TÜRK-ALMAN ÜNİVERSİTESİ
TÜRKISCH-DEUTSCHE UNIVERSITÄT

Project Plan: Comparing different Convolutional Neural Network models applied on a Traffic Sign Classification problem

Sedat Hatip
Department of Computer Science
Turkish-German University
e170503101@stud.tau.edu.tr

Abstract

In this paper, I will try to compare different Convolutional Neural Network (CNN) models applied on a Traffic Sign Classification Problem and compare their results respectively

1 Introduction

With the increase of the usage of deep learning in a lot of industry branches, such as the automotive industry, they gain a lot of attendance and start being used more frequently to automate different problems. A very important branch is the detection and classification of street signs, as it plays a crucial role in autonomous driving cars. They have to reliably detect and recognize them afterwards. In this paper however, I will solely compare the classification (recognition) part of traffic sign,

2 Dataset and Features

The Dataset I will be using is the "GTSRB - German Traffic Sign Recognition Benchmark" from Kaggle, which was created for a competition in 2011 (Link can be found in references). It contains 43 different street sign types of Traffic Signs found in Germany, with over 50 000 samples in total (approximately 39 000 training samples and 12 000 test samples). My train set consists of 80% of the train folder, while my validation set consists of the remaining 20%. The images do not come preprocessed and are a compilation of a 1-second video of a traffic sign at 30 frames per second, which results in 30 different angles for each training example. The resolution of the samples vary from 900 pixels all the way up to 20 000 pixels per image.

The Dataset itself is split into a Train, Test and Meta folder, while the latter merely shows a metapicture of each label (to know which label resembles which traffic sign). The Train folder is split into 43 subfolders where each one resembles one class The Test folder is solely a compilation a shuffled compilation of samples without labels. Additionally in the root folder of the Dataset, there is a CSV file for both the Train and Test set, which represents the top left, top right, bottom left, bottom right corner as well as the label and the path of each sample. In my code, I didn't make use of the coordinates in the CSV file, but rather only saved the picture and each corresponding label in matrixes and label vectors respectively.

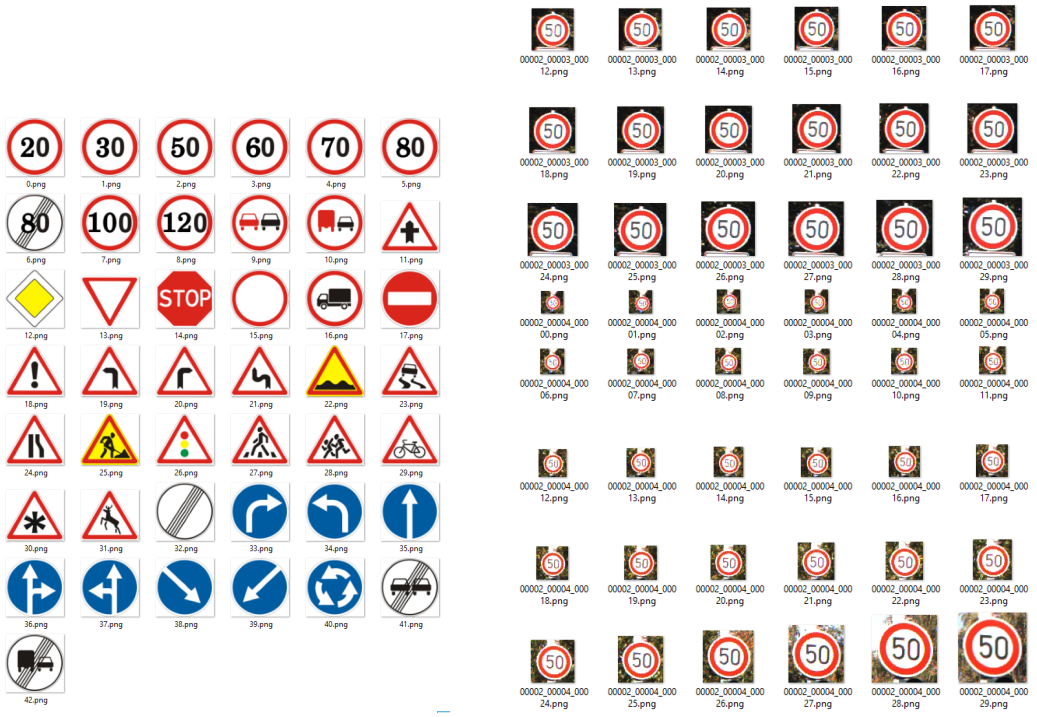


Figure 1: Meta Folder and Example of a Train subfolder (Label: 2, 50km/h speed limit sign)

It should be noted, that the dataset is unbalanced and doesn't have the equal amount of samples for each class (e. g. there are 210 samples for label 0, but 2250 for label 2)

3 Preprocessing and Data Augmentation

Before making use of the data, I preprocessed it by resizing all samples to a 32x32 as this resolution happens to have shown the best results for the classification problem of street signs. This results in lowering the resolutions for some pictures, while supersampling some of them, which have less than 1024 pixels (32x32)

Figure 2: Comparison of a downsampled and normal sized sample

I also normalized all samples by dividing each RGB value of all pictures by 255.

For data augmentation, I added random rotations of up to 10 degrees, random zoom-ins and -outs between 0,85 and 1,15, horizontal and vertical shifts of 0.1 pixels. Out-Of-Boundary pixels are filled with the nearest pixels' values

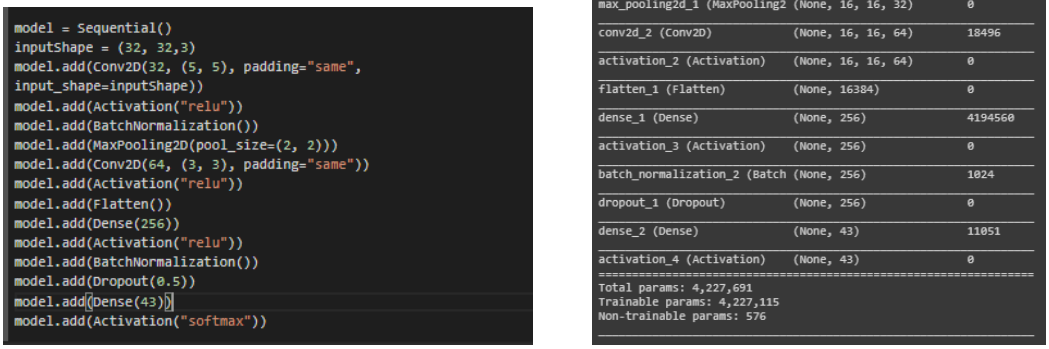
4 Hyperparameters

I used the Adam optimizer during backpropagation as it seems to be the most efficient in contrary to the likes of RMSprop. In a paper from Sebastian Ruder from 2016, he stated, that "Insofar, RMSprop, Adadelta, and Adam are very similar algorithms that do well in similar circumstances. [...] its bias-correction helps Adam slightly outperform RMSprop towards the end of optimization as gradients become sparser. Insofar, Adam might be the best overall choice".

I picked a batch size of 32, a learning rate of 0.003 with the standard learning-rate-decay of Keras's Adam optimizer and used the sparse-categorical-entropy loss. The reason, why I didn't use the categorical-entropy loss is because we will have one prediction at the end, which is mutually exclusive, e. g. one sample can only belong to one of the 43 categories.

5 Models

My own model consists of 2 Convolutional, 1 Max Pooling, 2 Batch Normalization, a Dropout Layer and finally 2 FC-Layers, while the second one resembles the 43 possible outputs. Each of these layers are activated using the ReLU function. In the output layer we use the softmax activation function, as we have a multiclass classification problem.



```
model = Sequential()
inputShape = (32, 32,3)
model.add(Conv2D(32, (5, 5), padding="same",
input_shape=inputShape))
model.add(Activation("relu"))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(Flatten())
model.add(Dense(256))
model.add(Activation("relu"))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(43))
model.add(Activation("softmax"))
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 32)        2432
_________________________________________________________________
activation_1 (Activation)    (None, 32, 32, 32)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 32, 32, 32)        128
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 64)        18496
_________________________________________________________________
activation_2 (Activation)    (None, 16, 16, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 16384)             0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               4194560
_________________________________________________________________
activation_3 (Activation)    (None, 256)               0
_________________________________________________________________
batch_normalization_2 (Batch (None, 256)               1024
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 43)                11051
_________________________________________________________________
activation_4 (Activation)    (None, 43)                0
=================================================================
Total params: 4,227,691
Trainable params: 4,227,115
Non-trainable params: 576
_________________________________________________________________
```

Figure 3: Own Model and Summary in Tensorflow/Keras

6 Results

Even though the hyperparameters aren't finetuned yet and the layers of the models were basically added randomly, the results are pretty good for a first network draft.
The model achieved an accuracy of approximately 95% for training accuracy and 97% for validation accuracy. The Test accuracy was at 92% which is also respectable, considering the paper by LeCun and Sermanet achieved a training accuracy of approximately 98% with their respective network during the competition in 2011 with the same dataset.

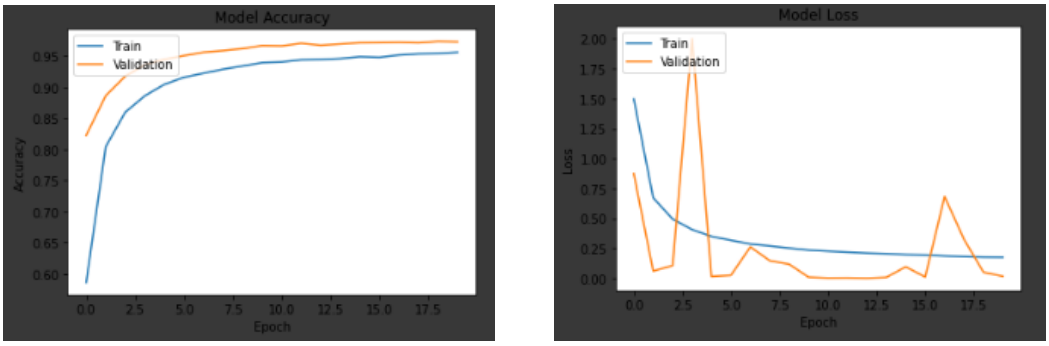


Figure 4: Accuracy and Loss of Train and Validation Data in own model

```
12630/12630 [==============================] - 1s 114us/step
['loss', 'accuracy']
[0.26236566040880227, 0.9221694469451904]
```

Figure 5: Test-Set Evaluation Accuracy and Loss

Figure 6: Manually cropped Image Accuracy Result

References

Dataset: GTSRB-German-Traffic-Sign

Code (only access with E-Mail @tau.edu.tr): Traffic Sign Classification_Sedat_Hatip Google Colab

Related Work

LeCun, Y. & Sermanet, P. (2011). Traffic Sign Recognition with Multi-Scale Convolutional Networks. *Courant Institute of Mathematical Sciences, New York University*
Link to .PDF

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *Insight Centre for Data Analytics, NUI Galway, Aylien Ltd., Dublin*
Link to .PDF

Subject to change: More papers will (and are going to) be added as a related work to this paper