CIFAR-10 picture recognition analysis
Lab report

Auer Thomas

September 28, 2022

Contents

1. Introduction

This report discusses neural network experimentation on the CIFAR-10 dataset. The implementation is written in the programming language Python, using the libraries Keras and SKLearn. The results by SKLearn are omitted within the report due to length limits but are still at the viewers disposal within the log files. The goal is to analyse various models and identify the impact of different parametrization of the underlying algorithms.

2. Methods and Resources

The datasets are downloaded from a remote destination using the keras.datasets package. Besides additional utility libraries, such as matplotlib and seaborn for displaying graphics, the Tensorflow library supplies the necessary algorithms for creating and executing neural network computations and hyperparameter evaluation.

After downloading the CIFAR-10 dataset it is necessary to perform some data transmutations. The data values are normalized from the value range of (0..255) to (0..1) by simple division. Then training and test labels are categorized from a class vector of integers to a binary class matrix. Lastly, the amount of the data is reduced to decrease the time necessary to run analytics. Afterward, the computation commences.

3. Results

The results stem from static configurations and dynamic parametrization for each training strategy. The training set was decreased from 50000 to 12500, the test set from 10000 to 2500 data points.

Notably, the displayed values are one of many produced results and may vary from execution to execution. The programs can be found in the accompanying IPYNB/PY files, located in the "src" directory of the repository. Results can be found in the "log" directories. The results referenced in the following context are taken from "log-win_advanced_2".

The Keras-driven neural network is the main focus of the experimentation. There are three main phases during the algorithm.

First, we create a standard, sequential model for generating results, which also serve as a reliable reference for other approaches. The basic model always uses the same parameters and layers.

Secondly, we run hyperparameter evaluation using the keras_tuner library, to find more accurate models. The model resembles the standard model, with deviations in activation functions, dropout and learning rates, and units in the dense layer, as well as randomized deactivation of whole layer structures. While almost every aspect of the model can be adjusted (see Overengineered Model), the more fundamental approach (Hyper model) allowed for better training times while being similarly efficient in its search. The Hyper model is used for the three search strategies including Random Search, Bayesian Optimization, and Hyperband

Thirdly, utilizing the findings by the hyperparameter search, a model is created and trained using the most optimal setup found during the evaluation. The model with the highest accuracy score of the three mentioned strategies is chosen as the most optimal model. For reference, the optimized model for the following results was automatically assigned by the algorithm, which is the Hyperband-Search Trial 47. It uses the following configuration:

```
Standard Model       |    Optimal Model
---------------------+--------------------
Conv2D(relu)         | Conv2D(relu)
Conv2D(relu)         | Conv2D(relu)
MaxPooling2D         | MaxPooling2D
Dropout(0.25)        | Dropout(0.3)
Flatten()            | Flatten()
Dense(256,relu)      | Dense(128,sigmoid)
Dense(256,relu)      | Dense(128,sigmoid)
Dense(10,softmax)    | Dense(10,sigmoid)
---------------------+--------------------
learnrate:  0.0001   |  0.001
```

	Standard Model		Optimized Model	
Training time	493.6614336967468s		290.2278664112091s	
Model	Dataset	Exec. Time	Accuracy	Loss
Standard Model	Train	7.391666650772095	0.7254980206489563	0.8031874299049377
Optimized Model	Train	2.7842016220092773	0.9039123058319092	0.302592009305954
Standard Model	Test	1.7537829875946045	0.6062424778938293	1.110999345779419
Optimized Model	Test	0.6126744747161865	0.6202480792999268	1.2597756385803223

Table 3.1: Training time, validation time, accuracy and loss for the standard and the optimized neural network model.

Rank	HPS Strategy	Accuracy	Loss
1	Random Search	0.42296919226646423	1.9702255725860596
2	Random Search	0.35334134101867676	1.8446855545043945
3	Random Search	0.27050819993019104	7.551784038543701
1	Bayesian Optimization	0.4233693480491638	2.0376853942871094
2	Bayesian Optimization	0.43017205595970154	2.0165903568267822
3	Bayesian Optimization	0.4337735176086426	2.024582624435425
1	Hyperband	0.6166466474533081	1.1819623708724976
2	Hyperband	0.5922368764877319	1.2619372606277466
3	Hyperband	0.5314125418663025	1.2906620502471924

Table 3.2: Small excerpt of the hyperparameter evaluation logs – The three best models per strategy and their performances on accuracy and loss.

Surprisingly, the keras_tuner algorithm considered the Bayesian Optimization with lower accuracy as the better model.

The following visualizations further aid the understanding of the results and depict the difference in model fitting – the optimized variant reaches a high point of validation accuracy very fast but does not improve afterward. Given the difference in training and validation, this may indicate overfitting. Interestingly, there is also a model (3.3 – see log-win_err_2) which seemingly had high accuracy. Yet, the actual mapping was entirely incorrect.

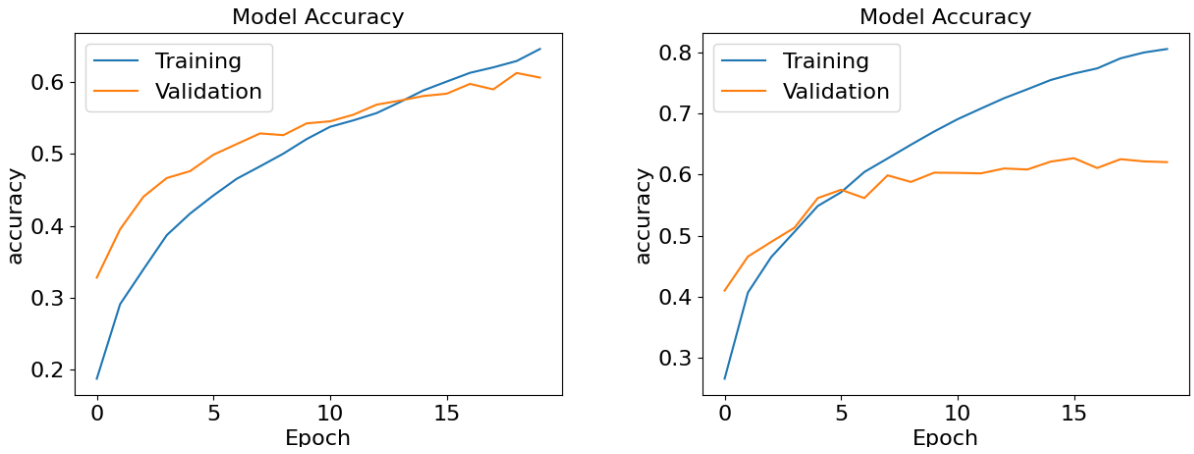

Figure 3.1: Training accuracy score comparison inbetween default network (left) and optimized network (right) parametrization.

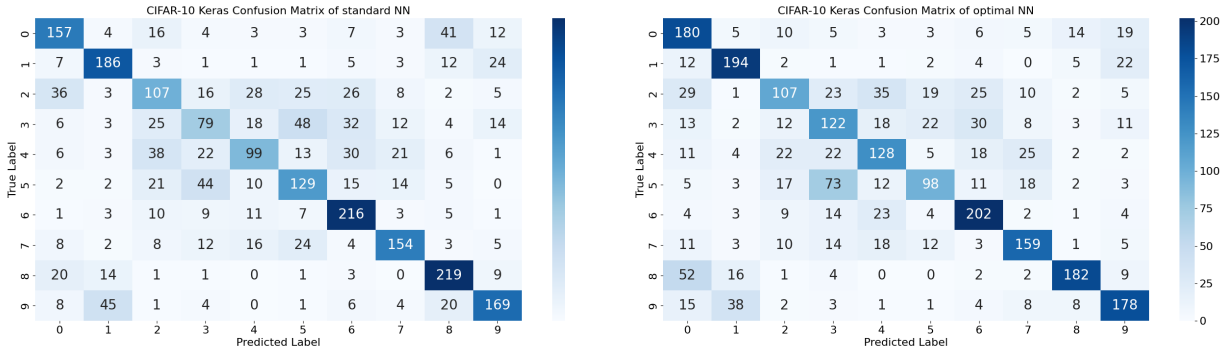

Figure 3.2: Confusion matrix comparison inbetween default network (left) and optimized network (right) parametrization.

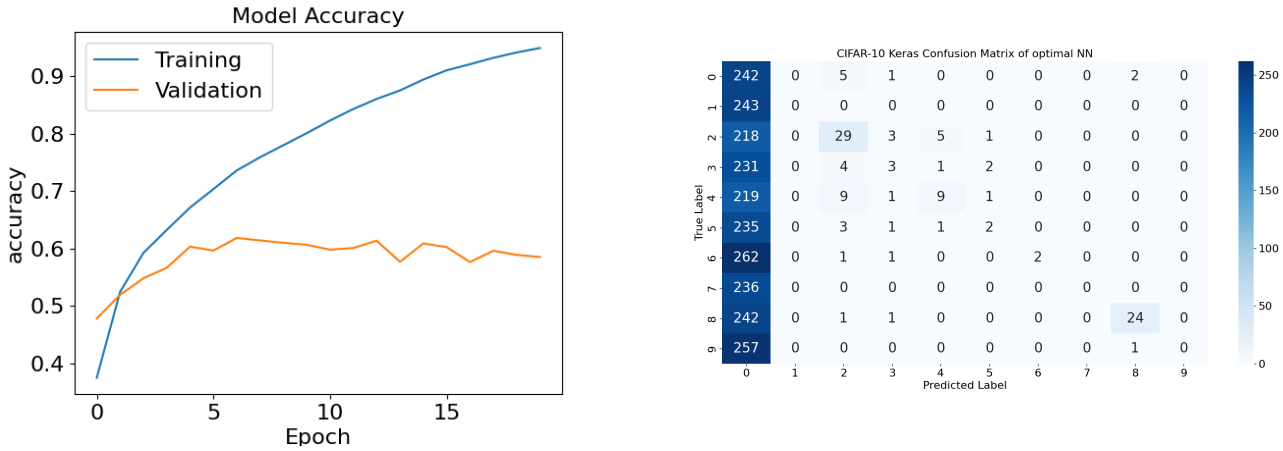

Figure 3.3: Training history (left) and coresponding confusion matrix (right) for a theoretically "good" model.

4. Discussion

The computation on the CIFAR10 dataset has produced various results, some of which were questionable due to overfitting issues. Many results show that a neural network is not necessarily optimal, even when utilizing hyperparameter evaluation on a given data structure. Not only is the evaluation extremely costly in terms of time and power, but the resulting models may even perform less efficiently than a single, well-specified model.

Over the many generations trained during this experimentation, the standard model did satisfactorily even compared to many optimal hyper models. For hyper models, Hyperband was the most consistent algorithm, with results often surpassing the standard model. On average, random search and bayesian optimization did worse than the standard model.

With respect to parametrization, most models do not significantly improve after 4 to 8 epochs. Hyperparameter evaluation further suggests, that reducing the number of layers decreases overall scores, rather than improving them. Different parametrization, such as dropout and learning rate, can slightly improve accuracy.

5. Conclusion

The evluation of neural networks, their structure and their variables, made it evident that searching for optimal models is a suboptimal approach to generate valuable data. For less vital purposes, it is better to create a neural network model specifically suited for the available data and then let it evaluate results, even if that will be less efficient than different models computed by various configurations and setups. Not only does it save a lot of resources but there is no real certainty that a better neural network model will be produced.