Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Irányítástechnika és Informatika Tanszék

Ádám Czifra

ITELLIGENT VISION WITH DEEP LEARNING

Traffic Sign Recognition

CONSULTANT

Márton Szemenyei

BUDAPEST, 2016

Tartalomjegyzék

1 Introduction

Today, the autonomous vehicles are in the spotlight. A lot of IT companies and automobile manufacturers are working with great efforts to revolutionize the 21st century's personal transportation.

Deep learning allows computational models that are composed of multiple processing layers to learn representations of data. Deep convolutional networks have brought about breakthroughs in processing images and videos.

My project laboratory's purpose was to create a deep learning system, which can successfully recognize traffic signs in Hungary.

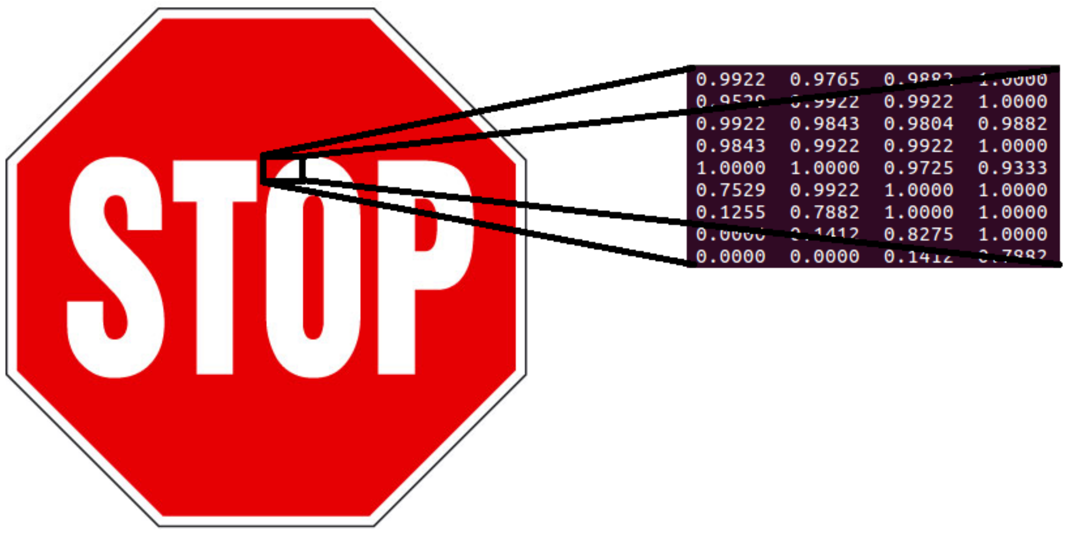
The study presents the basics of image classification and neural networks. Furthermore, it shows the advantages of torch framework. Moreover, it describes the challenges of the training and test datasets, then it presents how can you make a deep learning network. Later, it evaluates the predictions of the network.

At last, after summarizing experiences and evaluating solutions, the study presents the possibilities of further improvement and future plans.

1.1 Image Classification

In this section I will introduce the image classification problem, which is the task of assigning one label from a fix set of categories to an input image. This is one of the core problems in Computer vision.

Our task is to predict probabilities of labels for the images that we give to our computer. For example, we give an image of a stop sign to our classification model. We have to keep in mind that our computer will see a picture as 3-dimensional array of numbers.



1. figure: Stop sign image and how the computer sees it

In this example our stop sign is 400 pixels wide, 407 pixels tall and has 3 color channels because of it's an RGB (Red, Green, Blue) image. So it consists of 400 * 407 * 3 numbers, total of 488 400 numbers. Our task is to turn these numbers to probabilities of our fixed labels. Such as:

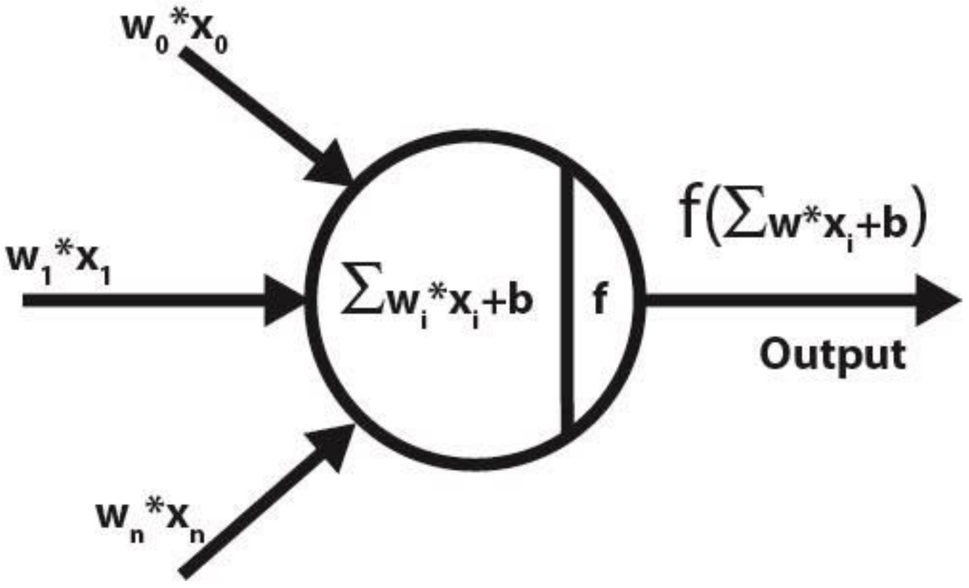
- 82% stop sign
- 16% no entry sign
- 2% speed limit sign

In this example we can assign “stop sign” label to our image, because of our predictions.

We have seen how our computer sees pictures, now I want to introduce the pipeline of the classification. At first, we need an input, this will consist of images, each labeled with one of our different classes. We will call this data as the training set. Then we need to use the training set to learn how our classes looks like. We will call this step the learning. At last, we will evaluate the quality of our learning model by asking it to predict labels to a new set of images, that has never seen before. We will call this set of images as the test set. Then we will compare this predicted labels with the true labels of each image.

1.2 Neuron and linear classification

In this section I want to introduce the basics of a single neuron and the relationship between a neuron and linear classification. Linear classification has 2 major components: score and loss function. Score function maps the data and compute class scores. Loss function quantifies the agreement between the predicted scores and the ground truth labels.



2. figure: Mathematical model of a single neuron

In the equation, which can be seen on the 2. figure, $\mathbf{x_i}$ is our input, for example an image. The **W** is a matrix, which is often called as weights. The **b** is a vector, which is called as the bias vector, because it influences the output, but without interacting with our input $\mathbf{x_i}$. **W** and **b** is our learning parameters, we can control the setting of the values of these parameters. Our goal will be to set these values in such way, that the computed class scores match our ground truth labels across the whole training set. The score of the correct class should be higher than the scores of incorrect classes. **f** is called as the activation function, which well set our output value.

The advantage of this method is when the learning is completed, we can discard the entire training set and only keep the learned parameters. That is because the images of the test set will be classified based on the computed class scores. Other advantage is that this make our evaluation faster, because we only use matrix multiplications instead of comparing the test set to all our training images.

Neural networks are modelled as collections of neurons that are connected in a graph. In other words, the output of a neuron is an input to other neurons. Neural network models are often organized into layers of neurons. Output layer is unlike all layers in neural network, because usually it doesn’t have an activation function. This is because the last layer often represents the class scores.

1.3 Torch

Torch is a scientific computing framework with wide support for machine learning algorithms. It is easy to use and efficient, thanks to an easy and fast scripting language, LuaJIT. The biggest advantage is that torch is open-source, which means anybody can develop new features and you can find a lot of libraries and function, which you can use immediately.

It has the following core features:

- a powerful N-dimensional array
- lot of routines for indexing, transposing, etc.
- linear algebra routines
- neural network models
- optimization routines
- fast and efficient GPU support
- Embeddable, with ports to iOS and Android

2 Training and test set

At first, we need to create a training set of images, which we can later train to our Neural Network. This set should contain a large variety of pictures, because we can have a lot of challenge in the classification. For example:

- Viewpoint variation
- Deformation
- Occlusion
- Illumination conditions

The training set should contain images, that have some of these challenges. We can create this challenges manually too, with adding noise and brightness to our original image, or just cutting a part of the image.

For training set, we need a huge dataset, with a large variety of conditions. Fortunately, there is no big difference between the traffic signs in the countries of Europe, therefore I could use some other countries big datasets, which I found through the internet. I sorted out the signs, which are not in use in Hungary, in this way I could made a Hungary-specific dataset.

The problem with these databases was, that the pictures had very different sizes. So at first I had to make them to the same size, I resized all the images to 32x32. I tried to compose my dataset such way, that it contains more challenges, therefore I modified some pictures (add plus noise and brightness). In this way I could make a large training dataset with 4311 images, which are consist of 42 type of traffic signs.

3. figure: Training dataset

Similarly, I created a test dataset. These images also consist of 42 type of traffic signs and this database contains a total of 2156 pictures.

3 Neural network

In this section, I want to introduce the neural network, that I used. It contains 9 layers, which are the following:

```
net = nn.Sequential()
net:add(nn.SpatialConvolution(3, 6, 5, 5))
net:add(nn.ReLU())
net:add(nn.SpatialMaxPooling(2,2,2,2))
net:add(nn.SpatialConvolution(6, 16, 5, 5))
net:add(nn.ReLU())
net:add(nn.SpatialMaxPooling(2,2,2,2))
net:add(nn.View(16*5*5))
net:add(nn.Linear(16*5*5, 42))
net:add(nn.LogSoftMax())
```

3.1 SpatialConvolution()

The convolution layer is the core module of a Convolutional Network, that does the most of the computation. The layer applies a 2D convolution over an input image composed of several input plane. First parameter is the number of expected input planes (n_1) in the image given into, the second parameter is the number of output planes (m_1), which the convolution layer will produce. The third and fourth parameter is the convolution kernel width (p_1) and height (p_2).

$y_i = x * k_i + b_i,$ i = 1, 2, .. , m_1 (*: convolution operator)

- x: 3D tensor: n_1 2D n_2 x n_3 (for example the image input, n_1 = 3 (RGB) n_2=width, n_3=height)
- y: 3D tensor: m_1 2D m_2 x m_3 (y_i feature maps (m_2 x m_3)
- k: 4D tensor: m_1 3D n_1 x p_1 x p_2 (kernels k_i (i = 1, … ,m1)
- b: 1D tensor: m_1 bias

The k and b parameters are the learnable parameters, the kernels and biases are first filled with random values, which are later modified to improve the class scores. In our network the output image size of the convolution layer can be calculated by the following method:

$$m_2 = n_2 - p_1 + 1 \qquad m_3 = n_3 - p_2 + 1$$

3.2 SpatialMaxPooling()

The pooling layer function is to progressively reduce the size of the representation to reduce the amount of parameters and computation in the neural network. The module applies 2D max-pooling (select the max value) operation in width x height regions by step size dWidth x dHeight steps. The number of output planes is equal to the number of input planes. The first and second parameters of the function are the width and height values. The third and fourth parameters are the dWidth and dHeight values.

3.3 ReLU()

The Rectified Linear Unit computes the function $f(x) = \max(0, x)$. So the module simply is zero when x <= 0 and then linear.

3.4 View()

This layer creates a new view of the input tensor using the parameter passed to the constructor. In this network, the layer reshapes our 3D tensor (16x5x5) into 1D tensor of 16*5*5.

3.5 Linear()

This module applies a linear transformation to the incoming data. The input tensor must be a 1D or a 2D tensor, this is why we had to use View() as the previous layer.

3.6 LogSoftMax()

The last layer is a LogSoftMax layer, this layer interprets the scores as unnormalized log probabilities for each class and then encourages the normalized log probability of the correct class to be high. LogSoftMax is defined as:

$$q(y_i) = \ln\left(\frac{e^{f_{y_i}}}{\sum_k e^{f_k}}\right)$$

This expression can be interpreted as the normalized log-probability assigned to the correct label y_i.

3.7 Criterion

When you want a neural network model to learn something, you give it a feedback on how well it is performing. Loss function computes an objective measure of the model's performance A typical loss function takes in our model's output and the ground truth and computes a value that quantifies the network's performance. The model then corrects itself to have a smaller loss.

In my network I used Negative Log-Likelihood criterion, because it is suitable for classification problems. This function expected to contain log-probabilities of each class, this is easily achieved, because our last layer is the LogSoftMax layer. The loss can be described as:

$$loss(pred, class) = -pred[class]$$

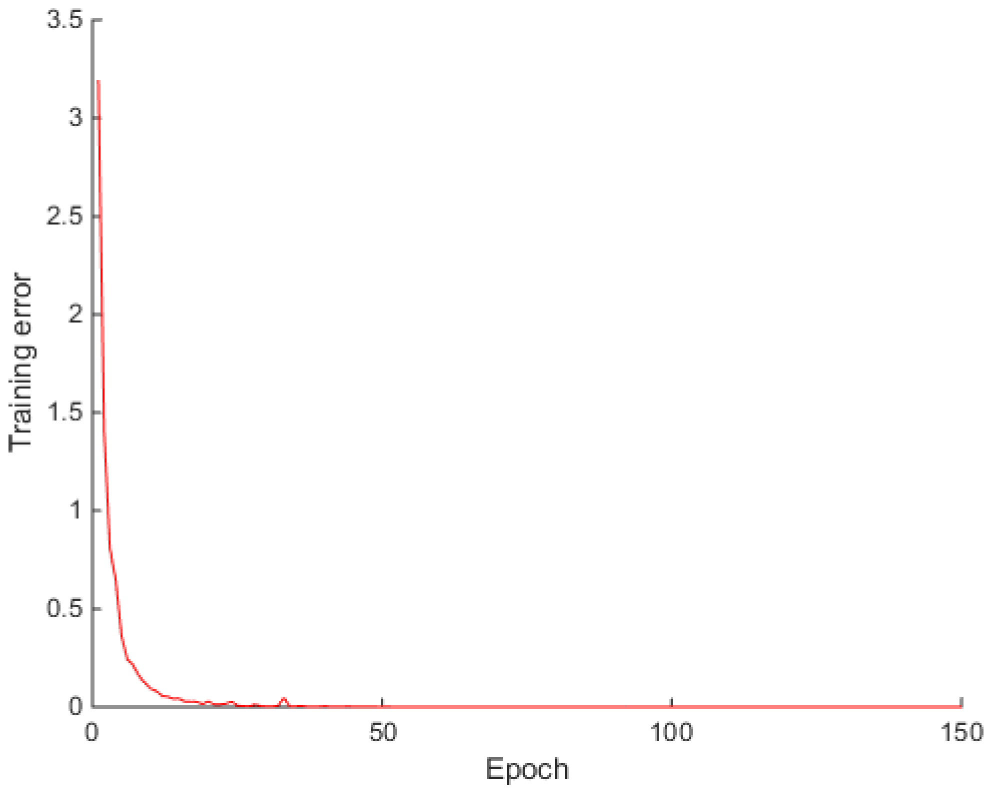
In the equation, "pred" is the predicted scores to our classes and "class" is the desired output (an integer from 1 to the number of our classes).

4 Training and evaluation

In this section, I want to present the result of the training method. At first, I would like to speak about the training error, then the accuracy of the model.

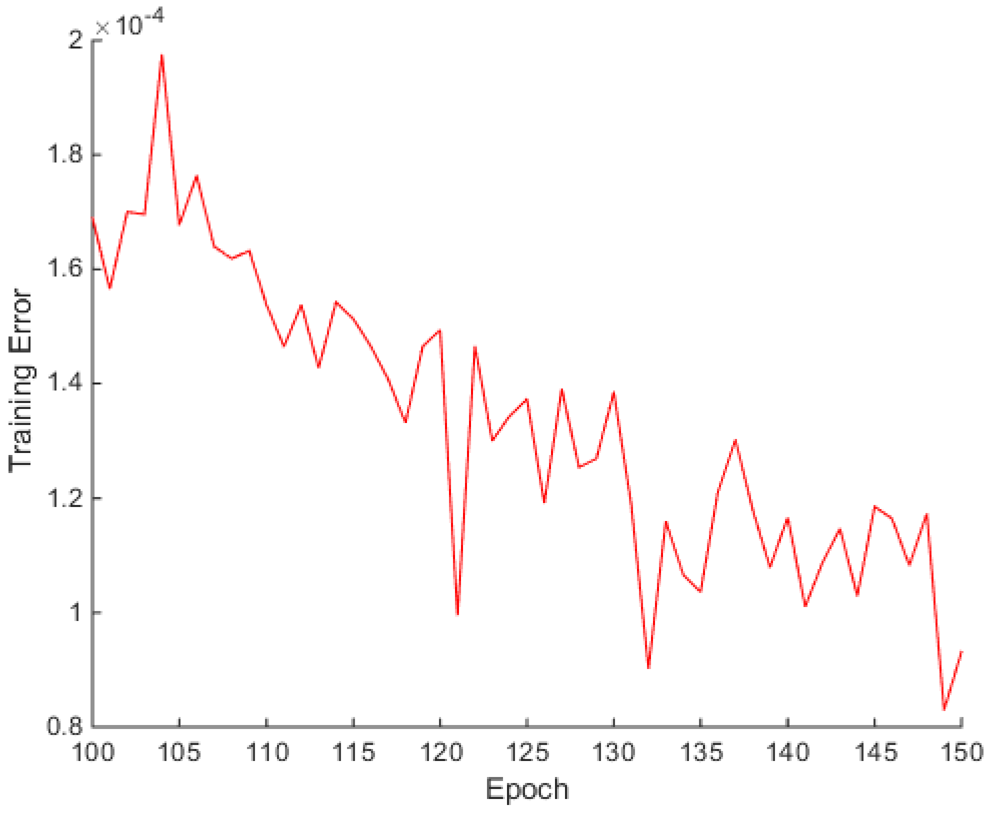
4.1 Training error

As we know, a model is trained by minimalizing its loss and maximizing its accuracy on the training dataset. I trained the model until 150 epochs. We can see the training error between 1 and 150 epochs on the following figure:



4. figure: Training error

On the 4. figure, we can see, that our error is decreasing and after 50 epochs it is very close to 0. This is very good, but how close is it, to zero?

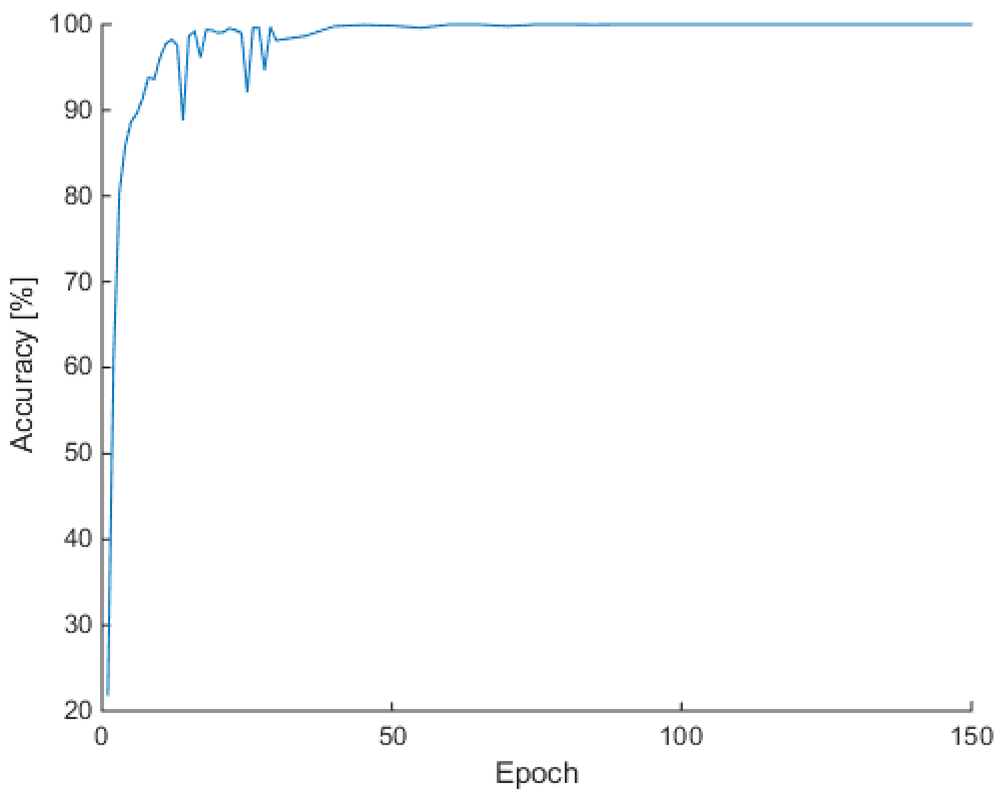


5. figure: Training error between 100 and 150 epochs

I've zoomed in a bit, and on the 5. figure, we can see that our training error is still decreasing between 100 and 150 epochs. Therefore, we can say, that our training is continuously improving.

4.2 Accuracy on the training set

Now let's see what is the network's accuracy on the training set. In this case, I give the training set to the trained model, and it will predict labels to each image, after this I’ll compare the results to the ground truth labels. Therefore, I could determine the accuracy of the network.

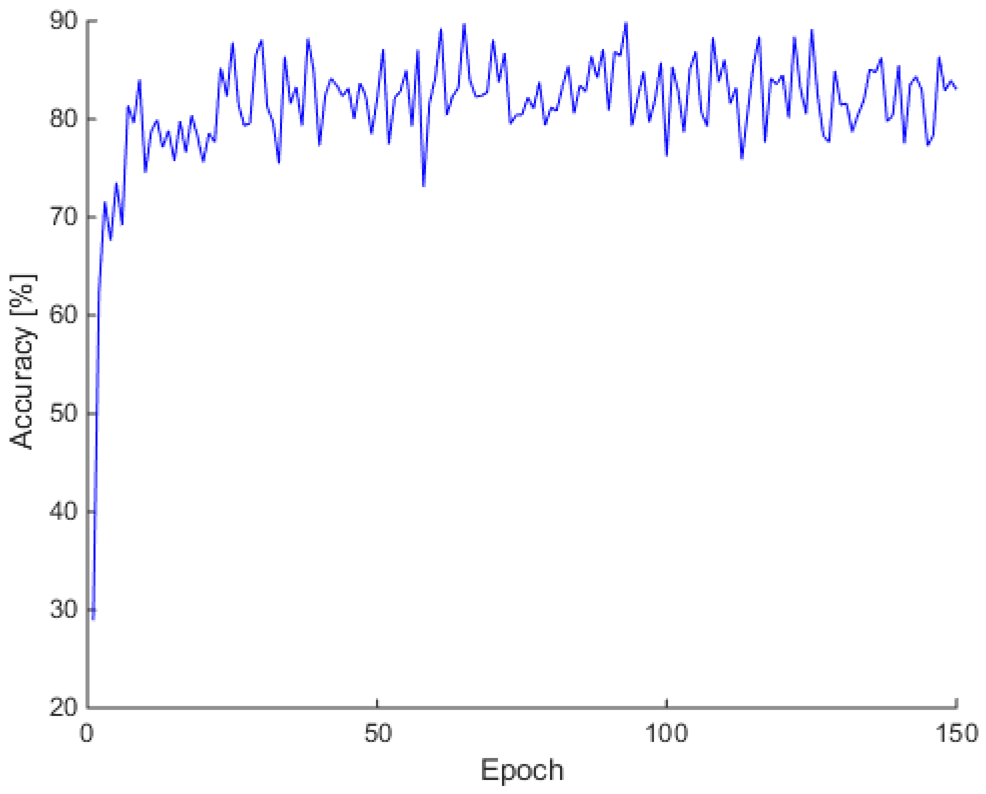


6. figure: Accuracy on the training set

On the 6. figure, we can see that our model has reached the 100% accuracy around 50 epochs. After 50 epochs, the accuracy is not changing, it will stay continuously at 100%. This means, that the predictions of the model are perfect on the training dataset.

4.3 Accuracy on the test set

In this section I want to see, what can the model do with new pictures, that it has never seen before. Therefore, I will give the test set to the model, and see how can it predict the labels to a new dataset.



7. figure: Accuracy on the test set

On the 7. figure, we can see that our accuracy is not developing after 30 epochs, it will stay around 80 %. This result is acceptable, but it is not good. We want our network to perform with more than 95% accuracy.

4.4 Overfitting

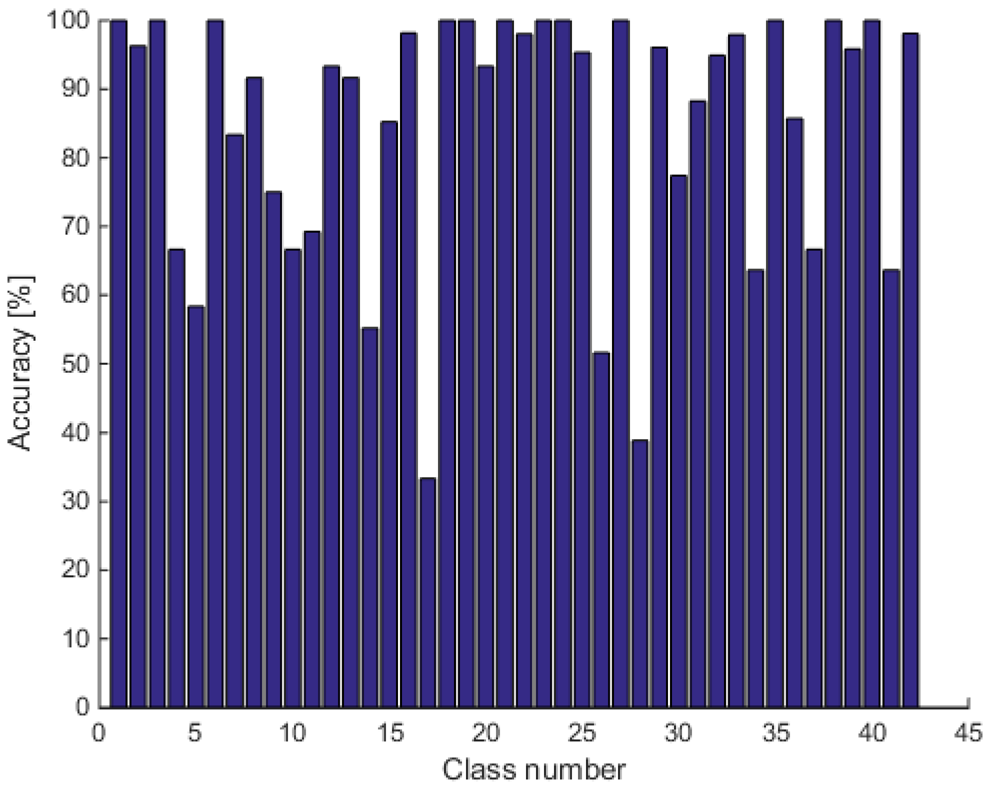
We have seen that our network correctly classified all 4311 training images. Meanwhile, our test accuracy tops out at just 89 %. So our model really is learning about peculiarities of the training set, not just recognizing digits in general. It's almost as though our network is just memorizing the training set, without understanding digits well enough to generalize to the test set. We say that our network is overfitting beyond 30 epoch.

This is because the number of parameters in our model is greater than the number of data in our training dataset, therefore the network can predict the output for training data by memorizing the entire dataset.

We can avoid overfitting by using a lot of data. In other words, we have to expand our training data set, this way the algorithm is forced to generalize and make a good network that suits all the data points.

4.5 Accuracy of the classes

In this section, let's see how are the classes performed, Furthermore, what are the classes that performed well, and the classes that didn't performed well.



8. figure: Accuracy of the classes

On the 8. figure, the classes are the following:

1 =	Uneven road	11 =	Other danger
2 =	Road hump	12 =	Road narrows on both sides
3 =	Slippery road	13 =	Road narrows on right
4 =	Bend to left	14 =	Crossroads
5 =	Bend to right	15 =	Equal crossroads
6 =	Double bend first to left	16 =	Give way to traffic on major road
7 =	Double bend first to right	17 =	Give priority to vehicles from opposite direction
8 =	Cycle route ahead	18 =	Stop and give way
9 =	Road works	19 =	No entry for vehicular traffic
10 =	Level crossing with barrier or gate ahead	20 =	No cycling

21 =	No trucks	32 =	Segregated pedal cycle and pedestrian route
22 =	No vehicles except bicycles being pushed	33 =	No waiting
23 =	No left turn	34 =	No stopping
24 =	No right turn	35 =	Priority over oncomming traffic
25 =	No overtaking	36 =	Parking place
26 =	Maximum speed	37 =	Handicapped parking place
27 =	Ahead only	38 =	Home zone entry
28 =	Keep Left	39 =	One-way traffic
29 =	Keep right	40 =	No through road
30 =	Roundabout	41 =	End of highway
31 =	Route to be used by pedal cycles only	42 =	Highway

We can see that there were classes that performed really well, such as the uneven road sign or the slippery road sign with 100 percent accuracy. However, there are 2 classes with under 50 percent accuracy, 'Give priority to vehicles from opposite direction' and the keep left sign did not perform well with 33% and 39% accuracy.

5 Overall

While I have worked on the project, I got to know the Linux environment and the basics of deep learning. Moreover, I could get an insight in the Lua programming language and the torch framework, that I used in the preparation of the deep learning model.

5.1 Evaluation

The completed deep learning system is suitable for basic traffic sign recognition, because its accuracy is over 80%. The torch framework proved to be a good choice, because creating a neural network is really simple with it.

The used images proved to be good choice, but need to expand the datasets over a total of 10 000 images to avoid overfitting.

5.2 Future opportunities

At first I want to improve the accuracy of the network, with expanding the training dataset to avoid overfitting. Moreover, I would like to improve my neural network to make better predictions.

Furthermore, I want to develop a deep learning system, that do not use images as test set, but it can recognize traffic signs in videos. After this I would like to create new functions, such as lane detection with image processing.

At last, I would like to transform an RC car to autonomous and build a test track to test the car with the created neural network.

References

[1] Belgian Traffic Sign Dataset, http://btsd.ethz.ch/shareddata/

[2] The German Traffic Sign Detection Benchmark, http://benchmark.ini.rub.de/?section=gtsdb&subsection=dataset

[3] Torch Cheatsheet, https://github.com/torch/torch7/wiki/Cheatsheet

[4] CS231n Convolutional Neural Networks for Visual Recognition, https://cs231n.github.io/