NOVA IMS Information Management School

Building a convolutional neural network for classifying 43 different categories of traffic sign images

Deep Learning Group Project - Summer Term 2021

Philipp Metzger
(m20201058@novaims.unl.pt)

Henrique Vaz
(m20200586@novaims.unl.pt)

Franz Michael Frank
(m20200618@novaims.unl.pt)

Table of content

Table of figures

I. Introduction

This report describes the designing and implementation of a Deep Learning model that is capable of classifying digital images of traffic signs. Traffic sign classification is used for example in cars to automatically provide information about the currently applicable traffic rules like the speed limit to the driver or an automated driving system. The images used to train and test the model were obtained from 'Institute Für Neuroinformatik' which has '*The German Traffic Sign Benchmark'* [1]. The type of Deep Learning model that was selected for this task is a convolutional neural network (CNN). The following chapter contains a definition of the task at hand.

II. Methodology

II. i. Task definition

The goal of this project is to construct an appropriate Deep Learning model that is capable of classifying digital images of traffic signs, in such a way that each image is mapped to an integer that represents a class, such as "Speed limit 20" or "Turn right". An appropriate model shall be found by testing a variety of architectures and parameter combinations of CNNs using an appropriate training data set that will be split into two parts, one for training and one for validation. The scope of this project was reduced to CNNs from the beginning, since the established literature suggests that they are most fit for the task of image classification (see for example chapter 5.5.6 of [2]). Next to the designing and implementation of the model, the evaluation of the same with appropriate test data and evaluation measures is another crucial part of this project. This will be further described in the following chapter.

II. ii. Evaluation measures

The initial training data set consists of 39,209 images which are categorized into 43 different classes. The corresponding test data set contains another 12,630 images. Hence, as this is a multiple classification problem, accuracy is the natural evaluation criterion. For the test images, a ground truth file is available, which enables the determination of the test accuracy, representing the key success criterion for this project.

Additionally, the training of Deep Learning models, depending on various factors such as model complexity or the number of epochs during training, is frequently highly time consuming [3]. Therefore, the running time is another measure by which the model is evaluated and a reasonable tradeoff between running time and accuracy is aimed to be achieved.

II. iii. Approach

One special characteristic of the training images is that they consist of contiguous series of 30 images each. The only exception is series 00019 of class 33, which contains only 29 images. All images within a series are nearly identical and differ only in terms of the resolution. For the initial subdivision of the images into a training and a validation data set, this is of

great importance to consider. The individual series should not be splitted in such a way that images from one series are in the training as well as in the validation data set. This would lead to an immense bias of the validation accuracy. Although this would increase the validation accuracy, the accuracy of the model on unseen data from the test data set would decrease strongly.

Therefore, for splitting into training and validation data, an approach is chosen that splits the images based on an approximate 70:30 ratio, but without dividing the individual series. Thus, for each class, a ratio is chosen which is as close as possible to 70:30, while ensuring that both sides are divisible by 30. For example, class 1, which contains 2,220 images, is divided into 1,560 training and 660 validation images, which corresponds to a ratio of 70.27:29.73.

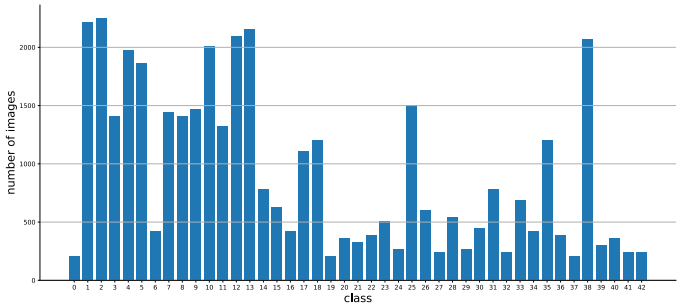

Figure 1: Number of images per class

An additional particularity of the training data is its imbalance. For example, the smallest classes 0, 19 and 37 contain only 210 images, while the largest class, 2, with 2,250 images is over 10 times that size (see Figure 1).

There are two different techniques applied in an attempt to solve this imbalance problem. The first approach is to use data augmentation to create new, slightly modified images within the individual classes. Thereby, all classes are upsampled to the size of the largest one.

The second way of approaching this is to weight the individual classes during training. This involves assigning larger weights to the smaller classes so that they are given more importance during training to compensate for the small amount of data. To calculate the factor of a class, the size of the largest class is divided by the size of the class under consideration. For example, for the smallest classes, which consist of 150 training images after the training validation split, the weighting factor is 10.4, while the factor for the largest classes is obviously 1.

Before building the model and training it on the image data, applying certain pre-processing steps might be useful. First of all, the images can be loaded either with 3-dimensional rgb colors, or only in a 1-dimensional grayscale. The latter can prevent overfitting and thus provide a better generalizing model, while at the same time shortening the training time considerably.

In addition, the input images are also normalized. The Deep Learning API Keras provides the two boolean arguments featurewise_center and featurewise_std_normalization within its class ImageDataGenerator. Through these arguments, the images can be normalized in that the input mean will be set to 0 over the dataset and the inputs themselves will be divided by the standard deviation of the dataset, both in a feature-wise manner.

At the end of the data preprocessing stage, the final image data generators can be initialized. Therefore, the two arguments batch_size and target_size have to be taken into account. Especially the target size is of great importance. Smaller sizes correspond to shorter running times of the training phase on the one hand, while the model's accuracy suffers considerably on the other hand. A size of 100x100 turned out to be a very suitable one.

Having the image data successfully prepared, the actual Deep Learning model is yet to be built. As already mentioned, a convolutional neural network is the most appropriate for the underlying problem. However, there are several different ways to implement it. A major challenge is to identify the most suitable number of convolutional layers. This choice is accompanied by a tradeoff between accuracy and efficiency [4].

Furthermore, to prevent the model from overfitting, it is useful to add one or multiple dropout layers. There are mainly two things to consider when implementing dropout, one being how many layers to add and the other one being the dropout rate of the respective layer. Likewise, to the convolutional architecture, also the decision for a dropout architecture and its specifications will have an impact on the accuracy versus efficiency tradeoff.

II. iv. Error analysis

The error analysis for this project consists of two experiments. The first experiment: Previously, during the training of the two final models, the state of the two final models was saved after each of the 30 epochs of their training. This enabled the authors to apply each of these 30 * 2 = 60 models to the test dataset and with the resulting accuracies and the training accuracies that were saved during the training process to gain knowledge on potentially present overfitting and thus the respective generalization ability of each of these 60 models. On the basis of this information, the final model was chosen.

The second experiment: In order to assess the fitness of the model to be deployed in a real-world scenario, the authors photographed 86 traffic signs in the city of Lisbon and applied the final CNN to these images. The traffic signs selected for this approach were traffic signs that had at least similar counterparts in the datasets used for training and testing, which are made up of photos of German traffic signs. An example for a traffic sign where the Portuguese version is only similar, not equal, to the German one is the one depicted in Figure 2. The results of this experiment were evaluated by comparing the predicted classes with the classes of the traffic signs that the authors found to be the German counterparts.

Figure 2: Comparison of a Portuguese „Caution: children" traffic sign (left) and a German one (right)

III. Results and discussion

III. i. Results of the model evaluation

As said in II.iii., many different CNN architectures were implemented. Within those configurations, also different preprocessing steps were applied. The process of reaching a final model included a vast trial and error methodology, always following relevant literature.

The first step taken in training was to implement a basic model with two convolutional layers, each followed by a max pooling layer plus two final dense layers following a flatten layer. This simple architecture could give the authors an idea of which would be the starting point. In this try, the authors worked only with a fraction of the available data in order to preserve model efficiency.

After looking at the first results, the authors rapidly identified large amounts of overfitting (Figure 3). Due to the amount of overfitting, the authors were forced to tackle that issue, having found two main ways of doing it: adding dropout layers (before the dense layers) and using data augmentation.

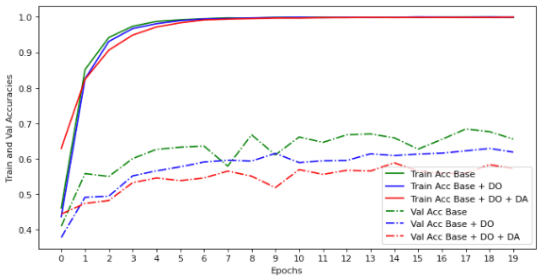

Figure 3: Base model vs base + dropout vs base + dropout + data augmentation

As seen in Figure 3, none of the models really outperformed the others. It's also fair to say that the worst performance belonged to the model where data augmentation was applied. Since no great improvements were achieved, the authors decided that using all data for training could be one good solution.

Considering what is said in II.iii. about training images series, the authors were aware that it could pose some issues regarding generalization capability of the model, therefore, the dropout layer was kept. In the first models trained using all data, two models were tested in order to decide which would be the best choice for the dropout rate. The values used in this testing were 0.3 and 0.9 (Figure 4).

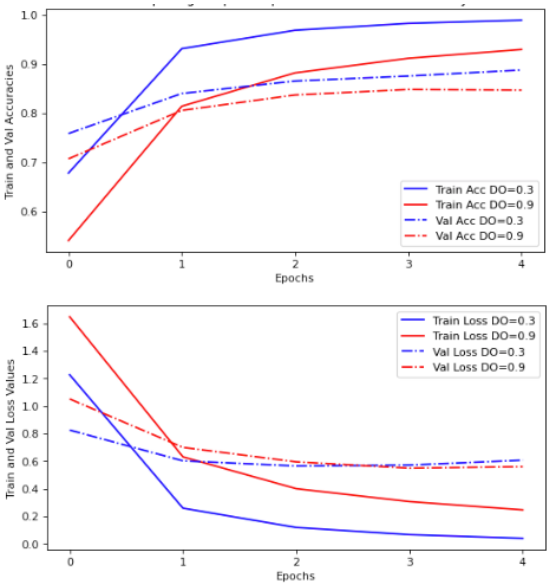

Figure 4: Base model with dropout rate equal to 0.3 and 0.9

After a brief look at the results obtained using all data, the authors quickly found that using either one of the two values a significant part of the overfitting would disappear. After getting these results, using class weights in training was tested. However, the results obtained didn't

differentiate significantly. Taking that into consideration, it was decided that the weighting could be a final step to take when a final architecture was found.

Moreover, and with the objective of increasing performance on the validation set, the authors decided to test the model using one more convolutional layers. The configuration of the model at this point is shown in Figure 5.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 149, 149, 32)	416
max_pooling2d (MaxPooling2D)	(None, 74, 74, 32)	0
conv2d_1 (Conv2D)	(None, 72, 72, 64)	18496
max_pooling2d_1 (MaxPooling2	(None, 36, 36, 64)	0
conv2d_2 (Conv2D)	(None, 33, 33, 128)	131200
max_pooling2d_2 (MaxPooling2	(None, 16, 16, 128)	0
flatten (Flatten)	(None, 32768)	0
dropout (Dropout)	(None, 32768)	0
dense (Dense)	(None, 512)	16777728
dense_1 (Dense)	(None, 43)	22059

Figure 5: Model architecture after adding 1 convolutional layer

In addition to the third convolutional layer added, that as mentioned above was used to increase performance, also grayscale was applied. The latter was introduced as a new attempt to tackle overfitting.

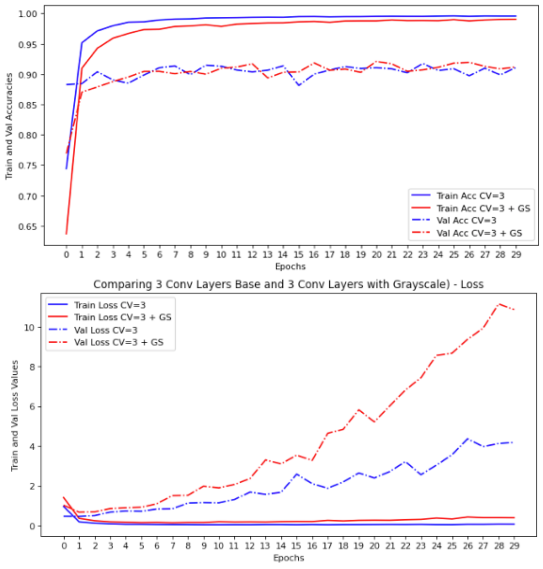

Figure 6: Model with 3 convolutional layers with and without grayscale

From Figure 6, the reader can conclude that the two additions applied in this step, one more convolutional layer and grayscale, resulted mainly in an increased model performance (overtaking the 90 % validation accuracy line) but the overfitting levels remained almost unchanged. Even though the loss values for the model without grayscale are more convincing, the authors still traded that for a slightly larger overall validation accuracy, keeping grayscale for the next steps.

Having achieved an increase in performance, the priority turned to reduce the amount of overfitting. The first idea to reduce overfitting would be to introduce more dropout layers in the current architecture. In order to find which were the optimal values to use in this step, the authors tried two similar configurations in which the only difference resided on the dropout rates. In each of the models a dropout layer was introduced after all convolutional layers. The combination of dropout values (from the first convolutional layer to the last) in the model with lower dropout rates was 0.15 – 0.15 – 0.1 – 0.3. As for the one with higher dropout rates it was 0.25 – 0.25 – 0.25 – 0.5.

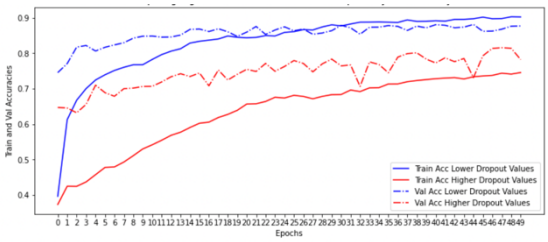

Figure 7: Higher vs lower dropout rates

In Figure 7, the reader is provided with each model's accuracies. In this case, the choice of model became easier with great trade-offs to be decided on. It is clear that in terms of validation and training accuracy, the model with lower dropout rates outperformed its opponent after 50 epochs.

From Figure 7, it also gets perceptible that at this point the overfitting issue seemed to be taken care of. That being said the next step would consider either improving accuracy, in order to reach values closer to 95 %, or improving performance in terms of time. Since times wasn't being an issue so far, the authors wanted to check the result of adding a fourth convolutional layer to the model. Being aware that this could also result in an increased amount of overfitting, a dropout layer was also added to this architecture. Two variants were tried in which the main difference was the number of filters used. The first had 32 – 64 – 64 - 128 filter structure (from first to last layer). The second one had two 128 filters layers instead of having two 64 filters layers, so it looked like 32 – 64 – 128 – 128. Unfortunately for the authors, the result was not positive. Although the second four convolutional layers model (2x128) quickly reached convergence in terms of accuracy (achieving values closer to the current best model), the loss values soared, which was not a good indicator. Therefore, the authors remained with the previous best model, with three convolutional layers and lower dropout rates in each dropout layer. The detailed results of the last testing can be seen in Figure 8.

Getting back to the older model, as said in II.iii., also image normalization was applied

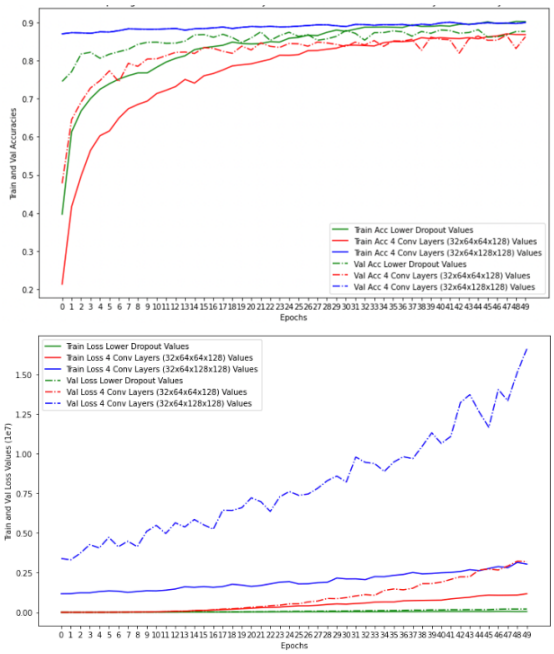

Figure 8: Two models with 4 convolutional layers against the older one

as a preprocessing step. Using 'featurewise_center' and 'featurewise_std_normalization', the authors tried to make the range of distribution of feature values similar between features. In order to evaluate if the goal would be reached, two exact same models were test. The only difference between one and the other was that one of them had image normalization as a preprocessing step, whilst the other did not. In Figure 9, the reader can find the results of such an experiment. From the graphics there isn't again a real trade-off. The model with image normalization outperformed the counterpart both in training and validation accuracies (this one remained constantly over 90 % accuracy on the validation set) and also it has much more encouraging loss values. Since the difference in the experiment was clear, the authors opted for adopting image normalization as a preprocessing step for future experiments.

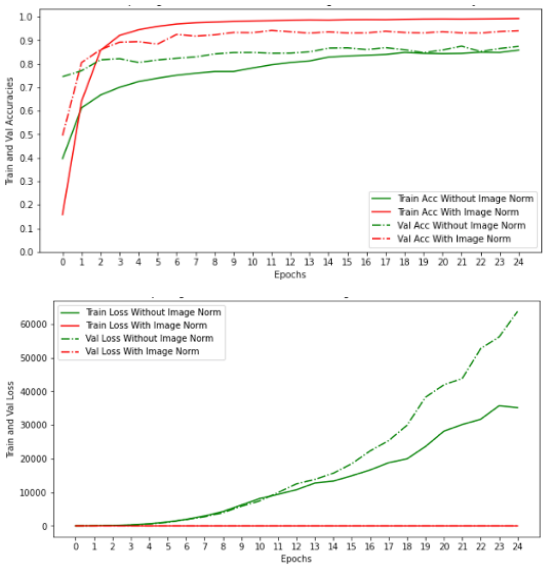

Figure 9: Testing a model with and without image normalization

In this stage the model was reaching a plateau, where no great improvement was taking place. In these cases, the way to overcome it is by making larger changes to the model rather than simply adding layers one by one or come up with new preprocessing techniques. Considering that, it made sense at this stage to draw a similar model but with some changes across the whole architecture. It would make sense to keep the three convolutional layers, however some parameters could be changed, such as the size of the filters and important dropout layers (for example, the first dropout layer is important since it is applied directly over the input convolutional layer). The resulting final model's structure is depicted in Figure 10.

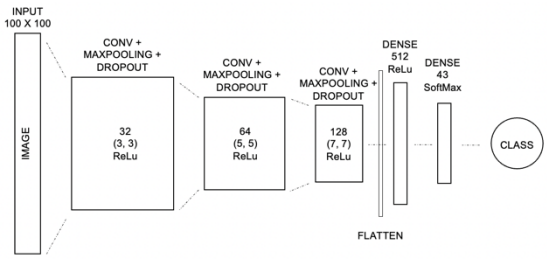

Figure 10: Chosen model architecture

Using this final architecture, and following the logic mentioned above, the authors were only left to test this model with and without class weighting. Having this done, a final model could be decided on.

For that, the authors replicated the same model and made use of the 'class_weight' parameter provided by the 'fit_generator', in which the user inserts a list with the classes' weights. The calculation of the weights is described in detail in II.iii.. The results of this last experiment can be found in Figure 11.

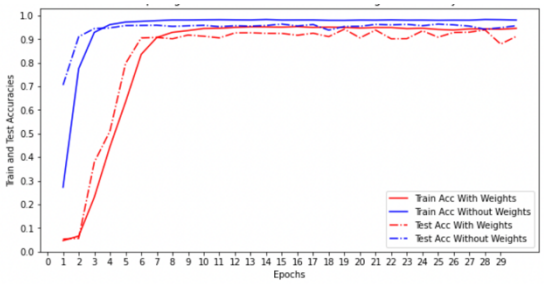

Figure 11: Testing the final model with and without class weights

From the graphic shown above, the reader can conclude that the model without class weights performed slightly better throughout all the train and testing as well. In this last experiment, the authors used all the data available for training (including data previously used for validation), leaving out only the test data.

Furthermore, and considering that for each epoch the model was saved, the authors had the opportunity to choose which epoch seemed the best to store the final model. In this case the choice was made for epoch 15. In III.ii., the reader can find the results of the model at this epoch, and further analysis of the model's performance.

III. ii. Results of the error analysis

Results of the first experiment: The model that the authors found to have the best combination of a high accuracy as well as a low amount of overfitting is the model without class weights applied during the training process and that was trained for 15 epochs. Its training accuracy is 98.14 % and its test set accuracy is 96.43 %. The difference between the training and test accuracies of 1.71 % was found to be acceptable by the authors.

Results of the second experiment: Of the 86 Lisbon traffic signs that were photographed in Lisbon by the authors 69 were classified correctly by the final CNN. This translates into an accuracy of 80.23 % for this dataset that contains some data that is rather unknown to the model developed in this project.

Additionally, even though there are some significant differences between the German and the Portuguese traffic sign shown in Figure 9, the final CNN was able to successfully classify photographies of this traffic sign. This shows the generalization ability of the final CNN and that it is versatile in different environments.

III. iii. Comparison to other results

The problem addressed by the authors is not something entirely new. In the introduction, it is mentioned that this type of model can be used by self-driving cars or systems that identify the traffic rules in certain locations. In order to generally classify the model, it gets pertinent to compare this model's performance with previous ones.

The first comparable model developed [5] in the past achieved an accuracy of 94.7 % on the test set. Other model designed to correctly identify traffic signs can be found in [6]. This one having reached a score of 99 % on the test dataset.

A third source [7] also reached values near 100 %. This project although, resorted to several object detection systems, rather than just simple convolutional neural networks.

IV. Conclusion

This project demonstrates the successful implementation of a Deep Learning algorithm in the form of a convolutional neural network for the classification of traffic signs, which requires about 4 hours of training time in order to be able to assign the signs to the correct classes with an accuracy of over 96 %. In order to achieve the best possible model, a wide variety of experiments through trial and error as well as referring to the current literature were used. Moreover, the developed model has such a strong generalization ability that self-made images of traffic signs around Lisbon, which were sometimes quite different from the training images or also dirty and defaced, could be categorized correctly for the most part.

Bibliography

[1] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, Christian Igel, German Traffic Sign Recognition Benchmark GTSRB.

[2] C. M. Bishop, Pattern Recognition and Machine Learning (Information Science and Statistics), Berlin, Heidelberg: Springer-Verlag, 2006.

[3] Scheidegger, F., Istrate, R., Mariani, G. et al, Efficient image dataset classification difficulty estimation for predicting deep-learning accuracy, The Visual Computer, 2020.

[4] Sarkar, A., Vandenhirtz, J., Nagy, J. et al, Identification of Images of COVID-19 from Chest X-rays Using Deep Learning: Comparing COGNEX VisionPro Deep Learning 1.0™ Software with Open Source Convolutional Neural Networks, SN Computer Science, 2021.

[5] https://towardsdatascience.com/traffic-sign-recognition-using-deep-neural-networks-6abdb51d8b70, viewed on 2 April 2021, 11:21.

[6] https://towardsdatascience.com/traffic-sign-detection-using-convolutional-neural-network-660fb32fe90e, viewed on 2 April 2021, 11:23.

[7] Álvaro Arcos-García, Juan A. álvarez-García, Luis M. Soria-Morillo, Evaluation of Deep Neural Networks for traffic sign detection systems.

GitHub

A completed set of developed and used source code as well as the entire results can be accessed through the following link to the GitHub repository:

https://github.com/ph1001/Deep_Learning_Project_Group_10