Machine Learning Engineer Nanodegree

Capstone Project

Stefan Merbele
March 31th, 2021

I. Definition

(approx. 1-2 pages)

Project Overview

Every car driver knows the problem. You have to inspect your environment all the time. What is the driver in front of me doing? What happens left and right of my car? Does the pedestrian want to cross the street? Furthermore, people inside the car want to conduct a conversation or need your focus.

So it is a complex task which needs all your focus to get to your aim save every day. A perfect area to use technology to support drivers and help to avoid accidents. That's why German researchers took the effort and created a huge dataset of traffic signs (GTRSB – German Traffic Sign Recognition Benchmark) in a drivers environment.

The process to help car drivers could be like this: The car's camera inspects the environment all the time. When it recognizes a traffic sign, it takes a photo and passes it to some sort of algorithm. This algorithm detects what kind of traffic sign is seen here. At the end of the process, the respective sign could be shown at the surface of the car. That would give the driver the opportunity, to see the speed limit for example, and ensures, that no sign is unseen. At a further step, the car itself could take action immediately if it is necessary – one more small step into automated driving.

Problem Statement

The project will focus on how to classify images of traffic signs correctly. This challenge is a multi-class, single-image classification challenge held at the International Joint Conference on Neural Networks (IJCNN) 2011. That means, that there are images with their respective label that holds information about what kind of sign is seen at the picture. Based on that, a model will be trained and subsequently evaluated to find out, if it can classify new pictures in the right way.

Over the past few years, research made great progress in image recognition through convolutional neural networks. That's why a solution with this approach will be shown to solve this challenge.

Metrics

In this case, classification accuracy seems to be the right metric to evaluate the success of the task. At the following formula, the calculation of accuracy is shown:

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions\ made}$$

To make this easy to understand, let’s go through this example:

There is a classification problem with 10 images. At the table, the label of every image can be seen (ground truth). The prediction shows, how the model estimates every picture. Out of the 10 labels, the model predicted 8 right. Using the formula, the accuracy, in this case, is 0.8 or 80%.

Ground Truth	A	A	B	A	B	B	B	B	A	B
Prediction	A	B	B	B	B	B	B	B	A	B

This makes clear, that the goal is to get an accuracy near to 1 because it would mean, that the model classifies every image right.

II. Analysis

(approx. 2-4 pages)

Data Exploration

Like mentioned earlier, in this challenge, the GTSRB - German Traffic Sign Recognition Benchmark is used. In Figure 1, there are a few examples of the dataset. Looking at these examples, it gets clear, that the signs are not photographed in a laboratory environment. The images are taken in a traffic environment. That makes the following tasks much more difficult because the model has to deal with huge variance because of different image shapes, the brightness of images, weather, etc. But it is also more interesting because it could be used directly to help car drivers.

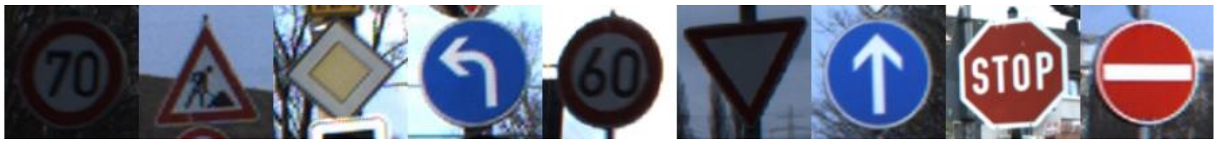

Figure 1: Example images of GTSRB

The dataset is provided from the University of Bochum in Germany. The training set has 39.209 images with 43 classes. The test set contains 12630 images.

Exploratory Visualization

It is known how many images there are provided in the dataset and how many classes they are to classify. But it is good practice to have a more detailed look at the distribution of the images per class. A visualization of the provided data helps a lot in this case. In this way, it is easy to see if there are any weaknesses in the dataset. That means if the number of pictures is distributed uniformly over all classes. If that is the case, the database is well balanced.

It is seen, that the difference of occurrences is about 200 to 2500 per label. The fewest labels per image can be found at labels 0, 19, and 37. There are only 210 images per label found in the dataset. At the other side of the range, there is label 2, 1 and 13. They can be found 2250, 2220, and 2160 times.

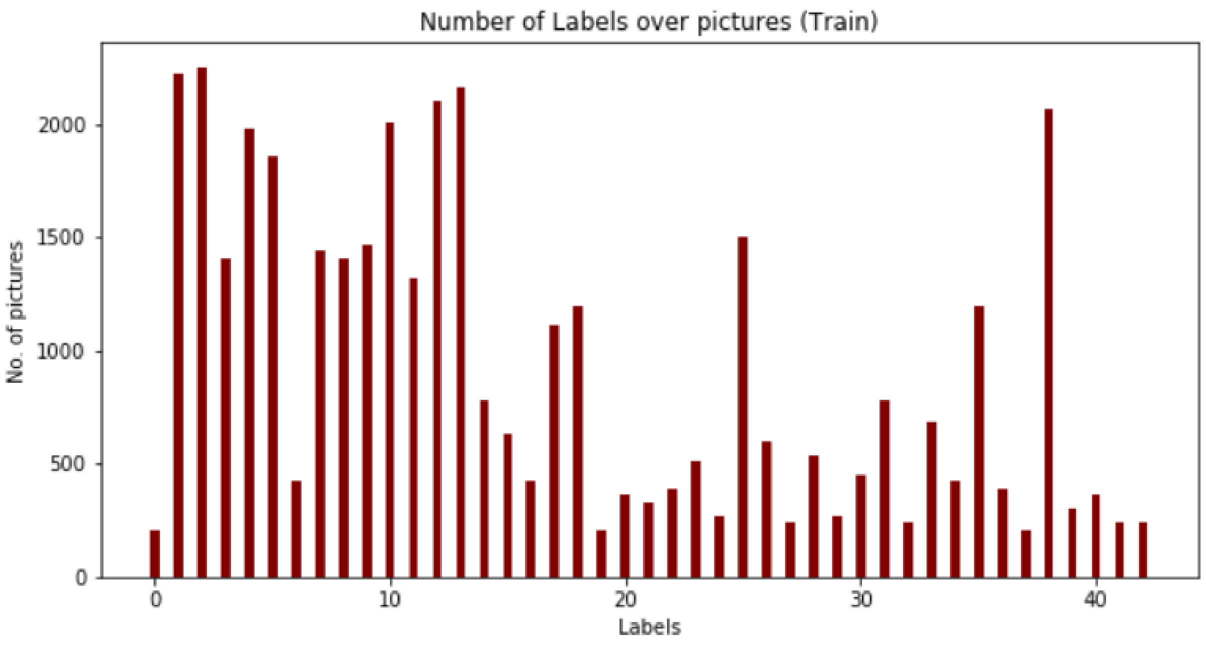

Figure 2: Number of images over labels

Algorithms and Techniques

The CNN has to be fed with arrays with the right shape. The preprocessing is described in more detail at “Data Preprocessing”. Like mentioned earlier in this report, a CNN is used to solve this challenge. The network has the following modules:

Convolutional2D Layer: In that technique a filter (kernel) is used to convolute the input of the neural network. This is very effective when it comes to image classification.

MaxPooling: This process is used, to find relevant information out of the picture. Furthermore, the extraction helps to make the calculation faster.

Dropout: This method helps to provide the model from overfitting. This is reached by removing randomized connections between neurons per training step.

Fully Connected Layer: These layers can be found at the end of the network. They are used to extract the classification out of the provided information.

The architecture of the network is following good practices from other classification problems. That means, that convolutional layers are used following a MaxPooling Layer and the dropout. Before the Fully Connected Layer can extract the provided information, the model has to be flattened. Because I will use a self-designed architecture, there is no chance to use transfer learning. Instead, I will train the model from scratch just with this dataset.

Benchmark

In this challenge, the goal is to get an accuracy of nearly 1. Other developers who trained from scratch got an accuracy of 94 to 97%. Those who used a pre-trained model got accuracies up to 99%. These neural networks have been trained on the ImageNet database which contains about 1,2 million images. That is quite an advantage. But like the net at the link at the bottom, they are much more complex and have over 20 Million parameters.

In this approach, I will train a much more simple CNN with less than 850.000 parameters to get a comparable result.

https://www.kaggle.com/basel99/gtsrb-model-with-accuracy-97

https://www.kaggle.com/marinovik/recognizing-traffic-signals-with-keras-cnn

https://github.com/surmenok/GTSRB

III. Methodology

(approx. 3-5 pages)

Data Preprocessing

At the first step, the images had to be uploaded and opened. After that, every image is resized to a size of 32 x 32 pixels. The smaller the images, the faster the training. But when images are smaller, important information could disappear. In the second step, every image is converted to a NumPy array. This is the right data format for the neural network to learn from. At the end of the uploading, resizing, and converting, there is one array with a shape of (39.209, 32, 32, 3). That means, that there are 39.209 images in the shape of (32, 32) and channel 3 which is representing the colors per pixel. After that process, the numbers 0 to 255 are representing the image. But these numbers are way to large for the network to learn from properly. That's why all numbers are divided by 255 to normalize the array. Now all numbers at the array are at the range between 0 and 1. That helps a lot to train the network efficiently.

But the CNN doesn't just need the "x" input at the beginning of the network. Like in every supervised learning task, the network must know what to estimate during the training process. That's why the corresponding labels are needed here. They are also known as "y". In the beginning, every image has a label between 0 and 42. But that's not the format how the CNN needs the data to be. So in this case, Keras is used to get a binary class matrix out of the labels.

Let’s say, there are 10 labels for example. They are reaching from 0 to 9. If there is a label 5 for a corresponding picture, the Keras “to_categorical” function would transform into a format like this:

Y = [0 0 0 0 1 0 0 0 0 0]

This kind of formatting is done for every label. This makes sure, that the format of the labels matches the shape of the CNN output.

The last step is shuffling. This makes sure, that the data is not sorted in any way. So it can be assumed, that training and validation data is distributed evenly over the labels. Finally, training and validation data are split up to the factor of 0.3. That means, that 70% of the pictures are used for training and 30% are used for validation.

Implementation

After the data preprocessing, there are a lot of details known from the dataset. Before the training is started, data augmentation is used. These are techniques used to increase the amount of data by adding slightly modified copies of already existing data. A Keras package helps to get this done easily. In this case, rotation range, with_shift_range, and height_shift_range are used to augment the dataset. This process helps to make the dataset larger and add more variety to it. It also helps to close the gap to the pre-trained network with the large ImageNet Dataset which is also used to solve this task.

Furthermore, the Keras package “ModelCheckpoint” is implemented. This ensures, that after every epoch the model is saved into a given folder. That has the advantage, that any model can be chosen from the training to see how it performs on the test set. Without this implementation, the model from the last training epoch has to be used no matter, if it is the best or not. Moreover, the model could be used to deploy it in an application.

At figure 2, it is seen, that the dataset is not balanced. There are labels, which just contain 210 images. In contrast to this, other labels contain up to 2250 images, which is about 10 times more. To counteract this unbalanced data, class weights are used.

This is used to take the skewed distribution of classes into account. It can be achieved by giving different weights to the majority and minority classes. These weights will influence the loss of classification during the training. The purpose of this is to penalize the misclassification made by the minority class by setting a higher class weight and at the same time reducing weight for the majority class.

Refinement

After preprocessing the data properly and using several methods to help CNN learn fast and efficiently, the training process has to be made. In Figure 3, the metrics of this process can be seen. The training is executed over 20 epochs. At first, let’s have a closer look at the accuracy which is represented by the red and orange lines. They are increasing very fast from 0.1 to 0.9

over the first 10 epochs. During the second half of the process, the accuracies are rising slowly to 0.96 (acc) and 0.99 (val_acc).

It seems surprising, that the validation accuracy is always higher than the accuracy with train data. The reason for this could be the augmentation of the training data. This adds some additional variety to the data, while the validation data is original and stays the same during the whole training process. Another could be, that the training data is larger and there could be some images which are very difficult to estimate.

Looking at the losses, it can be seen, that they are declining very fast during the first 10 epochs. That matches with the fast inclining accuracies. The losses are starting at approximately 3 and reaching 0.14 (loss) 0.02 (val_loss) towards the end. The val_loss is always lower than the training loss. But that is also explainable with the higher accuracy of the validation data.

Overall, the training can be seen as successful. The accuracies are reaching almost 1 and the losses are decreasing constantly. Looking at the metrics, the model of epoch 19 seems to be the best for deploying (loss: 0.14, acc: 0.96, val_loss: 0.02, val_acc: 0.99). Because of that, the following evaluation and validation is based on that model.

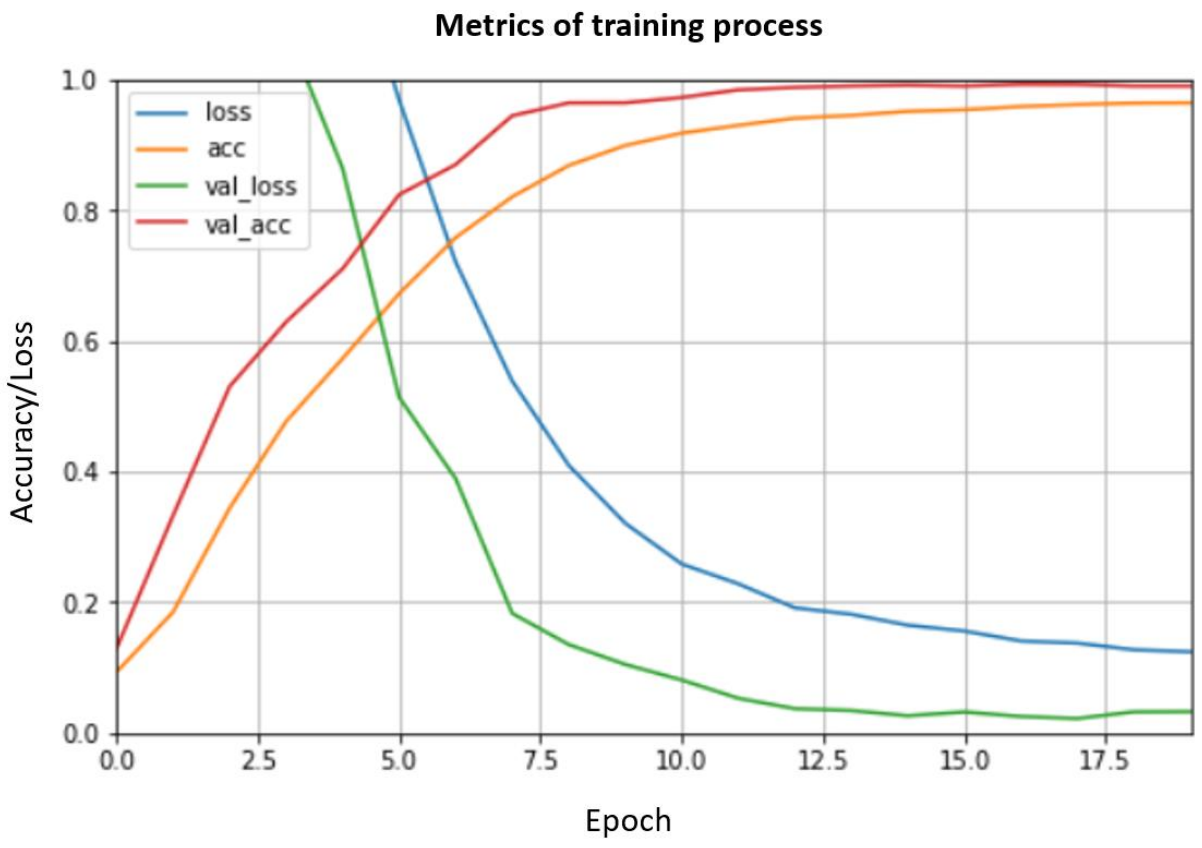

Figure 3: Training process

IV. Results

(approx. 2-3 pages)

Model Evaluation and Validation

As mentioned in the section before, the model of Epoch 19 is chosen to make further predictions. The accuracy for training (0.96) and validation data (0.99) are very good. Now it is important to apply the model to unknown data. In that way, the deployment can be simulated and it can be seen, if the model is suffering from overfitting.

Applying the model, there is an accuracy of 0.97 over 12.630 test images. This shows, that the model is not overfitted at all. It generalizes very well to unseen data located between training accuracy and validation accuracy. The overall high accuracies show, that the model can be trusted.

Justification

In the following table, the accuracy of the self-designed network is compared to other models mentioned in the section “benchmark”. The links to the notebooks can also be found there. The table is ranked by accuracy. As expected, the accuracy of the model using transfer learning is the highest. Here the most complex model is used and pre-trained to the large ImageNet Dataset. That leads to very high accuracy of 0.996.

The following models are self-designed. So there is a need to train them from scratch. The models of other comparable developers show accuracies of 0.976 and 0.939. The self-designed model is at 0.973.

So the model is not the best compared to the ones I found, but it is just 0.003 points worse than the best model trained from scratch. That is satisfying so far. With accuracies nearly 1, the challenge can be seen as solved.

Table 1: Accuracy comparison test data

	Accuracy	**Training**
surmenok	0.996	Transfer learning
Basel Ayman	0.976	From scratch
Self-designed model	0.973	From scratch
marinovik	0.939	From scratch

V. Conclusion

(approx. 1-2 pages)

Free-Form Visualization

Now it is time to have a closer look at the predictions. To have a look at the confusion matrix of test data gives the first overview. Because the accuracies are high, there isn’t that much outside the diagonal line, which is representing true predictions. Furthermore, all numbers outside of that line that is marked with a red box are filled with figures of 20 or higher. That shows the classes where the model has the highest problems of classifying correctly. As it can be seen, there are just four cases, in which the model has misclassified the image 20 times or higher to another single class over more than 12.000 images. The biggest single misclassification is found at classes 29 and 30 with 71 misclassifications.

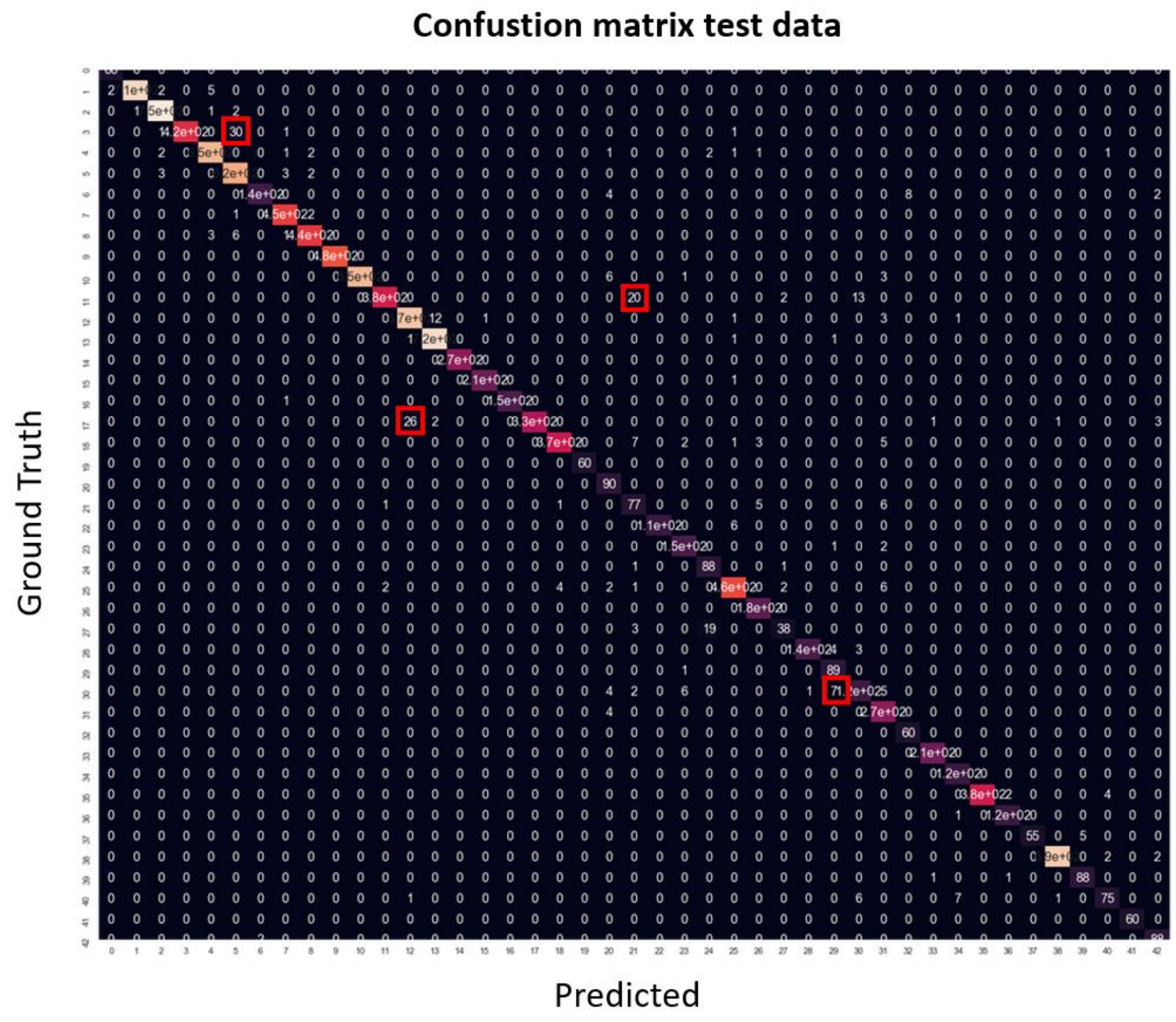

Figure 4: Confusion matrix test data

Let’s have a more detailed look to these classes. There are 270 images of class 29 on the dataset and 450 images of class 30. These are few compared to other classes. By having a look at Figure 5, it can be seen, that the classes look similar in shape and color. Just the sign in the middle is different. It seems, that all these reasons are leading to certain insecurity in predicting these classes.

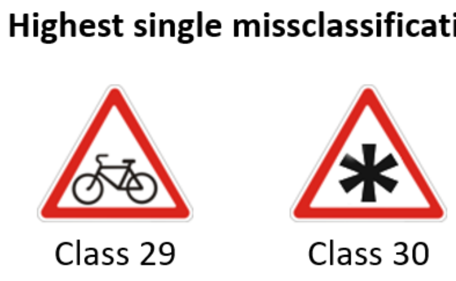

Figure 5: Comparison Class 29/30

Reflection

In the beginning, the images must be uploaded, reshaped, and converted into arrays. After dividing it into train and validation data, it has to be normalized. Data augmentation is used to create more data and more variety. The loss function is weighted to face the imbalanced data.

The model learned very quickly and I find it interesting, how near the simple model can get to the pre-trained model (difference in accuracy about 0.02).

Improvement

There isn’t much space to improve the model because the accuracy is already very high. But the architecture could be tested with different configurations. Here are some ideas:

What if I use more or less Convolutional2D-Layer?

Will the kernel size influence the accuracy?

Should I use more or less Fully-Connected Layer and how many neurons should they have?

Is there a better optimizer than “rmsprop”?