TÜRK-ALMAN ÜNİVERSİTESİ
TÜRKISCH-DEUTSCHE UNIVERSITÄT

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[1]. 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.[1]

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.

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)

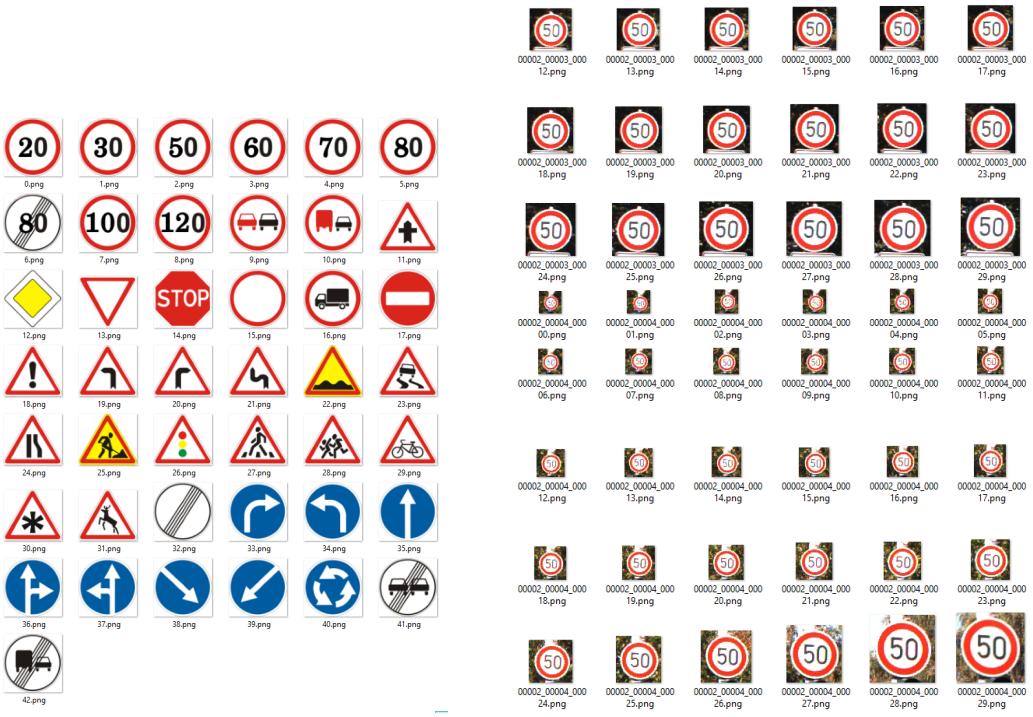

Figure 1: Meta Folder and Example of a Train subfolder (Label: 2, 50km/h speed limit sign)

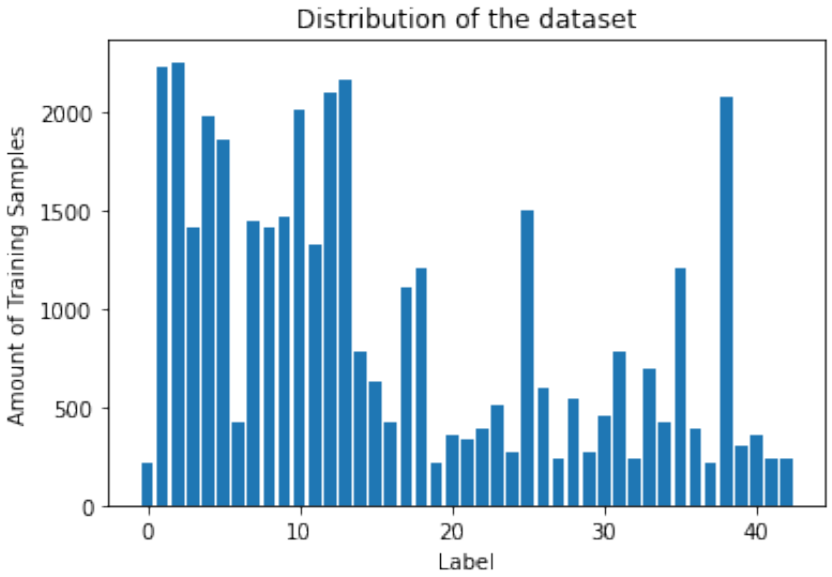

Figure 2: Distribution of Training samples

3 Related Work

As this was a competition back in 2011, there has been a lot of progress throughout the lifecycle of the dataset. LeCun, Y. and Sermanet, P. achieved a final test accuracy of approximately 98% during the competition with their own Convolutional architecture.[2] However the state-of-art and winner back then was a multi-column deep neural network by Cireşan D., Meier U., Masci, J. and

Schmidhuber, J. where they took the average result of several different deep neural network models which were trained with different preprocessed images. This resulted in them having a test-accuracy of approximately 99,5%[3]

4 Preprocessing and Data Augmentation

Before making use of the data, I preprocessed it by resizing all samples to 32x32 (for some Transfer-Learning Problems to 90x90) as this resolution happens to have shown the best results for the classification problem of traffic signs[2]. This results in lowering the resolutions for some pictures, while supersampling some of them, which have less than 1024 pixels (32x32)

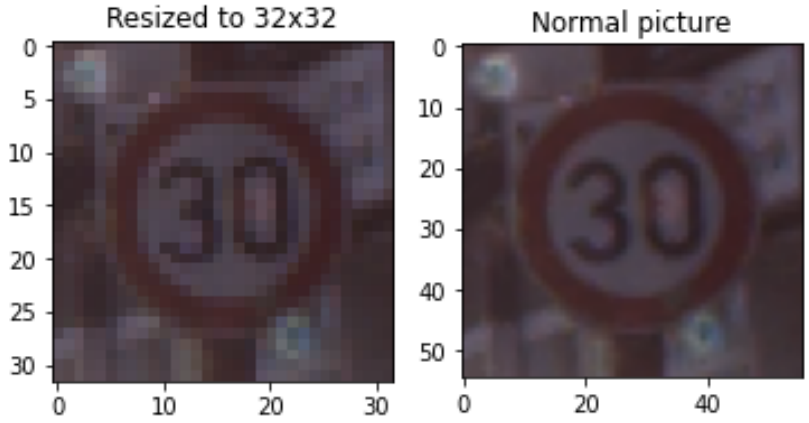

Figure 3: Comparison of a downsampled and normal sized sample

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. This was a heuristic approach by myself.

5 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"[4].

I picked a batch size of 256, a learning rate of 0.0015 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]

6 Models

For all models, which are trained using transfer learning, I added a 256-noded fully connected layer following a 43-noded fully connected layer as the output layer. For Chapter 6.2, 6.3 and 6.4, I always trained the model for 10 epochs first, each with only the last two custom layers only. After that I proceeded to make a few more of the last layers trainable and trained it for an additional 50 epochs. The reason why I made a few of the last layers trainable is because later/deeper layers contain more image-specific features, while for our problem I solely wanted the more general features such as edge detection. This is called fine-tuning.[6]

6.1 Own Model

My own model consists of different types of layers as seen in Figure 3. 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 with a single possible output. My own model was trained for 200 Epochs and was applied to samples at a resolution of 32x32.

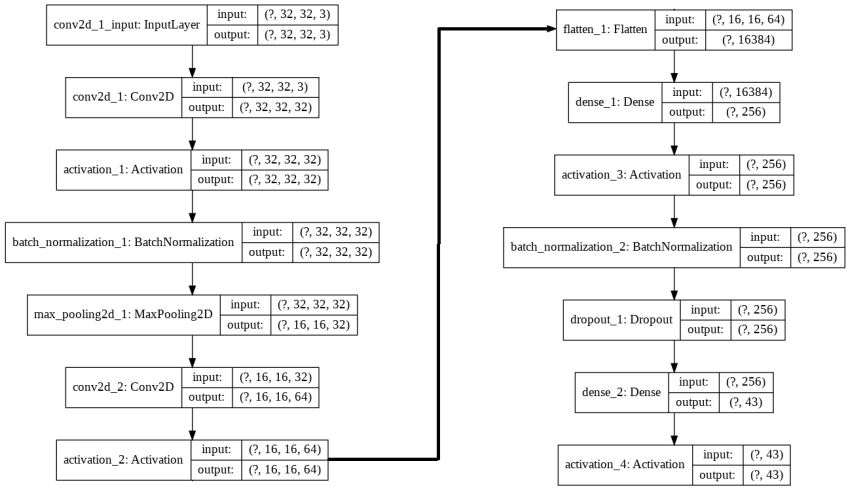

Figure 4: Visualization of own model

6.2 ResNet50

I used a ResNet50 model with pretrained weights from Imagenet. For this model, I had to upsample some of the samples to 90x90, as a lower input shape could lead to errors (e. g. the convolution layer's filters could be bigger than the picture itself). The ResNet50 model consists of 182 layers of which I made the last 60 trainable (approximately the last third of all layers) and the rest untrainable (frozen).

6.3 InceptionV3

As known, the Keras built-in InceptionV3 model consists of a total of 312 layers. The initial pre-trained weights are again from the Imagenet dataset. As it was for the ResNet50 model, I also had to use bigger resolution samples, so I also went for a 90x90 sample for the sake of being able to compare the models to each other at the end. Here I made the last 100 layers trainable (approximately the last third of all layers) and the rest untrainable again.

6.4 VGG16

For the last compared model, I picked the VGG16 model with pretrained weights from Imagenet. I made the last 6 of the 18 total layers trainable, while the first 12 were frozen. For the model, I Flattened the last layer of the VGG16 architecture, as the earlier layers were 3-dimensional.

7 Results and Conclusion

As seen in Figure 5, we can quickly see, that deeper Neural Network architectures lead to overfitting issues. This may be due to the lack of differences between each traffic sign. The deeper models (InceptionV3 and ResNet50 in this case) quickly adapt and learn the (rather unimportant) features such as noise in the samples: The model may perform extraordinary on the Datset it has been trained on, but performs way worse on the real problem. We can also clearly see, that the Inception Model performs worse than the ResNet50 architecture, which uses Residual Blocks to basically minimize the problem of the vanishing gradient which seem like the main reason in the end, why we have a

bad Test-accuracy for both of these models.[7] To counter this high variance problem, we could have added a few L2-Layers or Dropout layers in between each model, however in this case, using a flatter network seems the best way to go, as the results of my own proposed model and the VGG16 model overperformed the first two.

Model	Train-Accuracy	Validation-Accuracy	Test-Accuracy
Own Model	97,6%	98,8%	*93,2%*
VGG16	99,9%	99,5%	*96,7%*
InceptionV3	99,2%	52,0%	*45,9%*
ResNet50	99,9%	70,7%	*70,2%*

Figure 5: Accuracy of the different models

My own model achieved an accuracy of approximately 97% for training accuracy and 98% for validation accuracy. The Test accuracy was at 93% which fairly respectable, considering the paper by LeCun and Sermanet achieved a test-set accuracy of approximately 98% with their respective network during the competition in 2011 with the same dataset. The VGG16 model comes really close to that figure with a Test accuracy of approximately 97%. There is surely more room for better results if the hyperparameters are finetuned a little bit more to exceed that 98% accuracy mark.

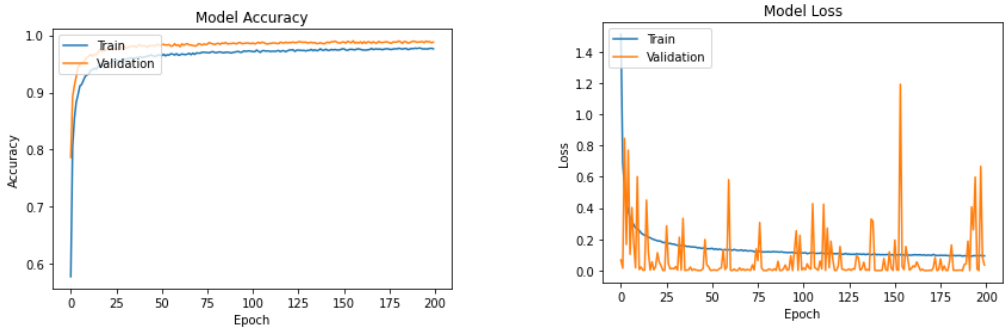

Figure 6: Accuracy and Loss of Train and Validation Data in own model

References

[1] Mykola. (2018). GTSRB - German Traffic Sign Recognition Benchmark.
Kaggle. https://www.kaggle.com/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign/

[2] LeCun, Y. & Sermanet, P. (2011). Traffic Sign Recognition with Multi-Scale Convolutional Networks.
Courant Institute of Mathematical Sciences, New York University. 2 pp.

[3] Cireşan D., Meier U., Masci, J. and Schmidhuber, J. (2012). Multi-Column Deep Neural Network for Traffic Sign Classification. *IDSIA - USI - SUPSI — Galleria 2, Manno - Lugano 6928, Switzerland* 11 pp.

[4] Ruder, S. (2016). An overview of gradient descent optimization algorithms.
Insight Centre for Data Analytics, NUI Galway, Aylien Ltd., Dublin. 10 pp.

[5] Jethwani, T. (2020, January 1). Difference Between Categorical and Sparse Categorical Cross Entropy Loss Function.
LeakyReLU. https://leakyrelu.com/2020/01/01/difference-between-categorical-and-sparse-categorical-cross-entropy-loss-function/

[6] fchollet. (2020, April 15). Transfer learning fine-tuning.
Keras.io. https://keras.io/guides/transfer_learning/

[7] Fung, V. (2017, July 15) An Overview of ResNet and its Variants.
Towards Data Science. https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035

Source Codes Repository: Traffic Sign Classification Github Repository