COMENIUS UNIVERSITY IN BRATISLAVA
FACULTY OF MATHEMATICS PHYSICS AND INFORMATICS

Testing the hierarchical neural network DBN in invariant object recognition

Master's thesis

2012 Bc. Milan Halabuk

COMENIUS UNIVERSITY IN BRATISLAVA
FACULTY OF MATHEMATICS PHYSICS AND INFORMATICS
DEPARTMENT OF APPLIED INFORMATICS

Testing the hierarchical neural network DBN in invariant object recognition

(Master's thesis)

BC. MILAN HALABUK

Study program: Cognitive Science
Study division: 2503 Cognitive Science
Supervisor: doc. Ing. Igor Farkaš, PhD.
Evidence number:

MEi:CogSci Middle European Interdisciplinary Master Programme in Cognitive Science

Bratislava, 2012

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname: Bc. Milan Halabuk
Study programme: Cognitive Science (Single degree study, master II. deg., full time form)
Field of Study: 9.2.11. Cognitive Science
Type of Thesis: Diploma Thesis
Language of Thesis: English

Title: Testing the hierarchical neural network DBN in invariant object recognition

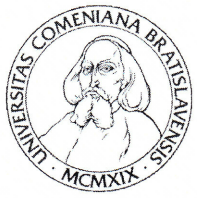
Aim:
1. Study the literature about deep architectures of neural networks with focus on Deep Belief Network model.
2. Experimentally investigate the properties of DBN trained on selected data sets, with respect to invariant recognition of objects, depending on properties of training data and on DBN parameters.
3. Compare the performance of DBN with another selected computational model on the selected data set.

Supervisor: doc. Ing. Igor Farkaš, PhD.
Department: FMFI.KAI - Department of Applied Informatics
Assigned: 09.11.2010

Approved: 14.10.2011 prof. RNDr. Pavol Zlatoš, PhD.
Guarantor of Study Programme

.....
Student

.....
Supervisor



Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Milan Halabuk
Študijný program: kognitívna veda (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: 9.2.11. kognitívna veda
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: anglický

Názov: Testovanie hierarchickej neurónovej siete DBN pri invariantnom rozpoznávaní objektov

Cieľ:
1. Naštudujte si problematiku hlbokých architektúr neurónových sietí, so zameraním sa na model Deep Belief Network.
2. Experimentálne preskúmajte vlastnosti DBN na vybratých dátových množinách, čo sa týka úspešnosti invariantného rozpoznávania objektov, v závislosti od vlastností trénovacích dát a od parametrov modelu DBN.
3. Porovnajte DBN s iným vybraným výpočtovým modelom na zvolených trénovacích dátach.

Vedúci: doc. Ing. Igor Farkaš, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky

Dátum zadania: 09.11.2010

Dátum schválenia: 14.10.2011

prof. RNDr. Pavol Zlatoš, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

DECLARATION

I declare that this thesis was composed by myself, that the work contained here is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

Bratislava . . . . . .

. . . . . .

Milan Halabuk

ACKNOWLEDGEMENTS

Thanks to my friends and parents, for their support, encouragement and help. I also want to thank to my supervisor Igor Farkaš for guidance and support. It has been a pleasure to work with such knowledgeable supervisors.

Abstract

Purpose of this work is to test ability of Deep belief network in object classification problems. This model is deep network with two phase training. The first one is unsupervised pre-training based on stack of Restricted boltzman machines. The second one is fine-tuning which uses back-propagation of error derivates. In this work are used two main datasets for experiments. The first dataset is composed of 11 leaves classes. This dataset includes rotational, color, size and noise variability. The second dataset is composed of normal and abnormal faces. The goal was to find out relationship among different network topologies, dataset parameters and final testing error. I found out that unsupervised pre-training, which is used for weight initialization helps to achieve better classification performance than random weight initialization. Significance of this help depends on type of dataset. In faces dataset this help is bigger (6.8%) than in leaves dataset (1.1%). I also confirm that a higher number of neurons and hidden layers increased classification performance.

Key words

Deep belief networks, object recognition, leaves dataset generator

Abstrakt

Cieľom tejto práce bolo otestovať schopnosti Deeb belief network pri úlohach z klasifikácie objektov. Model Deep belief network pozostáva z dvojfázového tréningu. Prvou fázou je trénovanie bez učiteľa, ktoré je založené na postupnom trénovaní a následnom spájaní boltzmanových strojov. Druhá fáza pozostáva z dolaďovania výkonu pomocou metódy spatného šírenia chyby. V tejto práci sú experimenty vykonávané na dvoch rôznych datasetoch. Prvý pozostáva z 11 tried listov. Tento dataset obsahuje listy, ktoré majú variabilnú rotáciu, veľkosť, jas a úroveň šumu. Druhý dataset je zložený z obrázkov normálnych a potencionálne nebezpečných tvárí. Cieľom bolo nájsť vzťahy medzi topológiou siete, parametrami datasetu a finálnou testovacou chybou. Ukázalo sa, že predtrénovanie, ktoré sa používa na inicializáciu váh pomáha dosiahnuť celkovo lepšie výsledky. Avšak veľkosť tejto pomoci výrazne závisí od zvoleného datasetu. Pri klasifikácii tvári bol prínos pretrénovania 6.8%. V prípade listov to bolo 1.1%. Taktiež sa potvrdilo, že vyšší počet neurónov a skrytých vrstiev má priaznivý vplyv na celkovú úspešnosť klasifikácie.

Kľúčové slová

Deep belief networks, rozpoznávanie objektov, generátor listového datasetu

Table of Contents

List of Figures

List of Tables

1. Introduction

The main target of this work is an object recognition. I choose two dataset to test real classification performance. The first one is leaves dataset, because it poses a nice real world example and it could exhibit typical problems of classification. Input data are highly variable and noisy. This variability includes class (different kinds of leaves) and intra class variability (different leave units from the same class). It also includes variability in size, color and rotation. Leaves datasets are created by my own data-generator, which is described below. The second dataset contains pictures of faces (there are two groups of faces – normal and abnormal). The goals of this project are to find appropriate features to learn, find an appropriate dataset parameters (like size and variability), find out learning parameters for classifier (to achieve sufficient results) and compare it to another method. As a classifier is used Deep Belief Network (DBN) because of good results in image classification [1, 17, 19, 21].

This document describes my work on project in two main steps. At the beginning there is theoretical part with explanation of fundaments. In the second part there are presented achieved results (Source codes and datasets are in attachments).

1.1 Connectionism and neural networks

Neural networks (artificial and non-artificial) are important part of recent cognitive science and are related to connectionist theory. They are very important part of computer based artificial intelligence. Neural network is universal mathematical approach in study and modeling of learning process, adaptation process and artificial cognitive systems. Whole concept of interconnected simple units is based on metaphor of human brain. They are biologically motivated mechanisms of knowledge acvisition and learning (applicable on different levels of abstraction) [23, 24]. Many of connectionists think that brain executes computations and that neural computing explain human cognition [24, 25]. In general, we suppose that we can use neural networks to explain mental processes. Connectionism in artificial intelligence and cognitive science is consider like a process of parallel information processing. Artificial neural networks have important role in cognitive science, linguistic [14], neuroscience [26], controlling of different processes of natural and social

science. In these wide spectrum of possible applications neural networks are not used only for modeling learning and adaptation process. They are also used for solving wide spectrum of different tasks and problems like object classification [17, 21], speech recognition [14], financial forecasting [27] and navigation [19]. But one of the main purpose of studying neural networks is their relation to human brain. In cognitive science and neuroscience neural networks are part of basic theoretical methods that model activity of our brain. In these two scientific disciplines are created basic connectionistic principles and is shown plausibility of neural networks for modeling different kinds of activities and aspects of human brain. One of the main purpose of studying artificial neural networks is finding relation between implemented mechanisms (interaction between neurons) and cognitive phenomena [26]. Connectionism represents important knowledge base which is able to interpret and explain different cognitive activities of human brain. This connectionistic representation of human brain is plausible with our knowledge about brain structure and it is supported by information about brain physiology.

1.2 Symbolic and sub-symbolic representation

In this part is explained difference between symbolic and sub-symbolic approach in studying of cognitive processes. In symbolic approach the central idea is a concept of symbol. The symbols are transformed to another symbols by using concrete hierarchical set of rules. Symbolic point of view is often consider like algorithmic. Many experts consider this approach as method with right level of abstraction (mostly in context of higher cognitive processes like reasoning, planing and usage of language). The main advantage is usage of mathematical language and therefore it is easy to read for human. Sub-symbolic approach is based on connectionism. The main concept is based on idea, that these processes, which occurs in neural network, include many of interconnected neurons. Neurons are elementary processing units and weighted connections between them represent long term memory. One of drawbacks of neural networks as non linear transformation system is that operations inside are not transparent and they resemble black box. Parameters of network – weights are not explicitly set. It is practically impossible. So it is necessary to provide suitable learning mechanism, which can network use for adjusting weights. Learning process is similar to human learning – it is based on examples.

During learning process weights are changed in appropriate way. Result of this process is requested behavior of neural network. In the connectionistic approach an information is represented by individual neurons, which are interconnected and form the network. The important aspect in representation of information is recent impulse, which is spread thought neural network. This impulse is essential in decision process (to discover active neurons). Spreading of this impulse is realized be simple calculations which are provided by each neuron.

2. Basic properties of neural system

A neuron cell is the basic building block of the nervous system (NS). The main difference between neuron and normal cell is that neurons are specializes to transmit information inside the body. Neurons are highly specialized cells and are responsible for processing of input information and communicating (sending) output information to next neuron or to specific organ (e.g. Muscle). Communication between neurons is based on electrochemical interaction. For purpose of this work and further explanation of artificial neural networks are important these key properties of typical neuron:

- **receiving** signals from other connected neurons
- **processing** of received signal
- **sending** processed signal in form of action potential to another interconnected neurons

2.1 Model of artificial neuron

Artificial neuron is a mathematical model, which is based on abstraction of biological neuron. Typically it is simplified simulation of biological neuron. There are many different models of neurons. These models differs in complexity. From simple models, which use simple discrete activation functions to models which are trying to model complex processes inside biological neural cell. One of the most common models of neuron is the model introduced by McCulloch and Pitts.

$$Y = Net\left(\sum_{i=1}^{N}\left(w_i x_i\right) + w_b\right) \qquad (2.1)$$

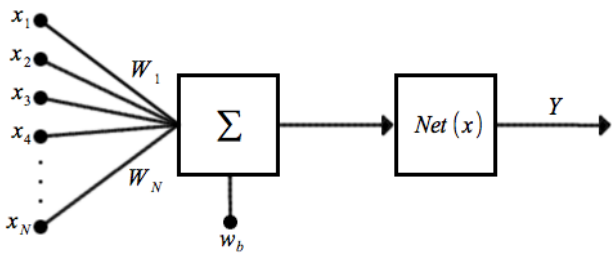


Figure 1: Visualization of Formula 1

In Figure 1 is presented how output of neuron is calculated. The whole process is mathematically described in Formula 2.1. x_i represents input of neuron (typical neuron has many inputs). w_i represents weights, which are like memory of neuron. In this memory is usually saved last experience of concrete neuron (it is like sensitivity for exact connection between two neural units). Higher weight also mean that this input is more important than input with lower weight. These weights influence each input vector so they actually influence all neural network. Special kind of weight is w_b . It is bias of each neuron. In biological neuron is bias also called threshold (for activation of neuron). That means if sum of all inputs (without bias) is less that threshold than the neuron stays inactive (passive state). $Net(x)$ is activation function of neuron and this function computes output of whole neuron. Result of whole process is typically send to another neuron for further processing and it is called Y .

2.1.1 Artificial neuron types

Based on type of output data we can divide neurons into 2 groups – **discrete** and **continuos**. Discrete usually support 2 types of output value. Typical discrete neuron is binary neuron (output values 0 and 1 or -1 and 1). Discrete activation function and visualization of this function is shown in Formula 2.2 and Figure 2. Continuos neurons have real number as an output. Standard continuos neuron uses sigmoid activation function (Formula 2.3, Figure 3).

2.1.2 Types of activation function

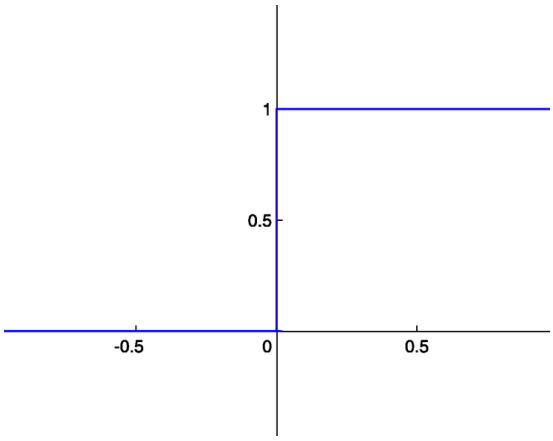


Figure 2: Step activation function

$$u=\sum_{i=1}^{n} w_i x_i \qquad y=\begin{cases} 1 & if\ u \geq \theta \\ 0 & if\ u < \theta \end{cases} \qquad (2.2)$$

Formula 2.2 represents step activation function. For input greater than threshold θ returns 1. For input smaller than threshold returns 0.

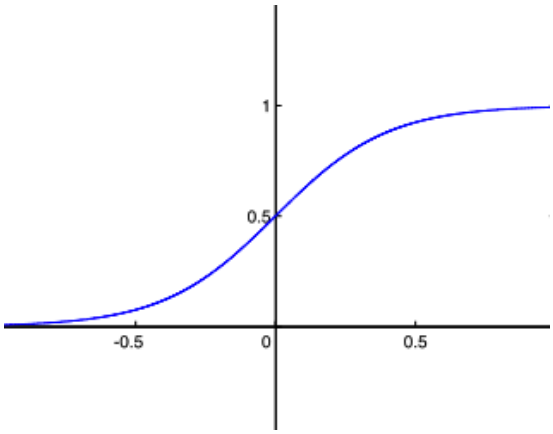


Figure 3: Sigmoidal activation function

$$f(x)=\frac{1}{1+e^{-kx}} \qquad (2.3)$$

Formula 2.3 represents sigmoidal activation function. For x close to infinity it returns 1. For x close to negative infinity it returns 0. When x = 0 then output is 0.5.

2.2 Artificial neural network (ANN)

Typically ANN is an adaptive system that is changing own structure based on information that is going thought the network during learning process. Artificial neural networks are tools for modeling non-linear statistical data. Usually are used for modeling complex relationships between input and output data vectors (finding patterns in data).

The word network referees to interconnection between neurons in different layers. Example of system with 3 layers is shown in Figure 4. This example has 3 layers of neurons. In the first layer are input neurons. These neurons represent input vector. Input neurons send data to second layer which is usually called hidden layer. At the end the signal from hidden layer is send to final or output layer. As is mentioned in section 2.1. the connections are weighted. The number of neurons may vary a lot. More complex systems have more layers of neurons.

2.2.1 Main properties of typical ANN:

- Interconnection pattern (it defines which neurons are interconnected)
- Type of learning procedure (defines the type of learning process that is used for updating the weights)
- Type of activation function (defines the basic behavior of neuron)

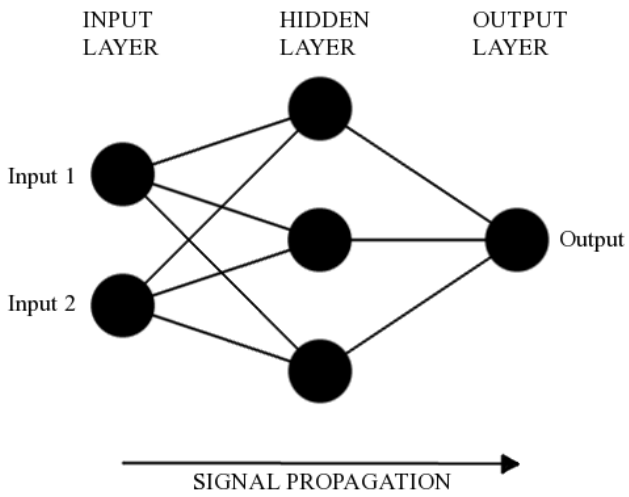


Figure 4: Example of interconnected neurons in multilayer neural network (2 input units, 3 hidden units, 1 output unit)

2.2.2 Learning of ANN

The main goal in learning process is to set up neural network in a way that will produce correct output (result). In biological networks is the experience of concrete neuron saved in dendrites. In ANN the previous experiences are saved in connection between neurons – in the weights. There are two main groups of learning methods (methods for changing weights).

First is supervised learning. In this type of learning the exact output value for each input vector is known. When real network output is different than expected – the weights are changed. Purpose of this change is to lower the difference between real and expected output. For each type of learning exists error function (it computes current error), that is minimized during process of learning (for detailed explanation of supervised learning and error function please refer to section 2.3.3.2). This process of changing weights is repeated until the error function result is not sufficient.

Another type of changing weights is unsupervised learning. During this process the network has not access to correct output results. ANN has to find out which input vector is connected to concrete output values. Typically these networks have to model similarities of training data and merge similar inputs into groups.

2.2.3 Multilayer perceptron

Typical example of artificial neural network is a multilayer perceptron (MLP). It is feedforward network. It means that signal is spread only in one direction. The signal flow is from input neurons to output neurons. MLP consists of multiple layers of neurons and each layer is fully connected to the next one. Fully connected means that for each neurons in first layer exist exact the same number of connections as a number of neurons in second layer (example in Figure 4). Each neuron uses nonlinear activation function (typical sigmoidal function). MLP uses supervised learning method called back-propagation.

Back-propagation training method

In the next chapter is described principle of the back-propagation learning rule. This learning algorithm is the essence of fine-tuning – training phase of deep belief network,

which is used in my experiments. The back-propagation algorithm consists of two main parts – forward pass and backward pass.

Forward pass – computes output of the network. It means that for one input vector the output of the each neuron in each layer is computed. This process is done one layer after another, starting from the lowest layer which uses the data vector as input. At the end the output of the network is then compared to the expected output vector. The final error can be represented as function of difference between desire output and real output (for more details see Formula 2.4).

$$E = \sum_{i=1}^{n} \frac{1}{2}(t_i - o_i)^2 \qquad (2.4)$$

Formula 2.4 represents mean square error of overall network (**n** is number of output neurons, **t** is desire output, **o** is current output of the network).

Backward pass – Derivatives of final error are propagated backwards through the weights. It means that error is computed for each neuron separately based on weights and error of previous neuron. In Formula 2.5 is shown how to compute error for neurons in final layer. Than error is propagated backwards through net and individual errors for all hidden neurons are computed. Computation of error for neurons in hidden layer is shown in Formula 2.6.

$$\delta_k = o_k(1 - o_k)(t_k - o_k) \qquad (2.5)$$

Formula 2.5 represents error on output neuron (**k** is number of current neuron, **δ** is error, **o** is current output, **t** is desire output)

$$\delta_h = o_h(1 - o_h) \sum_{k \in Downstream(h)} w_{kh}\delta_k \qquad (2.6)$$

Formula 2.6 is error on hidden neuron (**w** is the weight matrix, **δ** is error, **h** is number of current neuron, **k** is number of neuron from previous layer, **δ** is error, **o** is current output)

After computing all errors the weights can be changed.

$$\Delta w_{ji} = \eta \delta_j x_{ji} \qquad (2.7)$$

Formula 2.7 represents final change of weight between neurons **i** and **j**. It is end of one learning cycle. This act of change is actually the essence of learning procedure. The whole process continue until the overall net error (Formula 2.4) is on sufficient level or is not another stoping criterion called.

2.3 Introduction to deep architectures

Important feature of ANN is possibility to add multiple hidden layers. In many case it is sufficient to use only one hidden layer, but the same function can be represented in a much more compact way with a deeper net [30]. In the deep networks neurons form progressive and more complicated feature detectors. It has been proven that its better to do the classification with such deeper net [31]. *"It would be worthwhile to explore learning algorithms for deep architectures, which might be able to represent some functions otherwise not efficiently representable. Where simpler and shallower architectures fail to efficiently represent (and hence to learn) a task of interest, we can hope for learning algorithms that could set the parameters of a deep architecture for this task."* [30] Schmidhuber with his team claim that their Multicolumn deep neural network is in traffic sigh recognition benchmark better than human [35]. Deep nets architecture were inspirited by visual systems of mammals [32]. There are comparison of sparse DBN output to the V2 area of the visual cortex [13].

In next chapter is described Deep belief network (DBN) as example of stochastic deep architecture. The main motivation for creating such networks was poor performance of classical back propagation algorithm in nets with more layers [31]. Main problem of

back-propagation is initialization of weights, which cause tendency to get stuck in poor local optimum [31]. In the recent years, the way of training deep models is improved by using an unsupervised learning algorithm that is used by generative models called deep belief networks (DBN) one layer at a time [2].

2.4 Stochastic models

Now we take a look on stochastic models of neural nets. Especially we will describe DBN as a classical example of such network. First we need to describe basic concepts of probabilistic neuron, which is base of the network. Then we will describe Restricted Boltzmann machine (RBM) [33], which is composed of probabilistic neurons and represent one layer of DBN.

2.4.1 Model of stochastic neuron

Probabilistic neurons typically have a state of 1 or 0. The probability of turning on is determined by the weighted input from other neurons. Illustration of sigmoidal function in Figure 5 represents probability of firing of neuron s_i .

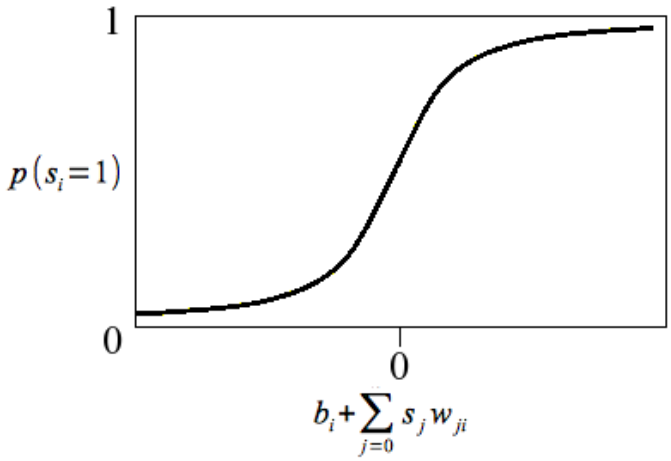


Figure 5: Activation of probabilistic neuron. (X axle represents sum of all inputs, Y represents probability of firing)

2.4.2 Restricted Boltzmann Machines

A Restricted Boltzmann Machine (RBM) is a form of artificial neural network. RBM is undirected graph, which is formed by one visible and hidden layer. Inside are symmetrically connected units that make stochastic (also called not deterministic or probabilistic) decisions about being on or off [34]. All units are fully connected to the units in the next layer. Difference between Boltzman machine and restricted Boltzman machine is in absence of visible-to-visible or hidden-to-hidden connections. Illustration of typical RBM is in Figure 6. Inventor of RBM was Smolensky (1986) [33]. RBM are base building blocks of more complex classification system like Deep Belief Network. In the next section are described fundaments of RBMs like sampling and training process.

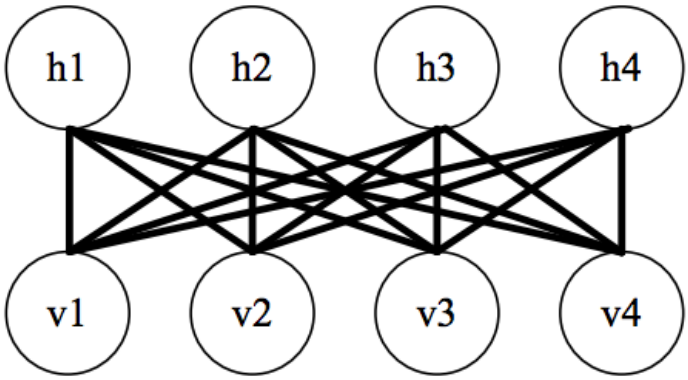


Figure 6: Illustration of bidirectional connection between hidden (h) and visible (v) units in Restricted Boltzman machine

The goal of RBM is to generate input vectors with high probabilities. This means that we want to learn network base features of input vectors and connect this features to input data (image).

There are **2 main phases** – data and reconstruction (Figure 7). At the beginning signal (e.g. an image) is initialized to input vector (time $t=0$). Then signal is spread to hidden layer. Neurons in Hidden layer are also called feature detectors. This spread of signal is special form of sampling, which is explained in formula 2.8. This formula

represents probability of firing of j–th neuron in layer h. Base on this probabilities are computed (sampled) output values (for more details check pseudocode below) for hidden and visible unit in data and reconstruction phase.

$$P(h_j=1|v)=\frac{1}{1+\exp(-\sum_{i=1}^{n}(w_{ij}v_i)-b_j)} \qquad (2.8)$$

Formula 2.8 is probability of firing for j–th neuron in hidden layer (n – number of inputs neurons, v – array of visible neurons, b – bias of hidden neuron, w – weight matrix).

After data phase there is second phase (time $t=1$), which is called reconstruction or phantasy. Signal is spread back to visible units and than goes back to hidden (Figure 7). Well trained RBM generates same data in both phases. Data and Reconstruction are important for changing of weights in learning process. Let see how it works.

Learning of RBM

The key element in learning process of a RBM is changing the weights. Result of this change is that the model gets higher probability of generating the requested data. This process could consist of multiple iteration of parallel updating visible and hidden neurons. However there is a much efficient way to train RBM. Hinton [34] discovered that learning works good only with use one step of reconstruction. He called this method **contrastive divergence**. This method is described in pseudocode 1-2 and Formulas 2.9-11. Key feature of this method is that only one reconstruction of model is needed.

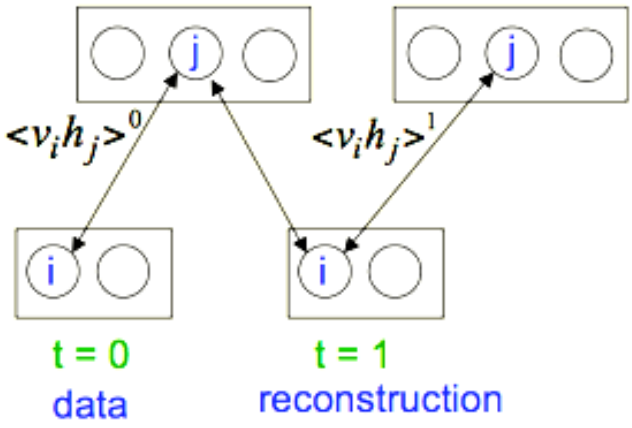


Figure 7: RBM (hidden and visible layers and connection between them)[4]

$$\Delta w_{ij} = \alpha\left(\langle v_i^0 h_j^0\rangle - \langle v_i^1 h_j^1\rangle\right) \qquad (2.9)$$

Formula 2.9 represents changing of weights during learning process.

$$\Delta b_j = \alpha\left(\langle h_j^0\rangle - \langle h_j^1\rangle\right) \qquad (2.10)$$

Formula 2.10 represents changing of bias for hidden units.

$$\Delta c_i = \alpha\left(\langle v_i^0\rangle - \langle v_i^1\rangle\right) \qquad (2.11)$$

Formula 2.11 represents changing of bias for visible units.

In Formula 2.9 the $\langle v_i^0 h_j^0\rangle$ represents the number of cases when i-th visible neuron and j-th hidden neuron fire together (for input vector v). As was mention before in contrastive divergence algorithm, sampling is typically done only once for each input. The model is driven by a reconstruction of the data [31, 34]. In other words the weight between visible and hidden units is strengthened if booth units are on. If both units are on during reconstruction phase the weights are weakened. Because of this preference of generating data instead of reconstruction the RBM learn to generate data instead of reconstruction. If

the data and reconstruction are identical, the weights are not changed. Similar procedure is applied on biases b and c, which are presented in Formula 2.10 and 2.11. Change of bias on j-th hidden neuron is represented as difference of activation of hidden neuron j during data and reconstruction phase. Analogically change of bias on i-th visible neuron is represented as difference of activation of visible neuron i during both phases.

Pseudocode 1: Example of update method in RBM [30, 36]

```
RBMupdate(v[0], alpha, W, b, c):
for all hidden units i:
  compute Q(h[0][i] = 1 | v[0])
# Formula 2.8 - sigmoid(b[i] + sum_j(W[i][j] * v[0][j]))
  sample h[0][i] from Q(h[0][i] = 1 | v[0])

for all visible units j:
  compute P(v[1][j] = 1 | h[0])
# Formula 2.8 - sigmoid(c[j] + sum_i(W[i][j] * h[0][i]))
  sample v[1][j] from P(v[1][j] = 1 | h[0])

for all hidden units i:
  compute Q(h[1][i] = 1 | v[1])
# Formula 2.8 - sigmoid(b[i] + \sum_j(W[i][j] * v[1][j]))
  sample h[1][i] from Q(h[1][i] = 1 | v[1])

W += alpha * (h[0] * v[0]' - h[1] * v[1]')
b += alpha * (h[0] - h[1])
c += alpha * (v[0] - v[1])
```

- **v[0]** is a sample from the training distribution for the RBM (v[x]' represent transposed matrix)
- **alpha** is a learning rate for the stochastic gradient descent in Contrastive Divergence
- **W** is the weight matrix
- **b** is the RBM biases vector for hidden units
- **c** is the RBM biases vector for input units

Pseudocode 1 shows example of training (updating weights W and biases b, c). In the first step are computed probabilities of firing of i-th neuron in hidden layer for input vector v[0]. These probabilities are computed for each unit in hidden layer (Formula 2.8). Base on this probabilities is created sample vector h[0] (one possible distribution of activations). Analogically are computed probabilities and samples for reconstruction of visible units v[1] and for hidden units h[1]. At the bottom of pseudocode is described change of weights (Formula 2.9-11).

Stacking of RBMs

After training of one layer of RBM this procedure can be applied again with additional layers. This is done by adding one layer on the top of the network. This method of stacking RBMs is called layer-wise [5]. As you can see in the Figure 8, newly added layer hd become new hidden layer and old(previous) hidden layer h1 produces inputs for it. Weights between previous hidden and visible layers are locked. This procedure can be repeated as many times as is needed (it depends on how many layers final network should contain). *"It can be proved that each time we add another layer of features we improve a variational lower bound on the log probability of the training data"* [29]. Adding more layers improve probability of generating training (input) data [30]. For more detailed explanation of stacking RBMs together refer to Pseudocode 2.

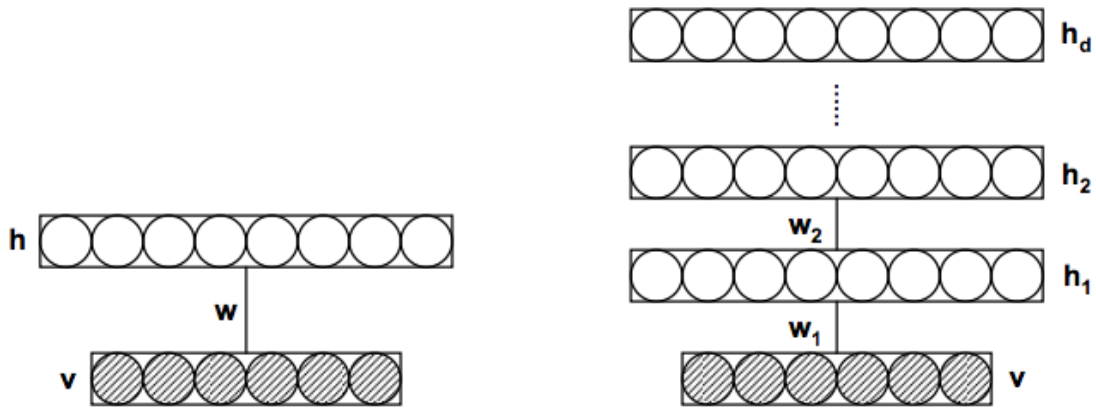


Figure 8: Stack of RBMs [14]

Pseudocode 2: Pre-training of DBN [30, 36]

```
PreTrainUnsupervisedDBN(X, alpha, L, n, W, b):
  initialize b[0]=0
  for l=1 to L:
    initialize W[i]=0, b[i]=0
    while not stopping criterion:
      sample g[0]=x from X
      for i=1 to l-1:
        sample g[i] from Q(g[i]|g[i-1])
      RBMupdate(g[l-1], alpha, W[l], b[l], b[l-1])
```

- **X** is the input (training) distribution for the network
- **alpha** is a learning rate for the stochastic gradient descent in Contrastive Divergence
- **L** is the number of layers to train
- **n**=(n[1], ...,n[L]) is the number of hidden units in each layer
- **W[i]** is the weight matrix for level i, for i from 1 to L
- **b[i]** is the bias vector for level i, for i from 0 to L

Pseudocode 2 illustrates the process of unsupervised pre-training of Deep Belief Network. The main part of the pseudocode is loop. Each iteration of this loop creates and train one layer of the network. Inside this loop is initialization of weights and biases (there is unique weight matrix for each layer). Another important matrix is g[l] which represents sampled outputs of network for layer l. The last part of the pseudocode is function ***RBMupdate*** (explained in previous section) which provide weight changes for the final layer.

2.5 Deep Belief Network (DBN)

*"**Deep belief nets** are probabilistic generative models that are composed of multiple layers of stochastic, latent variables. The latent variables typically have binary values and are often called hidden units or feature detectors."* [2]. DBN can be viewed as composition of restricted type of Boltzmann machine (learning module) and is useful for representation of high dimensional and complicate data. It is trained in a greedy layer-wise fashion (generative model with many layers of hidden causal variables). The common way how to use Deep Neural network for image classification is with cooperation of unsupervised pertaining and supervised fine-tuning. The main fields of use are dimensionality reduction and object recognition. Learning time of MLP does not scale well (problem with effective training of multiple hidden layers). Stack of Restricted boltzman machines is used for initialization of weights in network. This process of initialization is called pre-training. Each layer of the network tries to model the distribution of its input, using unsupervised training [3]. Second phase (after pre-training) is called fine-tuning. In this phase a final layer of neuron is added (each output neuron represent one category).

2.5.1 Unsupervised pre-training

Each layer of the network tries to model the distribution of its input, using unsupervised training in a Restricted Boltzmann Machine (RBM). The unsupervised greedy layer-wise training serves as **initialization**, replacing the traditional random initialization of multi-layer networks [2].

2.5.2 Fine-tuning (supervised training)

After a DBN has been initialized by pre-training (RBM), this procedure of fine-tuning will **optimize** all the parameters (weights). It can be performed by adding a final layer of variables that represent the desired outputs and **back-propagating error derivatives**. *"When networks with many hidden layers are applied to highly-structured input data, such as images, back-propagation works much better if the feature detectors in the hidden layers are initialized by learning a deep belief net that models the structure in the input data"*[2]. In other words the whole network is trained like MLP with back-propagation learning method [4, 5].

2.5.3 Advantages of unsupervised pre-training

In the next section is investigated why does unsupervised pre-training works successfully. *"Searching the parameter space of deep architectures is a difficult task because the training criterion is non-convex and involves many local minima. This was clearly demonstrated in recent empirical work [9] showing consistently that with hundreds of different random initializations, gradient descent converged each time to a different apparent local minimum, with solutions obtained from random initialization and purely supervised training consistently getting worse for architectures with more than 2 or 3 levels. This points to why, until recently, deep architectures have received little attention in the machine learning literature." [8]* In the other words we can look at pre-training like on regulation mechanism, that minimize variability and bias weight values towards such configuration of parameter space that is better for classification tasks (higher performance in classification).

Features of unsupervised pre-training:

- **Better generalization ability**.

  Results from various range of experiments with object recognition datasets (dataset examples size from 10000 – 50000) shows that for the same topology of network the test error is lower when unsupervised pre-training is used [8, 9, 10, 11, 12].

- **Smaller difference in test performance.**

  The difference in final test error between different test runs on the same dataset is lower when unsupervised pre-training is applied. This impact is bigger when more hidden layers are used. [8, 9]

- **Scaleability of network.**

  The advantage of using pre-training is visible when network topology is robust (many neurons in layer). When the network size is constrained to small size the performance is decreased (in some cases is even worse than without pre-training) [8, 9]

2.5.4 Application of DBN

In this section are described examples of implementations of Deep Neural Nets. This type of networks were invented in 2006 by G. Hinton and still are not very common. There is range of possible applications in generating, recognizing images [15, 17, 19], video sequences [16] and voice recognition [20]. Through cognitive science perspective there are interesting application for post-processing EEG signals [21].

Application in Image processing.

Raia Hadsell [19] with her group presented example of deep learning for long range vision application. They tried to classify complex terrain features from robot position up to the horizon of the area where robot was positioned. This classification of important features allow high-level strategy to be applied. Deep belief network was trained to learn

informative and important features of the input image. The main goal of this experiment was to predict traversability of concrete terrain in a realtime. They achieved impressive results.*"The classifier is able to see smoothly and accurately to the horizon, identifying trees, paths, man-made obstacles, and ground at distances far beyond the 10 meters afforded by the stereo supervisor." [19]*

Another example of DBN application is classification of 3D object presented by Nair and Hinton. [17] They used NORB database for evaluation of performance. This database contains of stereo paired images of different 3D object. There is variation in lightning conditions and viewports. Images are stereoscopic (2 pictures for each sample). Their model achieved 6,5% error rate on test dataset. This is good result compared to best published (5,9%). They also compared it to SVM which scores 11,6%.

Alex Krizhevsky [21] provided interesting experiment. He used convolutional DBN for classifying 1,6 million tiny images from CIFAR-10 dataset. This dataset contain of various images of objects from 10 categories. His best test classification success was 78,9% (The best published result was achieved with Multi-column Deep Neural Network [22] with 88,79% successfully classified images).

Application in speech recognition

A. Mohamed and his team experiment with speech recognition by using DBN. They showed that *"The DBNs learned using the sequence-based training criterion outperform those with frame-based criterion using both three-layer and six-layer models, but the optimization procedure for the deeper DBN is more difficult for the former criterion."*[20]

Application in EEG signals filtering

D. Wulsin and his team applied DBN to classification and abnormally data detection from EEG (electroencephalography). They found that DBN performance was comparable to another standard classifiers and classification was 1.7 – 103.7 times faster than comparable classifiers. *"These results indicate that DBNs and raw data inputs may be more effective for online automated EEG waveform recognition than other common techniques."* [21]

Leaves recognition application

One part of my experimental setup is test performance of DBN on my own leaves dataset. There are few projects in leaves recognition [6, 7]. But these project do not use DBN as classifier and use different approach in dataset creation (no huge variability and no noise). Also there is no widely used dataset, so it is hard to compare performance between different approaches (it is not good to compare results based on different datasets).

3 Experimental part

3.1 Motivation and goals

In this section are described experiments with DBN classifier. I used two different dataset classes (leaves and faces). At the beginning is described process of generating leaves. Then are presented experiments and comparisons between different setups of Deep network. This work is focused on how unsupervised pertaining helps to improve error rate of whole classifying system. I wanted to find out relation between number of neurons and layers (topology) and its influence on performance.

3.2 Creation of leaves dataset

I have decided to create my own dataset for testing because there is an absence of complex configurable leaves dataset, which is suitable for testing the performance of classifiers. There are popular datasets like CIFAR-10, MNIST or NORB, but these do not include leaves. I consider it like good example for testing classification performance of artificial neural networks. I wanted to test my ability for handle complex process from creation of dataset to successful classification. Leaves are nice example for possible application in real world environment (e. g. application for mobile phones). Another and main motivation for creating own dataset generator was ability to generate number of different kinds of training sets and find interesting relations between features of datasets and features of networks (e. g. how noise level or number of samples influence test performance). In next section is described how was this dataset generator created and how to use it. Generator is accessible for public use at the web page: http://dbn.meshmatrix.com .

3.2.1 Leaves gathering

For purpose of my work I have decided to collect leaves from my home town location Levice, Slovakia. The location is marked on Figure 9 with A sign.

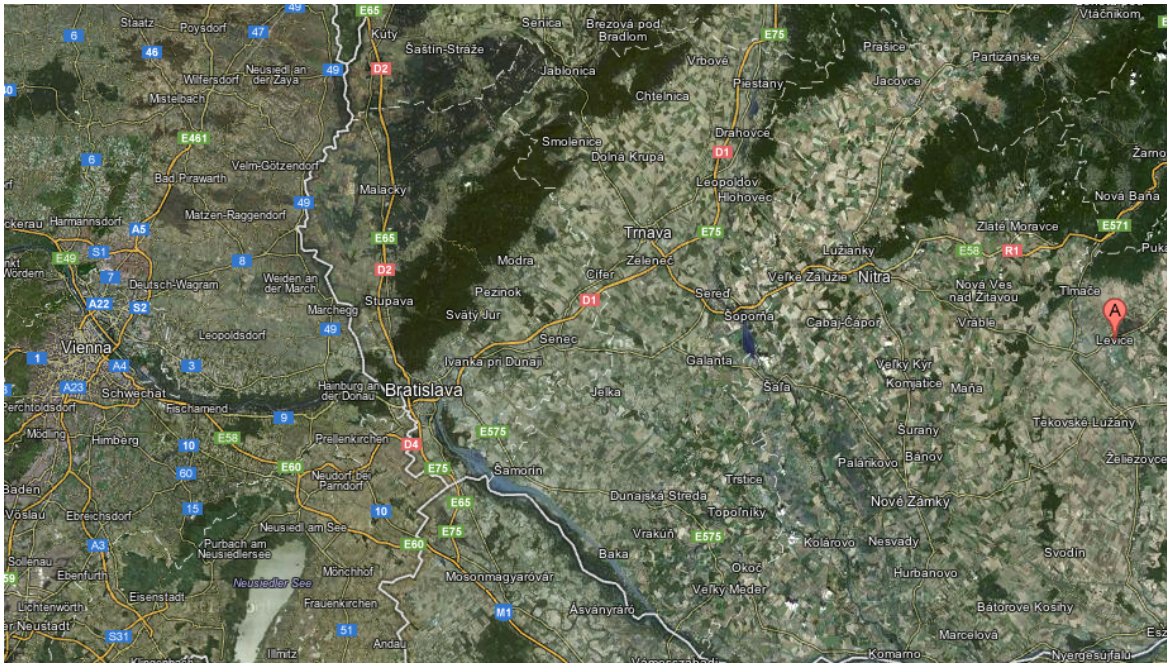


Figure 9: A sight represent Levice, Slovakia (GPS coordinates: 48.214526,18.606523) [google maps]

3.2.2 Photo method

For purpose of taking pictures has been used Sony HX-1 camera with tripod. I used 3 lights placed around the scene, because there was a demand to prevent shadows, which may cause problems for further processing. I used long exposures (1/15 of second) for brighter output pictures. For detailed information of camera setup please look at EXIF info of each picture (in attachment). Some of collected leaves were creased so I tried to flatten them by putting them into a book for 2 weeks. Finally I have created 2 databases of leaves images – with flattering and without. I have also some problems because some leaves change color from green to yellow or brown during this process. This was the reason why I was unable to use all leaves in the experiments

3.2.3 Leaves database example

I have collected more than 200 samples of 17 kinds of leaves. The example of these leaves is presented in table 1. It is example from non flattered picture dataset. For full leaves database with original source photos is in attachment.

 Apple	 Apricot	 Sweet Chestnut
 Field Maple	 Hornbeam	 Platonoides Maple
 Red Maple	 Oak	 Pear
 Cottonwood	 Willow	

Table 1: Example of raw leaves pictures

These images are essence for dataset generation. For further processing I chose 11 tree categories with best picture quality (e.g. walnut picture was bad, because of yellow color). Also key aspect why there was decision to reduce number of categories was insufficient quantity of some samples (I have decided to work with trees with 5 or more sufficient photos). Chosen tree categories are presented in Table 1.

3.2.4 Texture creation

In this process the raw picture from camera is transformed into a form suitable for dataset generator program. Main purpose of this process is to create a texture, which is used in next process. Texture is image of leaf which is placed into generator and transformed into

output for training. As you can see in Figures 10, 11, 12 and 13 (Table 2) this process contains of next steps:

- Cut leaf from base image
- Convert to black and white
- Normalize color
- Invert colors
- Increase contrast value
- Resize into 64 x 64 pixels.

Whole process was done in GIMP picture editor.

<table>
<tr>
<td> 
Figure 10: Raw image</td>
<td> 
Figure 11: Covered to black and white + colors are normalizes</td>
</tr>
<tr>
<td> 
Figure 12: Inverted color and increased contrast</td>
<td> 
Figure 13: Resized to 64x64 pixels</td>
</tr>
</table>

Table 2: Creation of final texture

3.2.6 Implementation of dataset generator

Dataset generator is created in javascript by using webGL for 3D graphics processing. I choose HTML5 and javascript because of multi-platform usage and for possibility to place this application on webpage. It is implemented base on http://learningwebgl.com/ source codes. For more information about implementation please look into source code (in Attachment). For running this web application you need OpenGL 2.0 compatible graphic card and also compatible web browser. I recommend Chrome or Firefox.

3.2.7 Functionality of dataset generator

In Figure 14 you can see top part of application interface. There are two main parts. Panel on the left side is collection of leaves where you can select leaves which will be used in generation process. Panel on the right side is for setting variables which influence final shape of output dataset. There are 4 main parts:

Rotation settings

In rotation settings you can choose variability level of rotations in all axles. X and Y axle simulates different view angle and are calculated like random number between zero and selected value. Z axle rotation is calculated like: selected value + random between 0 and 1. This small randomness in calculation is to prevent rotational periodicity in output data. As you can see, there is factor of randomness so when you run same setup 2 times there will be different outputs. My recommendation is to create datasets with more than thousand samples to cover more cases of state space (combination of randomness and many samples should lead to almost uniform distribution of different states). Z rotation variable influence size of output dataset (if value is lower than the size of dataset is bigger).

Size settings

Another setting area is called Size. Here user can set inner constants that represent start and end distance of object to camera. Step setting represents the decrease in distance after turning an object in 360 degrees (during generation process). By setting this variablel user

influence the size of output dataset (if there is smaller step size or difference between start and end size is higher → more samples will be generated)

Noise settings

Another setting area is called Noise. Here user can select color range for noise pixel, level of noise and also he can decide if noise will be applied only in background or in whole image.

Brightness settings

The last setting area is called Brightness. In this area user can choose the variability in brightness of the texture.

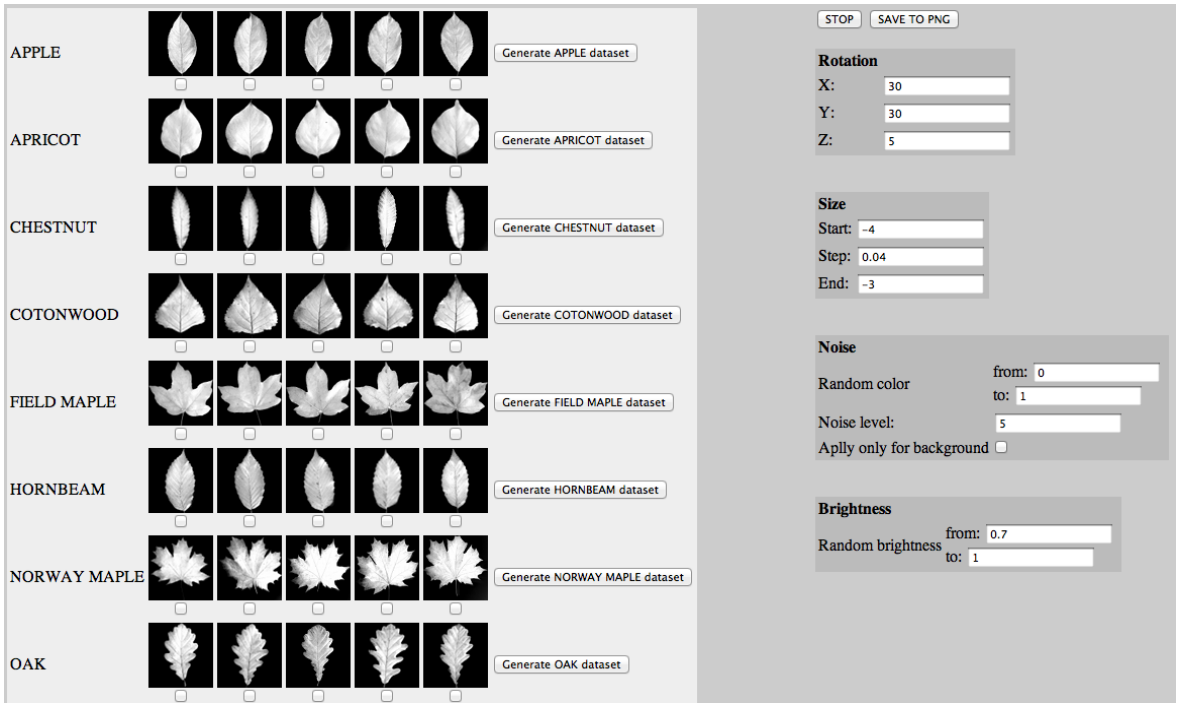


Figure 14: Screenshot of Dataset Generator application interface: http://dbn.meshmatrix.com/

At bottom part of application is output generated. It is generated in textual (Figure 16.) and visual (Figure 15) form. In textual form there are 1024 numbers, which represent color and

one number more, which represent category of sample. Visual form can be saved as PNG image file.

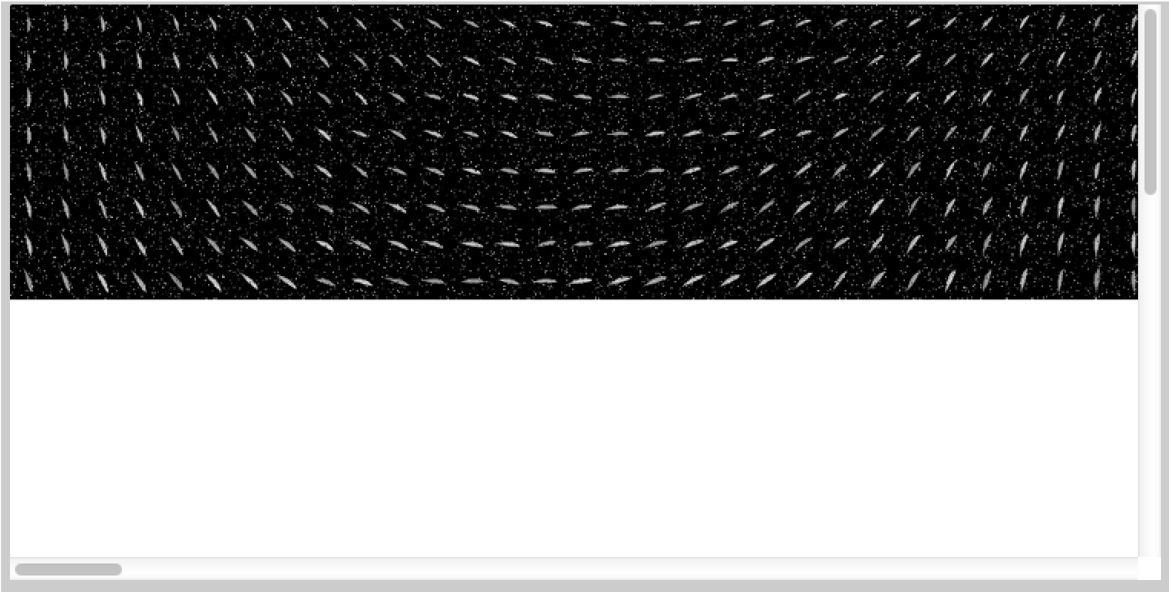


Figure 15: Screenshot of bottom part of application, which shows generated output in visual form

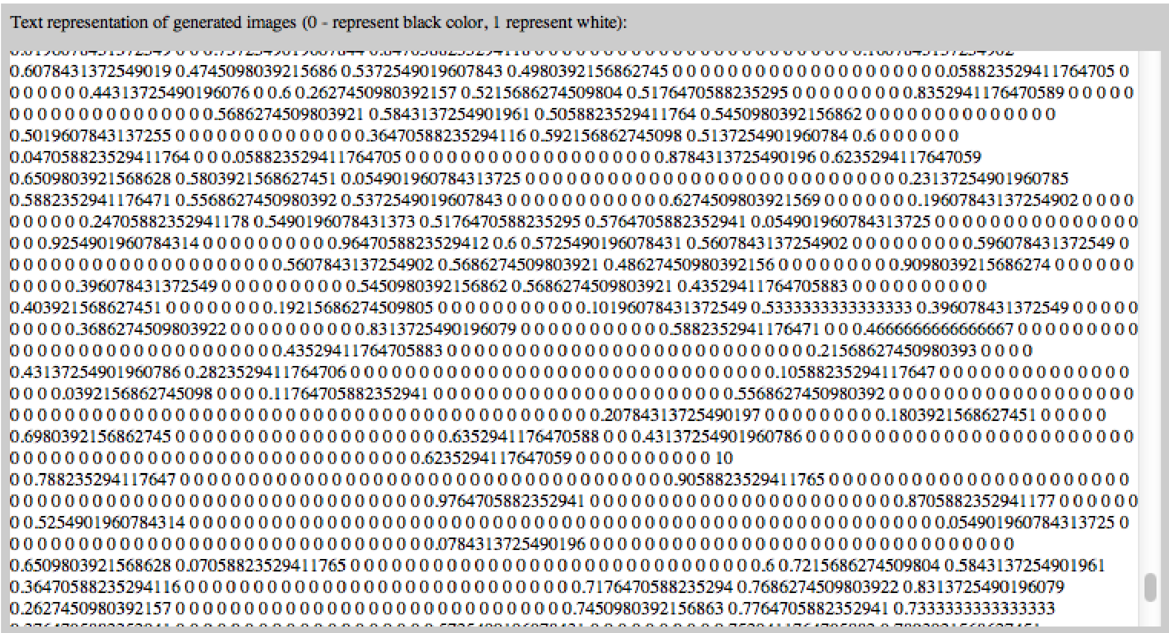


Figure 16: Screenshot of bottom part of application, which shows generated output in textual form

3.3 Faces dataset

Another testing dataset, which is used for experiments is faces dataset [18]. This dataset was created at Slovak Academy of Sciences. There are 2 face classes – normal (Figure 17) and abnormal (Figure 18). It contains 2280 samples. For purpose of this project the pictures were resized from 128x128 pixels to 64x64 pixels. Motivation for creation of this dataset was ability to recognize potentially dangerous or suspicious people near ATMs. The assumption was, that suspicious people wear some type of face cover.

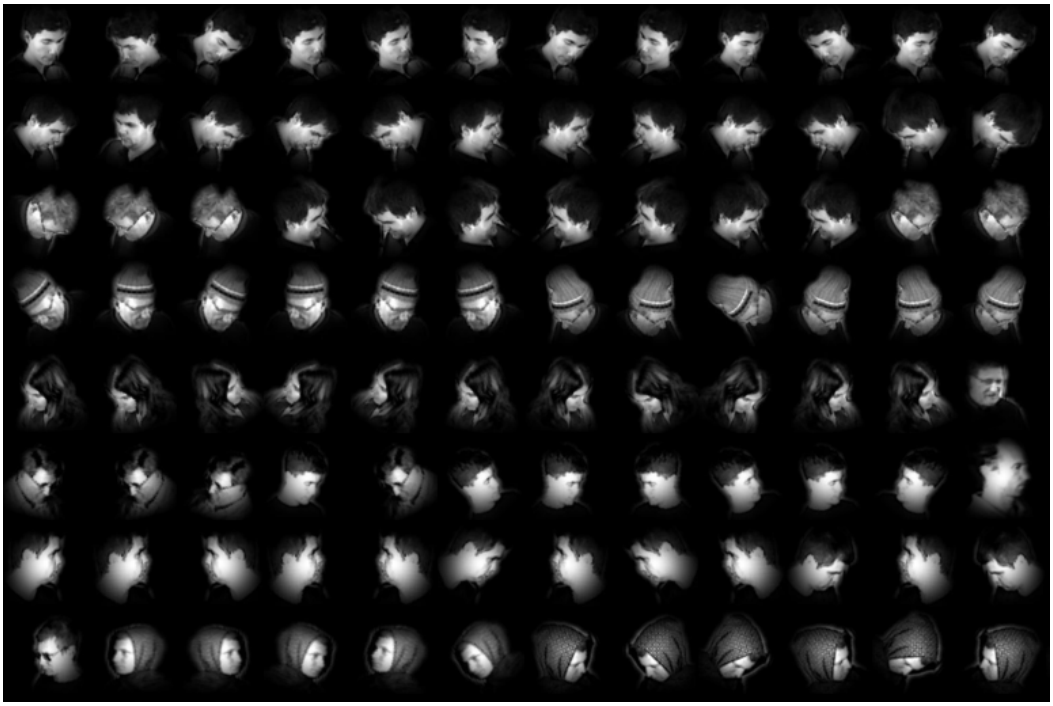


Figure 17: Example of normal face class from Faces dataset [18]

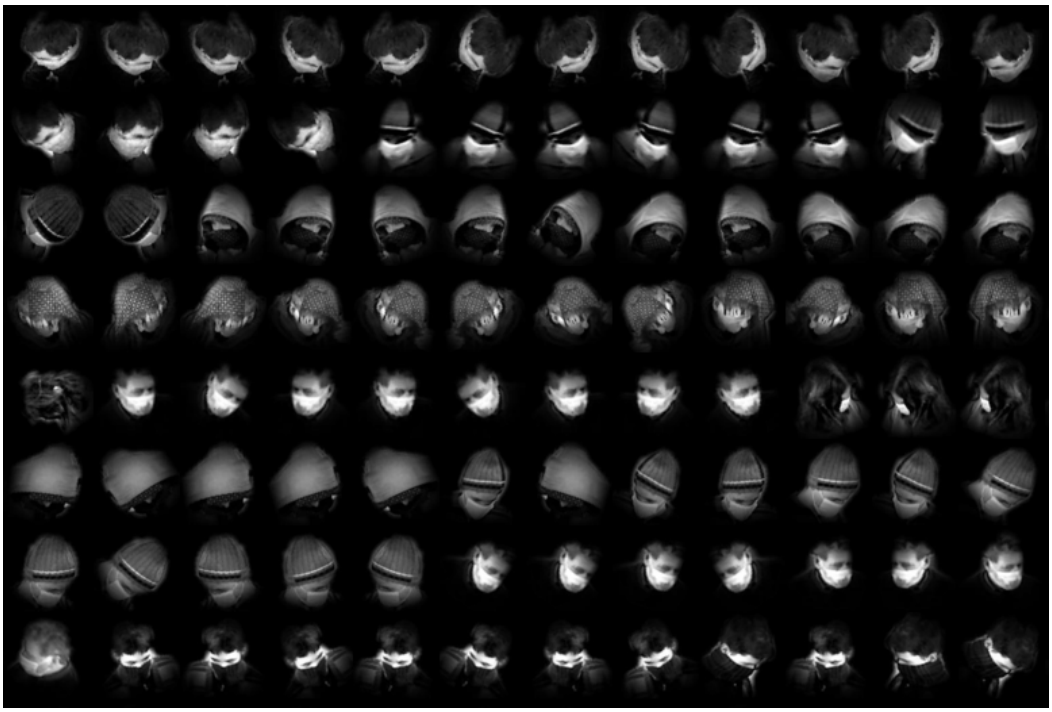


Figure 18: Example of abnormal face class from Faces dataset [18]

3.4 DBN implementation

DBN used in this project is based on *http://deeplearning.net/tutorial/DBN.html* . It is using fast Theano mathematical library, which is written in C. Whole program is written in Python. RBMs used to pre-train and initialize each layer of the network, that is used for classification (*"construct each RBM such that they share the weight matrix and the hidden bias with its corresponding sigmoid layer"* [36]). For more info refer to source code (in attachment).

3.5 Experiments on leaves dataset with DBN

In this section are presented 5 experiments with different variations of leaves dataset. The goal of this experiments is to find out how parameters of the network as number of hidden layers, number of neurons, number of fine-tuning epochs, number of pre-training epochs

and learning rate influence the final testing error. Variations in dataset parameters are in size (number of samples) and noise level. Each of these experiments has these common properties:

- Training, validation and testing sub-datasets were created from main dataset by random selection.
- Each experiment setup was run one time, expect last 5 setups in Table 6. They were run 4 times and in table is average value of these runs.

3.5.1 Experiment 1

Description of Dataset 1

All leaves (all 5 subclasses for each of 11 classes) are used for dataset generation.

- Training set = 2000
- Valid set = 400
- Test set = 504

(In Figure 19 is example of this dataset samples.)

DATASET GENERATOR SETUP:

Rotation		
x=30	y=30	z=5
Size		
Start=-4	Step=0.1	End=-2.8
Noise		
Random color from=0 to=0	Noise Level=0	Apply only for background=false
Brightness		
Random brightness from=0.7	to=1.0	

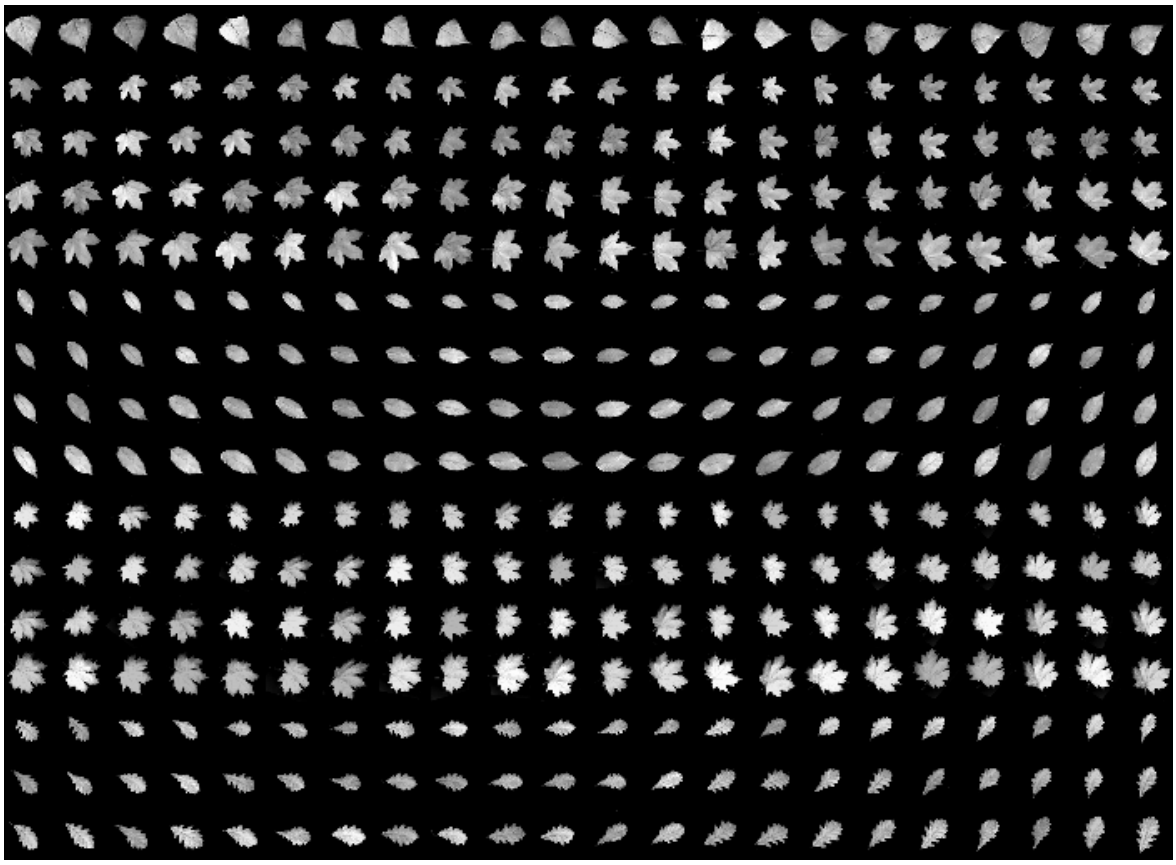


Figure 19: Example of Dataset 1 (For whole picture please see attachment)

HIDDEN LAYER TOPOLOGY	FINETUNING LEARNIG RATE	PRETRAINING LEARNING RATE	PRETRAINING EPPOCHS	FINETUNING EPPOCHS	VALIDATION ERROR	TEST ERROR
1024	0.0025	0.00001	500	500	39.8	42.4
1024	0.025	0.001	500	500	22.3	19.0
1024	0.1	0.01	500	500	18.8	20.0
500-500	0.1	0.01	500	500	14.5	16.4
500-500	0.025	0.001	500	500	20.3	19.8
1024-1024	0.1	0.01	500	500	16.5	17.6

Table 3: Results of experiment 1

TOPOLOGY	FINETUNING LEARNIG RATE	PRETRAINING LEARNING RATE	PRETRAINING EPPOCHS	FINETUNING EPPOCHS	VALIDATION ERROR	TEST ERROR
500-500	0.1	0.01	500	100	22.3	21.6
500-500	0.1	0.01	250	100	20.0	20.0
500-500	0.1	0.01	125	100	20.8	19.2
500-500	0.1	0.01	60	100	23.0	18.4
125-125	0.1	0.01	500	100	22.0	25.8
250-250	0.1	0.01	500	100	20.3	24.2
125-125	0.1	0.01	250	100	22.5	24.8
250-250	0.1	0.01	250	100	21.3	20.6
125-125	0.1	0.01	125	100	25.5	27.2
250-250	0.1	0.01	125	100	20.5	21.6
125-125	0.1	0.01	60	100	27.8	26.4
250-250	0.1	0.01	60	100	20.7	25.8

Table 4: Influence of number of pre-training epoch on final test error (Experiment 1)

1024	0.1	0.01	500	100	22.0	21.2
500	0.1	0.01	500	100	21.0	21.4
250	0.1	0.01	500	100	24.3	26.2
125	0.1	0.01	500	100	25.8	24.4

Table 5: Influence of the number of neurons on final error (Experiment 1)

500-500	0.25	0.01	60	500	15.3	21.8
500-500	0.1	0.01	60	500	17.5	22.0
500-500	0.05	0.01	60	500	17.0	17.8
500-500	0.025	0.01	60	500	17.8	19.4
500-500	0.01	0.01	60	500	21.8	26.2
500-500	0.005	0.01	60	500	33.3	38.6
500-500	0.5	0.01	100	100	18.8	15.0
500-500	0.25	0.01	100	100	18.8	22.2
500-500	0.1	0.01	100	100	20.5	23.6
500-500	0.05	0.01	100	100	26.8	23.8
500-500	0.025	0.01	100	100	38.8	32.2

Table 6: Influence of learning rate on final error (Experiment 1)

Evaluation of results for Dataset 1

Number of hidden layers

When we compare single hidden layer topologies from Table 5 (error 23.85%) to their double hidden layer equivalents from Table 4 (error 22.7%) we can see that 2 hidden layers have average performance 1.15 % better.

Number of neurons

When we look at Table 5 we can see that increase in the number of neuron helps decrease classification error. The similar trend occurs in Table 4 where are compared 3 different topologies. There is average error of 25.3% for topology 125-125, 21.9% for 250-250 and 20,8% for 500-500.

Number of fine-tuning epochs

Based on Tables 3. - 6. is obvious that more epochs results in better performance. Is important to notice that tested number of epochs was not high, so there was not over-fitting.

Number of pre-training epochs

Based on Table 5 there are these average test errors:

Number of epochs	Average error in %
500	22.7
250	21.5
125	22.5
60	23.7

These numbers shows that number of pre-training epoch have not significant influence on performance.

Influence of learning rate

Table 6 examines influence of learning rate on overall performance. The average of test and validation error shows that the best value is between 0.1 – 0.5. But its important to notice that this parameter is related to number of epochs.

3.5.2 Experiment 2

Description of Dataset 2

All leaves (all 5 subclasses for each of 11 classes) are used for dataset generation.

- Training set = 8000
- Valid set = 504
- Test set = 1000

(In Figure 20 is example of this dataset samples.)

Difference between Dataset 1 and Dataset 2 is in number of samples. This difference is created by decreasing rotational step and size step in dataset generator.

DATASET GENERATOR SETUP:

Rotation		
x=30	y=30	z=**2**
Size		
Start=-4	Step=**0.06**	End=-2.8
Noise		
Random color from=0 to=0	Noise Level=0	Apply only for background=false
Brightness		
Random brightness from=0.7	to=1.0	

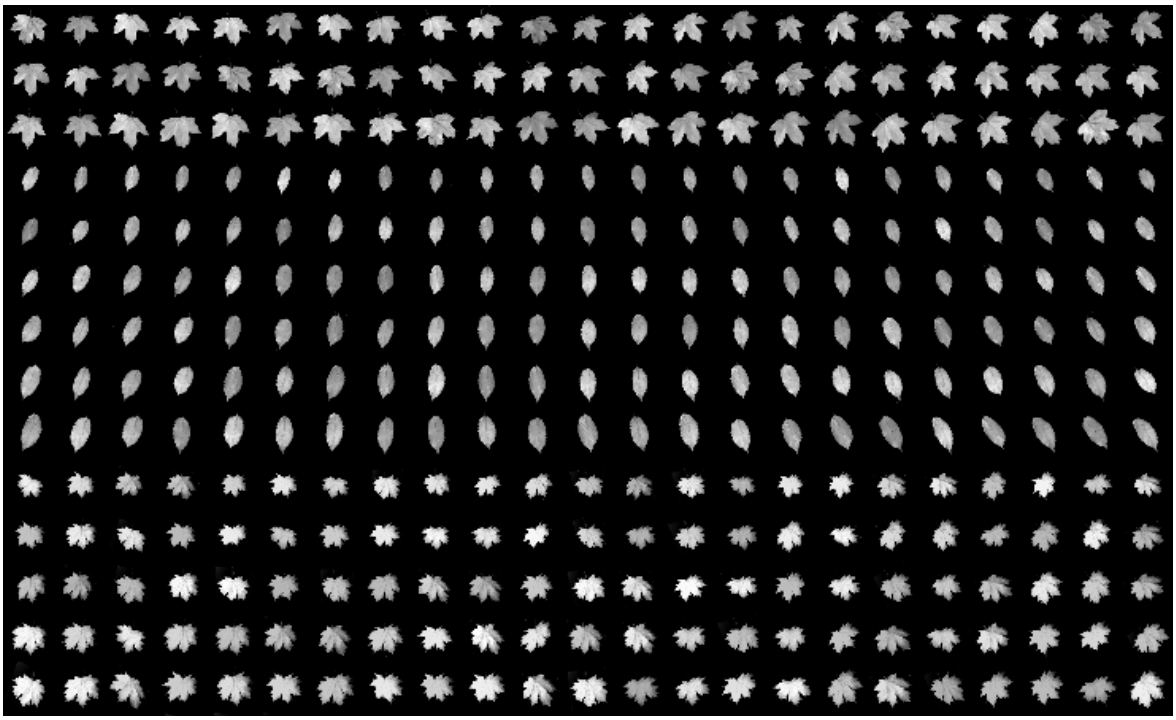


Figure 20: Example of Dataset 2 (For whole picture please see attachment)

TOPOLOGY	FINETUNING LEARNIG RATE	PRETRAINING LEARNING RATE	PRETRAINING EPPOCHS	FINETUNING EPPOCHS	VALIDATION ERROR	TEST ERROR
1024	0.0025	0.00001	500	500	15.2	14.4
1024	0.025	0.001	500	500	7.4	7.8
1024	0.1	0.01	500	500	6.8	5.5
500-500	0.1	0.01	500	500	6.6	8.1
500-500	0.025	0.001	500	500	5.4	6.1
1024-1024	0.1	0.01	500	500	7.4	4.7

Table 7: Results of experiment 2

TOPOLOGY	FINETUNING LEARNIG RATE	PRETRAINING LEARNING RATE	PRETRAINING EPPOCHS	FINETUNING EPPOCHS	VALIDATION ERROR	TEST ERROR
125-125	0.1	0.01	500	100	9.2	11.5
250-250	0.1	0.01	500	100	7.0	11.2
125-125	0.1	0.01	250	100	9.0	11.7
250-250	0.1	0.01	250	100	9.4	9.5
125-125	0.1	0.01	125	100	11.6	9.5
250-250	0.1	0.01	125	100	9.4	8.7
125-125	0.1	0.01	60	100	10.4	11.8
250-250	0.1	0.01	60	100	6.4	10.0
500-500	0.1	0.01	500	100	8.8	7.3
500-500	0.1	0.01	250	100	7.8	7.4
500-500	0.1	0.01	125	100	7.4	8.7
500-500	0.1	0.01	60	100	6.8	8.2

Table 8: Influence of number of pre-training epoch on final test error (Experiment 2)

1024	0.1	0.01	500	100	6.6	7.5
500	0.1	0.01	500	100	8.6	8.8
250	0.1	0.01	500	100	11.0	12.5
125	0.1	0.01	500	100	13.0	12.9

Table 9: Influence of the number of neurons on final error (Experiment 2)

500-500	0.25	0.01	60	500	6.2	5.2
500-500	0.1	0.01	60	500	5.8	7.2
500-500	0.05	0.01	60	500	7.6	6.2
500-500	0.025	0.01	60	500	8.2	8.1
500-500	0.01	0.01	60	500	8.4	8.9
500-500	0.005	0.01	60	500	10.6	10.9

Table 10: Influence of learning rate on final error (Experiment 2)

Evaluation of results for Dataset 2

Number of hidden layers

When we compare single hidden layer (125, 250, 500) topologies from Table 9 (error 11.1%) to their double hidden layer equivalents (125-125, 250-250, 500-500) from Table 8 (error 9,2%) we can see that doulble hidden layers have average performance 1.9 % better.

Number of neurons

When we look at Table 9 we can see that increasing the number of neuron helps decrease classification error. The similar trend occurs in Table 8 where we compare 3 different topologies. There is average error of 11.1% for topology 125-125, 9.9% for 250-250 and 7.9% for 500-500.

Number of epochs

Based on previous Tables 7.-10. is obvious that more epochs results in better performance. Is important to notice that tested number of epochs was not high, so there was not over-fitting. The best models have the high number of epochs (500).

Number of pre-training epochs

Based on Table 8 there are these average test errors:

Number of epochs	Average error in %
500	9.2
250	9.1
125	9.2
60	8.9

These numbers show that number of pre-training epoch have not significant influence on performance.

Influence of learning rate

There is strong trend (Table 10), which prefers higher learning rates.

3.5.3 Experiment 3

Description of Dataset 3

All leaves (all 5 subclasses for each of 11 classes) are used for dataset generation.

- Training set = 8000
- Valid set = 504
- Test set = 1000

(In Figure 21 is example of this dataset samples.)

Difference between Dataset 2 and Dataset 3 is in adding random noise (10% of the image is noise).

DATASET GENERATOR SETUP:

Rotation		
x=30	y=30	z=2
Size		
Start=-4	Step=0.06	End=-2.8
Noise		
Random color from=**0** to=**1**	Noise Level=**10**	Apply only for background=false
Brightness		
Random brightness from=0.7	to=1.0	

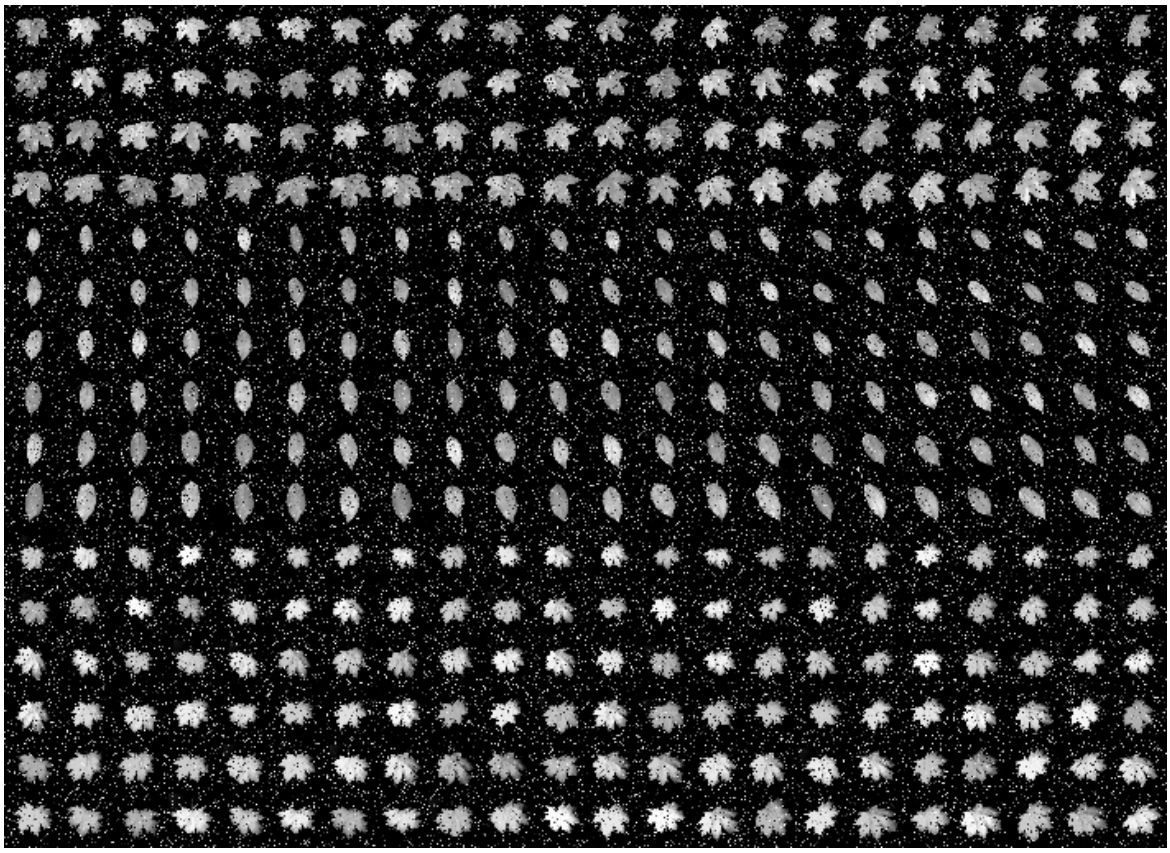


Figure 21: Example of Dataset 3 (For whole picture please see attachment)

TOPOLOGY	FINETUNING LEARNIG RATE	PRETRAINING LEARNING RATE	PRETRAINING EPPOCHS	FINETUNING EPPOCHS	VALIDATION ERROR	TEST ERROR
1024	0.0025	0.00001	500	500	28.8	31.6
1024	0.025	0.001	500	500	18.2	20.0
1024	0.1	0.01	500	500	23.4	24.7
500-500	0.1	0.01	500	500	22.2	20.1
500-500	0.025	0.001	500	500	18.0	20.1
1024-1024	0.1	0.01	500	500	19.8	22.1

Table 11: Results of experiment 3

TOPOLOGY	FINETUNING LEARNIG RATE	PRETRAINING LEARNING RATE	PRETRAINING EPPOCHS	FINETUNING EPPOCHS	VALIDATION ERROR	TEST ERROR
125-125	0.1	0.01	500	100	23.8	24.6
250-250	0.1	0.01	500	100	18.4	23.6
125-125	0.1	0.01	250	100	19.4	25.1
250-250	0.1	0.01	250	100	22.2	23.4
125-125	0.1	0.01	125	100	21.8	24.9
250-250	0.1	0.01	125	100	18.0	22.1
125-125	0.1	0.01	60	100	20.0	24.2
250-250	0.1	0.01	60	100	22.8	22.4
500-500	0.1	0.01	500	100	21.6	22.3
500-500	0.1	0.01	250	100	21.4	22.1
500-500	0.1	0.01	125	100	21.6	19.9
500-500	0.1	0.01	60	100	19.6	22.1

Table 12: Influence of number of pre-training epoch on final test error (Experiment 3)

1024	0.1	0.01	500	100	23.2	23.9
500	0.1	0.01	500	100	25.8	25.1
250	0.1	0.01	500	100	26.0	29.0
125	0.1	0.01	500	100	29.4	25.8

Table 13: Influence of the number of neurons on final error (Experiment 3)

500-500	0.25	0.01	60	500	15.0	22.3
500-500	0.1	0.01	60	500	21.6	19.6
500-500	0.05	0.01	60	500	19.8	22.5
500-500	0.025	0.01	60	500	19.2	21.5
500-500	0.01	0.01	60	500	23.0	22.3
500-500	0.005	0.01	60	500	22.6	23.1

Table 14: Influence of learning rate on final test error (Experiment 3)

Evaluation of results for Dataset 3

Number of hidden layers

When we compare single hidden layer topologies from Table 13 (error 26.9%) to their double hidden layer equivalents from Table 12 (error 22.4%) we can see that double hidden layers have average performance 4,5 % better.

Number of neurons

When we look at Table 13 we can see that there is a trend. Increasing the number of neuron helps decrease classification error. The similar trend occurs in Table 12 where we compare 3 different topologies. There is average error of 23% for topology 125-125, 21.6% for 250-250 and 21,3% for 500-500.

Number of epochs

Also in this experiment is trend. More epochs results in better performance. The model with best performance had 500 epochs.

Number of pre-training epochs

Based on Table 5 there are these average test errors:

Number of epochs	Average error in %
500	22.4
250	22.3
125	21.4
60	21.9

As well as in previous experiments these numbers show that number of pre-training epoch have not significant influence on performance.

Influence of learning rate

There is no significant influence of learning rate (Table 14). One of possible cause may be noise in dataset.

3.5.4 Experiment 4

Description of Dataset 4

All leaves (all 5 subclasses for each of 11 classes) are used for dataset generation.

- Training set = 17000
- Valid set = 2000
- Test set = 2120

Example of this dataset is in Figure 22.

Difference between Dataset 3 and Dataset 4 is in increasing number of samples in dataset.

DATASET GENERATOR SETUP:

Rotation		
x=30	y=30	z=1
Size		
Start=-4	Step=0.04	End=-3
Noise		
Random color from=0 to=1	Noise Level=10	Apply only for background=false
Brightness		
Random brightness from=0.7	to=1.0	

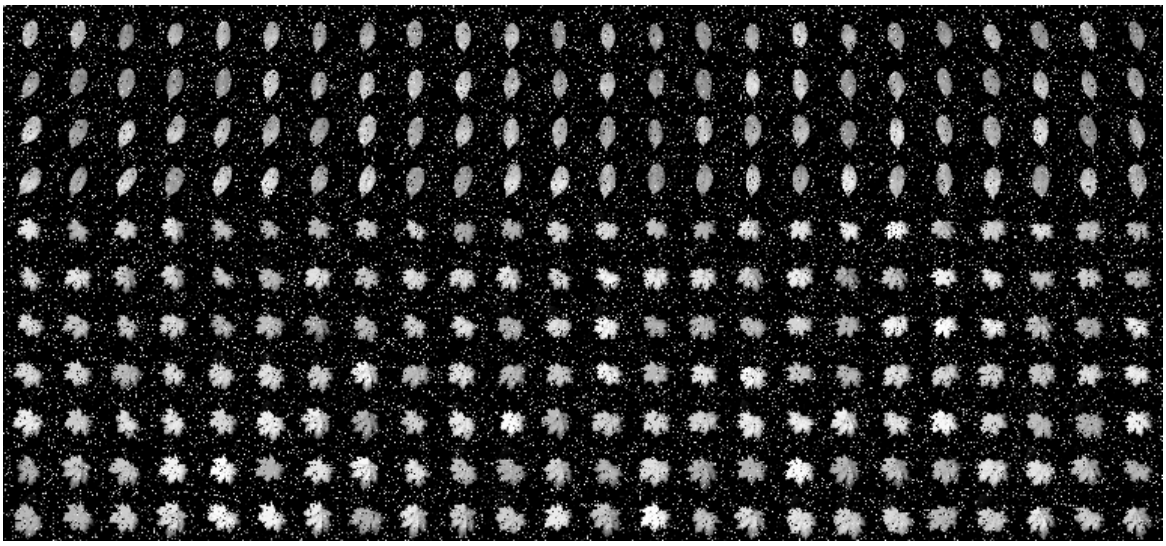


Figure 22: Example of Dataset 4 (For whole picture please see attachment)

TOPOLOGY	FINETUNING LEARNIG RATE	PRETRAINING LEARNING RATE	PRETRAINING EPPOCHS	FINETUNING EPPOCHS	VALIDATION ERROR	TEST ERROR
1024	0.0025	0.00001	500	500	20.5	19.1
1024	0.025	0.001	500	500	14.1	15.6
1024	0.1	0.01	500	500	17.2	17.2
500-500	0.1	0.01	500	500	17.5	17.2
500-500	0.025	0.001	500	500	14.9	15.9
1024-1024	0.01	0.001	500	500	13.9	15.0

Table 15: Results of experiment 4

TOPOLOGY	FINETUNING LEARNIG RATE	PRETRAINING LEARNING RATE	PRETRAINING EPPOCHS	FINETUNING EPPOCHS	VALIDATION ERROR	TEST ERROR
125-125	0.1	0.01	500	100	18.1	17.9
250-250	0.1	0.01	500	100	18.7	18.6
125-125	0.1	0.01	250	100	19.3	19.2
250-250	0.1	0.01	250	100	18.9	18.4
125-125	0.1	0.01	125	100	19.2	20.0
250-250	0.1	0.01	125	100	17.6	19.1
125-125	0.1	0.01	60	100	17.0	19.3
250-250	0.1	0.01	60	100	17.7	16.7
500-500	0.1	0.01	500	100	16.7	16.0
500-500	0.1	0.01	250	100	16.0	16.2
500-500	0.1	0.01	125	100	15.3	17.3
500-500	0.1	0.01	60	100	14.5	17.2

Table 16: Influence of number of pre-training epoch on final test error (Experiment 4)

1024	0.1	0.01	500	100	17.5	17.8
500	0.1	0.01	500	100	16.7	20.0
250	0.1	0.01	500	100	20.6	22.4
125	0.1	0.01	500	100	21.3	22.0

Table 17: Influence of the number of neurons on final error (Experiment 4)

500-500	0.25	0.01	60	500	15.1	14.3
500-500	0.1	0.01	60	500	15.6	15.7
500-500	0.05	0.01	60	500	15.5	16.3
500-500	0.025	0.01	60	500	15.1	17.1
500-500	0.01	0.01	60	500	15.6	16.3
500-500	0.005	0.01	60	500	14.6	15.6

Table 18: Influence of learning rate on final error (Experiment 4)

Evaluation of results for Dataset 4

Number of hidden layers

When we compare single hidden layer topologies from Table 17 (error 20.5%) to their double hidden layer equivalents from Table 16 (error 17.7%) we can see that 2 hidden layers have average performance 2.8 % better.

Number of neurons

When we look at Table 17 we can see that increasing the number of neuron helps decrease classification error. The similar trend occurs in Table 16 where are compared 3 different topologies. There is average error of 18.8% for topology 125-125, 18.2% for 250-250 and 16.2% for 500-500.

Number of epochs

The similar trend like in previous cases occurred. The top models are with the highest number of epochs.

Number of pre-training epochs

Based on Table 17 there are these average test errors:

Number of epochs	Average error in %
500	17.7
250	18.0
125	18.1
60	17.0

These numbers shows that number of pre-training epoch have not significant influence on performance. (There is no trend)

Influence of learning rate

Table 18 examine influence of learning rate on overall performance. However in this case, there is no significant trend. On of the possibility may be noise in dataset (similar to previous experiment).

3.5.5 Experiment 5

Description of Dataset 5

This experiment is slightly different from previous. The difference is that first four leaves from each class are used for training and validating. And the last one representant of each leave is used for creation of testing sub-dataset.

- Training set = 17000
- Valid set = 2000
- Test set = 2120

A) TRAINING + VALIDATION DATASET GENERATOR SETUP:

Rotation		
x=30	y=30	z=1
Size		
Start=-4	Step=0.04	End=-3
Noise		
Random color from=0 to=1	Noise Level=5	Apply only for background=false
Brightness		
Random brightness from=0.7	to=1.0	

B) TESTING DATASET GENERATOR SETUP:

Rotation		
x=30	y=30	z=5
Size		
Start=-4	Step=0.04	End=-3
Noise		
Random color from=0 to=1	Noise Level=5	Apply only for background=false
Brightness		
Random brightness from=0.7	to=1.0	

TOPOLOGY	FINETUNING LEARNIG RATE	PRETRAINING LEARNING RATE	PRETRAINING EPPOCHS	FINETUNING EPPOCHS	VALIDATION ERROR	TEST ERROR
1024	0.025	0.001	500	500	9,4	19,7
1024	0.1	0.01	500	500	8,8	21.0
500-500	0.1	0.01	500	500	8.0	21.0
500-500	0.025	0.001	500	500	9,1	19,6
1024-1024	0.01	0.001	500	500	7,1	19,2

Table 19: Results of experiment 5

Evaluation of results for Dataset 5

The network achieved more than 80% successfully classified cases. This experiment is very similar to real world situations, because it used different leaves subclass for testing. Interesting comparison is between Validation and Test error. Validation contains samples from the same subclasses as train dataset. Test dataset contains different subclass. We can see that using leaf subclass from non training set decreased performance from 92.9% to 80.8% (in the best case).

3.6 Investigation of how unsupervised pre-training influence performance

In this section is presented one experiment with Dataset 1. The goal of this experiments is comparison of DBN to Deep network with random weight initialization (without unsupervised pre-training).

TOPOLOGY	FINETUNING LEARNIG RATE	PRETRAINING LEARNING RATE	PRETRAINING EPPOCHS	FINETUNING EPPOCHS	VALIDATION ERROR	TEST ERROR
500-500	0.5	0.01	100	100	18.8	15.0
500-500	0.25	0.01	100	100	18.8	22.2
500-500	0.1	0.01	100	100	20.5	23.6
500-500	0.05	0.01	100	100	26.8	23.8
500-500	0.025	0.01	100	100	38.8	32.2
500-500	0.01	0.01	100	100	40.8	44.2
500-500	0.005	0.01	100	100	43.0	52.0
500-500	0.0025	0.01	100	100	56.0	55.4

Table 20: Results with unsupervised pre-training

TOPOLOGY	FINETUNING LEARNIG RATE	PRETRAINING LEARNING RATE	PRETRAINING EPPOCHS	FINETUNING EPPOCHS	VALIDATION ERROR	TEST ERROR
500-500	0.5	0	0	100	16.8	18.0
500-500	0.25	0	0	100	18.8	21.6
500-500	0.1	0	0	100	23	25.8
500-500	0.05	0	0	100	27.3	27.0
500-500	0.025	0	0	100	34.0	34.9
500-500	0.01	0	0	100	36.8	43.8
500-500	0.005	0	0	100	49.8	51.2
500-500	0.0025	0	0	100	55.5	56.4

Table 21: Results without unsupervised pre-training

3.6.1 Evaluation of results

The goal of this experiment was to measure help of unsupervised pre-training. In fact this was comparison between DBN and MLP (multilayer deep network without unsupervised weights initialization). As you can see in Table 20 and 21 the increase in performance is not significant. Advantage of pre-training is higher when are used higher learning rates. Average validation and test error for top four results from Table 20 is 21.2% and for Table 21 it is 22.3%. Increase in performance when unsupervised pre-training is used was 1.1% (in average).

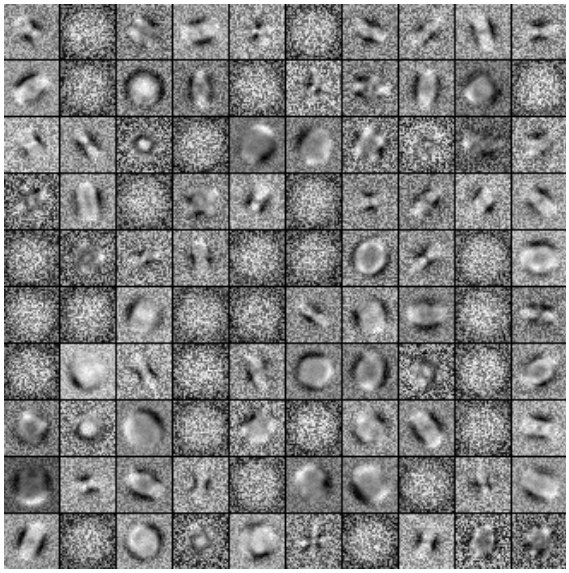


Figure 23: Example of feature detectors which are formed for hidden neurons

This is visualization of weights for each hidden neuron (there is 32x32 pixel input image therefore there is 1024 weight values for each neuron). Weights were transformed to values between 0 and 1. As you can see that there are not significant features. This lack of reasonable feature shapes is one of possibility why the unsupervised pretreating does not help to increase performance so much (in Dataset 1).

3.7 Experiment on faces dataset

3.7.1 Description of experiment

Experiment was performed on faces dataset [18]. Whole dataset (2280 samples) is divided to train (800 samples), validation (112 samples) and test (1368 samples) sub-datasets. The goal of this experiment is found out how unsupervised pre-training influence performance. For this purpose is used next training setup:

- fine-tuning learning rate = 0.1
- pre-training learning rate = 0.01
- pre-train epochs = 100
- fine-tuning epochs = 100

Each test was run 4 times and in Figure 24 is presented average of these runs.

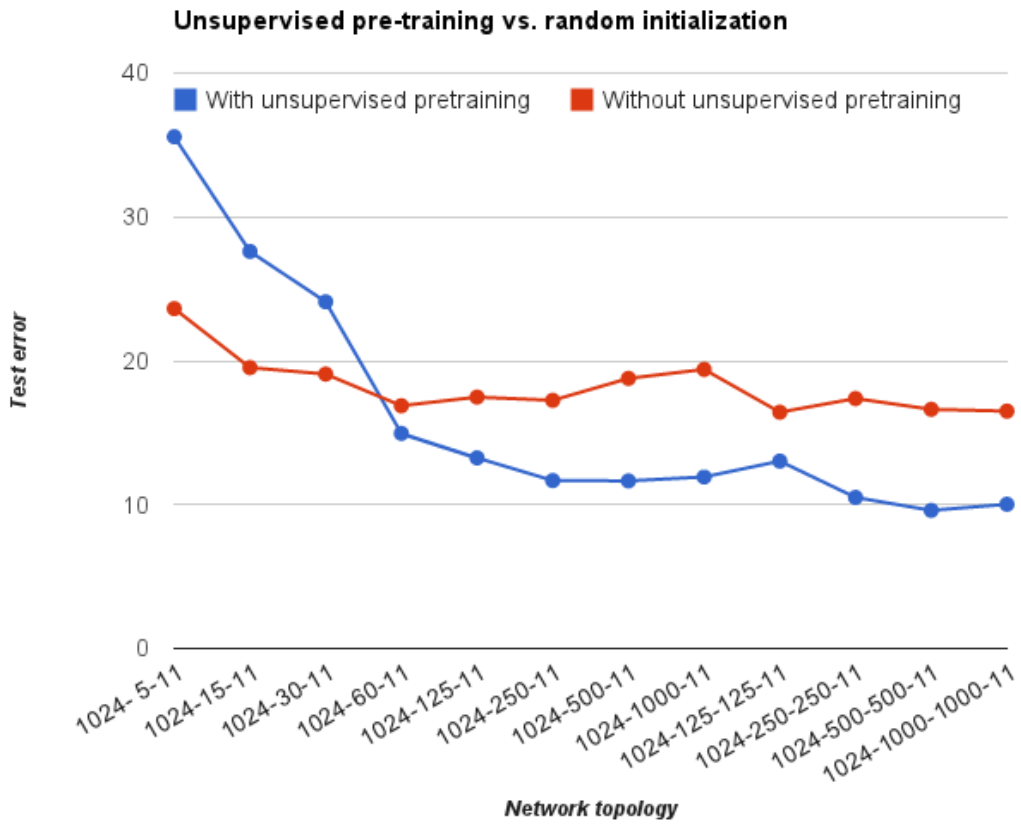


Figure 24: Performance comparison between unsupervised pre-training and random initialization

3.7.2 Evaluation of results

Figure 24 represents comparison between random weight initialization (typically used in MLP) and unsupervised pre-training weight initialization created by stack of RBMs. From this visualization is obvious that pre-training helps to achieve better classification performance. The best achieved test error for random initialization was 16.4 % and for RBMs initialization was 9.6 %. This result is better than result presented in original paper [18] where the best error rate for dataset with 836 train samples was 11.17%. They achieved this results by using HTM (hierarchic temporal memory) classifier.

4. DISCUSSION

In this part is described what actually above result tables means. There are presented 7 experiments, each with different setup (testing different variations and noise levels). The most important part of table is TEST ERROR. It represents percentage of mistaken classification in TEST dataset. The whole dataset was divided into 3 parts – for training, validation and testing. It is because of measure correlation between error on in-sample and out-of sample data. Experiment number 1. represents dataset with 2904 samples and without noise. It scores test error 16.4%. In the experiment number 2. is increased number of samples (9504) by decreasing step of rotation (it generates more sample). In Dataset 2. is achieved best error score of **4.7%**, which is good (also when we compare it to another works [6,7]). In experiment number 3. is added noise level 10%. The rest of setup is the same like in experiment number 2. The test error increased to 19.6%. After this experiment is clear that noise level had significant influence on whole performance. In experiment number 4. is increased number of samples to 21120 and noise level is 10%. The Test score is improved to 15%. Finally is provided experiment number 5. In this experiment are used 2 datasets. One for training and validating (leaves number 2-5 from each category) and second (leaves number 5 from each category) for testing. This experimental setup is simulation of real world data (during testing). Noise level is decreased to 5% and size of all two dataset is increased to 26928 samples. The final "real world" classification performance error on TEST dataset was 19,2%. One possible cause of this error may be the small resolution of input data and also small size of usable information in it (in some cases leaves are wide only 5 pixels). Maybe it would be interesting to test humans on this dataset and then compare to DBN.

When we look on the tables we can find interesting correlation between TEST ERROR and parameters of network and dataset. First correlation is between the size of topology and final error. In general if there is more neurons the error is lower. There is the same correlation between number of hidden layers and final error. The most important correlation is between size of dataset and error. For bigger datasets there is smaller error rate.

Difference in performance of unsupervised pre-training in faces and leaves dataset is significant. While in leaves experiments the pre-training did not help very much (1.1%) in Faces datasets it helps significantly (6.8%). This may be caused by huge variability and size of leaves dataset. It shows that advantage of using DBN is related to the parameters of training data.

5. CONCLUSION

The goal of this work was to show usability of deep belief network in object categorization problems. This goal was successfully achieved. There are provided sets of systematic experiments that show us a potential of this system. In some cases this potential is higher (faces dataset) in another cases where the variability is higher (and there are not so significant and clear features – leaves dataset) the potential is not so significant. Overall the network shows quite good classification skills and there is potential for real usage of such classification method in practice (e. g. portable cell phone application).

However there are also many another methods like HTM [18] or recently presented Multi-column Deep Neural Networks [22, 35]. Especially this recent network is very encouraging. Based on fresh results on image classification task like CIFAR-10 or MNIST Multi-column Deep Neural Networks achieved the best performance.

This field of research is very interesting for the application in robotics industries (e.g. road sign recognition [35]) and there is also huge potential for cognitive science, because these methods are inspired by biology and functionality of neural system.

REFERENCES

[1] Vinyals, O., Ravuri, S.V. (2011). Comparing multilayer perceptron to Deep Belief Network Tandem features for robust ASR, Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on , vol., no., pp.4596-4599, 22-27 May 2011 doi: 10.1109/ICASSP.2011.5947378

[2] Hinton, G. E. (2009). Deep belief networks. Scholarpedia, 4(5):5947.

[3] Erhan, D., Courville, A., Vincent, P. (2010). Why Does Unsupervised Pre-training Help Deep Learning ? Journal of Machine Learning Research,11(2007), 625-660. JMLR. org.

[4] Hinton, G. E. (2007). NIPS Tutorial on: Deep Belief Nets, Canadian Institute for Advanced Research, online: http://www.cs.toronto.edu/~hinton/nipstutorial/nipstut3.pdf

[5] Hinton, G. E., Osindero, S., Teh, Y. (2006). "A fast learning algorithm for deep belief nets", Neural Computation, vol 18, 2006

[6] Neto, J., Meyer, G., Jones, D., Samal, A. (2006). Plant species identification using Elliptic Fourier leaf shape analysis. Computers and Electronics in Agriculture, 50(2), 121-134. Elsevier Sci Ltd.

[7] Prasad, S., Kumar, P., Tripathi, R.C. (2011). Plant leaf species identification using Curvelet transform, Computer and Communication Technology (ICCCT), 2011 2nd International Conference on, vol., no., pp.646-652, 15-17 Sept. 2011 doi: 10.1109/ICCCT.2011.6075212

[8] Erhan, D., Courville, A., Bengio, Y., Vincent, P. (2010). Why Does Unsupervised Pre-training Help Deep Learning? In Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS) 2010, Chia Laguna Resort, Sardinia, Italy.

[9] Erhan, D., Manzagol, P.-A., Bengio, Y., Bengio, S., Vincent, P. (2009). The difficulty of training deep architectures and the effect of unsupervised pre-training. Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS 2009) (pp. 153–160). Clearwater (Florida), USA.

[10] Ranzato, M., Poultney, C., Chopra, S., LeCun, Y. (2007). Efficient learning of sparse representations with an energy-based model. NIPS 19 (pp. 1137–1144). MIT Press.

[11] Bengio, Y., Lamblin, P., Popovici, D., Larochelle, H. (2007). Greedy layer-wise training of deep networks. NIPS 19 (pp. 153–160). MIT Press

[12] Larochelle, H., Bengio, Y., Louradour, J., Lamblin, P. (2009). Exploring strategies for training deep neural networks. The Journal of Machine Learning Research, 10, 1–40.

[13] Lee, H., Ekanadham, C., Ng, A., (2008). Sparse deep belief net model for visual area v2, in Advances in Neural Information Processing Systems 20. MIT Press, 2008, pp. 873–880

[14] Mohamed, A., Dahl, G., Hinton, G. (2009). Deep Belief Networks for phone recognition. Deep Learning for Speech Recognition and Related Applications NIPS Workshop.

[15] Bengio, Y., Lamblin, P., Popovici, P., Larochelle, H. (2007). Greedy Layer-Wise Training of Deep Networks, Advances in neural information Processing Systems 19, MIT Press, Cambridge, MA.

[16] Sutskever, I. and Hinton, G. E. (2007). Learning multilevel distributed representations for high-dimensional sequences. AI and Statistics, 2007, Puerto Rico.

[17] Nair, V., Hinton, G. E. (2009). 3D Object Recognition with Deep Belief Nets. (Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, & A. Culotta, Eds.)Computer, 22, 1-9. Citeseer.

[18] Skoviera, R., Valentin, K., Farkaš, I., Stolc, S., Bajla, I. Detekcia anomálneho správania biologicky inspirovanou inteligentnou sietou,Vyskumna sprava UM SAV Bratislava

[19] Hadsell, R., Erkan, A., Sermanet, P., Scoffier, M., Muller, U. (2008). Deep belief net learning in a long-range vision system for autonomous off-road driving.2008 IEEERSJ International Conference on Intelligent Robots and Systems, 1(1), 628-633. Ieee.

[20] Mohamed, A., Yu, D., & Deng, L. (2010). Investigation of Full-Sequence Training of Deep Belief Networks for Speech Recognition. Architecture, (September), 2846-2849.

[21] Krizhevsky, A. (2010). Convolutional Deep Networks on CIFAR-10. Machine Learning, 1-9.

[22] Cires, D., Meier, U. (2012). Multi-column Deep Neural Networks for Image Classification. Applied Sciences, (February), 20.

[23] Haykin, S. (2008). Neural Networks and Learning Machines. Pearson Prentice Hall New Jersey USA 936 pLinks (p. 906).

[24] Farkaš I. (2011). Konekcionizmus v náručí výpočtovej kognitívnej vedy. In Kvasnička V. et al. (eds.), Umelá inteligencia a kognitívna veda III. 19-62.

[25] Bechtel, W., Abrahamsen, A. (2002). Connectionism and the Mind: Parallel Processing, Dynamics, and Evolution in Networks. Artificial Life (pp. 295 – 305). Wiley-Blackwell.

[26] O'Reilly, R.C., Munakata, Y., Frank, M.J., Hazy, T.E., and Contributors (2012). Computational Cognitive Neuroscience. Wiki Book, 2nd Edition.

[27] Ribeiro, B., Lopes, N. (2011). LNCS 7064 – Deep Belief Networks for Financial Prediction. Processing, 766-773.

[28] Hinton, G. E., Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. Science, 313(5786), 504-507. AAAS.

[29] Hinton, G. E. Video lecture.

URL http://carbon.videolectures.net/2009/singles/07/jul09_hinton_deeplearn/jul09_hinton_ deeplearn.pdf.

[30] Bengio, Y. (2009). Learning Deep Architectures for AI. Foundations and Trends® in Machine Learning,2(1), 1-127. Now Publishers Inc.

[31] Hinton, G. E. (2010). Learning to represent visual input. Philosophical Transactions of the Royal Society of London - Series B: Biological Sciences,365(1537), 177-184. The Royal Society.

[32] Felleman, D. J., Van Essen, D. C. (1991). Distributed hierarchical processing in the primate cerebral cortex. (A. T. Smith & R. J. Snowden, Eds.)Cerebral Cortex,1(1), 1-47. Oxford Univ Press.

[33] Smolensky, P. (1986). Information processing in dynamical systems: Foundations of harmony theory. In D. E. Rumelhart & J. L. McClelland (Eds.),Parallel Distributed Processing Explorations in the Microstructure of Cognition(Vol. 1, pp. 194-281). MIT Press.

[34] Hinton, G. E. (2007) Boltzmann machine. Scholarpedia, 2(5):1668.

[35] Ciresan, D. C., Meier, U., Masci, J., Schmidhuber, J. (2012). Multi-Column Deep Neural Network for Traffic Sign Classification. Neural Networks, in press.

[36] online: http://www.iro.umontreal.ca/~lisa/twiki/bin/view.cgi/Public/DBNPseudoCode

[37] online: http://deeplearning.net/tutorial/DBN.html

APPENDIX

Attachments of the thesis are saved on the attached DVD.

Medium contains:

- source files
- source codes
- Master's thesis in digital form (PDF)