**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 autonomously driving system. The type of Deep Learning model that was selected for this task is a Convolutional Neural Network (CNN).

**II. i. Task Definition**

The goal of this project is to find an appropriate Deep Learning model that is capable of classifying digital images of traffic signs, such 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. 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 [1]).

**II. ii. Evaluation Measures**

The initial training data set consists of 39209 images which are categorized into 43 different classes. The corresponding test data set contains another 12630 images that are not pre-classified. 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 [source]. 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 splitting 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 2220 images, is divided into 1560 training and 660 validation images, which corresponds to a ratio of 70.27:29.73.

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 2250 images is over 10 times that size (see Figure 1).

*Figure 1: Number of images per class*

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 [source].

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 [source].

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. They are set to 20 and 150x150 respectively, both usual values that can be found in recent literature [source]. The latter one could also be chosen to be smaller, which would on the one hand significantly shorten the running time of the model, while on the other hand the accuracy would suffer considerably.

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 [source].

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.

# Bibliography

|  |  |
| --- | --- |
| [1] | C. M. Bishop, Pattern Recognition and Machine Learning (Information Science and Statistics), Berlin, Heidelberg: Springer-Verlag, 2006. |