

CIFAR-10 picture recognition analysis

Lab report

Auer Thomas

September 21, 2022

Contents

1	Introduction	2
2	Methods and Resources	2
3	Results	2
3.1	Keras neural network	3
3.2	SKLearn multi-layer perceptron	5
4	Discussion	6
5	Conclusion	6

1. Introduction

This report discusses neural network experimentation on the [CIFAR-10 dataset](#). The implementation is handled in the programming language [Python](#), using the libraries [Keras](#) and [SKLearn](#). 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 imported from a remote destination with the use of the [keras.datasets](#) package. Beside 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. A secondary approach using [SKLearn multi-layer perceptron](#) serves as additional reference in terms of computational performance and use. Notably, SKlearn implements [GridSearch](#) for parameter search.

Approach

After the download of the CIFAR-10 data, it is required to perform some data transmutations on the set. The data values are normalized from the value range of (0..255) to (0..1), by simple division. For the Keras computations, the training and test labels are categorized from a class vector of integers to a binary class matrix. SKLearn requires a distinct approach by reshaping the available data of the form (Images, X-Pixel, Y-Pixel, RGB-Channel) to (Images, {X-Pixel, Y-Pixel, RGB-Channel}). Optionally, the size of the used data is reduced in order to reduce the time necessary to run analytics. Afterwards, the main computation on the data commences.

3. Results

The results stem from a common configuration pattern for both SKLearn and Keras, as well as distinct parametrization of algorithms for each. The training set is reduced from 50000 to 6250 datapoints, while the test set is reduced from 10000 to 1250. The reduction was deemed necessary to ensure adequate expenditure of time for the algorithms.

Notably, the displayed values are one of many produced results and may vary from execution to execution. The different computational approaches can be found in the accompanying IPYNB/PY files, located in the "src" directory of the repository. All aggregated results can be found in the "log" directories, each produced from a linux-system and a windows-system with hardware specifications as depicted in the projects "README.md"

3.1 Keras neural network

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 basic information gathering, that also serves as a reliable reference for further approaches. The basic model always uses the same parameters and layers.

Secondly, we run hyperparameter evaluation using the [keras.tuner](#) library, to find models that are more accurate. 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 (Hypermodel) allowed for better training times while being similarly efficient in it's search. The Hypermodel is used for the strategy of a Random Search, a Bayesian Optimization and a [Hyperband](#)

Thirdly, utilizing the findings by the hyperparameter search, a model is created and trained using the most optimal setup found during the evaluation. In the current implementation, the model with the highest accuracy score of the three mentioned strategies is chosen.

For reference, the optimized model for the following results were automatically assigned by the algorithm, which is the Hyperband-Search (Trial 12) on the Windows device. It uses the following configuration:

Standard Model		Optimal Model
-----	+	-----
Conv2D(relu)		Conv2D(relu)
Conv2D(relu)		Conv2D(relu)
MaxPooling2D		
Dropout(0.25)		
Flatten()		Flatten()
Dense(256,relu)		Dense(480,sigmoid)
Dense(256,relu)		Dense(480,sigmoid)
Dense(10,softmax)		Dense(10,sigmoid)
-----	+	-----
lr:	0.0001	0.001

	Standard Model		Optimized Model	
Training time	99.97050666809082s		114.58384299278259s	
Model	Dataset	Exec. Time	Accuracy	Loss
Standard Model	Train	1.7971601486206055	0.5135221481323242	1.3555371761322021
Optimized Model	Train	1.5238370895385742	0.8758201599121094	0.4363606870174408
Standard Model	Test	0.3897683620452881	0.4715772569179535	1.470583200454712
Optimized Model	Test	0.40241503715515137	0.5268214344978333	1.4896273612976074

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

Log	HPS Strategy	Accuracy	Loss
WIN	Random Search	0.46357086300849915	1.4989255666732788
LNx	Random Search	0.40512409806251526	1.7340798377990723
WIN	BayesianOptimization	0.5212169885635376	1.5947179794311523
LNx	BayesianOptimization	0.3010408282279968	6.0242438316345215
WIN	Hyperband	0.5004003047943115	1.6119012832641602
LNx	Hyperband	0.5252201557159424	1.8868718147277832

Table 3.2: Small excerpt of the hyperparameter evaluation logs – Each strategy and their best performances on accuracy and loss for each machine.

Surprisingly, the Bayesian Optimization strategy on linux appeared to have major issues for reasons not further investigated, while it competes well on windows. Given the size limit of this report, please refer to the hyperparameter-tuning-log.txt files for concise information on all strategies and their models.

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 afterwards. Given the difference in training and validation, this may indicate overfitting.

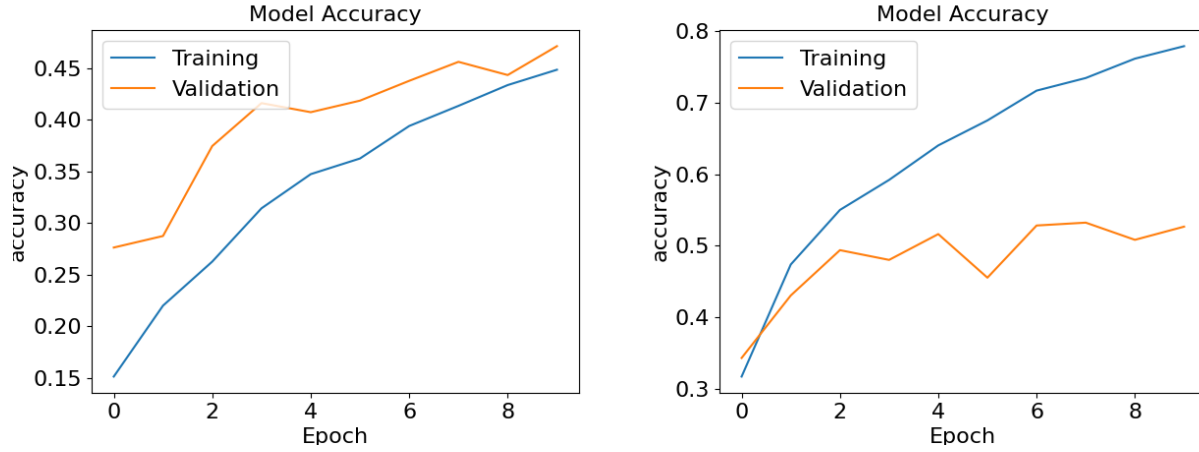


Figure 3.1: Training accuracy score comparison inbetween default network (left) and optimized network (right) parametrization.

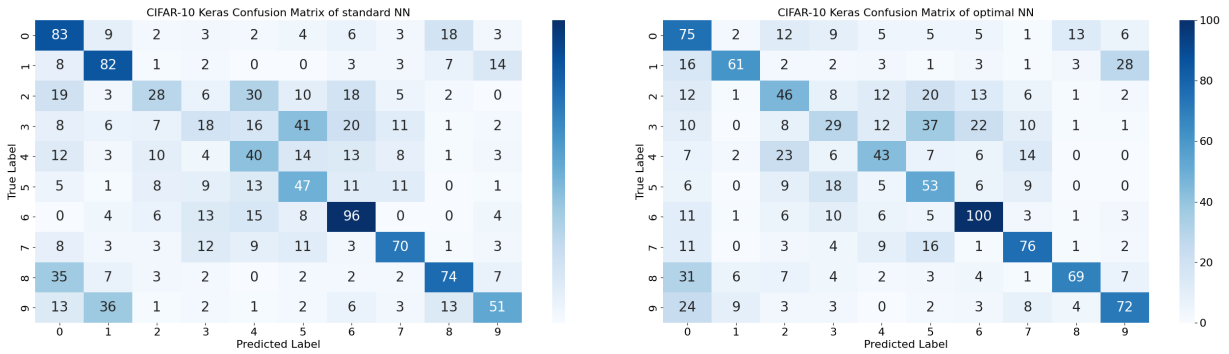


Figure 3.2: Confusion matrix comparison inbetween default network (left) and optimized network (right) parametrization.

3.2 SKLearn multi-layer perceptron

To further investigate the performance on the CIFAR10 dataset, we utilize the multi-layer perceptron of the SKLearn library to add an additional viewpoint on training and testing.

4. Discussion

5. Conclusion