Image Classification on Standford Cars Dataset

Patrick Brus

30.08.2021

1 Introduction

Cars are the most important means of private transportation in the current era. There were 276 million registered vehicles in the U.S. in 2019, according to Statista [1]. This number also includes motorcycles, busses and so on, but gives a clear indication how important vehicles for private transportation are. In this project, a car model classifier is developed. This car model classifier takes an image of a car as input and outputs the model name including the manufacturer of the car, as long as it is known to the model. This helps in case someone finds a nice car he maybe wants to buy or wants to gather more information about it, but does not know the model name or the manufacturer of the car. The open-source Standford cars dataset [2] is used for training the model.

Deep Learning is applied in order to solve this image classification. Before applying Deep Learning, the data set is explored in order to understand the data and to get the best performance of the algorithm. This report summarizes the findings after performing some Exploratory Data Analysis (EDA). In addition, pre-processing is applied in order to prepare the dataset for Deep Learning. Afterwards, a training set and a testing set is created, where the already created splits of the creators of the dataset are used. The test set should only be applied after the final model is trained and optimized. The aim of the hold-out test set is to check the performance of the model in the real world.

In the methodology section, different convolutional neural network (CNN) architectures are compared in order to find the best one. Data augmentation is used in order to increase the variance in the dataset and to avoid over-fitting of the model. As a next step, different image sizes are used to train a model and the performances are then compared. This is to find the best suited image size regarding performance and over-fitting. Afterwards, the training set is over-sampled, because the dataset itself is imbalanced. The performance of the over-sampled model is then again compared to the not over-sampled model. As a last optimization step, the hyperparameters are optimized using Bayesian Optimization.

In the results section, the final model is trained by using all the optimal parameters found in the methodology section. The final model is then tested on the hold-out test set in order to check the performance in the real world. Furthermore, the final model is stored in a file such that it can be easily imported in case someone wants to use the cars classifier.

The conclusion section briefly summarizes this project, followed by a future work section, which discusses some possible further steps that could be done to maybe further improve the performance of the final model.

The code and all required files can be found on my Github page [1] in Jupyter notebooks.

2 Exploratory Data Analysis and Pre-Processing

The dataset consists of 16184 images in total. For each image there is also a label, the coordinates for the corners of the bounding boxes and a flag indicating whether the image belongs to the test set or not. In a first step, the bounding boxes are discarded, because they are not required in order to train a cars classifier. The dataset contains 196 different classes, while each class corresponds to a car model. Figure 1 shows the distribution of the full dataset, while one can see that the dataset is slightly imbalanced. Therefore, over-sampling can be applied to balance the dataset and to check, whether this leads to an improvement or not. Almost 50% of the dataset is put into the test set, while preserving the class distribution. This is very important, because this ensures that the test set reflects the real distribution and that all classes are present in the test set. Otherwise, it could happen that the performance on the test set is quite good, but in the real world the model would deliver bad performance on some not good represented classes. In a next step, the image sizes of some randomly chosen images are printed. This is to check whether all images are of the same size or if some adaption is required. But all image sizes printed are of a different shape. This means, that the images must be resized to a fixed image size later in the input pipeline. All images are RGB-images. Figures 2 and 3 are showing two example images of this dataset.

The class labels are stored as integer labels. Therefore, they are one-hot encoded first. Afterwards, the training set is further split into training and validation sets, while using 20% of the initial training set for the new created validation set. A stratified split is used in order to preserve the same distributions between the training and validation sets (Figure 4).

As pre-processing, all images are normalized such that all pixel values are in the range of [0, 1]. This is applied in the input pipeline using Tensorflows "ImageDataGenerator" class. This class has the advantage, that images can be loaded batch wise during training by using the "flow_from_dataframe" function, which helps to save a lot of RAM memory during the training process.

3 Methodology

This chapter briefly describes the chosen strategies for training the model and shows some evaluation results. In machine learning, all strategies can be compared to each other and the best performing can be chosen for the final training strategy. The data set is slightly imbalanced (chapter 2). Therefore, the f1-score is used as target metric for comparing the different training strategies to each other. The next sub-chapters are describing the different training strategies and their results when they are used for training a model. The first sub-chapter describes the used data augmentations (chapter 3.1). The next sub-chapter contains the results of the comparison of different CNN-architectures (chapter 3.2). Afterwards, different image sizes are compared and the best one is used as target image size for the final model (chapter 3.3). Then over-sampling is used in order to balance the dataset and it is checked, whether this helps to boost the performance of the model (chapter 3.4). As a last step, the optimal hyperparameters are searched by using a Bayesian hyperparameter search (chapter 3.5).

[1] link to Github: `https://github.com/patrickbrus/IBM_Machine_Learning_Professional/tree/master/Deep_Learning`

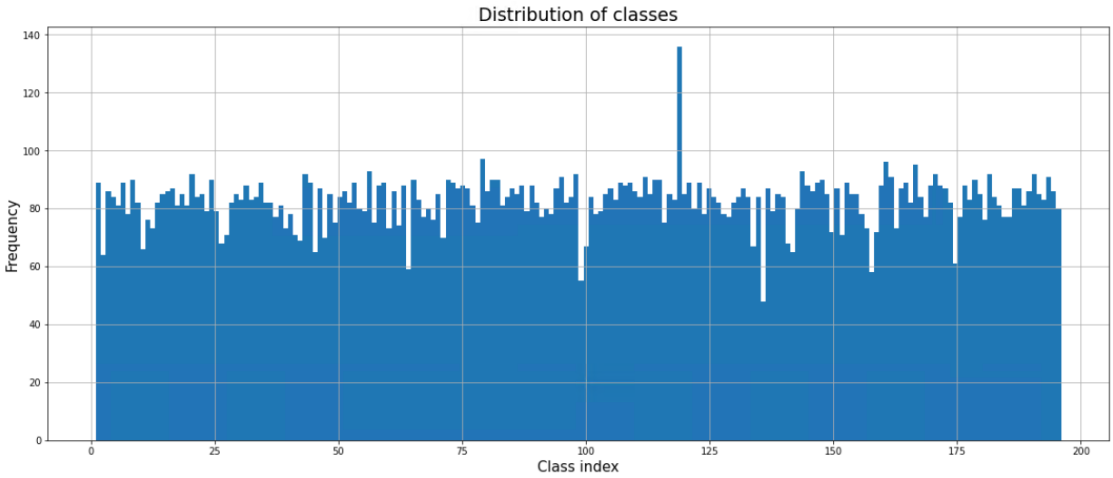

Figure 1: Overview of the dataset and the distribution of all classes.

Figure 2: Example image of a Dodge Dakota Club Cab from 2007.

Figure 3: Example image of a BMW 3 Series Wagon of 2012.

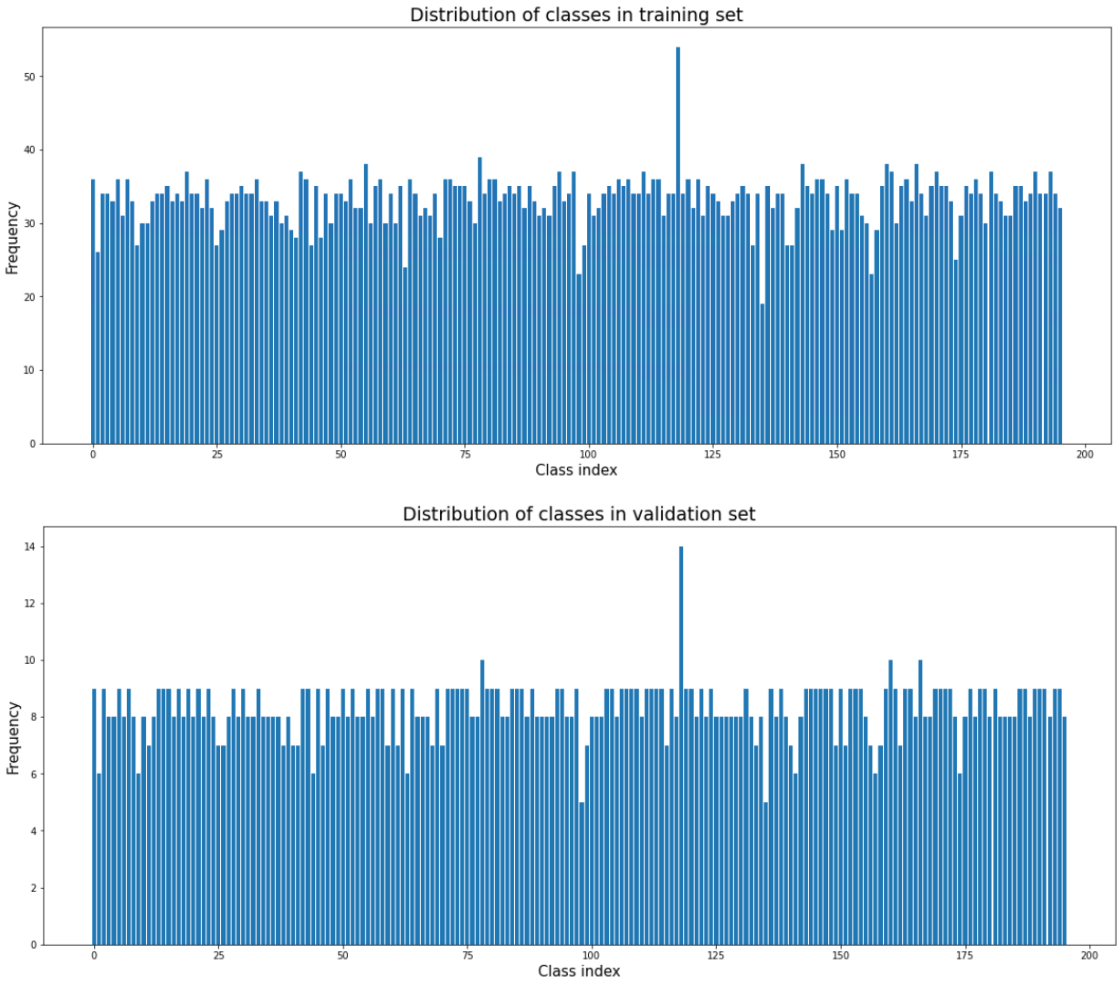

Figure 4: Distribution of the training set (upper) and the validation set (lower). As one can see, the distributions are very similar of training and validation set, which is achieved using a stratified split.

3.1 Data Augmentation

Data augmentations are used in order to increase the variance in the training dataset and to therefore avoid over-fitting and increase the performance on real world data. The data augmentations used for this project are chosen manually by applying an augmentation on an example image and evaluating whether the image still makes sense or not. The evaluation results of the different applied augmentations can be seen in the Jupyter notebook, called "Check_Augmentations.ipynb", in the above mentioned Github repository. In general, one could use these augmentations and could train a model with one additional augmentation at a time and compare its performance to a baseline model without any augmentations. But for the sake of this project, I decided to not make this evaluation, because a model without any augmentations has a very poor performance and so I concluded, that every augmentation leads to an improvement in comparison to the baseline model. The following augmentations are used throughout this project:

1. Flip image left right
2. Multiply pixel values with offset
3. Salt and pepper
4. Gamma contrast change
5. Add offset to pixel values
6. Add additive gaussian noise
7. Apply motion blur
8. Apply an affine transformation
9. Rotate the image
10. Apply an elastic transformation

Augmentation "Add" and "Gamma Contrast" are never used at the same time, because this could lead to unrealistic images. In total, each augmentation is only applied with a certain probability to ensure that in total approximately 20% of the images are not augmented and therefore the model has seen the original images.

3.2 Encoder Search

There are plenty of different available CNN architectures, that could be used for the car classifier. In this project, six different CNN architectures (Table 1) are chosen and compared to each other. Each architecture is used for training a car classifier with the hyperparameters given in Table 2. In the end, the best validation f1-score is stored and used to find the best CNN architecture. Here, only the best validation f1-score is used and it is ignored, that each architecture has a different training complexity and that maybe one architecture leads to only slightly less good validation f1-scores, but has way less parameters than the model which reaches a higher validation f1-score. But for a future optimization, the training complexity could also be considered and could be included into the decision, which encoder architecture to use for the final car classifier. Table 1 also shows the final results. As one can see, the DenseNet121 achieves the best validation f1-score and is therefore used as the final encoder architecture for the cars classifier and also for the other evaluations of this report. Figure 5 also shows the f1-scores during the training process for the different encoders and Figure 6 the loss values during the training process.

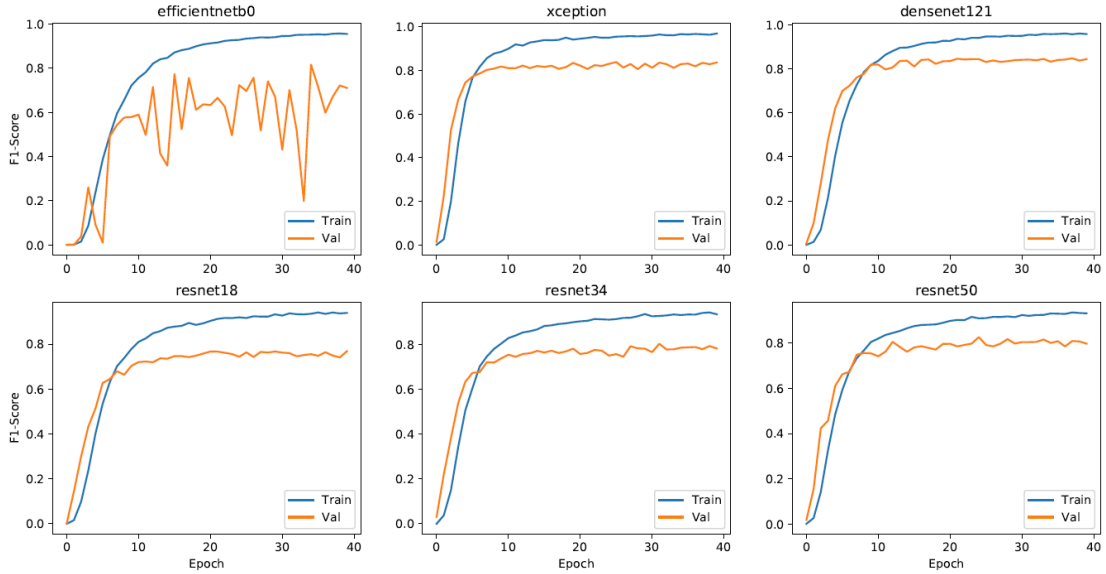

Figure 5: F1-Scores during the training process of the different encoder architectures.

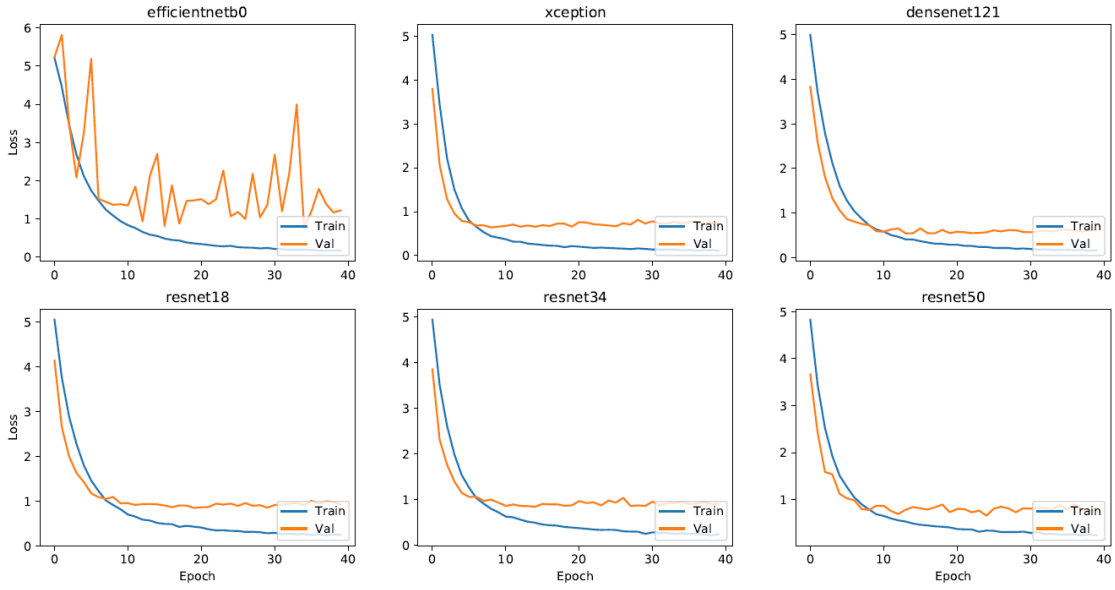

Figure 6: Loss values during the training process of the different encoder architectures. As loss function, the categorical cross-entropy is used.

Architecture Name	Best Validation F1-Score in %
DenseNet121	84.8
Xception	83.9
ResNet50	82.7
EfficientNetB0	81.6
ResNet34	80.4
ResNet18	076.9

Table 1: Used encoder architectures and their best validation f1-scores after training.

Hyperparameter	Value
Number of epochs	40
Optimizer	Adam
Learning rate	1e-4
Batch size	8

Table 2: Used hyperparameters for training the different encoders.

3.3 Image Size Comparison

The optimal image size should be found next on the way to the optimal training strategy. Four different options will be evaluated and compared. A DenseNet is trained for each of the four image sizes with the hyperparameters from Table 2. The results can be found in Table 3, where again the best validation f1-score is taken as a metric for the decision. Figures 7 and 8 are showing the results of the f1-scores and loss values during the training process of the different image sizes. The model that is trained with 256x256 pixels achieves the best performance. Thus, this image size is used for the final cars classifier.

Image Size used for Training	Best Validation F1-Score in %
128x128	65.0
192x192	79.8
224x224	82.5
256x256	84.8

Table 3: Different image sizes used for training and their best validation f1-scores after training.

3.4 Oversampling

As mentioned in chapter 2, the dataset is slightly imbalanced. Therefore, over-sampling is used in order to better balance the dataset and to avoid the risk that the model is more biased towards the majority classes and is less accurate in predicting the minority classes (Figure **??**). A DenseNet121 is then trained using the over-sampled dataset and the hyperparameters from Table 2, except that the number of epochs is reduced to 30 epochs. This is due to the oversampling and therefore the reason that the model should learn faster, because the same images are available more than once within one epoch of training. The best validation f1-score of the over-sampled model is at 84.6%, which is almost the same as in the case where no oversampling is applied. Therefore, oversampling is not used for training the final image car classifier, because it also requires more training time

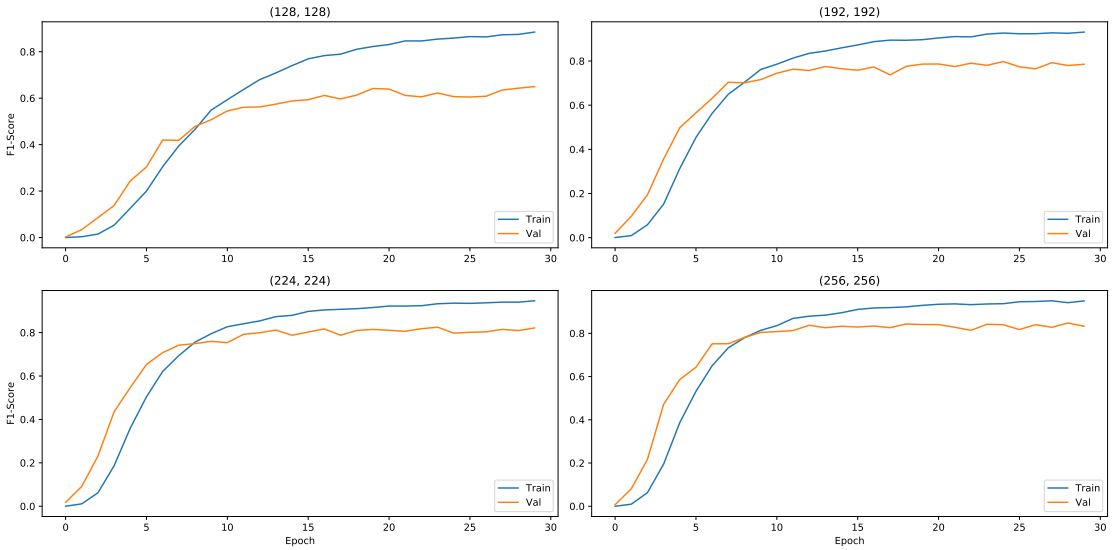

Figure 7: F1-Scores during the training process of training a DenseNet with different image sizes.

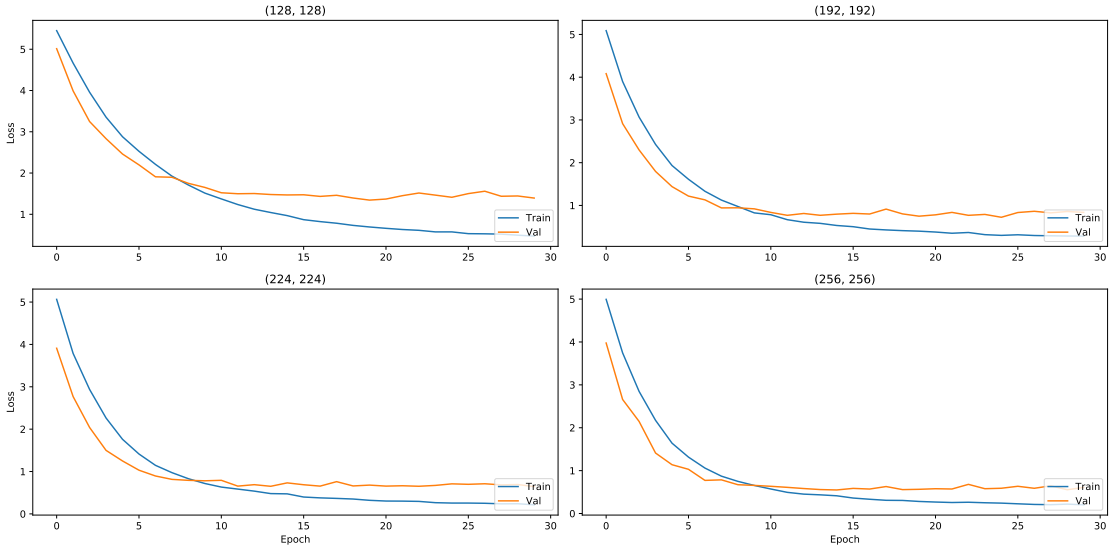

Figure 8: Loss values during the training process of training a DenseNet with different image sizes. As loss function, the categorical cross-entropy is used.

Figure 9: Training dataset after applying oversampling. The training set is now perfectly balanced, which is achieved by copying the same image of the minority classes as long as each class has the same amount of samples.

and is in general more prone to over-fitting.

3.5 Bayesian Hyperparameter Search

As a last step towards finding the optimal training strategy, the hyperparameters are optimized using Bayesian hyperparameter search. The Bayesian search has the advantage, that it is more efficient in finding the optimal hyperparameters than a random search and it requires less iterations than a grid search. A Gaussian Process with a kappa of 3 is used as optimization strategy. Four initial points are provided manually, which shall help to direct the optimization process into the optimal direction. The Bayesian optimization is executed for 12 iterations. The acquisition function called "Lower Confidence Bound" is used and it gets the best validation f1-score as metric to optimize. The Lower Confidence Bound tries to minimize its optimization metric. Therefore, the negative best validation f1-score is used. Table 4 shows the used search space for the hyperparameters, while Table 5 shows the best parameters.

Hyperparameter	**Search Space**
Optimizer	(Adam, RMSProp)
Learning rate	[1e-5, 1e-3]
Batch size	(8, 16)
Decay rate	[0.95, 1]
Decay steps	[1, 3]

Table 4: Optimized hyperparameters and their search space for the Bayesian hyperparameter optimization.

4 Results

The cars classifier is trained with all the findings found in the Methodology section. The cars classifier is now trained for 50 epochs and the best model according to the validation f1-score is

Hyperparameter	Search Space
Optimizer	RMSProp
Learning rate	1e-5
Batch size	16
Decay rate	0.95
Decay steps	3

Table 5: Best hyperparamters found by the Bayesian hyperparameter search.

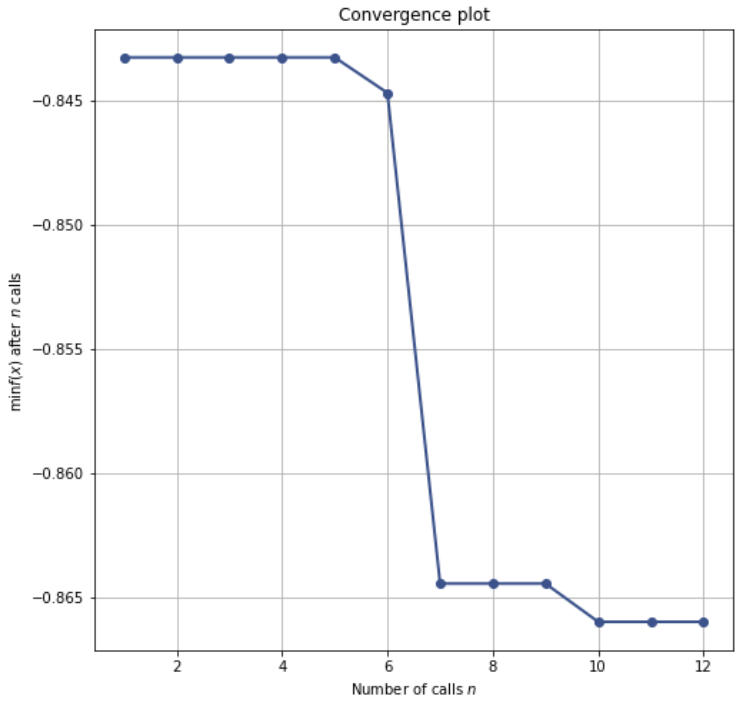

Figure 10: The convergence plot of the Bayes optimization. This plot shows the number of iterations and the achieved optimal f1-score.

stored as tensorflow model (Figure 11). Afterwards, the best model, with a validation f1-score of 87.2%, is loaded and evaluated on the hold-out test set in order to check the performance of the final car classifier on the real world data. The best model achieves a f1-score of almost 87% on the hold-out test set, which is almost the same as the best validation f1-score that is achieved during the training process. This shows, that the model performs very well on real-world data and that there is no over-fitting of the model on the training and validation data.

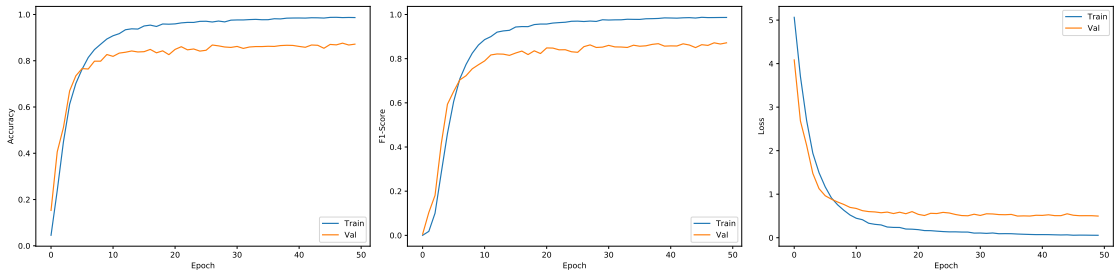

Figure 11: The training metrics of the training process for training the final cars classifier.

5 Conclusion

The final cars classifier achieves a f1-score of almost 87% on the hold-out test set. As underlying CNN architecture, the DenseNet121 is used, because of its good performance. As target image size, the size 256x256 pixels is used. The training set is not over-sampled, because the model trained on the over-sampled training set achieved approximately the same validation f1-score as the model which was trained on the not over-sampled training set. Some hyperparameters are optimized by utilizing Bayesian hyperparameter search.

6 Future Work

Currently, the final cars classifier is only available as Keras file and if someone wants to use it, he or she first has to create the complete input pipeline and has to write a Python program to feed the image into the model and get the prediction. So as a possible future work, the final cars classifier could be deployed on a web server and an API could be programmed, such that a user can directly upload an image of a car and get directly the prediction of the cars classifier without first having to build the application. Another possible future work step is to plot the class activation mappings for the car predictions in order to better understand what the model has learned and what the model uses in order to make the prediction.

References

[1] Statista Research Department. *U.S. vehicle registrations 1990-2019*. 2021. URL: `https://www.statista.com/statistics/183505/number-of-vehicles-in-the-united-states-since-1990/` (visited on 08/31/2021).

[2] jkrause. *Cars Dataset*. 2013. URL: `https://ai.stanford.edu/~jkrause/cars/car_dataset.html` (visited on 08/31/2021).