
JINR Summer Student Programme 2016

# Classification of the compressed images using Deep Learning

JINR Summer Student Programme project report

**Author:**
Goncharov Pavel
Belarus, Gomel state technical university
Faculty of Automation and Information Systems

**Supervisors:**
Gennadiy Ososkov

**Participation period:**
18 July - 15 August

2016

# Contents

# 1 Intoduction

## 1.1 Nonlinear PCA

Image classification is one of the main spheres of machine learning. There are a large number of methods and algorithms of the image classification problem using artificial neural networks, such as:

- perceptron [1];
- adaptive resonance networks [2];
- radial basis function networks [3];
- convolution neural networks [4]

However, using of all image's pixels as input vector makes the training process too long and, at the same time, increases a number of interneural connections which brings a bad convergence of cost function.

In this work the method of extracting image features is proposed to use those features for the further classification. As it shown by M.Kramer [5], the nonlinear principal component analysis (NLPCA), which is similar to the well-known method of principal component analysis(PCA), using autoassociative neural networks(ANN) works better than the regular PCA, because it helps to find nonlinear dependences in data. On the figure 1 the ANN model of NLPCA realization is presented.

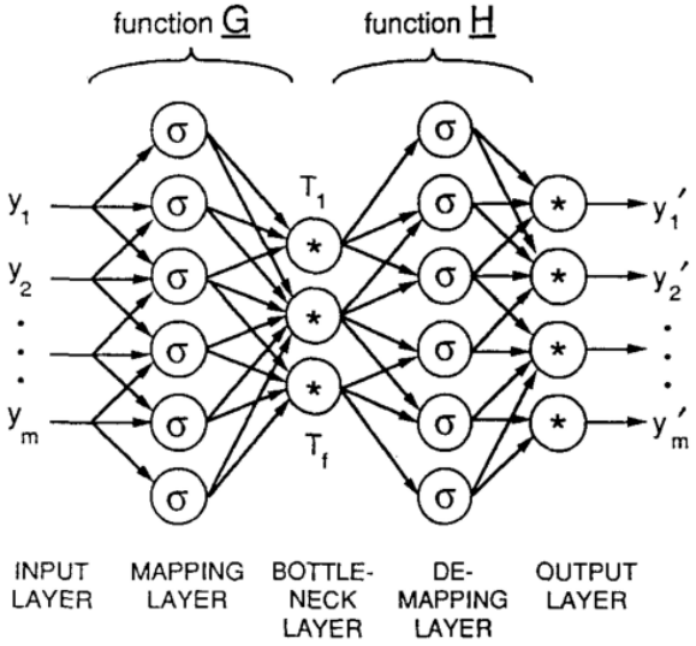



Figure 1: Autoaccosiative neural network for simultaneous determination of $f$ nonlinear factors, $\sigma$ *indicates sigmoidal nodes,* $*$ *indicates sigmoidal or linear nodes*

Bottle-neck layer $T$ extracts $f$-principal components. The ability of the neural network to fit arbitrary nonlinear functions depends on the presence of a hidden layers with nonlinear nodes.

To train the combined network, the weights appearing in the networks representing are optimized so that the reconstructed outputs $\underline{\underline{Y'}}$ match the inputs $\underline{\underline{Y}}$ as closely as possible. Training is complete, when $E$, the sum of squared errors, qiven in equation 1, is minimized [5]:

$$E = \sum_{p=1}^{n} \sum_{i=1}^{m} \left(Y_i - Y_i'\right)_p^2 \tag{1}$$

After training, the network has an ability to rescale input images into the smaller-dimensional features space.

## 1.2 Deep Belief Network

Images are transformed to the eigenvectors (vectors of the principal components) should be classified. To obtain maximum quality of the classification the number of hidden layers should be increased. However it would inevitably cause the bad convergence of the cost function. To solve this problem the deep belief networks (DBN) were proposed[6].

DBNs are probabilistic generative models that are composed of multiple layers of stochastic, latent variables [6]. The schema of the DBN is presented on the figure 2.

Net runs a pretraining procedure, using latent restricted Boltzmann machines (RBM) layers (see below). Discriminative fine-tuning can be performed by adding a final layer of variables that represent the desired outputs and backpropagating error derivatives [6].

### RBMs and Contrastive Divergence

A Boltzmann machine is a type of stochastic recurrent neural network and Markov Random Field, for which the bilinear energy function is determined [7]:

$$E(v,h) = -\sum_{i \in visible} a_i v_i - \sum_{j \in hidden} b_j h_j - \sum_{i,j} v_i h_j W_{i,j}, \tag{2}$$

where $v_i$, $h_j$ are states of visible unit $i$ and hidden unit $j$, $a_i$ ,$b_j$ are they biases and $W_{i,j}$ is the weight between them.

RBMs further restrict BMs to those without visible-visible and hidden-hidden connections. On the figure 3 the comparison of BMs architectures is presented.

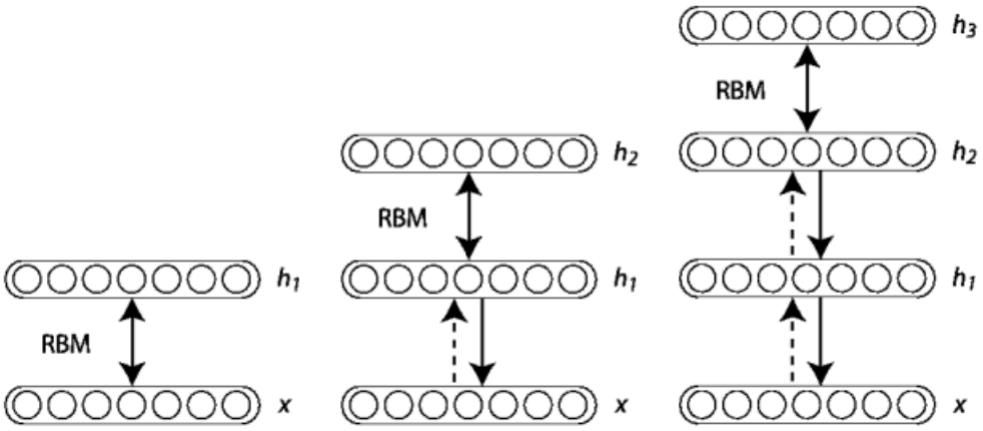
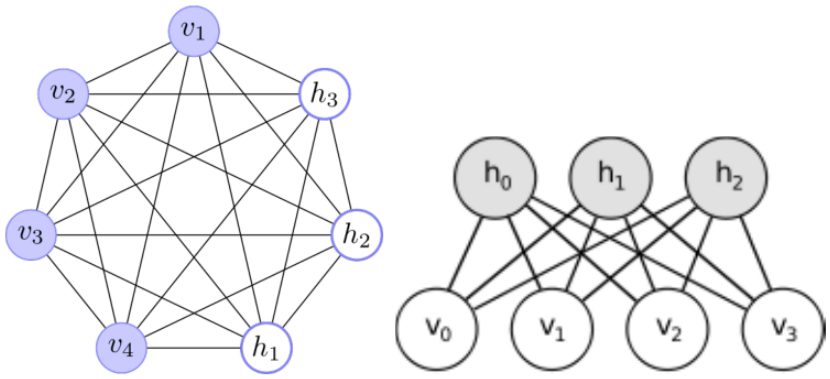



Figure 2: Deep Belief Network model




(a) Boltzmann machine (b) RBM

Figure 3: Comparison of the BMs model's architectures

The network assigns a probability to every possible pair of a visible and a hidden vector via this energy function:

$$p(v,h) = \frac{1}{Z} e^{-E(v,h)}, \tag{3}$$

where the «partition function», $Z$, is given by summing over all possible pairs of visible and hidden vectors:

$$Z = \sum_{v,h} e^{E}(v,h) \tag{4}$$

The probability that the network assigns to a visible vector, $v$, is given by summing over all possible hidden vectors:

$$p(v) = \frac{1}{Z} \sum_{h} e^{E(v,h)} \tag{5}$$

The probability that the network assigns to a training image can be raised by adjusting the weights and biases to lower the energy of that image and to raise the

energy of other images, especially those that have low energies and therefore make a big contribution to the partition function. The derivative of the log probability of a training vector:

$$\frac{\partial \log p(v)}{\partial w_{i,j}} = \langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model}, \tag{6}$$

where the angle brackets are used to denote expectations under the distribution specified by the subscript that follows. This leads to a very simple learning rule for performing stochastic steepest descent in the log probability of the training data, where $\eta$ is a learning rate:

$$\Delta w_{i,j} = \epsilon \left( \langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model} \right) \tag{7}$$

Because there are no direct connections between hidden units in an RBM, it is very easy to get an unbiased sample of $\langle v_i h_j \rangle_{data}$. Given a randomly selected training image, $v$, the binary state, $h_j$, of each hidden unit is set to 1 with probability [Eq.8]. And it is also very easy to get an unbiased sample of the state of a visible unit, given a hidden vector [Eq.9].

$$p(h_j = 1|v) = \sigma \left( b_j + \sum_i v_i w_{i,j} \right) \tag{8}$$

$$p(v_i = 1|h) = \sigma \left( a_i + \sum_j h_j w_{i,j} \right), \tag{9}$$

where $\sigma = 1/(1 + exp(-x))$.

Getting an unbiased sample of $\langle v_i h_j \rangle_{model}$, however, is much more difficult. It can be done by starting at any random state of the visible units and performing alternating Gibbs sampling for a very long time [8].This procedure is called Markov chain [Fig.4].

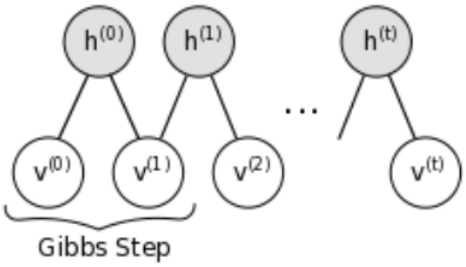



Figure 4: Sampling in an RBM

Such approach is impossible realize, however, there is a «trick» to speed up sampling process – contrastive divergence(CD) [9] learning algorithm:

- since we eventually want $p(v) \approx p_{train}(v)$, we initialize the Markov chain with a training example;
- CD does not wait for the chain to converge. Samples are obtained after only k-steps of Gibbs sampling. In practice, $k = 1$.

## 1.3 Datasets

As a database for training and testing networks, two well-known datasets will be used:

- «MNIST handwritten digit database»;
- «FERET face database».

MNIST handwritten digits database includes 70 thousands of handwritten digits images, which format is $28 \times 28$ pixels [Fig.5]

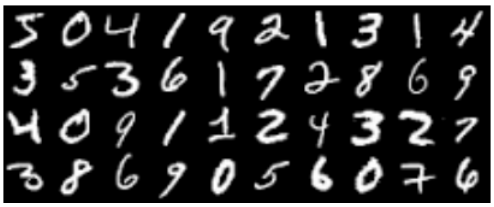


Figure 5: MNIST handwritten database sample

Face database FERET includes of 400 people's faces photos (40 people to 10 photos for each) taken from different angles, which format is $55 \times 45$ pixels [Fig.6].

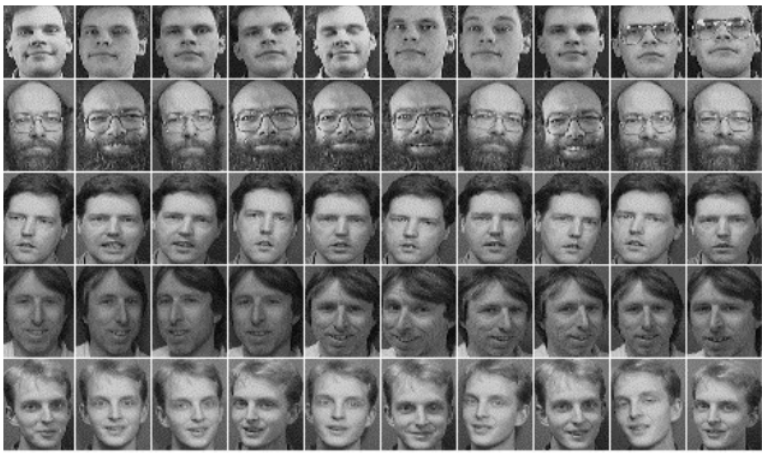


Figure 6: FERET face database sample

## 1.4 The aim of the work

The aim of this work is to develop a scientific programme complex, which will help to obtain maximum accuracy on the classifications made for both test sets of using images databases. This program should include two artificial neural networks:

- Autoassosiative neural network for the implementing Nonlinear PCA;
- Deep Belief Network for the final classification.

Tune the model's hyperparameters to increase accuracy and test complete models on two datasets. Make a comparison of the results of classification using different types of networks: DBN and the regular perceptron with one hidden layer.

# 2 Methods and realization

## 2.1 The problem of convergence

Deep Learning is equal to many hidden layers in the artificial neural network. It means that the network be affected by the problem named «vanishing and exploding gradients» [10]. To avoid this problem in the work some techniques will be used.

### Autoencoder

Autoassociative neural network(ANN) that were used for simultaneous determination of nonlinear factors has three hidden layers. It means they are also affected by this problem. Therefore network training with the backpropagation algorithm brings a bad convergence. On the figure 7 there is an example of the autoencoder's output. Tied weights and the Adam Optimization [11] were used to solve this issue.

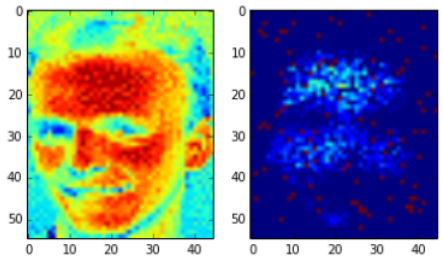



Figure 7: Example of the autoencoder's output that was trained with the backpropagation and regular weights

Regular formula for a feed forward neural network with a single hidden layer:

$$f(x) = \sigma_2 \left( b_2 + W_2 \sigma_1 (b_1 + W_1 x) \right) \tag{10}$$

The problem of this architecture is that, since the weight matrices $W_1$ and $W_2$ are independent, the autoencoder can easily learn the identity given a big enough hidden layer. A way around this is to use a tied weights which has the formula:

$$f(x) = \sigma_2 \left( b_2 + W_1^T \sigma_1 (b_1 + W_1 x) \right) \tag{11}$$

Here we set $W_2 = W_1^T$ eliminating a lot of degrees of freedom.

Using of this technique we the global minimum of the cost function 1. Recovered by autoencoder from 64 nonlinear principal components images are shown on the figure 8.

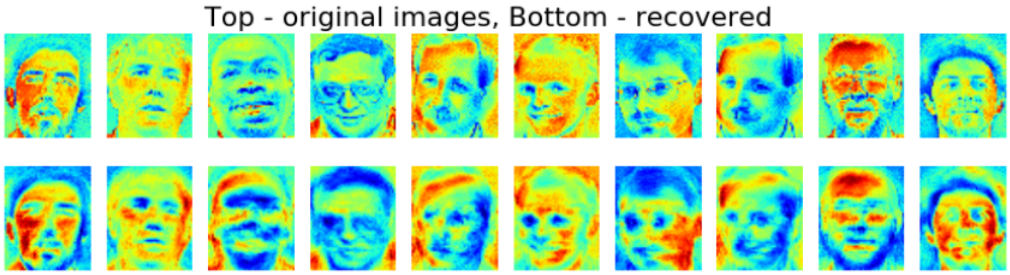



Figure 8: Images from the FERET database recovered with the autoencoder from 64 nonlinear principal components

## Deep Neural Network

The main reason of using DBN in the given work is that this network does not suffer from «vanishing and exploding gradients» problem. But some learning algorithms (in particular unsupervised learning algorithms such as algorithms for training RBMs by approximate maximum likelihood) are problematic in this respect because we cannot directly measure the quantity that is to be optimized (e.g. the likelihood) because it is intractable. On the other hand, the expected denoising reconstruction error is easy to estimate (by just averaging the denoising error over a validation set) [12]. Furthermore, a training time of the DBNs is too long. To speed up the training time and to avoid the convergence problem a Deep Neural Network(DNN) with weights normalized initialization [10] will be used.

Consider a weight matrix $W \in \mathbb{R}^{m \times n}$ where each element was drawn from an independent and identically distributed(IID) Gaussian with variance $Var(W)$. And there is no correlation between input and weights and both are zero-mean.

If we consider one filter (row) in $W$, say $w$, then the variance of the output signal over the input signal is:

$$\frac{Var(W^T x)}{Var(X)} = \frac{\sum_n^N Var(w_n x_n)}{Var(X)} = \frac{nVar(W)Var(X)}{Var(X)} = nVar(W) \tag{12}$$

It is reasonably to keep variance of the signal going forward in the network to remain the same, thus it would be advantageous if $nVar(W) = 1$.

So the formula of the normalized initialization will be:

$$W \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\right], \tag{13}$$

where $U[-a, a]$ is the uniform distribution in the interval $(-a, a)$ and the n is the size of the previous layer(the number of columns of $W$).

The DNN will be much faster than DBN, because of lack of a pretraining procedure(RBM hidden layers). After updating network models with these techniques we give the first predictions on the test set which consists of two parts: handwritten digits and faces. The result is shown on the table 1.

Table 1: Comparison of the different estimators

| | Perceptron | Deep Neural Network | Deep Belief Network |
|---|---|---|---|
| MNIST | 0.7977 | 0.8713 | 0.8482 |
| FERET | 0.8750 | 0.9583 | 0.9416 |

As we can see the results of the predictions are good and an accuracy of the perceptron with one hidden layer is already lower than in the deep learning models. But we can improve these results if will use cross validation for a hyperparameter optimization.

## 2.2 Model selection

Firstly, we need to choose an optimal number of principal components for a data rescaling. And a boundary of the number of principal components is equals to 250. It prevents the model from a growing complexity. On the figure 9 is shown the dependence of the accuracy on the number of neurons in the bottle-neck layer.

As we can see on the figure 9, the optimal number of the components begins with 60 components. Now lets take a look on how many epochs is needed to train the network with the different numbers of principal components.

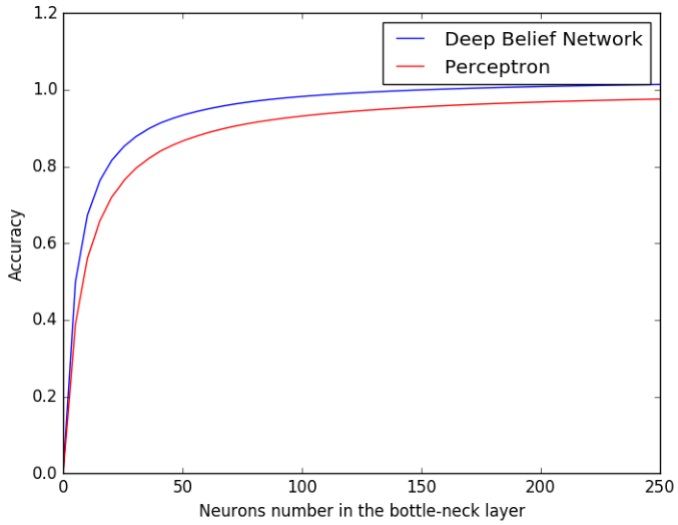



Figure 9: Dependence of the accuracy on the number of neurons in the bottle-neck layer

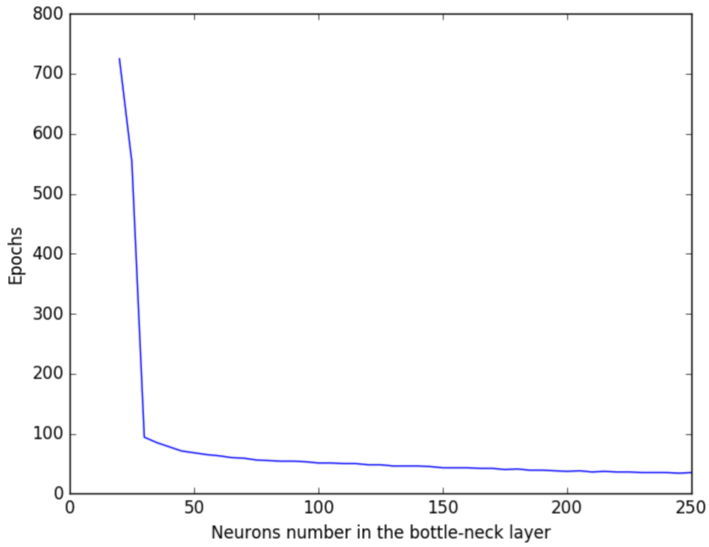



Figure 10: Dependence of the epoch's number on the number of neurons in the bottle-neck layer

The graphic 10 shows that the epochs number inversely proportional to the bottle-neck neurons number. The main reason is the insufficient number of neurons for the input data reconstruction. The curve stabilizes around 70 neurons.

First two tests showed that the optimal components number lies between 50 and 200. The figure below displays training time dependence on the bottle-neck layer neurons number. It is worth noting that the epochs number and training time value are not the same things. This test proves that the optimal value is around 70-80 neurons in the bottle-neck layer.

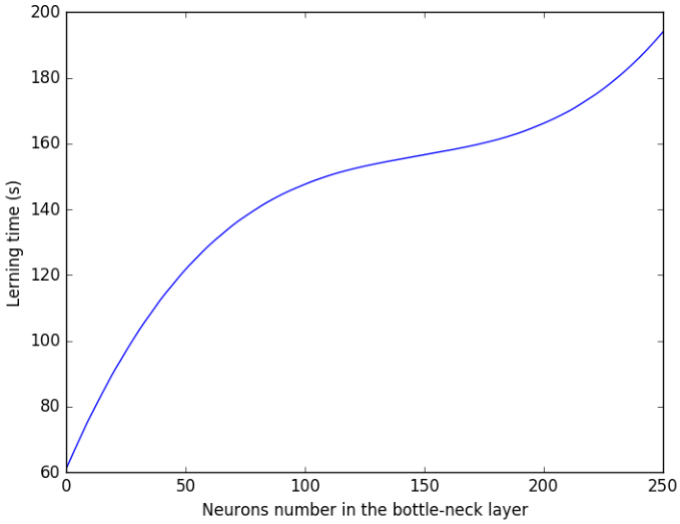



Figure 11: Dependence of the training time on the number of neurons in the bottle-neck layer

Experiments with different numbers of principal components showed that 64 is a good choice, because it helps to ensure accuracy value around 95 percent and, as it said before, great number of neurons increases the model complexity. Therefore the number of the principal components which is chosen is 64. A final model of the autoencoder consists of 5 layers: input and output layers with the same dimensions(length of the input vector), mapping and demapping layers with 256 neurons per layer and the bottle-neck layer, which number of the neurons equals to 64.

### DNN and DBN tuning

A good quality of deep architectures is that they can model high-level abstractions in data that allows to obtain high prediction quality. But the model can be easy overfitted if wrong parameters were choosed.

If the model is not complex enough, it may not be powerful enough to captur all of the useful information necessary to solve a problem. However, if the model is very complex (especially if there is a limited amount of data), it is suffer from overfitting. Deep learning takes the approach of solving very complex problems with complex models and taking additional countermeasures to prevent overfitting such as: dropouts and regularization. But lets tune the model complexity without this aid methods.

And the first is necessary to choose the number of the hidden layers. On the figure 12 above we can see two curves that represent the dependence between the accuracy and the number of the hidden layers. As it is shown the best choice is two hidden layers. So we have chosen the hidden layers number, it's time to

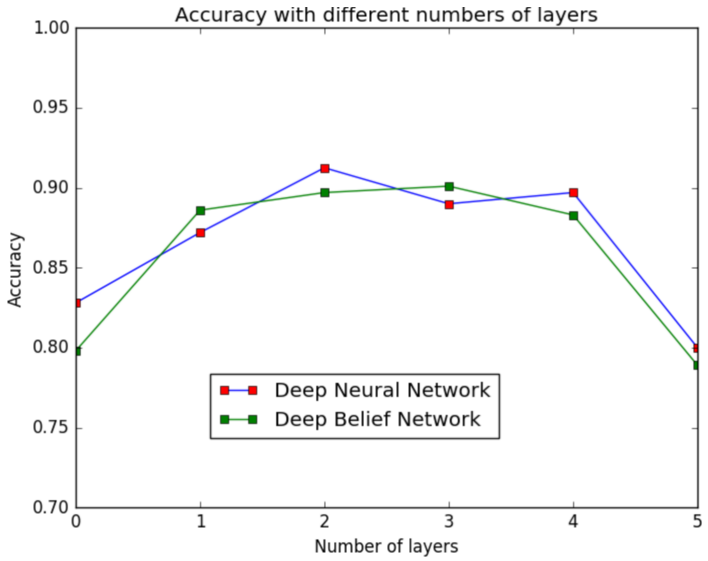



Figure 12: Dependence of the accuracy on the number of the hidden layers

determine an optimal combination of the hidden neurons per layer.

On the figure 13 the accuracy distribution dependence on the number of neurons in the hidden layers is showed. On axes we can see the number of neurons in the first hidden layer, and on the vertical axis - the number of neurons in the second hidden layer. But it is not clear on this plot which combination of the hidden neurons is better. Therefore lets print a table [2] with the best combinations and their accuracies. As we can see in the table, two last combinations bring us the same accuracy, but 100 in the second layer better than 180. Thus, the optimal numbers of the hidden neurons per layer are 300 and 100 respectively.

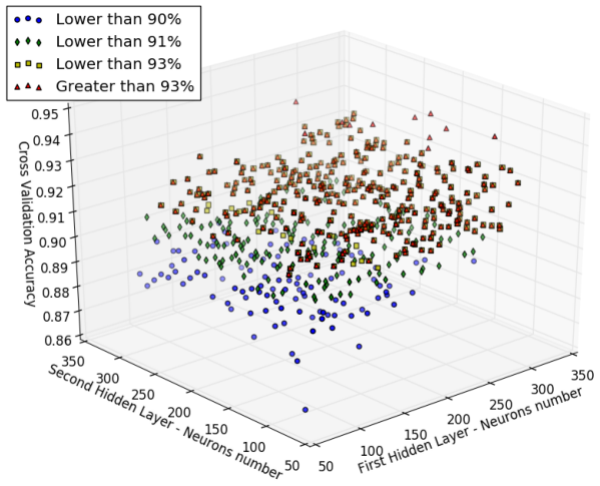



Figure 13: Accuracy distribution depends on the number of neurons in the hidden layers

Table 2: 10 best combinations of the hidden neurons numbers

| Neurons 1st layer | Neurons 2nd layer | Cross-validation accuracy |
|---|---|---|
| 200 | 230 | 0.939286 |
| 250 | 300 | 0.939286 |
| 260 | 210 | 0.939286 |
| 300 | 160 | 0.939286 |
| 300 | 200 | 0.939286 |
| 300 | 100 | 0.942857 |
| 300 | 180 | 0.942857 |

The other hyperparameters were tuned using the grid search cross-validation (see *scikit-learn.org*). The following chapter presents the results of testing the final tuned models.

## 2.3 Testing

Since we set up the neural network models, it is the time to test them. Firstly, lets take a look, how good the ANN can rescale the input data from the 64 nonlinear principal components. In figures 8,14 are shown the examples of the data rescaling.

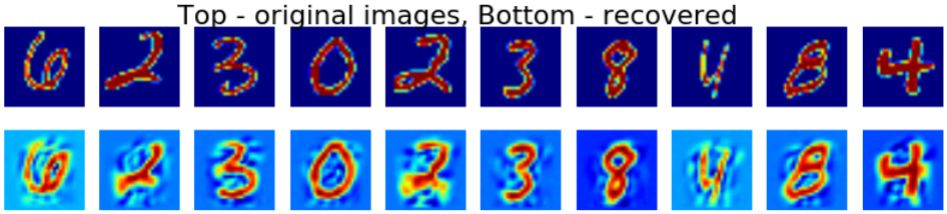
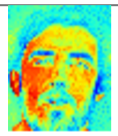
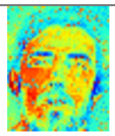
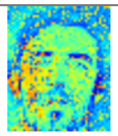
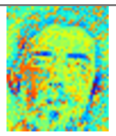
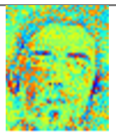
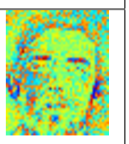
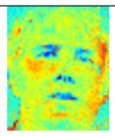
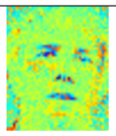
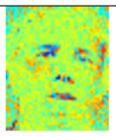
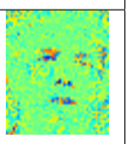
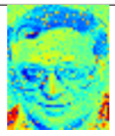
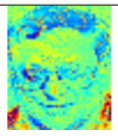
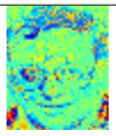
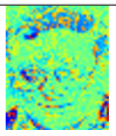
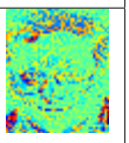



Figure 14: Images from the MNIST database recovered with the autoencoder from 64 nonlinear principal components

The main test is that how the model will rescale the input data with the different percent of noise on the images. In table 3 we can see the example of the noisy images compared with the original ones.

Generally, to clear images from the noise the ANN had to be trained to minimize the sum of square errors between the output image, produced by feed forward noisy image through the net, and the clear image that noisy image refers to. Eventually, the network will learn how to clear images(denoising autoencoder). The ANN in this work doesn't have this quality, but nevertheless it can recover even the noisy images.

Table 3: Example of the noisy images

| Original images | Noisy images | | | | |
|---|---|---|---|---|---|
| | 10% | 20% | 30% | 40% | 50% |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

On the figures 15, 16 below the examples of the recovered noisy images are shown.

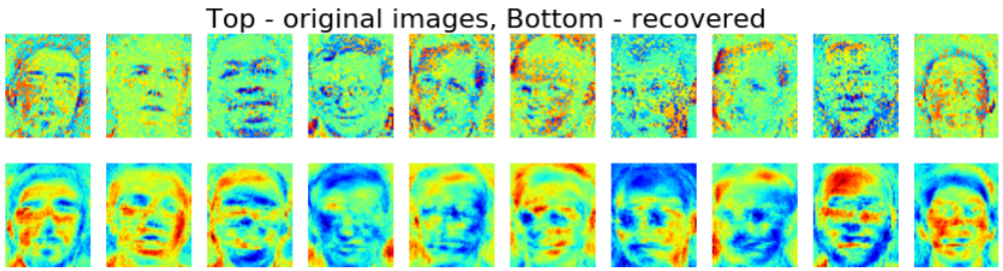



Figure 15: Noisy images from the FERET database recovered with the autoencoder from 64 nonlinear principal components

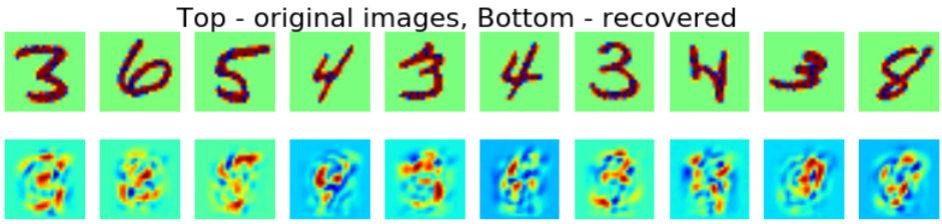



Figure 16: Noisy images from the MNIST database recovered with the autoencoder from 64 nonlinear principal components

### Classifiers

The images rescaled to the 64-dimensional eigenvectors were classified by three different artificial neural networks:

- perceptron with one hidden layer;
- deep neural network with random initialized weights;
- deep belief network.

The example of the incorrect classifier predictions (as classifier the DBN were used) [Fig.17]. It should be pointed out, the classification of the handwritten digits is a very difficult task even for the human.

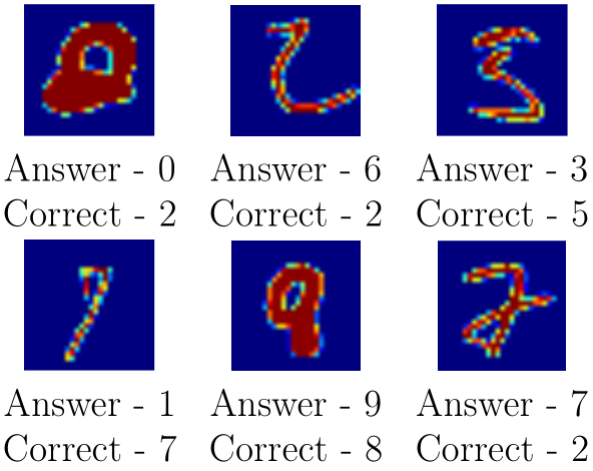



Figure 17: Incorrect predictions on the MNIST test set

Table 4 contains the results of the testing different classifiers with the two datasets:

- MNIST handwritten digits dataset;
- FERET faces datases.

The results are very good. Deep learning techniques provide to obtain the 100 percent accuracy on the faces database and the 92 –98 percent accuracy on the handwritten digits dataset. And if it is not a problem for the human to differ faces, then the DBN had made better predictions, even in the case of digits, than the human can.

Table 4: Final comparison of the different estimators

| | Perceptron | Deep Neural Network | Deep Belief Network |
|---|---|---|---|
| MNIST | 0.7977 | 0.9285 | 0.9839 |
| FERET | 0.8750 | 1.0000 | 1.0000 |

# 3 Conclusion

In this work the programme complex, consists of three neural networks: ANN for the NLPCA realization, DNN and DBN classifiers, - for the image recognizing problem was developed. Significant steps to achieve the goal of high predictions accuracy were done:

- the network models were choosed;
- DNN classifier was added to compare with DBN;
- the convergence problem was solved;
- all hyperparameters were tuned with grid search cross-validation method;
- models were tested and the classifiers comparison table was presented.

The obtained results are quite satisfactory. Proposed model allowed to achieve the 100 percent accuracy on the FERET database and the best results for the MNIST handwritten digits dataset is 98.39%, which the DBN provides, versus 79% with perceptron. It proves that deep learning deals with classification tasks much better than the regular machine learning algorithms due to its capability to define complex nonlinear dependencies in data.

Final comparison shows that DBN is the best model, but, it worth to note, that DNN model trains much faster even using contrastive divergence algorithm for the RBMs layers pretraining.

The developed system can be improved to cope with noisy images classification and it can be speeded up using GPUs.

# 4 References


[1] *Rosenblatt, Frank.* The perceptron, a perceiving and recognizing automaton Project Para / Frank Rosenblatt. — Cornell Aeronautical Laboratory, 1957.

[2] *Carpenter, Gail A.* Adaptive resonance theory / Gail A Carpenter, Stephen Grossberg. — Springer, 2011.

[3] *Broomhead, David S.* Radial basis functions, multi-variable functional interpolation and adaptive networks / David S Broomhead, David Lowe. — 1988.

[4] Gradient-based learning applied to document recognition / Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner // *Proceedings of the IEEE.* — 1998. — Vol. 86, no. 11. — Pp. 2278–2324.

[5] *Kramer, Mark A.* Nonlinear principal component analysis using autoassociative neural networks / Mark A Kramer // *AIChE journal.* — 1991. — Vol. 37, no. 2. — Pp. 233–243.

[6] *Hinton, Geoffrey E.* Deep belief networks / Geoffrey E Hinton // *Scholarpedia.* — 2009. — Vol. 4, no. 5. — P. 5947.

[7] *Hopfield, John J.* Neural networks and physical systems with emergent collective computational abilities / John J Hopfield // *Proceedings of the national academy of sciences.* — 1982. — Vol. 79, no. 8. — Pp. 2554–2558.

[8] *Hinton, Geoffrey.* A practical guide to training restricted Boltzmann machines / Geoffrey Hinton // *Momentum.* — 2010. — Vol. 9, no. 1. — P. 926.

[9] *Carreira-Perpinan, Miguel A.* On Contrastive Divergence Learning. / Miguel A Carreira-Perpinan, Geoffrey Hinton // AISTATS / Citeseer. — Vol. 10. — 2005. — Pp. 33–40.

[10] *Glorot, Xavier.* Understanding the difficulty of training deep feedforward neural networks. / Xavier Glorot, Yoshua Bengio // Aistats. — Vol. 9. — 2010. — Pp. 249–256.

[11] *Kingma, Diederik.* Adam: A method for stochastic optimization / Diederik Kingma, Jimmy Ba // *arXiv preprint arXiv:1412.6980.* — 2014.

[12] *Bengio, Yoshua.* Practical recommendations for gradient-based training of deep architectures / Yoshua Bengio // Neural Networks: Tricks of the Trade. — Springer, 2012. — Pp. 437–478.