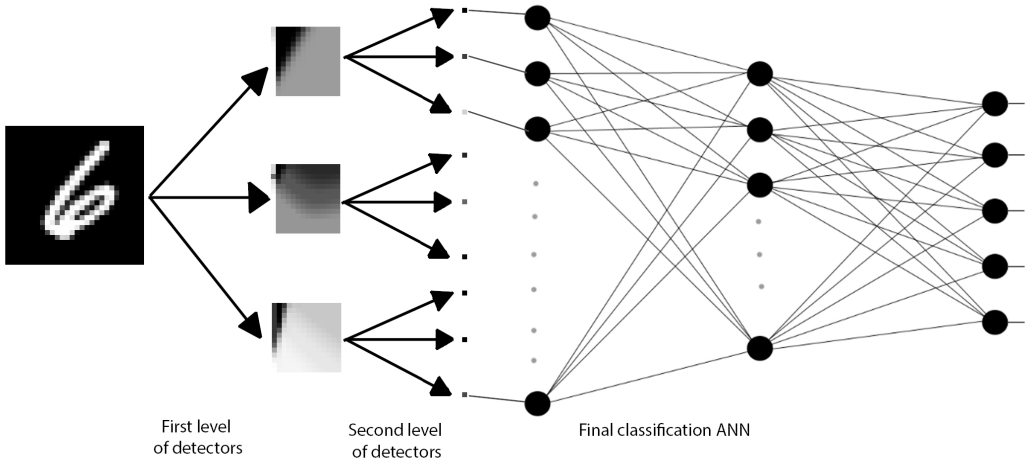


Figure 3.8. The illustration of the tree like architecture. The first level contains three detectors, where each of them transform original images into a space of 14x14 pixel images. The second level contain three detectors for each transformed image space of previous layer. The second level detectors transform the images into single value outputs. At the end a final ANN combine all outputs of the second level together and train itself on them by RProp.

In this architecture each population of detectors is generated over the images generated by a single detector from the previous layer. Whenever we reach the layer where detectors have only one output, we collect it all together by the final ANN, which has same number of inputs as is the total number of detectors in the last level. This final

ANN is trained by Resilient propagation and outputs 10 values, where the ID of the most activated neuron is considered as a classification of the image.

We can use several levels of detectors, but to make it simple for experimenting we define only two levels. In the first level there are detectors which transform the original image into 14x14 images and the second level detectors transform it into single value output.

3.6.2 Behavior of complex detectors

The behavior of a simple detectors was defined straight forward, but when we have the complex detectors, it is difficult to define behavior metric which would be simple and useful. Therefore we propose a new behavior metric.

First let us consider few things what we need from the behavior. We can look at the transformed images as an original images, therefore we would like to have as distinct images of different classes and also as similar images of different versions of one digit. The behavior should take it into account.

What we get from the detector, that is used for the computation of the behavior? There is up to 60 000 of transformed images labeled by 10 classes. This is too much information to evaluate, not to mention that computation of the behavior should be as fast as possible. Therefore we average all images belonging to each of the classes separately. For each pixel we take its average over all values of the same pixel in the images of the same class.

This way we get averaged 10 images, one for each class. In the figure 3.9 you can see the effect of averaging on the original images in the MNIST database.

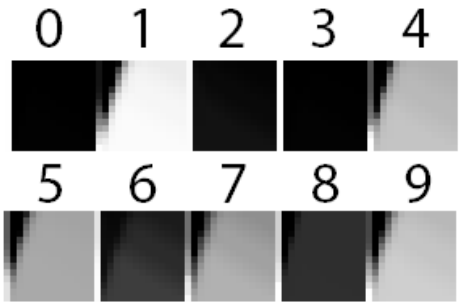
Figure 3.9. The averaged images from the MNIST database. Each digit is averaged over all images of the same digit.

We have also experimented with the use of standard deviation for each pixel, but that was not useful. Therefore we consider the 10 averaged images as a representation of each class in the transformed space after using the detector.

During our research we first considered those 10 images as a behavior of the detector and compared the behavior as a squared distance of all pixels in those images. But that proofed to be a dead end and the generated detectors were not distinct at all. That was the reason to redefine the behavior of the complex detectors as follows.

As a behavior of the complex detectors we use a matrix of 10x10 values. Each cell $[i, j]$ in the matrix is defined as a distance of the averaged image of class i from the averaged image of class j. Therefore the matrix contain distances of all pairs of averaged images. The distance of two averaged image is a sum of squared distances of pixels in both images on the same position. This matrix has 0 values on the diagonal and is symmetric. All values in the behavior are divided by the number of all pixels, so they

are always between 0 and 1 (it is important for the HyperNEAT and NS). You can see an example of behavior in the figure 3.10.



	0	1	2	3	4	5	6	7	8	9
0	0	0.79	0	0	0.47	0.37	0.03	0.35	0.03	0.52
1	0.79	0	0.72	0.78	0.04	0.08	0.53	0.09	0.52	0.03
2	0	0.72	0	0	0.41	0.32	0.02	0.3	0.02	0.46
3	0	0.78	0	0	0.46	0.36	0.03	0.34	0.03	0.51
4	0.47	0.04	0.41	0.46	0	0.01	0.27	0.01	0.27	0
5	0.37	0.08	0.32	0.36	0.01	0	0.2	0	0.19	0.01
6	0.03	0.53	0.02	0.03	0.27	0.2	0	0.18	0	0.31
7	0.35	0.09	0.3	0.34	0.01	0	0.18	0	0.18	0.02
8	0.03	0.52	0.02	0.03	0.27	0.19	0	0.18	0	0.3
9	0.52	0.03	0.46	0.51	0	0.01	0.31	0.02	0.3	0

Figure 3.10. On the left side there are averaged outputs of single detector, on the right side is the behavior matrix computed from those images.

This metric represents which digits can the detector differentiate from each other and which not. During the evolution we do not care how it differentiate but we are trying to find a set of detectors where each can differentiate different set of classes.

The novelty of a detector needed during the evolution is calculated based on its distance from behaviors of other detectors. This distance is calculated as follows:

$$distance(B^a, B^b) = \sum_i \sum_j \left(B^a_{i,j} - B^b_{i,j}\right)^2$$

where B^a and B^b are behavior matrices of detectors a and b.

3.6.3 Minimal criteria of detectors behavior

When we have investigated the complex detectors that are evolved in our approach, we have found out that many of them are useless and contained too much zeros in the behavior matrix. That means that they cannot distinguish two classes from each other. This happened more often than we would thought, see the figure 3.11, where are averaged images from three different detectors, which have nearly all values in the behavior zero.

Those detectors are useless, since we cannot get any information from images that are unrecognized from each other. Those detectors should be omitted from the population, but because of the Novelty Search they are also considered as a novel, because they have some non-zero values in different positions the the behavior.

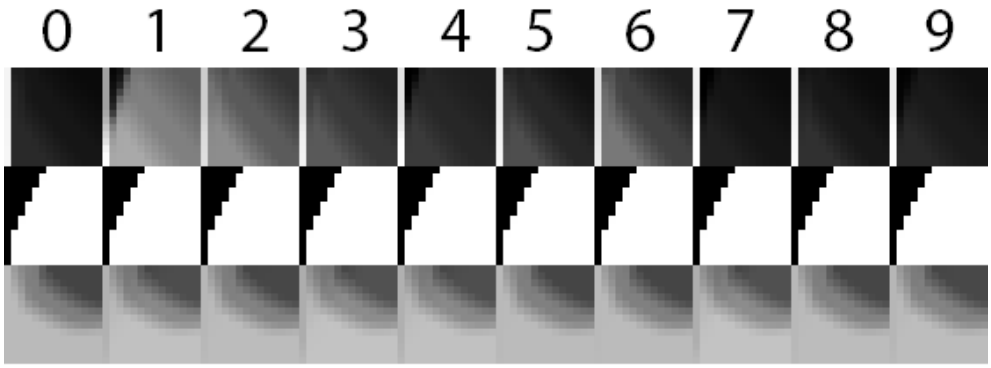


Figure 3.11. There is example of 3 distinct detectors, that have nearly zero behavior matrices.

In the work [17] Lehman and Stanley suggested improving the Novelty Search with a minimal criteria. It is a threshold value which each of the individuals in the population must reach to maintain in the population. It ensures that computation resources are not wasted on useless individuals. In their work they showed great improvement in the performance of NS.

We can use this improvement in our work to avoid useless detectors from being left in the population and wasting the computation power. Note that we do not want to remove all useless detectors, when they can distinguish at least little between classes.

The behavior of detector is a matrix, which cannot be compared to the threshold. To find out if the detector is useless, we first sum all values in its behaviors matrix top triangle (the bottom triangle is the same). This way we get one value somewhere between 0 and 45 (actually in reality it cannot reach the 45 value, the best detectors usually has the sum around value 10). We define the threshold to have a value of 0.01, this way it discards from the population only detectors that cannot recognize nearly any class from others. We have defined the threshold value empirically, because the 0.01 had the best results.

Chapter 4
Implementation

In this chapter we describe the implementation part of this work. We describe main issues and how have we overcame them. All codes are available on the attached CD with all results of experiments and other binary files used for this work.

This work is a research work, therefore a lot of code with various dead-end or parts that had to be rewritten could not be presented in the final implementation. In this chapter we describe the final implementation as it is presented in the source codes.

4.1 Architecture

I have programmed the code with the Java programming language in the Netbeans IDE. The code is separated into several packages: main package, simple detectors, complex detectors, support classes, experiment settings, artificial neural networks.

The architecture has due to the research nature of the work separated classes for each part of training, so they can be run separately. There is separate class (NoveltyEvolution.java) for running the NeuroEvolution, then another class (NoveltyStorage.java) handles selecting subset of detectors from the population or from the storage and at last separate class (ANNmain.java) handles training the final ANN.

Since the architecture use the blocks of codes I have needed a structure to link those parts together. For this purpose I have created a class ImageData.java, which at the beginning loads a complete MNIST database, then after NeuroEvolution it transforms all images into the feature vector based on the selected images and at the end it is used for training the final ANN and then the classification. Moreover after each subtask it can save the intermediate results and continue next time from the saved point.

This architecture proved to significantly improved the research process, since mostly only parts of the whole process needed to be evaluated.

There is several supporting classes, for example for visualization purposes, thread handling or methods which allows me to track and interfere with currently running code. Most of those methods were used during the research and are not important for the final methods.

4.2 HyperNEAT and Artificial Neural Network

For the HyperNEAT I am using Java implementation called Another HyperNEAT Implementation (AHNI). It was produced by Oliver Coleman at The University of New South Wales by extending the pre-existing ANJI version of NEAT (which was written by Derek James and Philip Tucker). He also published an undergraduate thesis [2] that explains different HyperNEAT experiments in detail.

The main part of my research uses AHNI. The HyperNEAT evolves detectors and use them to transform images into the vector of features, where each feature is output of one detector. Then it supplies the feature vector of each image to simple Artificial Neural Network with one hidden layer, which produces the final classification.

For the classifying final ANN I am using advanced machine learning framework Encog [1]). It was created by Heaton Research. It supports different learning algorithms such as the Bayesian Networks, Genetic Algorithms, Support Vector Machines and mainly Artificial Neural Network algorithms. The Encog contains classes to create a wide variety of networks, as well as support classes to normalize and process data for these neural networks.

Figure 4.1. The logo of Encog framework. You can find more about the framework at http://www.heatonresearch.com/encog

I am using this framework in the final step of learning phase. Artificial Neural Network process the feature vectors generated by HyperNEAT and it returns vector of length 10. The highest value in the vector is considered the classification for particular feature vector. As a training technique for the final ANN I have experimented with Backpropagation, Quick propagation, Resilient propagation and Scaled Conjugate Gradient propagation. I have decided to use the Resilient Propagation as a training technique because it has proven to give best results in shortest time over other mentioned methods.

I have made several modifications in those frameworks. For example the implementation of Novelty search in AHNI did not preserved individuals in the archive, therefore I have amended it in a way that the individuals could be pull off the archive at the end of evolution. Other modifications are not that significant and they are commented in the code. I have also repaired a bug in the AHNI library and I have submitted this bug to the author. It made the Novelty Search exit with an error.

The AHNI uses a .properties files to save the settings. I have build on that and added some of my own settings to those files.

4.3 Detector storage

We can evolve detectors every time we need them. But since they are independent from the rest of the learning process we can generate them in advance and then use them whenever they are needed. This way we can do longer and more thorough evolution and then only select a subset of detectors from the evolved population.

[1]) `http://www.heatonresearch.com/encog`

For this purpose we define structure, which we call *storage*. It is represented as a linked list with several function for handling the data. There are several methods to generate a subset of detectors from the set of stored detectors. Those methods are described in the research chapter.

We can define a threshold for adding new detectors into the storage. Whenever new detectors are inserted into the storage it compare it to all detectors already in the storage and if it finds a detectors with behavior closer than the threshold it forbid adding the detector into the storage. This way we prevent overpopulation of the storage with similar detectors.

For the experiment purpose we have generated a storage with size of 5191 detectors. It contains simple detectors without any hidden layer. We have generated it by running the evolution several times with different parameters. The runs contained population size ranging from 10 to 100, number of generations ranging from 50 to 500 etc. But all detectors in the store have the same structure without any hidden layers.

The main purpose of this structure is to accelerate calculations. Another possible feature is that whenever HyperNEAT generates population it does not have to distribute all individuals uniformly over the whole behavior space, but when we put together detectors from several separated evolutions they can be more diverse. But in the experiment section we show, that HyperNEAT actually provide diverse population of detectors which covers the space uniformly. Therefore the storage with detectors from many evolutions is not that much superior over the use of only one evolution.

4.4 Handling testing labels

The MNIST database contains labels for both the training and the testing images and it could happen that I would make a bug while programming and use testing labels in learning phase. This would jeopardize the whole research since the model would be learned on the testing labels. To prove that my results are not affected by any mistake I have done the following measure.

In the code there is only one class that directly handles MNIST database. It prevents to access testing labels during training phase. After the learning phase it classify all testing images and save the results into a separate file. Then I run the separate code that loads these results and compare them with correct labels to calculate actual testing error.

I am aware of inefficiency of this precaution. I have decided to use it for two reasons, the first is that it proves that the results are unaltered by the testing labels and the second reason is that this way all results of experiment can be attached on the CD.

4.5 10th fold Validation

As mentioned earlier I do not use the testing images during training phase and therefore I cannot check performance of the algorithm to decide when it is good enough to stop training phase of the Resilient learning. We could run the training for enough epochs and use the last state to classify all testing images. But this approach could induce problem with over-fitting, where the network learns perfectly how to recognize training set, but has poor performance over the testing data.

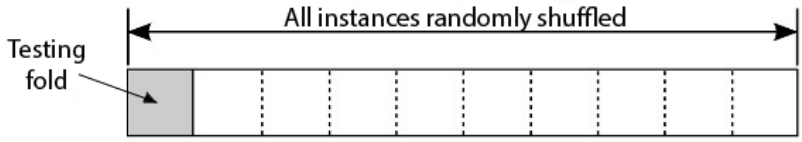


Figure 4.2. The illustration of the 10th fold validation

Therefore at the beginning of each experiment I randomly split data into 10 folds and then select the first one as a testing set and the rest nine folds together as a training set. Then after each epoch of the Resilient propagation I compute the error over that testing set. For the final evaluation I am using the state where the network has the smallest error over the testing fold. In that state I am classifying the actual testing images provided in the MNIST database.

I could also perform 10-fold cross-validation, where we are switching which fold is considered as a testing. But I have rejected this method since we have enough images in the training set and the 10-fold cross-validation would take more time and the improvement would be insignificant.

4.6 Selecting subset of images

The evolution of HyperNEAT and Resilient propagation runs longer with more images over which we are training it. But we do not need to train it over all 60000 images, it is sufficient enough to select a subset of 1000 images for example. We could select all the images at random, but that could select more images of one class over other.

Therefore we start by selecting random images and we count how many images of each class was selected and whenever we select more than 1/10th of images of some class we block that class and forbid selecting more images of that class. This way we get nearly same number of images for each class. It is not exactly 1/10th of images for each class, because we are allowing to add one image over the limit. This way we do not need to wait until random selection select the last missing class at the end of selection. Note that the behavior of detectors is averaged over the number of images in each class separately.

Chapter 5
Experiments and Results

In this chapter we experimentally evaluate proposed methods and compare the results. We are only concerned in the smallest error rate over the set of testing images and do not consider the time complexity. Because this is a new approach we are more concerned with the results than with time complexity which could be improved in future works or whenever someone would like to implement it for real life purposes.

The research part was done on my personal computer which has 4 cores. The use of Metacentrum in the research part showed up to be unsuitable, since after computing several experiments, the tasks had to wait in the Metacentrum queue for several days before starting. That would slow down the research process significantly. On the other hand the experiment part, where we needed experiments to be averaged over several runs, was performed on the Metacentrum cluster. I could use there one CPU with at most 24 cores (the remaining 8 cores were left for the maintenance). There all experiments run several times faster than on my personal computer, but I had to wait for the results for several days.

All used settings of HyperNEAT can be found in the appendix A, those settings did not change during all experiments. Other settings, which were changed from the experiment to the experiment, are mentioned with the particular experiment.

The settings of experiments and all our results are saved in the attached binary files for the future work or a reviewing purposes.

All results show an error rate of particular method. The error rate is a percentage of incorrectly classified testing images. All experiments use whole database of 10 000 testing images for the computation of the error rate.

5.1 Comparison of Detector Selection Methods

The first experiment compares three different methods for detector selection. All three methods were described in the Research section. Those methods are used whenever we need to select a subset of detectors from evolved population.

The first method is a simple random selection. This method should have reasonable results, since all detectors are already diverse. We consider it as our baseline. Second method is the maximal difference method which select detectors that recognize one particular class more probably than others. Third method selects detectors with the smallest entropy of the behavior. And the last method is the maximal distance. It selects a set of detectors whose behaviors are as far apart as possible.

The results are averaged over 60 runs. We are selecting detectors from the storage, where all methods have access to the same set of detectors. This storage contains over 5000 detectors evolved from several evolutions. Each run consists of selecting 50 detectors and then training the final ANN. This ANN has one hidden layer consisted

with 50 neurons and is trained by Resilient propagation for 1000 epochs. Note that the final ANN is learned on 9/10 of the training data and the test data were evaluated only for the output with the smallest error on the last 10th fold of the training data as described in the implementation chapter. The results are in the table 5.1.

Method	Error rate
Random selection	13.32 %
Maximal difference	13.06 %
Smallest entropy	11.98 %
Maximal distance	10.48 %

Table 5.1. Results of experiment comparing detector selection methods.

It is a surprise that the maximal difference method performed nearly the same as the random selection, we have supposed better behavior. The smallest entropy method shows improvement from maximal difference. But the last method, which selects the most diverse set of detectors, proves to be the best way of selecting detectors.

Therefore whenever in the following experiments we are selecting a subset of detectors we always use the maximal distance method, because it had the best results in this experiment.

5.2 Detectors from the storage versus newly generated

In this experiment we compare the use of the detector's storage and the generation of detectors. There is over 5000 detectors in the store generated from several runs. We are trying to compare it with detectors which are newly generated to find out if the evolution covers the space uniformly. For the detector selection we are using the maximal distance method.

Main goal of this experiment is to show if the HyperNEAT evolved detectors cover the behavior space evenly or if detectors from several evolutions have significantly better performance.

For the generation of detectors we allow maximum of 1000 generations, the size of a population is equal to the number of detectors needed. We do not need a complete database of images during the evolution of detectors, therefore we use a subset of 1000 images to speed evolution up. All detectors in this experiment do not contain any hidden layer. Results are averaged over 20 runs. Final training over the outputs of detectors is done by the ANN learned by Resilient propagation for 1000 epochs with hidden layer of 50 neurons.

Number of used detectors	From storage	Evolved detectors
50 detectors	10.48 %	11.67 %
100 detectors	8.87 %	9.04 %
200 detectors	8.69 %	8.73 %

Table 5.2. Results of the experiment showing error rate for experiment comparing the use of the store and evolving detectors.

By comparing the result we can see that the use of storage, which contains detectors from several runs and contain over 5000 detectors, is not that better than detectors evolved from the single evolution. This suggests that evolution of detectors covers the space of behaviors evenly and we do not need to select detectors from several separate evolutions.

The detectors evolution is time consuming, that is the main reason why we are using the storage in our experiments. We need to evolve detectors only once and then for different purposes we can select different subset of detectors.

5.3 Structure of detectors

In this experiment we compare the performance of detectors with different number of hidden layers. We experiment with several structures of detectors: no hidden layer, one hidden layer of 5x5 neurons, two hidden layers where the first layer has 5x5 neurons and the second hidden layer has 10x1 neurons; and detectors with three hidden layers where the first hidden layer has 12x12 neurons, the second has 5x5 neurons and the third hidden layer has 10x1 neurons.

Results are averaged over 15 runs, the final ANN trained by Resilient propagation contain one hidden layer with 50 neurons and it can learn for at most 5000 epochs.

To make the conditions as similar for each structure we evolved one population in the single evolution of HyperNEAT, with 5000 individuals and running up to 1000 evolutions. At the end of evolution the final population and the individuals from the archive are connected into the single population from which we select different subsets of detectors for each of the runs.

Number of hidden layers	Zero	One	Two	Three
50 detectors	8.87%	9.68%	8.21%	8.2%
100 detectors	7.77%	7.35%	6.77%	6.43%
250 detectors	7.11%	5.35%	4.77%	4.64%
500 detectors	5.85%	4.87%	4.01%	3.9%
1000 detectors	4.6%	4.4%	3.67%	3.53%

Table 5.3. Results for the experiment showing impact of number of used detectors on classification. The first columns shows error rate for population of detectors without hidden. The second column is when the detectors has one hidden layer of 5x5 neurons. The third column shows results for detectors with two hidden layers, the first has 5x5 neurons the second has 10x1 neurons. The last column shows results for detectors with three hidden layers 12x12, 5x5 and 10x1.

The more hidden layers the detectors have the better performance they have. It would be interesting to experiment with more hidden layers, but we had reached a problem with the time complexity since all layers are fully connected. For example there is 20000 connections in a detector with two hidden layers and there is nearly 120000 connections in the detector with three hidden layers, i.e. 6 times more after adding one hidden layer. The HyperNEAT becomes significantly slower with each added hidden layer, because it needs to calculate values for all connections.

5.4 Simple ANN classificator

This experiment is designed to show that our method is better than simply training the ANN with Resilient propagation on the original images. For this purpose we create ANN with 784 neurons in the input layer (each neuron will get one pixel of 28x28 images), with one hidden layer of 50 neurons and output layer with 10 neurons. The structure is similar as the one used in final part of classification of our methods, except that instead of detector's outputs we feed the ANN with pixels from the original images. The figure 5.1 shows the error rate of the ANN over 1000 epochs of Resilient propagation. The values are averaged over 30 runs for each epoch.

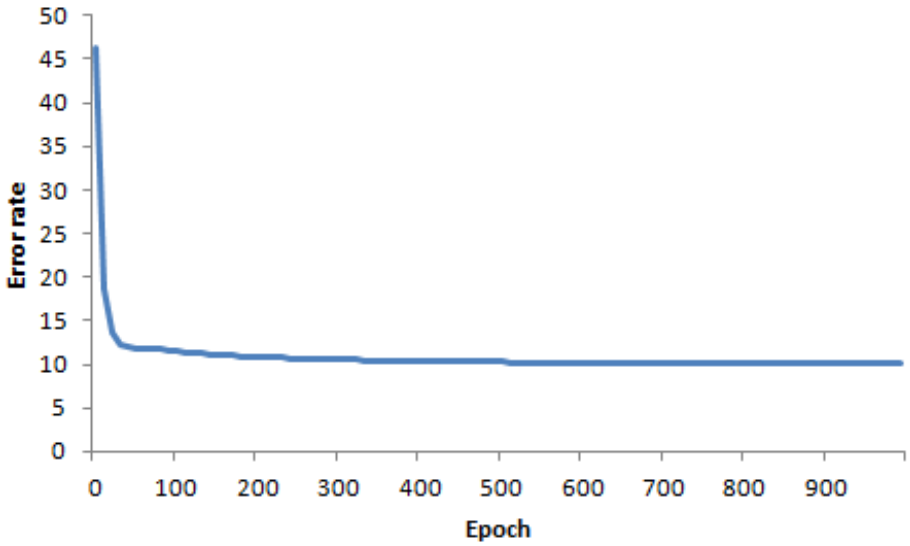


Figure 5.1. The results of experiment without any detectors. We are training the ANN over pixels from original images.

The ANN quickly adapts to the data but has final error rate of 10.03%. If we compare this result with results of our method it perform rather poorly. For example our method needs only 50 detectors to have better performance (see table 5.3). Moreover if we compare it to the same number of outputs as has the ANN from this experiment, i.e. 784 detectors, we would get even better results between 4% and 6% (compared to the results in the table 5.3).

This experiment shows us that the use of detectors to detect features in the image improves classification. Therefore it is reasonable to study it more deeply. It also proves that the quality of the classification is not a result of using only the final ANN trained with RProp.

5.5 Comparing results with related work

In the work [5] (2013) they have experimented with the use of HyperNEAT in the image recognition. They also used the MNIST database. To our knowledge this is the only paper considering this approach in the image recognition domain. They developed two architectures. The first one is similar to ours, where they developed a population of feature detectors and the most fit individual was fed into the final ANN. They got a

resulting error rate of 41.6%. If we compare this result with any of ours, we can see that our methods are significantly better than their.

The second approach they developed was based on the CNN. In this approach they got an error rate of 7.9%. Therefore in this experiment we perform similar experiment as they did, to show if our method improves their results. We perform an experiment with similar settings as their first approach, which has similar architecture to our direct approach.

The settings for this experiment is defined as closely to those in the [5] as possible. The population of 256 individuals is evolved for 2500 generations. The structure of each detector is similar as in their work, i.e. input layer of 28x28 and 4 hidden layers 16x16, 8x8, 6x6 and 3x3. One change is that their output layer contained 100 neurons, but in our work we have detectors with only one neuron in the output layer.

In the fitness based approach described in that paper only one detector is selected for the final classification, but in our work we need to select a set of detectors, therefore we select 256 selectors from the final population as was the size of initial population (we could select all detectors ever evolved and get even better results). In their work the final ANN was trained over 250 epochs, but it was calculated over and over during the evolution process. We need to run it only once at the end of evolution, therefore we have decided to let it train over 2000 epochs, which was sufficient enough to converge to the result without further improvements. Only 300 images was used in the evolution, the final ANN was trained over 1000 images. The result were averaged over 10 runs.

With this setting, we got an error rate of 6.45%. The Novelty search significantly improves the results of the HyperNEAT evolution in comparison to the error rate of 41.6% and it was even able to outperform their best result using CNN of 7.9% in the paper [5]. Moreover they presented that result as a best they could get, but with our method and the same settings we could get even better result if we would not select only 256 detectors from the population but many more. Usually after the evolution we got around 1500 detectors in the archive and the final population and selecting more detectors from it could improve the results significantly as shown in the experiment 5.3.

We were able to reach an error rate of 3.53% with our settings (see table 5.3), which is much better than the mentioned 7.9% in the [5].

5.6 Grid based Multi-Level Architecture

In previous chapter we have defined two approaches of creating a grid based multi-level architecture. Each of the approaches uses the simple detectors which it orders into a grid. We can use those detectors selected into the grid in the comparison with the direct approach by taking them separately from the grid and training the final ANN on them. This gives us clear way to compare if the grid based architecture is promising or not.

The settings of this experiment is the following. The detectors are first selected from the store based on the grid approach definition. Then the grid is created out of them and the training images are converted via the grid to the new image space. On the new images we evolve diverse population of 500 detectors. The evolution runs for 1000 generations. Last step is to use the evolved detectors to generate feature vector on which is the final ANN trained. That ANN consists of hidden layer with 50 neurons

and it is learned for at most 1000 epochs of Resilient training. The results are averaged over 20 runs.

We compare the results with the results of direct approach. We take exactly the same detectors used in the grid and train on their outputs the final ANN with the same parameters as in the grid based approach.

Approach	Error rate	Error rate for direct approach
Rows depending 10x10	12.38 %	9.72 %
Rows depending 20x20	9.72 %	7.09 %
Rows and columns depending 10x10	12.03 %	9.77 %
Rows and columns depending 20x20	12.72 %	8.85 %

Table 5.4. The performance of the grid based architecture. The first column contain results for different approaches of grid architecture and in the second column are results of experiment where we used detectors from the grid directly for training the classification ANN.

We can see from the results that if we do not add a layer with a grid and directly use the detectors, we get better results. This suggests that the grid based architecture is not promising at all.

The experiment also revealed that making the grid bigger for the rows and columns depending approach does not improve performance as we thought. It even selects worse set of detectors than the rows depending approach with bigger grid.

We could experiment with different grids or with more detectors generated on images in the new space, but we can see that this approach is probably not the correct way for improvement. In the following chapter we try to discuss problems with this architecture and resonate why this approach performs poorly.

5.7 Architecture with complex detectors

In this last experiment we examine the architecture with the complex detectors. The architecture uses two types of detectors. The first type transforms original images into 14x14 images, the second type takes the output of the first type and outputs one value. At the end all outputs of second type detectors are taken together as an input to the final ANN trained with RProp. The final ANN contain 50 hidden neurons as in the other experiments.

The first type of detectors has an input layer of 28x28 neurons, one hidden layer of 20x20 neurons and one output layer of 14x14 neurons. The second type of detectors has an input layer of 14x14 neurons, the first hidden layer of 5x5 neurons, second hidden layer of 10x1 neurons and the output layer contain only one neuron.

The detectors of the first type were selected from a population evolved after 1000 generations, with the population of size 1000. The second type detectors are evolved for each individual output of the first type detectors with population of size 100 over 500 generations.

The whole process of training is as follows. At first a set of n first type detectors is selected. For each of them all training images are evaluated, therefore we get n-times training images of 14x14 pixels. Then for each of the new training sets the second type

detectors are evolved and from the final evolution m of detectors is selected. Therefore we have $n * m$ single value outputs. Those outputs are put as an input vector for the final ANN which is trained with RProp for 1000 epochs. The results are averaged over 10 runs.

Structure	Size of feature vector	Error rate
n=30, m=10	300	16.12%
n=20, m=20	400	19.03%
n=50, m=10	500	12.86%
n=30, m=30	900	16.13%
n=100, m=10	1000	10.91%

Table 5.5. The performance of the architecture with complex detectors. In the first column the number n shows number of first type detectors, the m shows number of second type detectors. The second column shows how many outputs we get from this architecture, this number of outputs is then fed into the final ANN. We can compare the results based on the number of those outputs with previous approaches.

The results of the experiment are worse than results from the other approaches. It could be because we do not use enough detectors in each layer, but to compute n=200 and m=200 would create unreasonable huge output of 40000 values which would have to be fed into the final ANN as an output and training ANN with such huge input layer would not be plausible. But more probable is that this approach has some unsolved issues.

We can notice that with more detectors in the first layer the better the result is even when the total size of feature vector is smaller. It could suggest that the first type detectors are good but the second type detectors should be improved. You can compare the third row with the fourth and see that with more detectors in the first layer we can get better results than with more detectors completely. This gives an opportunity for the future work.

In the following chapter we discuss possible drawbacks and problems that could lead to this poor performance.

Chapter 6
Discussion

In this chapter we discuss the results of this work, what we have achieved and what are problems with some of those approaches.

6.1 Success of direct approach

The direct approach proved to have the best results. It could be because of the straightforward nature of the approach. Our goal was not to create a state of the art method, but on the other hand in comparison with the work [5] we have shown significant improvement. The combination of HyperNEAT with Novelty Search is much better than simple HyperNEAT used on the domain of image recognition.

Even though it uses only one level of detectors, the resulting ANN is considered as a deep, because it has 9 layers with ten-thousands of neurons. If we consider the detectors with 4 hidden layers and the final ANN with one hidden layer, where intermediate output of detectors and input layer of final ANN can be considered as a hidden from outside point of view, we get network with 7 hidden layers and one input and one output layer.

6.2 Why Grid based architecture was a failure?

Let us first think about the impact of the grid based architecture. This approach takes dozens of distinct detectors and assigns them a place in the grid. Those detectors were generated for the purpose to be as distinct as possible. But this approach selects only some of them who satisfy the criteria, for example has biggest difference in behavior for the most detected class and the second most detected class. This selection already loses a lot of potential of detectors, due to the fact that the detectors which for example strictly detect numbers 6, 8, 0 and not any other number cannot be selected even that they have a high diverse value. Moreover it assumes that selected detectors are already good classifiers on their own and assign them a position in the grid based mostly on their performance over most recognized class.

After the grid is created it transforms all images into the new image space. The example of this transformation is shown in the figure 6.1. You can notice that if the original image is a well-recognizable it has white region in appropriate row (or row and column in second approach), but it also has a lot of white dots everywhere in the new image. On the other hand when the image is not typical, for example little bit distorted, it makes the outputs noisy and you cannot determine anything about the original class.

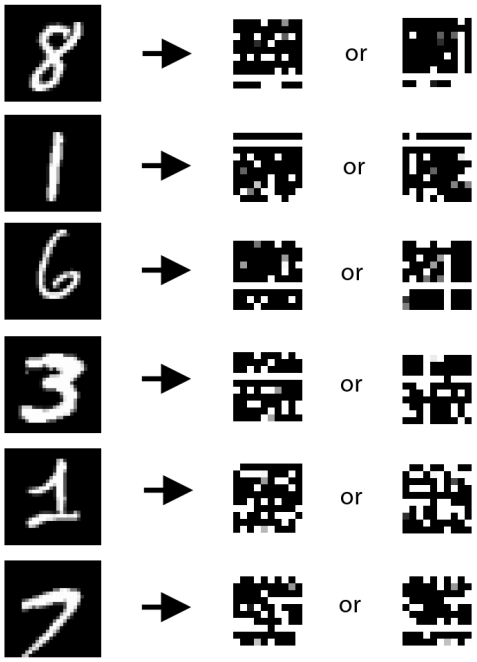


Figure 6.1. The transformation of the image with the grid architecture. First column is original image, second column is row based approach and the third column is the row and column approach. When the number is written well then the transformation has white region in the appropriate place. But when the image looks little bit non-standard then the output is too noisy to be used for recognition.

When we have the transformed images we apply the HyperNEAT from direct approach. It tries to evolve as much distinct detectors as possible with different behaviors over the transformed images. But most of the evolved detectors will be useless. It is because they are trying to find some novel geometric features in the image space, but which except of the dots and the straight lines does not contain any other geometric data.

Since each row of the grid consists of detectors which detect the same class most probably. Then most useful feature would be detector detecting totally straight line on particular row. But then even a simple sum of rows and classification based on the highest value of the sum should has best classification.

As we seen in experiments you can get much better classification with the direct approach which does not lose any information due to the grid. This is most probably the main reason of the failure of this approach and its bigger improved versions. Therefore we would not suggest to advance in this approach any further.

6.3 Problems with complex detectors

The idea of this method is to improve simple detectors used in the direct approach. If we could generate diverse population intermediate detectors we could select the most diverse ones and make the evolution easier for the following levels.

In the experiments with the complex detectors it has been shown that they do not have good results in the comparison with the direct approach. In this section we try to discuss possible causes of this failure.

Initial justification could be that this method is incorrectly defined and that the whole idea is ineffective. Let us investigate the output of the first layer of detectors. In the figure 6.2 you can see the output of the set of 10 diverse detectors. The output is averaged for each class, before it is used to calculate the behavior as was described earlier in this work.

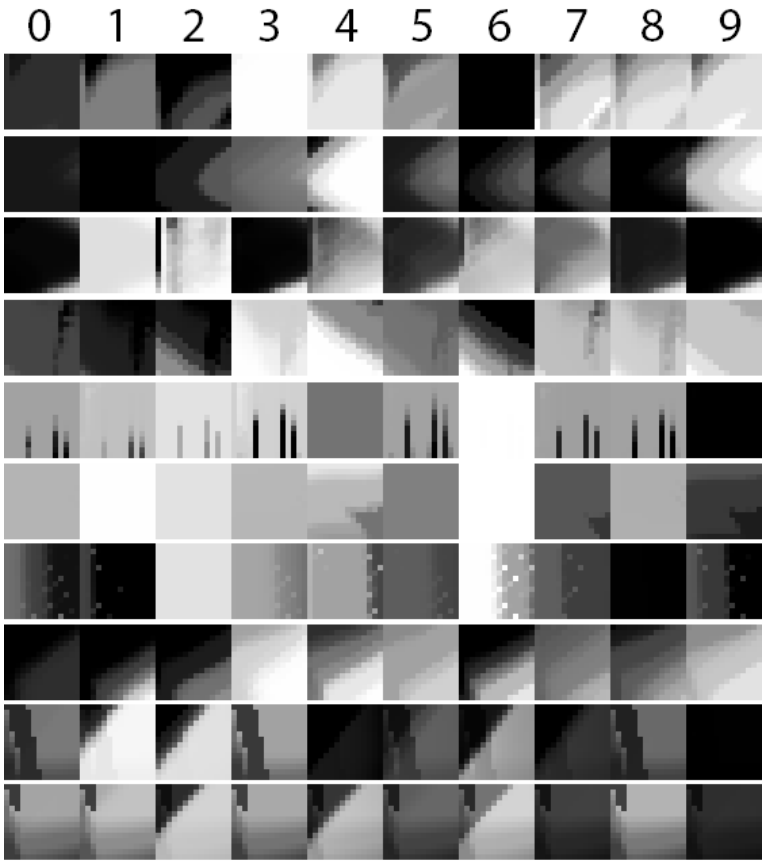


Figure 6.2. Each row of images contain averaged outputs of complex detectors. Each column correspondent to one of the digits.

As was required from the method, the detectors are diverse from each other. It is difficult to answer if they could be even more diverse. During the research most of the evolved detectors looked similar to those presented, they were always less complicated than the original images, i.e. they mostly did not contained any complicated structure, it was usually some transition from white to black. Therefore if we accept that this is as diverse population of detectors as it can be evolved, then we need to investigate second parameter and that is the diversity between classes detected by particular detector.

If we examine the differences between classes (columns in the figure 6.2) detected by one detector, then we can see that the distinction is usually in different hue than different structure. That is huge difference from the initial images, where each class has different structure. That could be a problem since the structure between classes does not change significantly. On the other hand there are differences between some subset of classes and the rest. For example in the first detector in the figure 6.2 the classes 0, 1, 2, 6 could be clearly distinguished from classes 3, 4 and possibly 7, 8, 9. This information could be used by generating different level of detectors after this one.

We have experimented with several behavior definitions of complex detectors and the one presented here has best results, but it does not mean we have found the best behavior metric. Possible improvement of the behavior metric could have an impact on the results. One of the parameters that could be used within the behavior could be the standard deviation in combination with the mean, which is used right now. We try to define a behavior with standard deviation and a mean, but the performance was worse than only the mean.

Another problem could be that the computation of behaviors is too slow and therefore we could not experiment with more hidden layers, longer evolutions or more individuals in the evolution. Note that for each behavior we have to first compute 10 matrices of averaged images over each class and then compare each matrix to each other. Therefore it would be suggested in the future work to develop faster behavior metric and try it on longer evolutions or bigger populations.

There is also a question about the total size of used detectors. Since we used at most 100 detectors in the first level, then comparing it to the direct approach it could be not enough. In the direct approach we have used several hundreds up to thousand detectors. But as mentioned previously it was too time consuming and with the computer I have available it was impossible to test in reasonable time and Metacentrum cluster was not useful for researching, since usually the experiment was started after several days from adding it to the queue which would slow the research.

Compared to the grid based architecture, the complex detectors seem to be interesting method and with possible improvements they could lead to better performance. There is a whole variety of definitions of behavior and detectors structure to be used.

6.4 Complexity of the approaches

Let us discuss the complexity of our approaches with the respect to different number of classes. Mainly the time and space complexity of calculation of behaviors. The complexities of evolution, resilient propagation and etc. are the same as in the original papers. We are considering same size of the original images and the same structure of the detectors.

In the direct approach we have to compute the behavior, which is a vector of length same as the number of classes, in that way it has a linear space complexity based on the number of classes. The time complexity of behavior calculation is linear to the respect for number of images. Since each image is processed by the ANN in a constant time and the output of the ANN is simply added to the behavior, therefore we need n evaluations.

There is no change in the space and time complexity of calculation behaviors in the grid based architecture, since it uses the detectors from the direct approach.

The complexity of the behavior of complex detectors is much different from simple ones. The space complexity to represent the behavior is quadratic with the respect to the number of classes, because we need to store a matrix of differences between all classes. The time complexity for computing the behavior is:

$$O(n + n * c^2) = O(n * c^2)$$

where n is a number of images and c is a number of classes. In the constant time the original image is transformed into the new smaller space and we need to transform n images. When we have all images transformed we need to sum same classes together, this way we get c matrices, where we have to compute difference between each other therefore the c^2. Therefore it is quadratic in the respect to the number of classes.

We have considered only the scalability for different number of classes.

6.5 Confusion matrix

In this section we present the confusion matrix for a direct approach with a resulting error rate of 3.17%. In the table 6.1 you can notice that most problematic is to distinguish between $3 \leftrightarrow 5$ and $4 \leftrightarrow 9$.

	0	1	2	3	4	5	6	7	8	9
0	962	0	4	0	0	1	7	0	4	3
1	1	1123	4	0	1	0	1	3	1	5
2	3	2	998	7	3	7	2	8	2	2
3	2	2	7	971	0	21	1	3	10	9
4	1	0	4	1	956	0	4	4	4	12
5	2	0	1	9	0	850	7	3	6	1
6	3	1	3	0	2	5	930	0	2	2
7	3	0	8	9	2	3	0	996	3	10
8	2	7	2	9	1	3	6	2	940	8
9	1	0	1	4	17	2	0	9	2	957

Table 6.1. The rows represent the classification of the ANN, the columns shows correct classification.

Chapter 7
Summary

7.1 Future Research

In our work we have showed that Novelty Search in combination with HyperNEAT has better performance than HyperNEAT driven by fitness function. It has promising results and further research could improve this approach even more.

One of the possible future improvements would be to code the methods from scratch and significantly optimize it. We did not cared about the time complexity and used frameworks which are not that optimized. This could by also one of the problems why we could not continue with more complicated substrates. For example multi-thread approach and the computing on GPU could improve the time performance significantly and could allow to experiment with more complex networks. Another approach could be to use super-computers or a computer clusters. I have used a Metacentrum cluster, but mainly at the end of research for getting results from experiments, the research was done on a personal laptop, where I usually reached the memory limitations. Moreover the Metacentrum cluster that I have used was only around five times faster than my personal computer. In our case the use of a faster cluster would need the whole application to be coded in that purpose on mind, the use of the frameworks limited the use on the computer cluster to the use of only single processor.

It would be interesting to experiment with the detectors evolved on digits for different domains, for example images of letters. Because the detectors do not care about the classes and only detect some feature in the image. It could have good results for different domains of images. Simply we could recompute behavior of all evolved detectors in the population for new training images and then the rest of the process would be unaltered. The HyperNEAT algorithm even helps this idea because it has an ability to re-size the ANN for different inputs.

Another possible improvement could be to combine detectors with different structure into one classificator. It is problematic to combine them in one population, but if we would run several separated evolutions of detectors, each with different structure, and at the end we would put all evolved detectors into one set. The most diverse subset could have better results than when we use only one structure of detectors. The detectors that have zero hidden layers are better on detecting linear parts in the image, but have problems with detecting nonlinear part. On the other hand detectors with multiple hidden layers can more easily detect nonlinear features, but it is more difficult for them to detect the linear information because of the vast state space of weights settings.

The second approach the grid based architecture is probably useless to research any further, but we believe that the third approach with complex detectors could be further improved. The idea behind using the Novelty Search on many levels of training

is reasonable, maybe we did not proposed correct metric and with better metric all our problems would disappear and the method would have great performance.

7.2 Conclusion

In this work we have investigated ways of searching for image feature from the image space with the use of indirect encoding of neural networks. We have improved its performance by combining it with the novelty search and showed its promising results. Moreover we were able to reach the error rate of 3.53% which is better than result presented in similar work in the paper [5] (2013), where they achieved the best error rate of 7.9%. In our approach we have used Deep Neural Architecture and also experimented with Convolutional Neural Networks on the domain of images from the MNIST database.

The application for experimenting with this structure was coded during the research. It could be used for the purposes of the future works, where we could build on that application more complicated algorithms

Since this was a novel research a lot of time was spent creating and testing our methods. We have got into many blind-spots and had to remake whole sections from the beginning. The approach with complex detectors took a lot of time of researching, since we were trying to make it work, but at the end the direct approach was still better.

The use of Novelty Search in the NeuroEvolution method has shown to have better results than the fitness based approach. Maybe the idea to forget the goal and only evolve most diverse population is better way of finding the best solutions. We have presented several approaches and showed that the Novelty Search could play an important role in the image recognition. We have proposed several ideas for the future improvements and further research.

The Novelty Search is an interesting idea which bring new light on evolutionary algorithms. Sometimes looking for the goal is not the best approach and maybe the evolution in the nature is also not fitness based, but instead it is based on the principle that the novel behaviors lead to the more fit individuals.

References

[1] Kenneth O. Stanley Jason Gauci. Generating Large-Scale Neural Networks Through Discovering Geometric Regularities. *Proceedings of the Genetic and Evolutionary Computation Confrence (GECCO-2007)*. 2007,

[2] Oliver Johan Coleman. *Evolving Neural Networks for Visual Processing*. Master's Thesis, The University of New South Wales, School of Computer Science and Engineering. 2010.

[3] Risto Miikkulainen Matthew Hausknecht, Piyush Khandelwal, and Peter Stone. HyperNEAT-GGP: A HyperNEAT-based Atari General Game Player. *Genetic and Evolutionary Computation Conference*. 2012,

[4] Kenneth O. Stanley Phillip Verbancsics. Constraining Connectivity to Encourage Modularity in HyperNEAT. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2011)*. 2011,

[5] Josh Harguess Phillip Verbancsics. Generative NeuroEvolution for Deep Learning. *CoRR, volume abs/1312.5355*. 2013,

[6] Faustino Gomez Jan Koutnik, Jurgen Schmidhuber. Evolving Deep Unsupervised Convolutional Networks for Vision-Based Reinforcement Learning. *he Genetic and Evolutionary Computation Conference*. 2014,

[7] Faustino Gomez Jan Koutnik, Juurgen Schmidhuber. Online Evolution of Deep Convolutional Network for Vision-Based Reinforcement Learning. *Lecture Notes in Computer Science Volume 8575*. 2014,

[8] Yoshua Bengio Yann LeCun, Leon Bottou, and Patrick Haffner. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE, 86(11):2278-2324*. 1998,

[9] Jean-Philippe Urban Philippe Smagghe, Jean-Luc Buessler. Novelty Detection in image recognition using IRF Neural Networks properties. *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*. 2013,

[10] Jurgen Schmidhuber Dan Ciresan, Ueli Meier. *Multi-column Deep Neural Networks for Image Classification*. . Dalle Molle Institute for Artificial Intelligence.

[11] Heinrich Braun Martin Riedmiller. A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm. *Proceedings of IEEE International Conference on Neural Networks*. 1993,

[12] Kenneth O. Stanley Joel Lehman. Abandoning Objectives: Evolution through the Search for Novelty Alone. *Evolutionary Computation journal, (19):2, pages 189-223, Cambridge, MA: MIT Press*. 2011,

[13] Kenneth O. Stanley Joel Lehman. Novelty Search and the Problem with Objectives. *GENETIC PROGRAMMING THEORY AND PRACTICE IX*. 2011,

[14] Risto Miikkulainen Joel Lehman, Kenneth O. Stanley. Effective Diversity Maintenance in Deceptive Domains. *Proceedings of the Genetic and Evolutionary Computation Conference*. 2013,

[15] Loukas Georgiou Paulo Urbano. Improving Grammatical Evolution in Santa Fe Trail using Novelty Search. *Complex Adaptive Systems*. 2013,

[16] Daniel Toropila Peter Krcah. *Combination of Novelty Search and Fitness-Based Search Applied to Robot Body-Brain Co-Evolution*. . Computer Center, Charles University, Prague.

[17] Kenneth O. Stanley Joel Lehman. Revising the Evolutionary Computation Abstraction: Minimal Criteria Novelty Search. *Proceedings of the Genetic and Evolutionary Computation Conference*. 2010,

[18] Risto Miikkulainen Kenneth O. Stanley. Evolving Neural Networks through Augmenting Topologies. *The MIT Press Journals*. 2002,

[19] Kenneth O. Stanley Jason Gauci. A Case Study on the Critical Role of Geometric Regularity in Machine Learning. *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*. 2008,

[20] Sebastian Risi David B. D'Ambrosio, Joel Lehman, and Kenneth O. Stanley. Evolving Policy Geometry for Scalable Multiagent Learning. *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*. 2010,

[21] Kenneth O. Stanley Jason Gauci. Indirect Encoding of Neural Networks for Scalable Go. *Proceedings of the 11th International Conference on Parallel Problem Solving From Nature*. 2010,

[22] Kenneth O. Stanley Phillip Verbancsics. Evolving Static Representations for Task Transfer. *Journal of Machine Learning Research 11*. 2010,

Appendix A
Parameters of HyperNEAT

Probability of adding a neuron	0.25
Probability of adding a connection	0.8
Probability of removing a connection	0.02
Probability of each weight being modified	0.1
Weight mutation magnitude standard deviation	0.5
Survival rate	0.3
Proportion of reproduction from crossover versus asexual	0.4
Minimum individuals to select as elite from each species	1
Minimum species size to select elites for	0
Maximum stagnant generations before removing species	15
Disjoint genes compatibility factor	2
Excess genes compatibility factor	2
Weight delta compatibility factor	1
Target number of species	population size ^ 0.6
Maximum weight range for substrate	[-1, 1]
Recurrent connections	disallowed
Functions available for CPPN	sigmoid, gaussian, sine, absolute, ramp, linear, sign, multiply, divide, power
Include deltas in CPPN inputs	true
Include angle in CPPN inputs	false
Bias for substrate neurons	true
Connection expression threshold	0.2
Layer connection topology	fully-connected
Activation function of substrate	sigmoid
Average distance to K nearest neighbors in NS	10

Appendix B
Abbreviations

ANN	Artificial Neural Network
CNN	Convolutional neural network
CPPN	Compositional Pattern Producing Network
ČVUT	České Vysoké Učení Technické v Praze
DNN	Deep Neural Network
EA	Evolutionary Algorithms
FEL	Fakulta Elektrotechnická ČVUT
HyperNEAT	Hybercube-based NeuroEvolution of Augmenting Topologies
NE	Neuroevolution
NEAT	NeuroEvolution of Augmenting Topologies
NS	Novelty Search
RNN	Recurrent Neural network
RProp	Resilient Propagation

Appendix C
CD content

The content of the attached CD:

```
root
-Source ... contain all uncompiled source codes and experiment settings
-Thesis ... the text of thesis with its source in the plain TEX
-Experiments ... contain classification from all experiments
--BestDetectorSelectingMethod ... results of different selection methods
--ComparisonWithVerbancsics ... comparison with the Verbancsics (2013)
--ComplexDetectors ... results of experiment with complex detectors
--DifferentStructureDetectors ... experiment with structure of detectors
--GridBasedArchitecture ... results of grid based architecture
--ImportanceOfStore ... store and newly generated detectors comparison
--OnlyResilientANN ... experiment without detectors
```