Assignment 1: Applied Deep Learning

Jan Janiszewski, UCNumber:2023198681*
Yorick Scheffler, UCNumber: 2023176104
jasiekj1@gmail.com
yorick.scheffler@student.hpi.com
University of Coimbra
Coimbra, Coimbra, Portugal

ABSTRACT

This report accompanies the two Colab-Notebooks that were created to solve the task given in this assignment. This is a practical introduction to work with Convolutional Neural Networks (CNN).

KEYWORDS

CNN, Neural Networks, Machine Learning, CIFAR10, Image Classification

Hyperparameter	mCA	F1	Precision	Recall
LR: 0.001	79.95	79.86	79.95	79.78
LR: 0.01	39.91	40.34	58.22	39.91
LR: 0.01 (Without StepLR)	27.71	24.55	22.04	27.71
LR: 0.00001 (Without StepLR)	62.74	62.37	62	62.74
Epochs: 20	77.77	77.89	78.01	77.78

Table 1: Results for different parameters (values in %)

1 CREATING & TRAINING A CNN (PART1)

Architecture and Implementation

In the beginning, we started with preparing the data. This is relatively straightforward for the CIFAR10-Dataset. The different data splits can be obtained via functions in this dataset. Therefore it's clear, which data to use for training and validating during the training process. Also, the part of the data is called the test set for evaluating the trained network. As architecture, we used an adapted architecture of the network that was used in the tutorial class of CNNs. We adapted the architecture to the challenges of the CIFAR10-Dataset. For example, previously a black-and-white dataset was used, so for the CIFAR-Dataset, we need to adapt the number of input channels to three, because it is an RGB-Image. After making sure the layout fits the new problem setting, we started a kind of random search to try out different adaptions to increase the performance in terms of the accuracy from 64% to the required at least 75%. The simplest way to increase the accuracy was to add more convolutional layers to the network. These additional layers in combination with a changed activation function and the use of some dropout layers were enough to reach around 73% accuracy. This is still not the required amount of accuracy so we had to find other options to change the network to reach a better accuracy. For that, we took inspiration from the ordering of the layers in VGGNets. So we reduced the amount of pooling layers used to two and always used two convolutions following each other. That architecture then reached around 80%, this seemed enough for the project purpose. Additionally, we included batch-normalization layers with the idea that the neurons learn equally important parameters and don't focus too much on single weights. The motivation for using leaky RELU as an activation function is similar. They didn't seem to hurt the learning process as we started to use them, but they also didn't increase the performance in a significant way, since they help with the dying gradient problem, especially in deeper networks, so using this activation might not be necessary, but it also didn't harm the results. Figure 4 displays the architecture of the network that was used to produce the numbers for the following questions in part 1 of the report.

In table 1 the network is tested with different hyperparameters, to see how the learning process is influenced through these. They are chosen in a way to highlight particular scenarios focusing on underfitting or overfitting scenarios.

The image 13 shows how the model is progressively learning and improving its weights for better accuracy, but the amount of epochs with 10 is not enough with this learning rate to achieve as good results as with the learning rate of 0.001, where nearly 80% is reached after 10 epochs. The higher learning rates have loss curves that increase again in between epochs, which indicates that the updates to the weights with this high learning rate are too much sometimes, which makes the results worse. The use of the StepLR function increases the performance there for a good amount, because in the later epochs, the learning rate is not as huge anymore, which then allows for more meaningful updates.

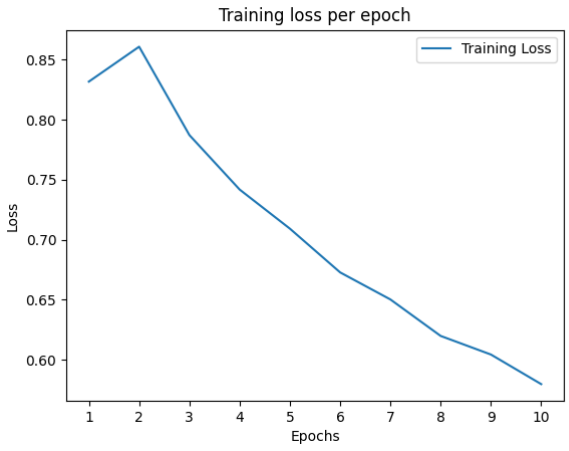

Figure 1: Loss Curve LR: 0.00001

*Both authors contributed equally to this research.

The best run is the first in the table there, we used the learning rate of 0.001 and 10 epochs. The increase to 20 epochs didn't improve the results using the test data split. The monitoring of the learning process showed an accuracy of above 90%, but since it doesn't replicate it with the test data split. It is a strong indicator that it is an overfitting on the training data and an increased amount of epochs would only make sense in combination with data augmentation techniques to increase the amount of diverse data during training. We didn't implement data augmentation since our achieved accuracy with the architecture depicted in figure 4 is already 79.95%. The following shows the resulting loss curve in figure 2 and confusion matrix in figure 3 for this run. The confusion matrix indicates that the models struggle with distinguishing between classes 3 and 5, which corresponds to cats and dogs. This shows that the most common classification mistake is recognizing cats as dogs and vice versa. Their data augmentation as a technique would also help to improve the models' performance in that regard.

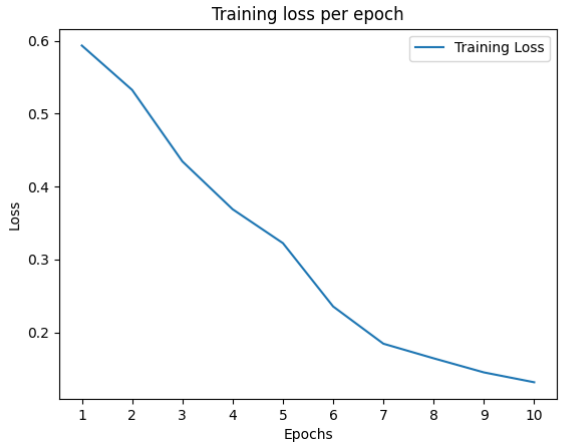

Figure 2: Loss Curve LR: 0.001

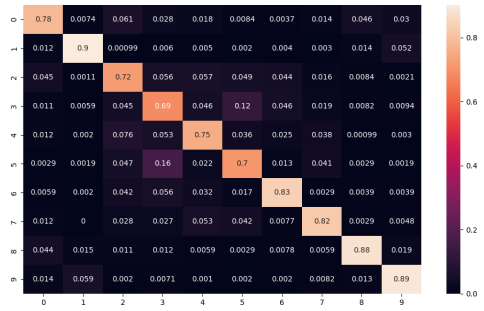

Figure 3: Confusion Matrix LR: 0.001

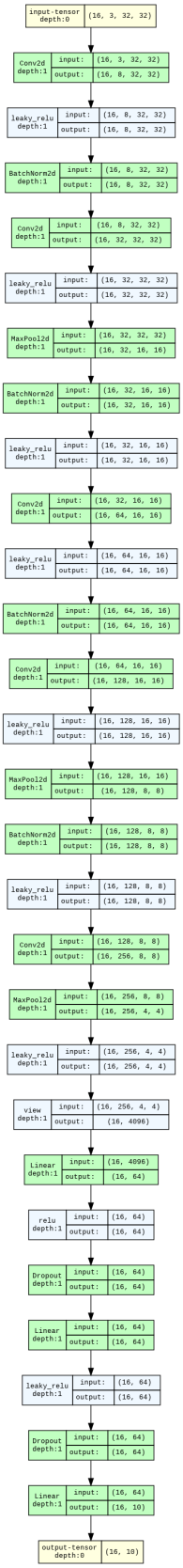

Figure 4: Model-Architecture

2 ALEXNET AND MOBILENET (PART2)

The next task was to train networks with the AlexNet and MobileNetV2 architecture on the EuroSat dataset. This dataset consists of satellite images of the Earth belonging to ten classes. We ran the following scenarios for each of these networks :

- Training from scratch
- Training using weights trained on ImageNet dataset as a starting point.

2.1 Modifications of the network architecture

To adapt the networks to the specificities of our dataset, the following modifications to their architecture were made:

- Since the dataset includes RGB images number of input channels in the first convolution layer was set to 3
- Last layer, fully connected one, output was set to the number of classes in our dataset - 10.

2.2 Networks training

The dataset was split into training and testing data in an 80/20 ratio, the split was done at random. From the training data, the validation data were randomly separated, representing 20% of the training data. The hyperparameters used when training each network are listed in table 2.

Hyperparameter	value
Number of epochs	10
Learning rate	0.001
Optimizer	Adam

Table 2: Hyperparameters used when working with MobileNet and AlexNet

Network model	mCA	F1	Precision	Recall
MobileNet	92.72	92.44	92.85	92.44
AlexNet	92.53	92.36	92.35	92.36
MobileNet (pretrained)	97.44	97.40	97.49	97.32
AlexNet (pretrained)	97.01	96.94	97.05	96.83
Late fusion approach	97.75	97.71	97.78	97.63

Table 3: Obtained results for different network models

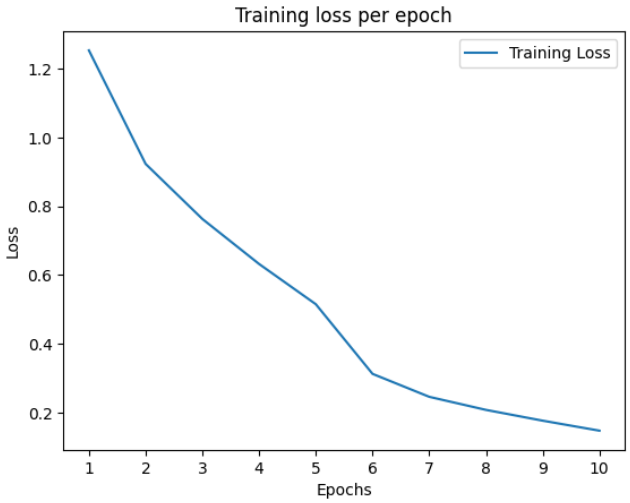

Figure 5: Loss Curve MobileNet

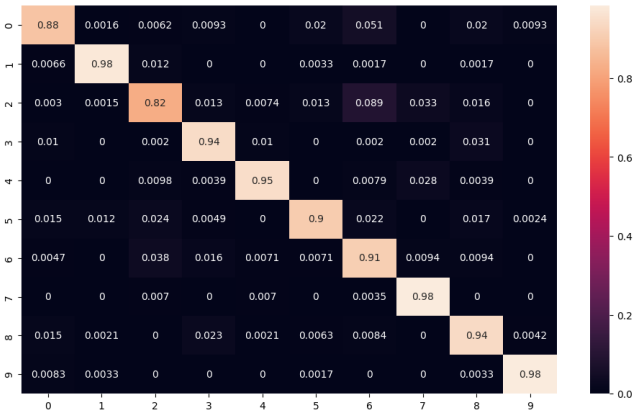

Figure 6: Confusion Matrix MobileNet

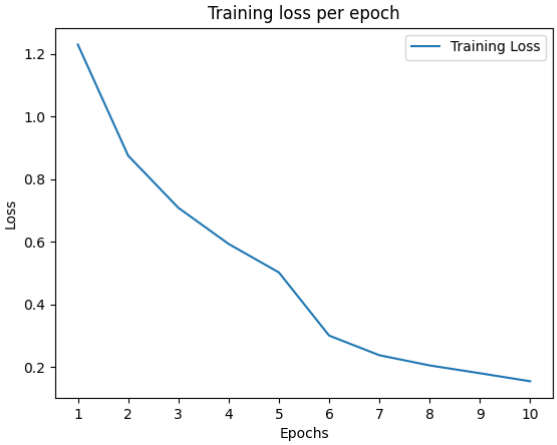

Figure 7: Loss Curve AlexNet

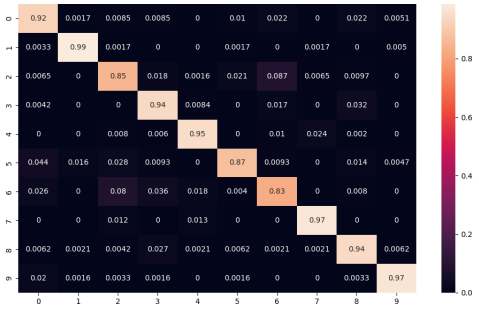

Figure 8: Confusion Matrix AlexNet

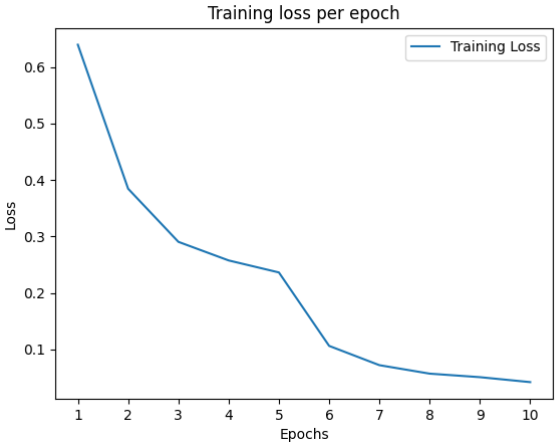

Figure 9: Loss Curve pretrained MobileNet

2.3 Late fusion approach

Using the pretrained AlexNet and MobileNet models, we implemented a 'late fusion approach' technique, this enables the prediction of two networks to be combined to increase classification accuracy. This can be done in a number of ways, we chose to add the vector with the predictions of the two networks together and then select the class corresponding to the maximum value.

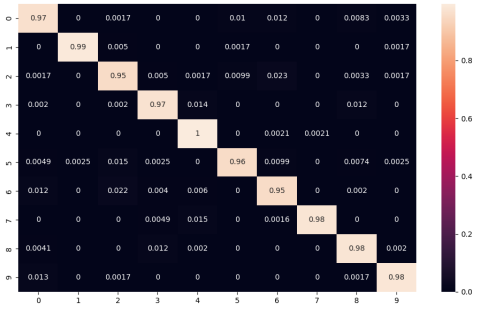

Figure 10: Confusion Matrix pretrained MobileNet

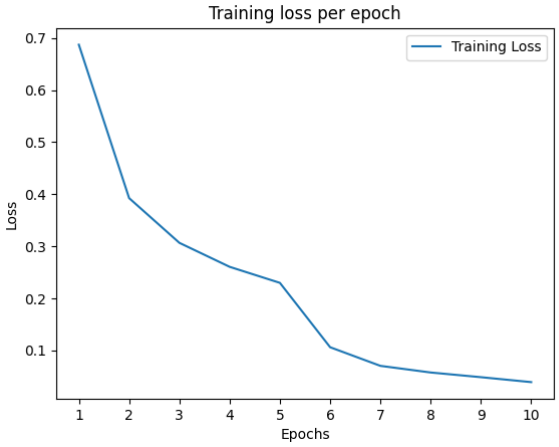

Figure 11: Loss Curve pretrained AlexNet

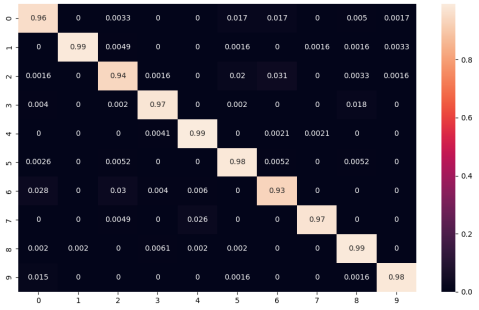

Figure 12: Confusion Matrix pretrained AlexNet

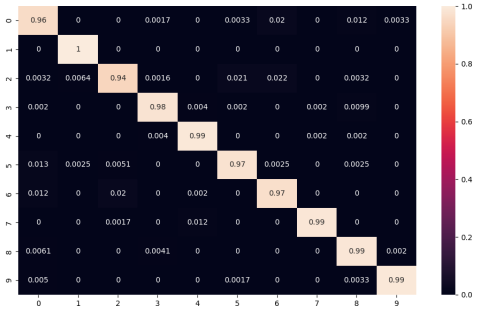

Figure 13: Confusion Matrix of late fusion approach

2.4 Analysis of the results

The best results were obtained using the late fusion approach. However, they are not significantly better than the results obtained using pretrained networks. Networks that have been pretrained have about five percent higher mCA than those trained from scratch. Networks that have been pretrained have about five percent higher mCA than those trained from scratch. For both training from scratch and pre-training, MobileNet performed better than AlexNet while this is not a significant difference. The training loss curve has a similar shape in each case. Comparing the loss curves between from scratch trained and pretrained seems to indicate that the learning process for pretrained models jumps a bit more since only a few weights get actually updated. This seems to be not an unexpected effect because the training process from scratch has a lot of weights to update per epoch, which probably makes the loss go down more smoothly. This is particular shown comparing the decrease of the loss in the epochs 4 to 5.

Analyzing the confusion matrices between the pretrained models and the models trained from scratch, it is noteworthy that class 2 performs the worst in both cases with ALexNet and MobilNet. The reason seems to be a few more misclassifications between class 2 and class 6 compared to the error rates for other classes. Since it occurs in both from scratch trained networks, it might caused by the way the data is split, but stating that for sure, would need further investigations.

3 GOOGLE COLAB WITH CODE

All code that was used can be found here: Assignemnt