Jonas Johansson jonasjo5@kth.se 890829-2975
Marcus Jonsson Ewerbring marcusew@kth.se 950328-7337

Neural network in Java

1. Initial study

Training

Functional requirements:

Given training data consisting of .png files, the system should be able to perform supervised learning.

The network will be able to handle grey scale images of numbers between 0-9 and empty (black) images. The images must have a black background and the numbers must be written in white.

After each epoch the system will output the amount of correct answers, number of files tested and elapsed time in seconds.

After a completed training session, the network is saved.

Non-functional requirements:

The whole program must be written in Java, only using the Java standard libraries with no addons.

The save function must not overwrite previously saved networks.

The training of 10 000 .png-files (one epoch) with a resolution of 28x28 pixels must not take longer than 5 minutes.

When tested against MNIST database of 50 000 handwritten numbers between 0-9 and 5000 additional black images, the network should have a success rate of at least 85% after 10 epochs. http://yann.lecun.com/exdb/mnist/

Using the network

A user can graphically choose and load a previously saved network.

A user can graphically choose and load a .png-file.

When a .png-file has been loaded, the predicted number and mean cost of the output layer should be shown as output together with the image file.

After a result has been shown, the user should be able to choose another .png-file.

Jonas Johansson jonasjo5@kth.se 890829-2975
Marcus Jonsson Ewerbring marcusew@kth.se 950328-7337

Non-functional requirements:

When a 28x28 .png image has been loaded, the system should show the result within a second.

Additional functionality, could be implemented if time permit

Functionality to detect several numbers in one image.

2. Design

We designed the neural network so that each perceptron in the input layer represents a pixel in the image loaded from the training data. The network consists of 1 input layer with as many perceptrons as the image has pixels, 2 hidden layers each with 30 perceptrons and an output layer with 11 perceptrons.
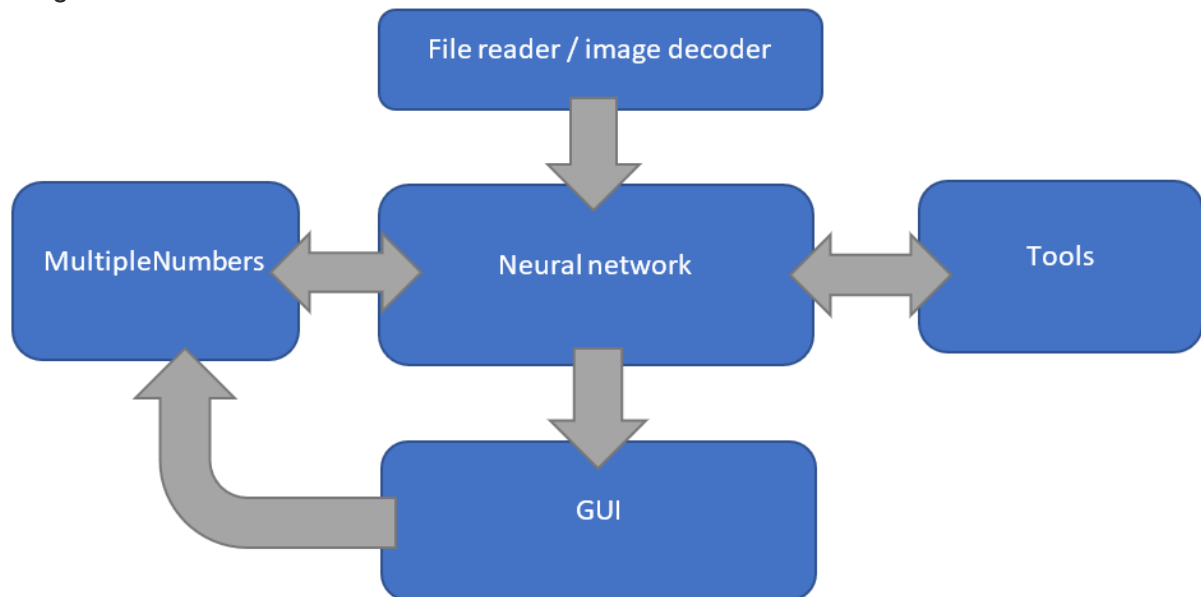
In the output layer perceptron 0-9 is used to output number 0-9 and perceptron 10 is used to recognize black images.
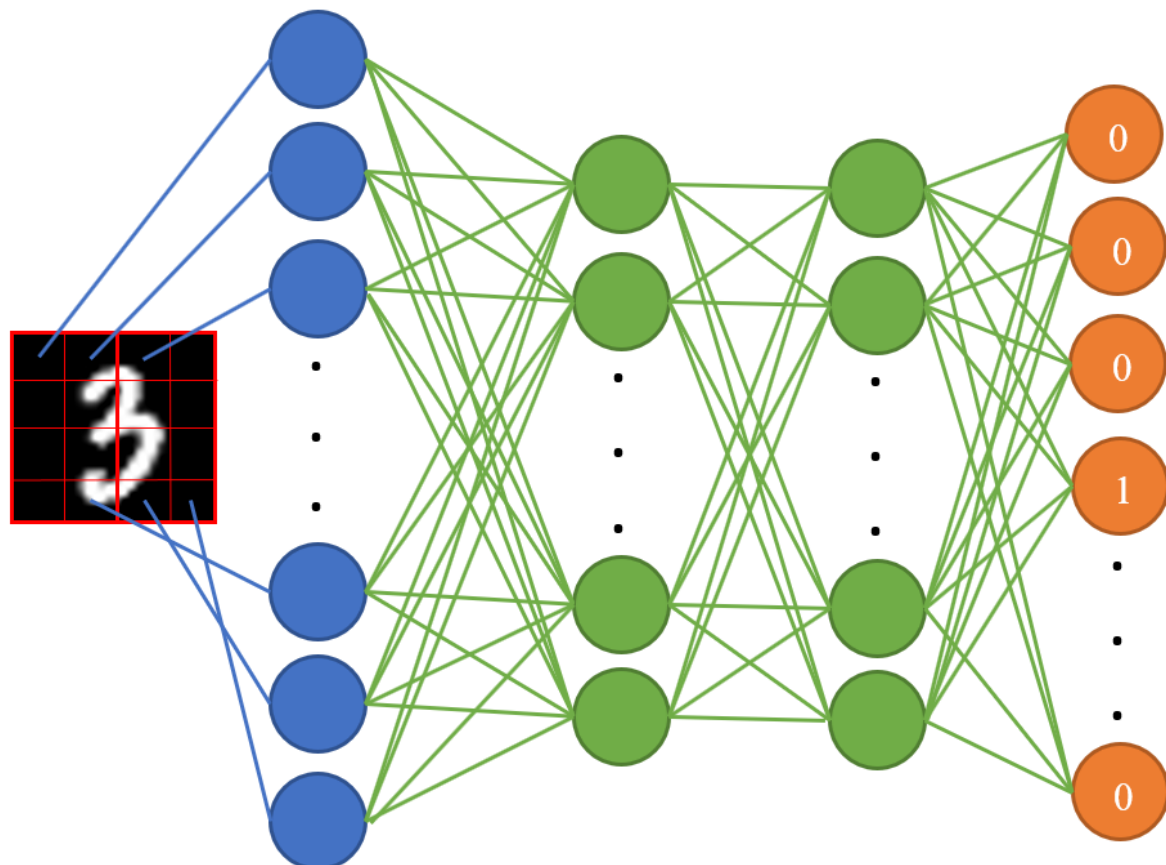
Picture 1

The prototype has a module for file handling and reading images, a neural network core, a tool module with support functions, a GUI and a module for handling multiple numbers in one

Jonas Johansson jonasjo5@kth.se 890829-2975
Marcus Jonsson Ewerbring marcusew@kth.se 950328-7337

image.

Picture 2

3. Implementation

The entire project has been implemented in java and it consists of several classes. These classes are presented below.

Classes:

- Interface, ActivationFunction
- ReLu
- Sigmoid
- NeuralNetGraphics
- Dataset
- Readimage
- TestRead
- Layer
- MultipleNumbers
- Neural_Network
- Perceptron
- Holder
- MyMath
- SaveAndLoadNetwork

Activation Functions

Activationfunction is an interface that the neural-network use, it enables the program to switch between different activation functions by extending the interface. In the program we have defined two activationfunction sigmoid and relu. All activation functions has to have the following methods: calculateActviation(double x) and getDerivative(double x).

Jonas Johansson jonasjo5@kth.se 890829-2975
Marcus Jonsson Ewerbring marcusew@kth.se 950328-7337

CalculateActivation(double x) returns the activationfunction applied on x and getDerivative(double x) returns the derivative of the function applied on x.

Graphics

The project contains one simple graphics class NeuralNetGraphics, the only purpose of this class is to give a visual interface of how the neural net is interpreting the image. If the input image has the pixel size 28x28 then the program will call the chosen neural network one time and print out the answer and the cost. If the picture width or height is bigger than 28 the program will apply the probabilistic class multipleNumbers on the image.

NN

The neural network consists of 4 classes, Layer, MultipleNumbers, Perceptron and Neural_Network. The Neural_network has two functions, train a network and run a loaded network. The training process consist of 4 steps:

- Initialize the network
- Forward propagation
- Backward propagation
- Save Network

First the network initializes all the required data structures, each layer contains a number of perceptrons and between each layer it is a matrix that represent the weight of each edge. This is visualized in picture 3.
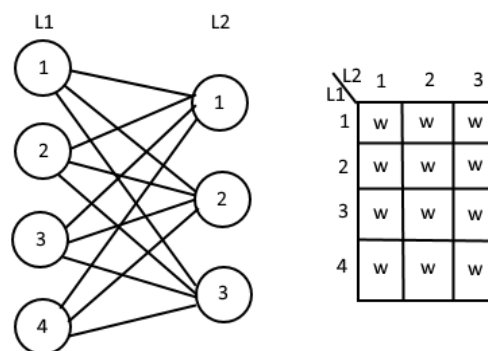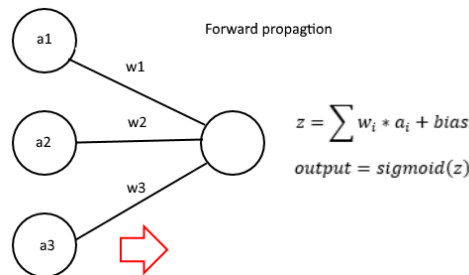

Picture 3

After the initialization the training process begin. The network gets a random image from the training data and a solution that it decodes in to the input layer. The assigned value to the perceptron in the input image is calculated by following formula: (r+g+b)/(255*3) where r = red value, g = green value and b = blue value, this will result in a value between 0 and 1. The program uses forward and backward propagation to calculate the error in the weights and biases. This process is repeated for each training image in a random order.

Forward propagation is done by calculating the output value for each perceptron in the hidden layer and output layer. The formula for the calculation is following:

$$z = \sum w_i * a_i + bias$$

Jonas Johansson jonasjo5@kth.se 890829-2975
Marcus Jonsson Ewerbring marcusew@kth.se 950328-7337

$$output = sigmoid(z)$$

the z value is stored in the perceptron because it is used later in backpropagation.
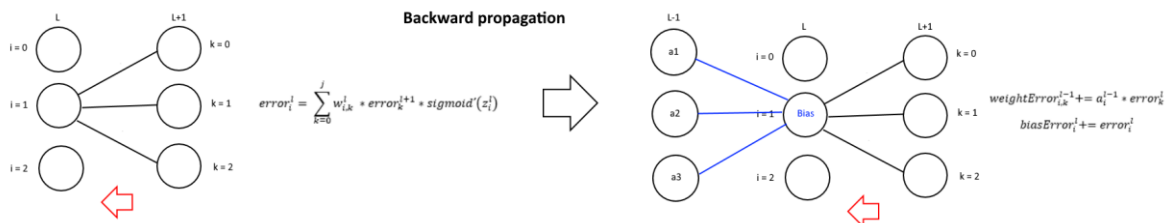

Picture 4

The backpropagation uses gradient descent to calculate the adjustments. The function is calculated by 2 steps, first calculate the error in the output layer and secondly calculate the error in the hidden layer. The error is used to calculate how much the weights and biases should be adjusted. This is the formula for the hidden which is represented in picture 5:

Note: we show the layers as elevated, this however not an exponential to the variables.

$$error_i^l = \sum_{k=0}^{j} w_{i,k}^l * error_k^{l+1} * sigmoid'(z_i^l)$$
$$weightError_{i,k}^{l-1} += a_i^{l-1} * error_k^l$$
$$biasError_i^l += error_i^l$$

Picture 5

The formula for the output layer uses the derivative of a cost function to calculate the error, we used the quadratic formula where the cost is calculated by: $(a_i^{Output\ L} - y_i)^2$ and the derivative is: $2 * (a_i^{Output\ L} - y_i)$ the error is then calculated by:
$$error_i^{Output\ L} += \left(a_i^{Output\ L} - y_i\right) * sigmoid'(z_i^{Output\ L})$$

The error is then used to adjust the weights between the last layer and the output layer and to adjust the biases at the output layer.

$$weightError_{i,k}^{Output\ L-1} += a_i^{Output\ L-1} * error_k^{Output\ L}$$
$$biasError_i^{Output\ L} += error_i^{Output\ L}$$

Jonas Johansson jonasjo5@kth.se 890829-2975
Marcus Jonsson Ewerbring marcusew@kth.se 950328-7337
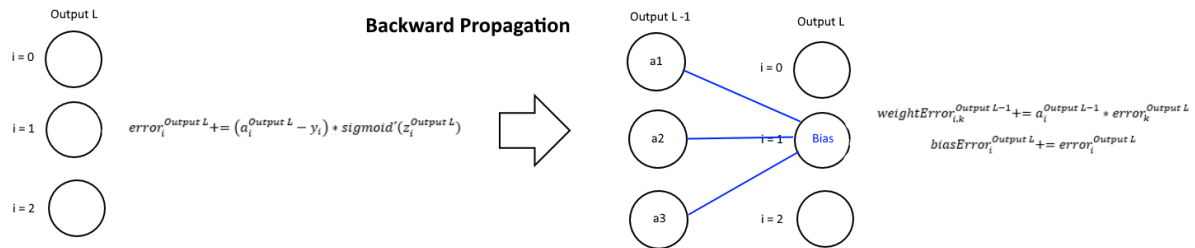
Picture 6

The training is applied in a series of mini batches where each batch contains the result of 10 learning sessions. The error is multiplied by (learning_rate/size_of_minibatch) the advantage we get by using this method is that we do not need to apply training after every image and speed up the process. We can also adjust the learning_rate to get the network to avoid local minimum in the gradient descent by increasing it or decrease it if the program jumps to far. Formula for training:

$$w_{i,k}^l -= \frac{learning\ rate}{minibatch\ size} * weightError_{i,k}^l$$
$$bias_i^l -= \frac{learning\ rate}{minibatch\ size} * biasError_i^l$$

multipleNumbers

The class multipleNumbers is used by the graphics class to analyze an image that is larger the 28x28 pixels. The class cuts out a 28x28 for every x and y position in the image, for example if we have a 28x30 image we will get 3 images. Each of these images are use as input to the neural network and retrieves the results. The images with the lowest cost that does not overlap with each other are picked. This gives us a greedy algorithm.

ImageTools

The class readImage reads a directory and creates a list of search paths to images. The directory has to have subfolder because they are used to give the image inside a solution. All images in the folder will have the same solution and the program assumes that the subfolders 0, 1...,9 and black is in the directory.

When the function getImage is called it will return a random dataset with a image and solution, if it´s no datasets left it will return null. The list can be reset by using the command reset.

Tools

The program uses several support classes to hold information, save and load data.

Jonas Johansson jonasjo5@kth.se 890829-2975
Marcus Jonsson Ewerbring marcusew@kth.se 950328-7337

4. Testing

For training we used MNIST database of handwritten numbers. The database contains 50000 training examples and 10000 testing examples, these images were converted to png to be compatible with our code.

Training

To track the improvements of the network during training we print the results from each epoch. The result is presented as number of correct answers and number of images. Also, the mean cost of the output is printed. We use the printouts to get an estimate of how good the network performs when trying to identify handwritten numbers.

We tested that the network is saved properly after training and can be loader at a later occasion. After the training was finished, we checked that a new file containing the network was created. We closed the software and loaded the file in the gui of the TestNetwork program. We then tested the network, see below.

By running the training program 30 epochs with 10 000 images in each epoch we checked that the performance requirements were met.

To improve the network's performance, we manually tweaked the number of hidden layers, amount of perceptrons in the hidden layers, number of epochs, learning rate and size of mini batches.

Network

We also tested the network by making our own png images in paint.net. We made 3 different kinds of images to test:

Black images, without any numbers (28x28 pixels).

Images with a black background and a handwritten white number (28x28 pixels).

Images with a black background and several handwritten white numbers in the same image (various sizes).

The images were then tested through the GUI of the TestNetwork program.

Result

On 28x28 images with numbers 0-9 and empty black images, we managed to get our network to perform 96% correct answers on average. The network performs worse with several numbers within the same image. This is because our function to extract sub images sometimes crops out parts of the numbers, making it hard for the network to recognize them.

Since recognizing several numbers isn't part of our main requirements or goals for this prototype, we do not have any performance requirements either. We therefore didn't test the functionality as thorough and have no exact statistic of the performance.

Jonas Johansson jonasjo5@kth.se 890829-2975
Marcus Jonsson Ewerbring marcusew@kth.se 950328-7337
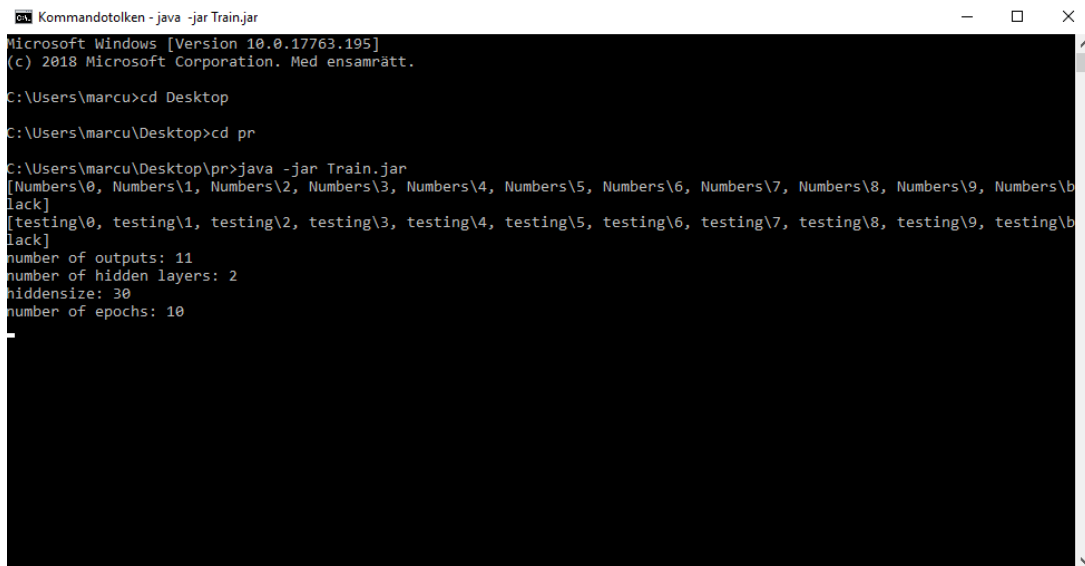
5. Delivery

How to use the program:

In the folder NeuralNetwork it should be 2 jars (Train.jar and TestNetwork.jar) and 4 directories (Numbers, SavedNetworks, Several and testing). In the Numbers folder it should be several subfolders (0,...,9 and black), all the images in these folders are used to train the network. The folder testing should also have similar structure as Numbers, but this folder is used to test the network. In the folder SavedNetorks it should be 1 pre-trained networks, the file Neural-Net-10-Epochs-2-Hidden-Black-96,08 is a network trained to also recognize black images with no numbers in. In the directory Several there should be some images with several numbers in the image.

How to train:

If you are using windows open the cmd and move to the folder where the 2 jars and the 3 folders are located. Then write following *java -jar Train.jar* and press enter, it should now appear some information about the training, see picture 7.
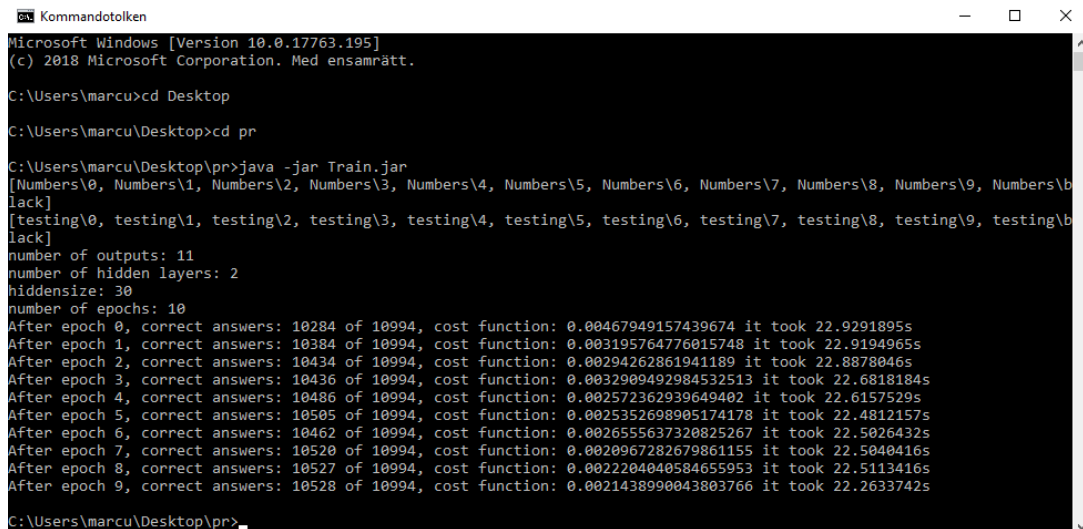
Picture 7

After a while, depending on how fast your computer are you will get a result from the epoch. The program will terminate after 10 epochs and save the network as Neural-Net in the SavedNetworks folder, if you already have a network in that folder with the name Neural-Net the program will save the network as Neural-Net1.

Jonas Johansson jonasjo5@kth.se 890829-2975
Marcus Jonsson Ewerbring marcusew@kth.se 950328-7337
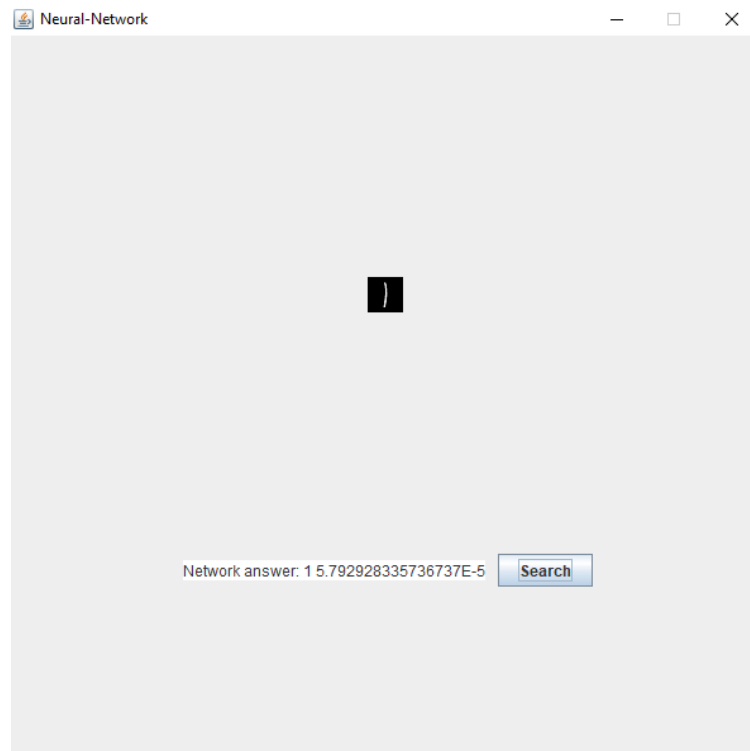
Picture 8

How to Test the network:

Run the test program by writing *java -jar TestNetwork.jar* and press enter, it should now appear a window on the screen that want you to select which network you want to use.
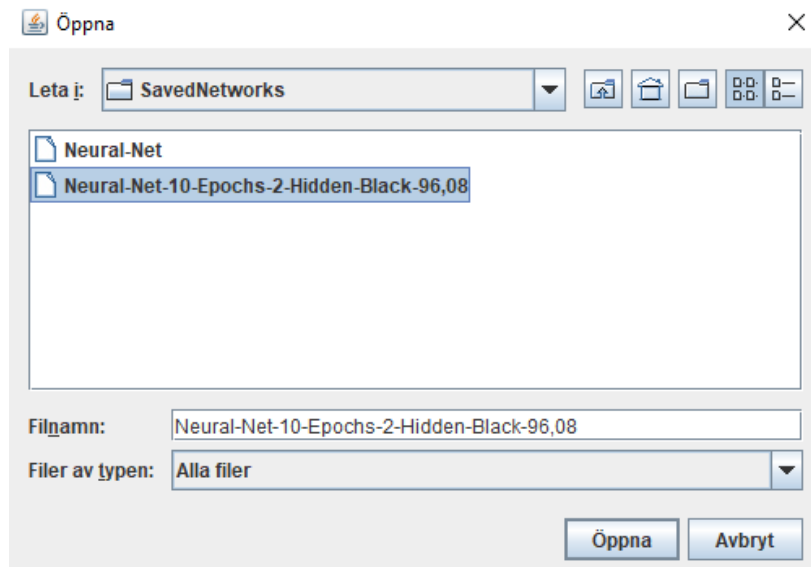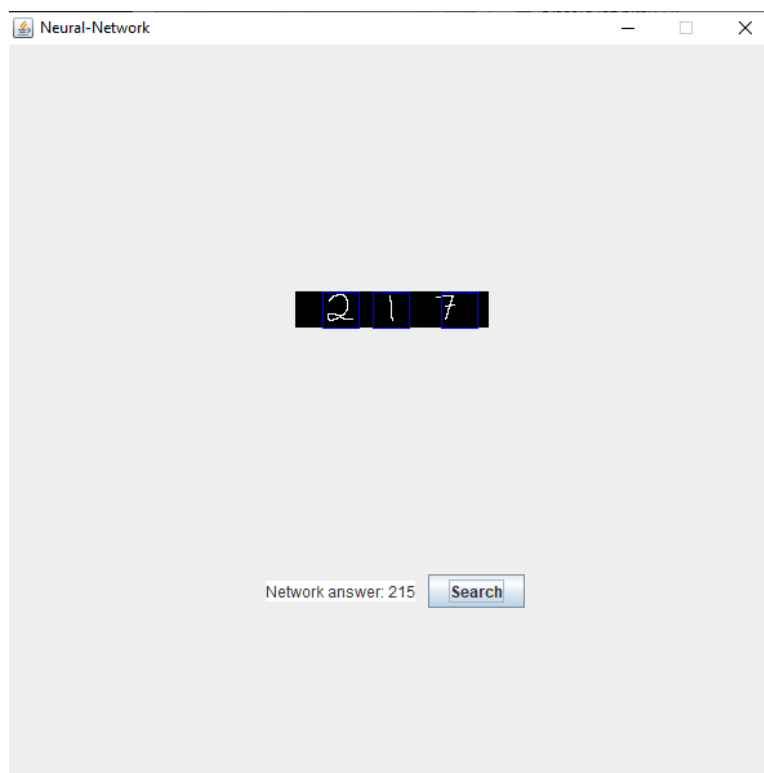

Picture 9

You should now have a blank window with a search-button in it, if you click the search-button the program will open the testing folder, select an image and the program will print what it thinks the picture is representing, the output should be interpreted as "Network answer: answer cost-function".

Jonas Johansson jonasjo5@kth.se 890829-2975
Marcus Jonsson Ewerbring marcusew@kth.se 950328-7337

Picture 10

If you select an image that is larger than 28x28 the program will print blue squares at the position the program thinks the number is located. We highly recommend using an image with the size: height = 28 and width >= 56 to try this function.

Picture 11

Jonas Johansson jonasjo5@kth.se 890829-2975
Marcus Jonsson Ewerbring marcusew@kth.se 950328-7337

Sources used in the project

Training and test data:
http://yann.lecun.com/exdb/mnist/

Links to homepages where we learned the math used in the project.
http://neuralnetworksanddeeplearning.com/chap1.html
http://neuralnetworksanddeeplearning.com/chap2.html
https://www.youtube.com/watch?v=aircAruvnKk
https://www.youtube.com/watch?v=IHZwWFHWa-w&index=3&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi&t=1018s
https://www.youtube.com/watch?v=Ilg3gGewQ5U&index=4&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi&t=477s
https://www.youtube.com/watch?v=tIeHLnjs5U8&index=5&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi&t=267s