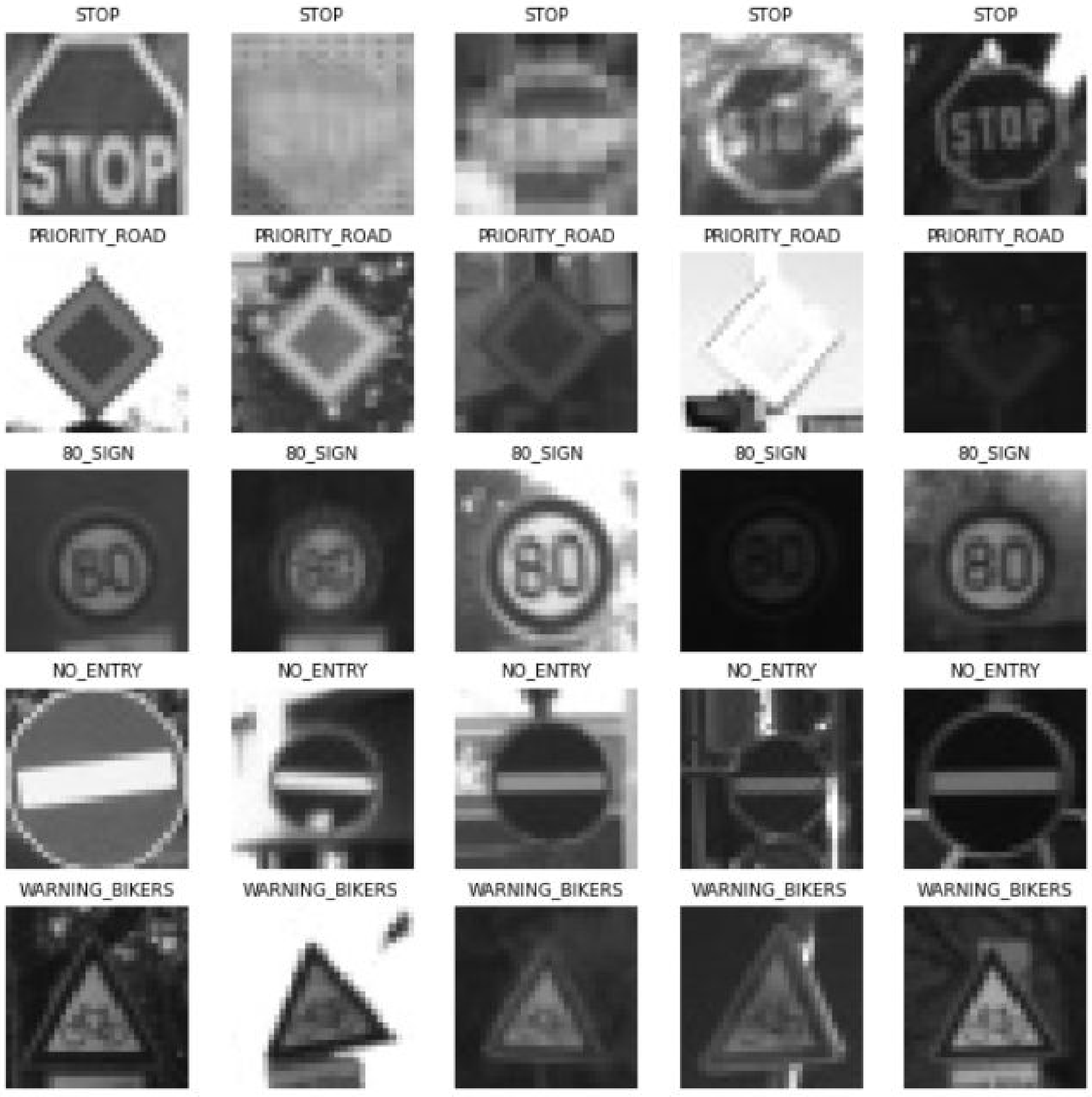
AARHUS UNIVERSITET

Detection and classification of traffic signs

Date
8.November 2020

Datasets

In this project were used 4 datasets in total merged into one. Some Classes that contain a small amount of sample images were removed. Some datasets were made mainly for object detection so using python scripts they were transferred into classic classification images. All data samples were resized into 30x30 pixels dimension and grey scaled. Dataset image examples (5 classes of 5 images):



Some of the samples are resized from very small images but that at least adds the opportunity for neural networks to learn from very bad visible signs.

Because 4 datasets were merged into one bigger one, some inequalities occured in the number of samples per class, some classes had less than 100 samples per class and others over 2000. Because of this, some classes had to be removed.

Dataset consists of pictures of traffic signs taken mostly from dash cameras attached on cars. On the internet one can find many traffic signs datasets but most of them are taken in China. Those signs look different from our european signs. According to dataset creators most of the used samples were taken in Germany and Sweden.

Dataset summary

Number of classes: 16
Number of samples: 25969
Train/Validation ratio: 90%/10%

LABEL NAME	SAMPLES
120_Sign	1488
30_SIGN	2234
50_SIGN	2384
60_SIGN	1441
70_SIGN	2145
80_SIGN	1943
GIVE_WAY	2276
NO_ENTRY	1478
PASSING_FORBIDEN	1470
PEDESTRIAN_CROSSING	1199
PRIORITY_ROAD	2292
STOP	2709
TRAFFIC_LIGHT_AHEAD	600
TRUCKS_FORBIDDEN	420
WARNING_BUMPS	390
WORK_ON_ROAD	1500

Dataset references:
https://www.cvl.isy.liu.se/research/datasets/traffic-signs-dataset/
https://github.com/hoanglehaithanh/Traffic-Sign-Detection
https://git-disl.github.io/GTDLBench/datasets/lisa_traffic_sign_dataset/
http://benchmark.ini.rub.de/?section=gtsrb&subsection=dataset

Dataset augmentation

Images contain text like “STOP”, or numbers like “80”, “100” even arrows which refer to the road direction. Because of that I cannot augment my dataset by flipping or rotating images. (Rotate may work only with some max rotate angle maybe around 25 degree). One of the possible augmentation also is adding noise to my dataset which can drastically increase the amount of data per class. But because i have enough training data ~2000 images per class i dont think augmentation is currently needed.

Framework and tools

In this project I tried to experiment with many tools that help with neural networks designing and calculations.
Main framework that was used is PyTorch which is a python based scientific library that contains many neural network functions. Together with CUDA library with nvidia i was able to push all neural network calculation into local GPU. All tests performed in this framework were running on a local GTX 1070 (8GB) graphic card.

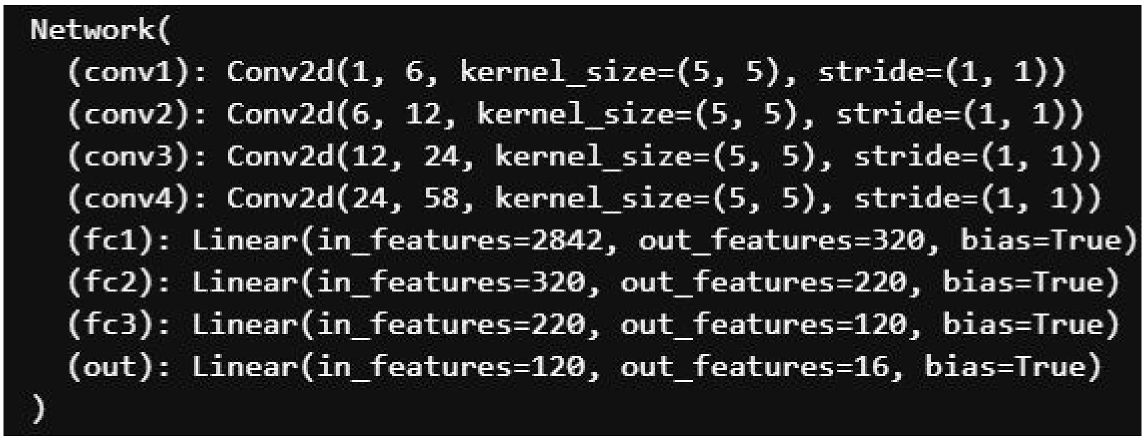
Later, YOLOv5 framework will be used for object detection which is also based on pytorch.

During the project I also discovered many interesting applications that offer services for deep learning. Website Roboflow offers dataset management but it's not free and Deep Learning studio is IDE which offers drag and drop neural network designing, training, testing and exporting it into many frameworks. Those were used just from curiosities. Everything is based on Python and PyTorch in this project.

Neural network (Classification only)

In this section various tests on Convolutional neural networks were performed and compared performance of training and validating, Tests will be performed using different learning rates, different optimizers techniques and different batch sizes.

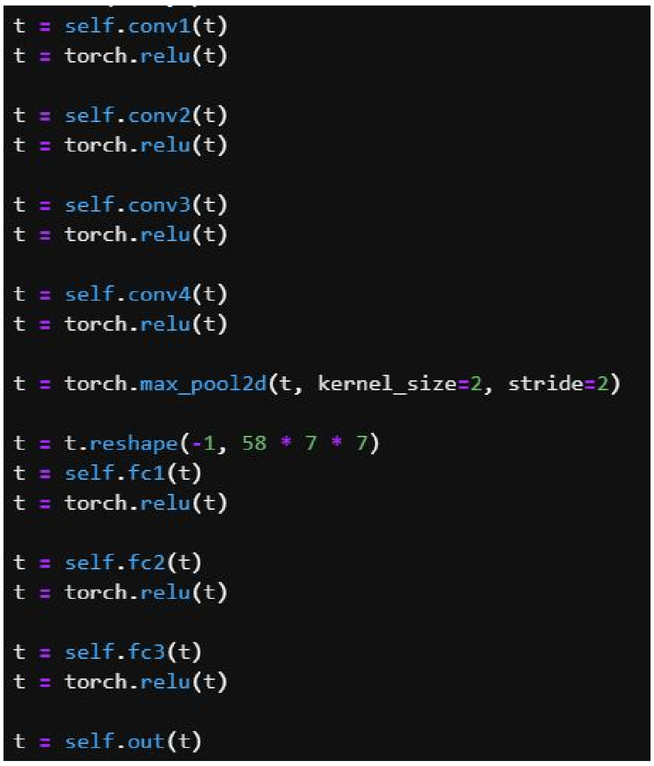
Model



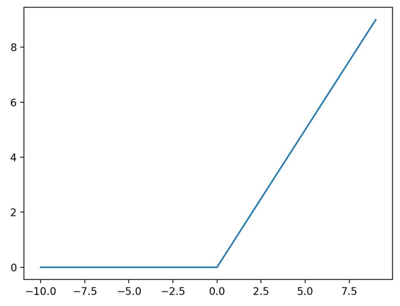
```
Network(
  (conv1): Conv2d(1, 6, kernel_size=(5, 5), stride=(1, 1))
  (conv2): Conv2d(6, 12, kernel_size=(5, 5), stride=(1, 1))
  (conv3): Conv2d(12, 24, kernel_size=(5, 5), stride=(1, 1))
  (conv4): Conv2d(24, 58, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=2842, out_features=320, bias=True)
  (fc2): Linear(in_features=320, out_features=220, bias=True)
  (fc3): Linear(in_features=220, out_features=120, bias=True)
  (out): Linear(in_features=120, out_features=16, bias=True)
)
```

Model designed for classifying signs contains 8 layers in total. 4 of them Convolutional and 4 linear fully connected layers. Between convolutional and linear layers there is “Pooling operation” which reduces the number of features to half.

“ReLU” was used as an activation function on every layer which outputs input directly if it's positive and 0 for negative values.



```
t = self.conv1(t)
t = torch.relu(t)

t = self.conv2(t)
t = torch.relu(t)

t = self.conv3(t)
t = torch.relu(t)

t = self.conv4(t)
t = torch.relu(t)

t = torch.max_pool2d(t, kernel_size=2, stride=2)

t = t.reshape(-1, 58 * 7 * 7)
t = self.fc1(t)
t = torch.relu(t)

t = self.fc2(t)
t = torch.relu(t)

t = self.fc3(t)
t = torch.relu(t)

t = self.out(t)
```



ReLU

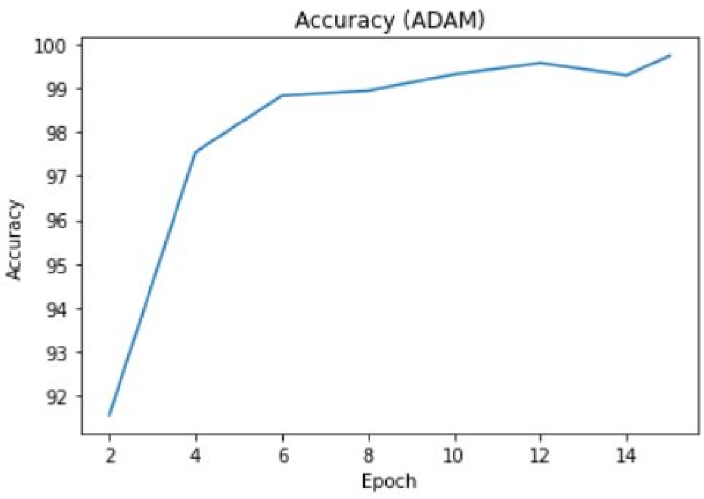
HyperParameters

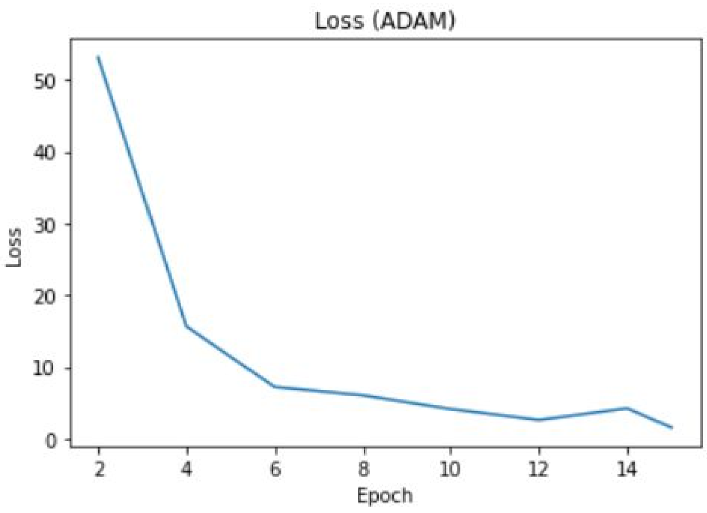
Optimizer

2 Optimizers were used in experimenting. First “Adam” which uses various techniques for adaptive learning rate. From this optimizer it is expected to work well without

specifying any of the parameters. Then we will use SGD Optimizer and we try to outperform Adam with a proper setting of learning rate and number of epochs.

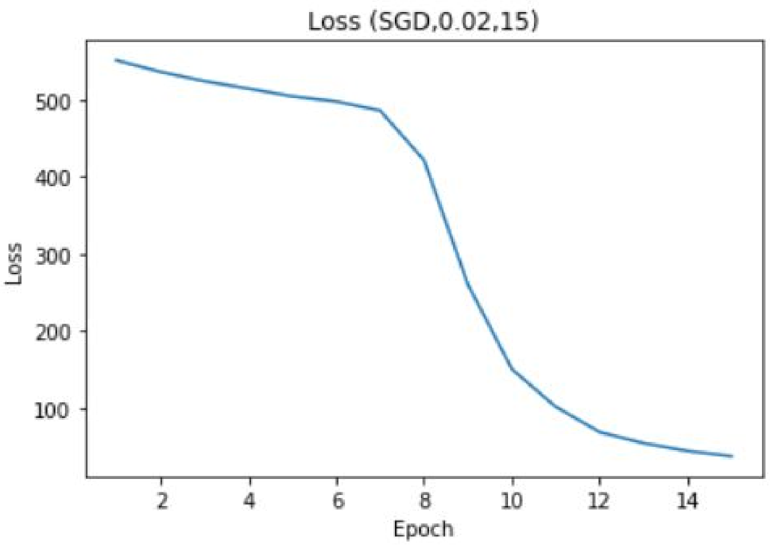
Performance

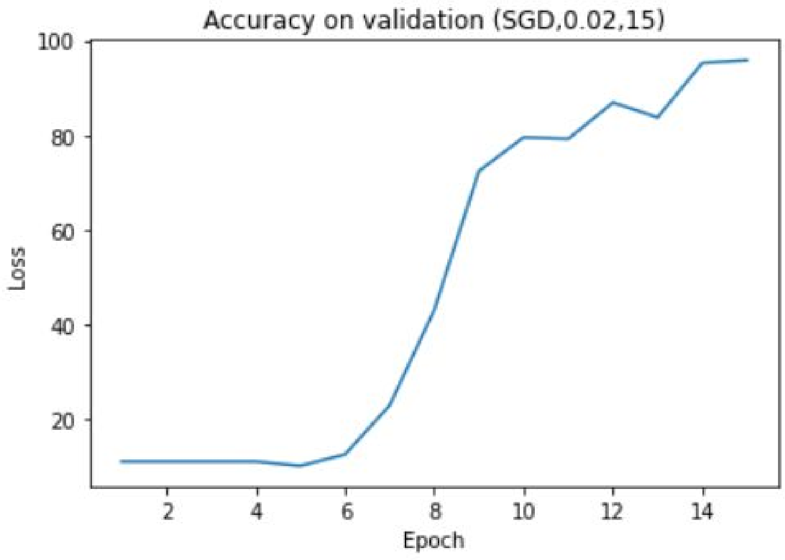




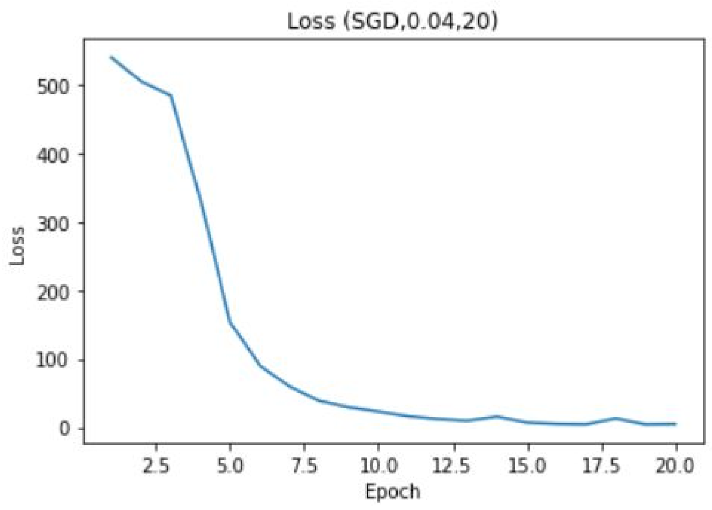
As expected, Adam optimizers perform very well, batch size is set to 125 and Accuracy after 2 epochs was over 91% which is very impressive. In a total of 15 epochs this model reached accuracy 99.73% on training data and 98.65% On validation data. Those results vary on every run mostly because of randomized initial weights and adaptivity of the optimizer.

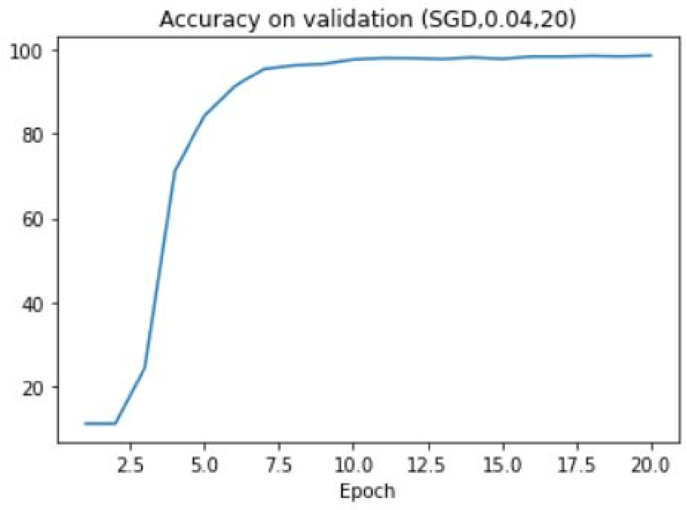
Using SGD Optimizer we need to specify learning rate and adjust the number of epochs in a way where training performs well but we do not overfit our model. Initial test with one of the very common learning rate values (0.02) and same number of epochs as for Adam performed quite well.





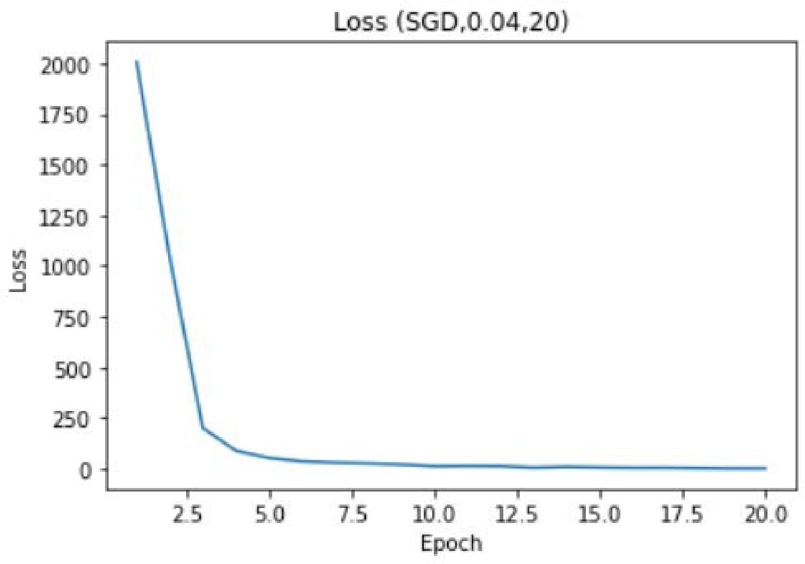
Here we can see that in the first 6 epoch there were some troubles to advance. Loss was decreasing very slow and accuracy on validation data barely improved. Later there is huge improvement. This optimizer was able to reach 95.5% accuracy on train data and 95.8% accuracy on validation data. Those results are not bad at all but adjusting the hyperparameter we can probably get better.

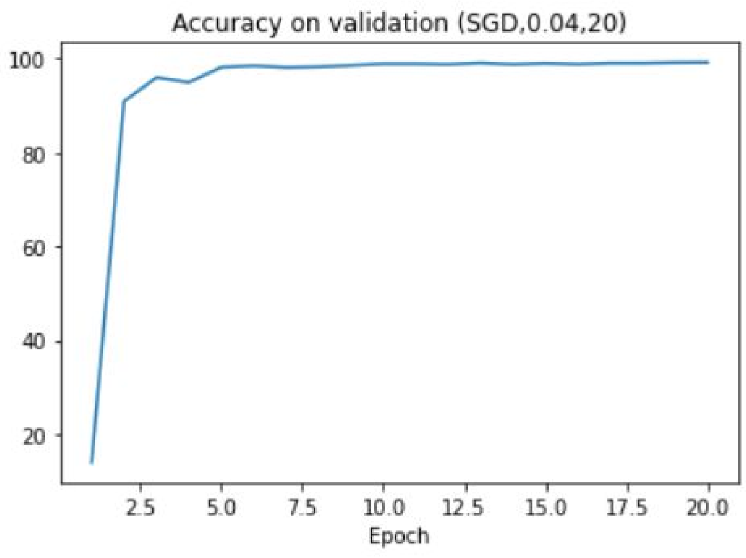




After increasing the learning rate to 0.04 and also increasing the number of epochs to 20 we managed to get slightly better performance than Adam optimizer. Increasing learning rate also fixed the problem where we struggled to decrease loss in the first few epochs. Using those parameters we reached 99.40% accuracy on training data and 98.72% accuracy on validation data.

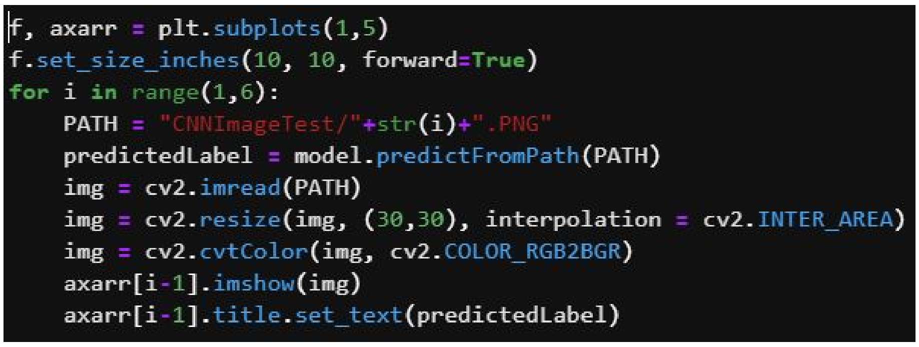
Decreasing batch size for training significantly increases time of training but we are able to get much quicker and better performance. Decreasing batch size from 125 to 32 increased accuracy of our neural network to over 99%



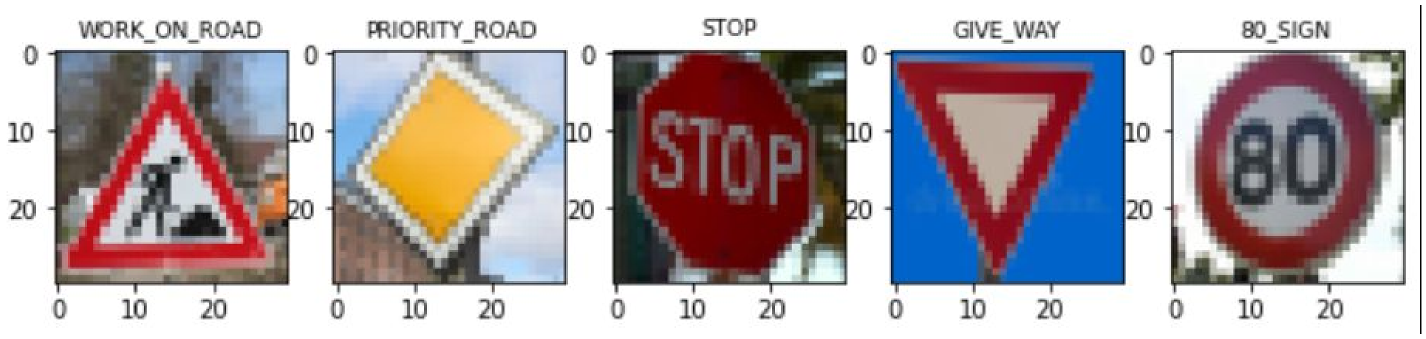


Train accuracy reached 100% and validation accuracy reached 99.80%. We can notice that accuracy was over 90% just after 2 epochs. Other 18 epochs were slightly increasing accuracy to almost 100%. More epochs could probably overfit our model and it would perform bad on validation sets. I can say that I am very satisfied with those results. Weight matrices from this test were saved. So next initialization will just load weights and training is not needed.

For a variety of tests, 5 images of signs were downloaded from the internet and forwarded through a trained network.

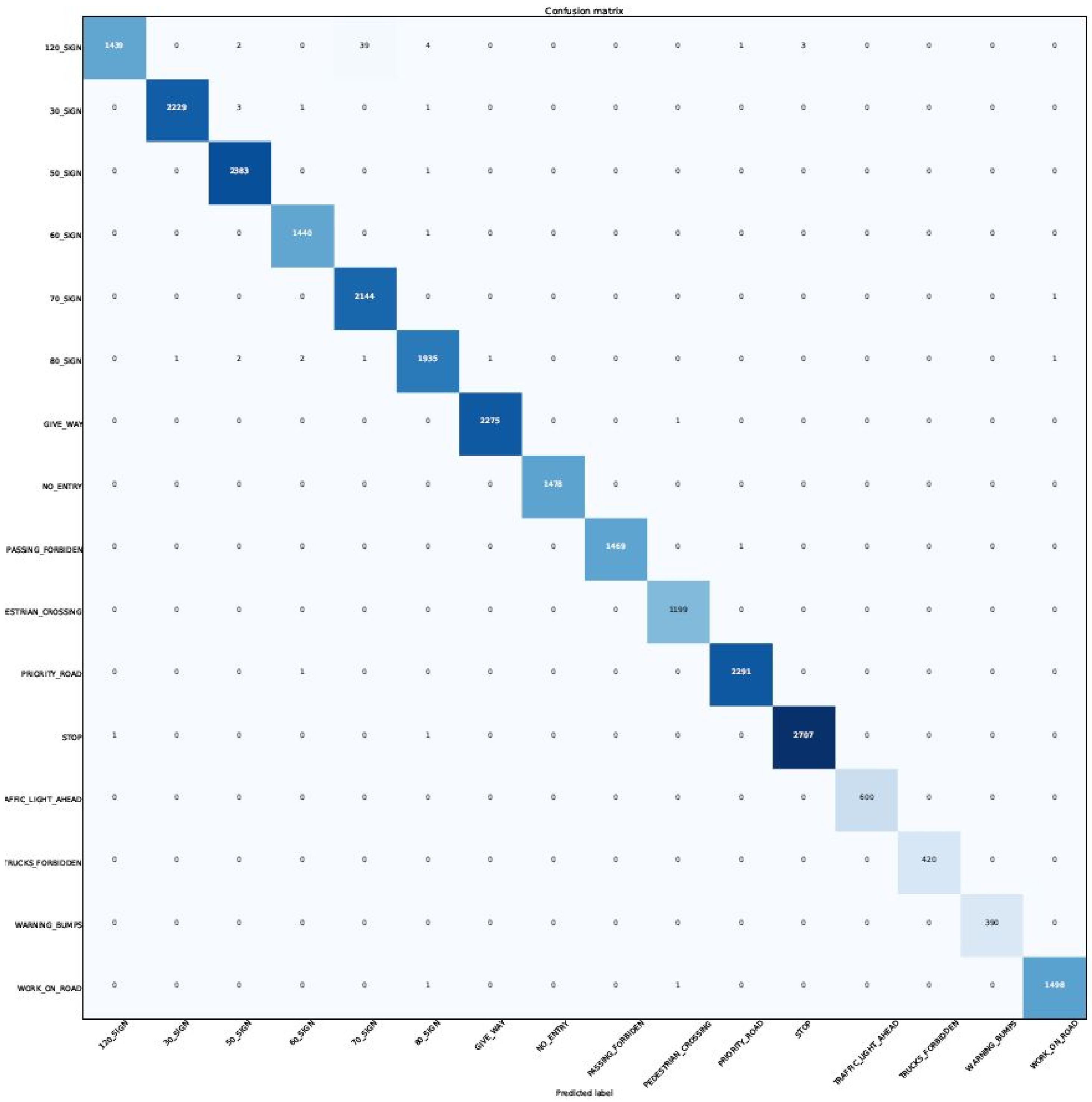


```
f, axarr = plt.subplots(1,5)
f.set_size_inches(10, 10, forward=True)
for i in range(1,6):
    PATH = "CNNImageTest/"+str(i)+".PNG"
    predictedLabel = model.predictFromPath(PATH)
    img = cv2.imread(PATH)
    img = cv2.resize(img, (30,30), interpolation = cv2.INTER_AREA)
    img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
    axarr[i-1].imshow(img)
    axarr[i-1].title.set_text(predictedLabel)
```



All 5 signs were classified correctly. Note that this model will automatically resize the image to 30x30 pixels and change the color scheme to grayscale.

Confusion matrix



	120_SIGN	30_SIGN	50_SIGN	60_SIGN	70_SIGN	80_SIGN	GIVE_WAY	NO_ENTRY	PASSING_FORBIDEN	PEDESTRIAN_CROSSING	PRIORITY_ROAD	STOP	TRAFFIC_LIGHT_AHEAD	TRUCKS_FORBIDDEN	WARNING_BUMPS	WORK_ON_ROAD
120_SIGN	1439	0	2	0	39	4	0	0	0	0	1	3	0	0	0	0
30_SIGN	0	2229	3	1	0	1	0	0	0	0	0	0	0	0	0	0
50_SIGN	0	0	2383	0	0	1	0	0	0	0	0	0	0	0	0	0
60_SIGN	0	0	0	1440	0	1	0	0	0	0	0	0	0	0	0	0
70_SIGN	0	0	0	0	2144	0	0	0	0	0	0	0	0	0	0	1
80_SIGN	0	1	2	2	1	1935	1	0	0	0	0	0	0	0	0	1
GIVE_WAY	0	0	0	0	0	0	2275	0	0	1	0	0	0	0	0	0
NO_ENTRY	0	0	0	0	0	0	0	1478	0	0	0	0	0	0	0	0
PASSING_FORBIDEN	0	0	0	0	0	0	0	0	1469	0	1	0	0	0	0	0
ESTRIAN_CROSSING	0	0	0	0	0	0	0	0	0	1199	0	0	0	0	0	0
PRIORITY_ROAD	0	0	0	1	0	0	0	0	0	0	2291	0	0	0	0	0
STOP	1	0	0	0	0	1	0	0	0	0	0	2707	0	0	0	0
AFFIC_LIGHT_AHEAD	0	0	0	0	0	0	0	0	0	0	0	0	600	0	0	0
TRUCKS_FORBIDDEN	0	0	0	0	0	0	0	0	0	0	0	0	0	420	0	0
WARNING_BUMPS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	390	0
WORK_ON_ROAD	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	1498

Note: Confusion matrix is hard to read because of its size. Full size version can be found in project src files as PDF.

Confusion matrix shows us the count of every correct and false prediction regarding the original label. We can expect that when 2 classes are very similar to each other with many common features, misclassification can happen. In this dataset we have only very few misclassified samples and that are speed limitation signs which are the same by shape, just the number in them is different.

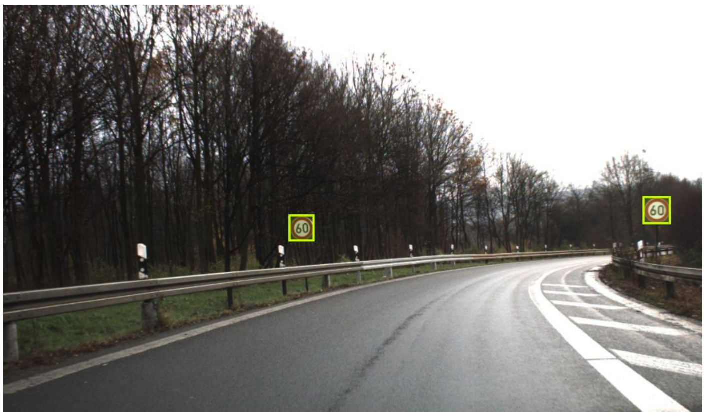
YOLOv5 network (Detection and classification)

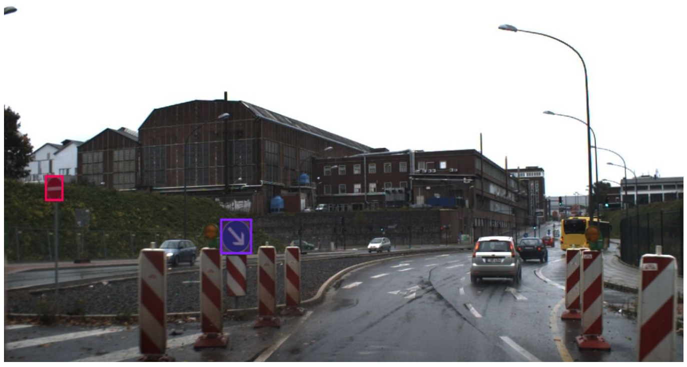
YOu Look Once, or YOLOv5 is a real time object detection framework based on PyTorch. It outperformed most of the object detection framework by frames per second analysed.YOLO can easily reach performance of 60-140 frames per second which make it very suitable for real time detection. It is highly recommended to use GPU instead of CPU while training a neural network.

Dataset

YOLO is an object detection framework. Thus it uses different styles of datasets. Instead of one whole image as a sample. It uses images and coordinates or “boundary boxes” where a specific object is located to train. Because of that my dataset is not compatible with the YOLO framework. Because of that I found another dataset. Dataset used for detection is made of 740 samples of images with 4 classes. [INFORMATIVE,WARNINGS,OTHER,MANDATORY]. Those classes contain many types of signs. Which can make great as universal sign detection. Convolutional neural network (Classifier) has a much better dataset for classifying signs.

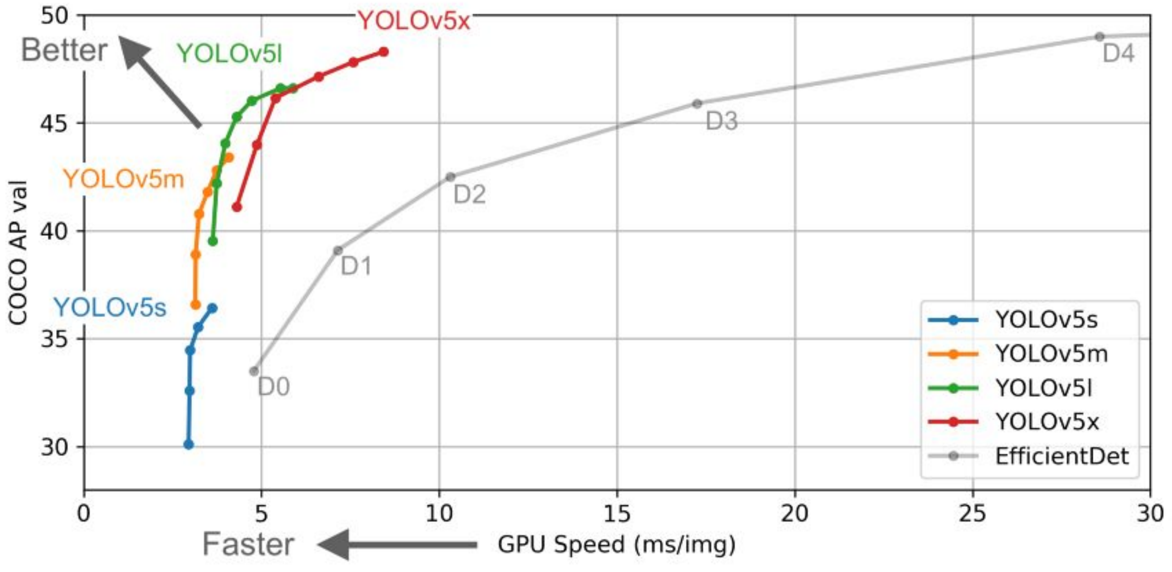
All 4 classes were merged into single class “SIGN” using roboflow web application and exported as YOLOv5 compatible dataset.





Model

YOLOv5 comes with a few predefined models that the user can choose or design his own model. In our case we chose the fastest and smallest predefined model which is “YOLOv5s”.

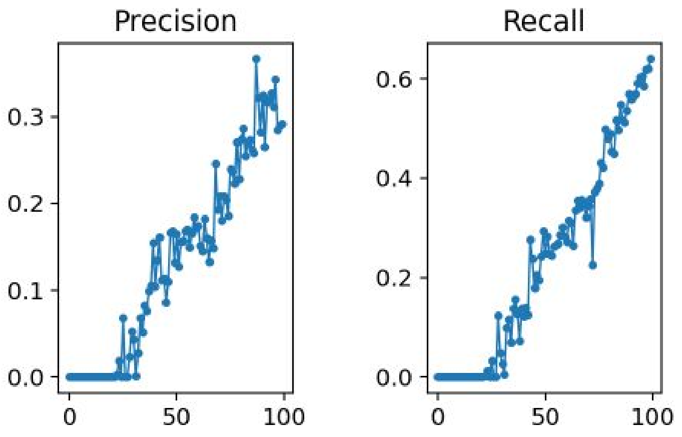


HyperParameters

Similar to the previous convolutional network. Many combinations of hyper-parameters to find the best combination. In the YOLOv4 framework main parameters we should adjust are batch size and number of epochs. Of course we are able to change many more parameters in the source code of the framework but we will not go into this more deeply. As mentioned in a Dataset section we are using another dataset with much less number of samples. This will force us to increase the number of epochs while we still need to be carefully about over fitting our model..

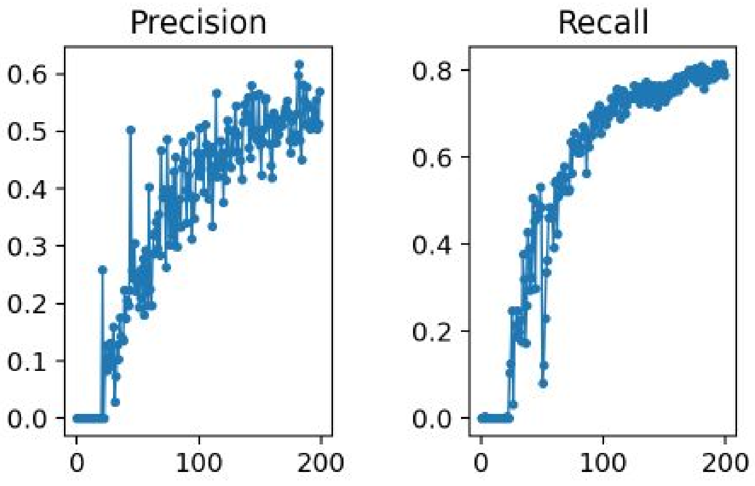
Performance

Batch size 16 & 100 epochs



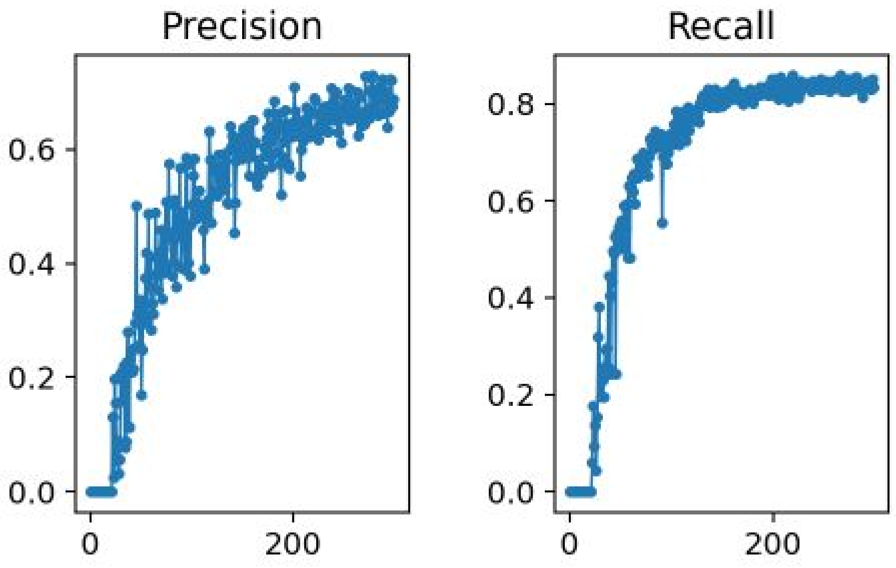
In our first test we tried very common values for training. (16 batch size and 100 epochs). Results were not so great. We can see that precision is increasing almost every epoch but we managed to get only precision of around 35% and ~63% recall. From those results it is obvious we are able to increase the number of epochs to get better results.

Batch size 16 & 200 epochs



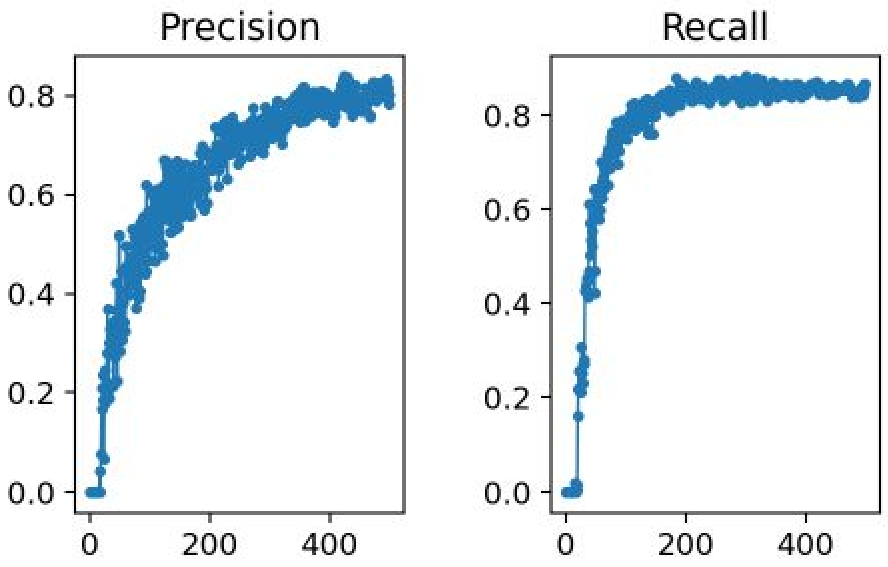
Making our number of epochs double made precision much better. With 200 epochs we got precision of ~60%. We are still able to increase the number of epochs.

Batch size 16 epochs 300



Increasing epochs did show slightly better results with precision of around 70% but we can see on graphs that after epoch 200 our learning rate goes much slower.

Batch size 8 epochs 500

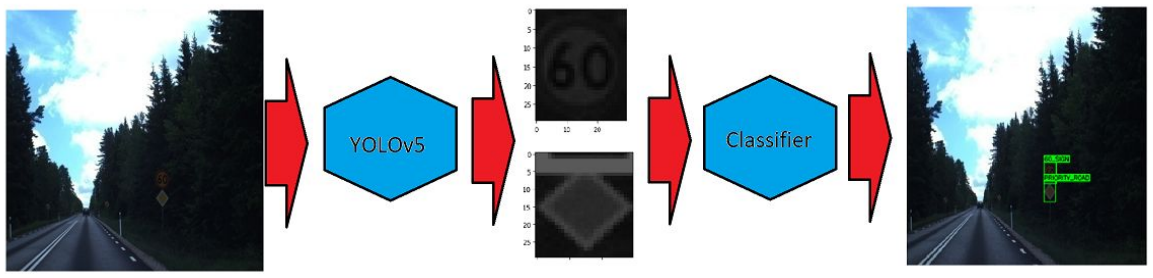


Best results were achieved by using batch size of 8 and 500 epochs. Training took a long time because of the small batch size and many epochs. Results from this experiment are used as the final version of the detection neural network.

Conclusion

With a proper dataset of signs for object detection and recognition I could replace the whole classification neural network with YOLOv% which current jobs is to detect any signs on a picture. But because the classification dataset had much more samples I stayed with this method.

Input image is forwarded into YOLOv5 network. This will return cropped images of detected signs and their bounding boxes from original input images. Cropped images which are resized to 30x30px greyscale which is mandatory input for classifier is then forwarded into classifier which will output labels. Labels and bound boxes are drawn on the original image.



YOLOv5 with YOLOv5s model can reach ~ 60 frames per second performance which would be great with a real time detection system like a car's dashcam. In this project it would be less because of another neural network running together with YOLOv5.

Look forward

It would be interesting to see how another neural network as a classifier lowers performance of YOLOv5 but for this case implementation to process videos would have to be implemented. Therefore this compassion is left out of the project.

System like that could be implemented in a car's dash cam and connected with navigation for example. This can read and notify drivers of changes in traffic rules. Big use of systems like that is I believe in fully autonomous cars where the network is detecting signs, traffic lights, road lines and other cars.

As mentioned early we could increase performance by moving the classifier network into YOLOv5. Project was started as a classifier and later it was late to look for a new dataset for a different type of neural network.

References

PyTorch tutorial https://pytorch.org/tutorials/beginner/blitz/neural_networks_tutorial.html [10.11.2020]

YOLOv5 https://github.com/ultralytics/yolov5 [21.11.2020]

NN Optimizers
https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6#:~:text=Many%20people%20may%20be%20using,order%20to%20reduce%20the%20losses.
[12.11.2020]

YOLOv5 tutorial https://medium.com/@michaelohanu/yolov5-tutorial-75207a19a3aa
[22.11.2020]