Deep learning lab - Assignment 2

Christian Altrichter

November 11, 2022

1 Image Classification Using ConvNets [90/100 points]

1.1 Question 1.1

1.1.1 Question 1.1.1

Inspect the number of images The training data set has 50.000 images and the test data set has 10.000 images.

This question has been solved using the "torchvision.datasets.CIFAR10" function. Subsequently, I iterate through the images in the training set loader and separate the images from the labels. With the image classifications from Question 1.3.6 we add titles and labels to the images. The images are 32 by 32 pixels, have however been increased to a figure size of 128 by 128.

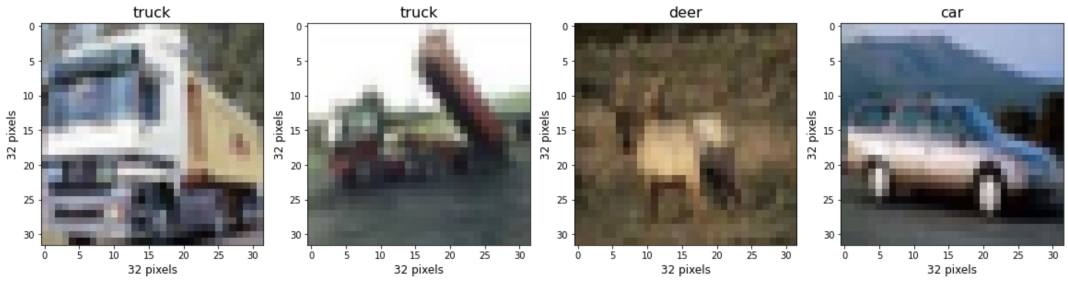

Figure 1: Question 1.1.1, plotting 4 images of the CIFAR10 training-dataset

1.1.2 Question 1.1.2

For this question I used the function "transforms.Compose()" with the Normalization parameters given in the exercises. This transformation has then been passed to both the train_set and the test_set which were then assigned a new variable. Subsequently, I have passed the new train_set_norm to a new data loader for later use in the model.

To verify whether our normalization is correct, I have used the function in 1.1.3 to compute the mean and std of the normalized train loader. If mean is close to 0 and stdev is close to 1 (close to due to rounding errors due to machine precision), then the normalization has been performed correctly.

After normalization we achieve the following means and stdev respectively:

$$mean = [-0.e+00, -2.e^{-04}, 1.e^{-04}]$$
$$stdev = [1., 1.002, 1.002]$$

1.1.3 Question 1.1.3

Please refer to the code written to compute the standard deviation and the mean. In the function, we output a message whether the given mean equates to the calculated mean, thus, checking the correctness of the given statistics.

The function used here is called "mean_and_std".

1.1.4 Question 1.1.4

With regards to this question, I have used the function "data.random_split()". Here I split the training data set as requested into training and validation sets and load the individual sets respectively.

1.2 Question 1.2

1.2.1 Question 1.2.1

Please refer to the source code with regards to the individual implementations.

1.3 Question 1.3

Note: For the training I have created an outer function "image_classification()" with parameters:

1. *learning_rate*
2. *momentum*
3. *epochs*
4. *dropout_value*

The function returns the model, the loss evolution for both training and validation as well as the accuracy evolution for both training and validation. This will be passed to a function to display the graphs.

1.3.1 Question 1.3.1

The training pipeline has been implemented in the code. I monitor the training loss and accuracy every n steps (where n = 200). Furthermore, at the end of each epoch I check whether the best_validation_accuracy for the current epoch is better than the previous one, if so, the tuple will be overwritten and output at the end of the training.

1.3.2 Question 1.3.2

The model has been trained with the following parameters:

1. learning rate: 0.001
2. Momentum: 0.9
3. Epochs: 20
4. Dropout: 0 (as introduced in 1.3.5, thus not used here).

1.3.3 Question 1.3.3

With the above mentioned parameters, I have achieved the following results:
Here, a total accuracy of 70.8% was achieved and the highest validation accuracy of 71.1% was achieved in epoch 17.

1.3.4 Question 1.3.4

Note: To depict the graphs I have created an outer function "plot_loss_and_acc()" that takes the return values of "image_classification()" from 1.3 to draw the graphs respectively.

Comment: The loss shows graph depicts an over fitting of the model due to the fact that the loss of the validation data set increases. This also leads to the result of a mere validation set accuracy relatively lower to the training set accuracy. Here, the final test accuracy was 70.42%.

With the result of 1.3.3 I have depicted the training and validation losses as follows:

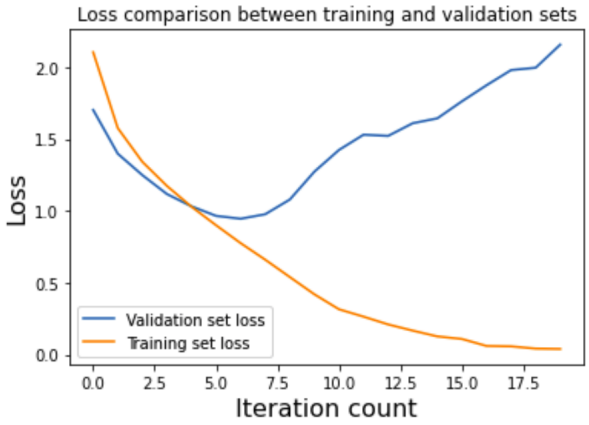

Figure 2: Graph for Question 1.3.4, plotting the loss of our training model from 1.3.3

And the training and validation losses as follows:

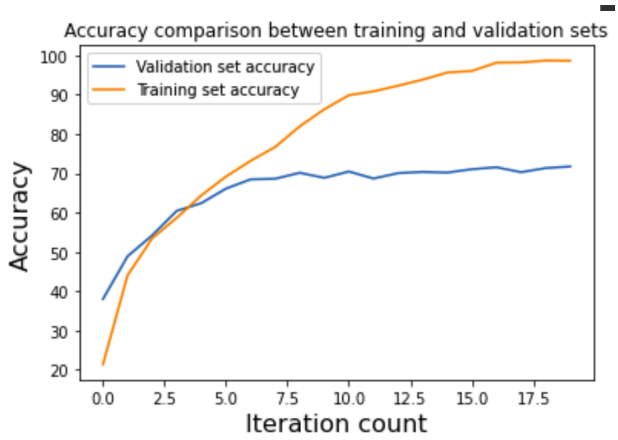

Figure 3: Graph for Question 1.3.4, plotting the accuracy of our training model from 1.3.3

1.3.5 Question 1.3.5

The argument "p" in nn.Dropout defines how many neurons are eliminated in every layer, thus not included in the model. For the upcoming layer these dropouts will not be considered. This means that specific weights are set to 0. This proves itself to be quite effective when training neural networks.

Comment on dropout of 0.2 vs no dropout: The dropout of 0.2 does not fundamentally impact the behaviour of the model in terms of accuracy and loss. Thus, the differences are negligible for the sake of commenting. Here, the final test accuracy was 70.69% (compared to no dropout of 70.42%).

Here the loss (Figure 4) and accuracy (Figure 5) for the model with a dropout of 0.2:

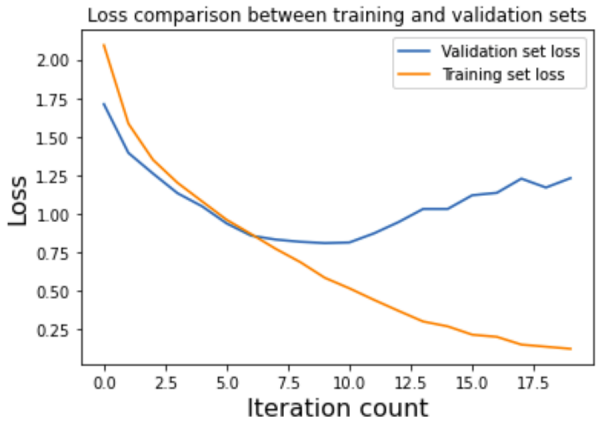

Figure 4: Graph for Question 1.3.5, plotting the loss of our training model, dropout = 0.2

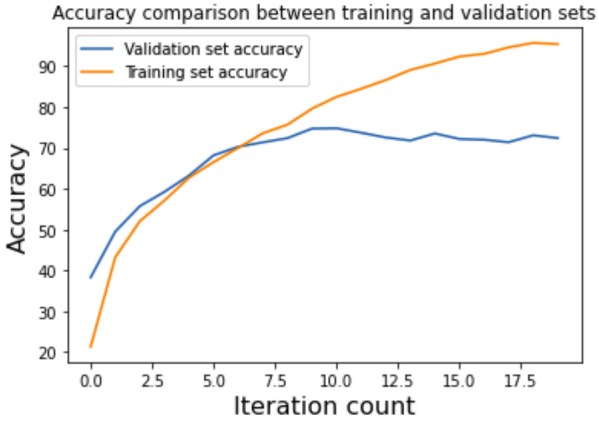

Figure 5: Graph for Question 1.3.5, plotting the accuracy of our training model, dropout = 0.2

Here the loss (Figure 6) and accuracy (Figure 7) for the model with a dropout of 0.5:

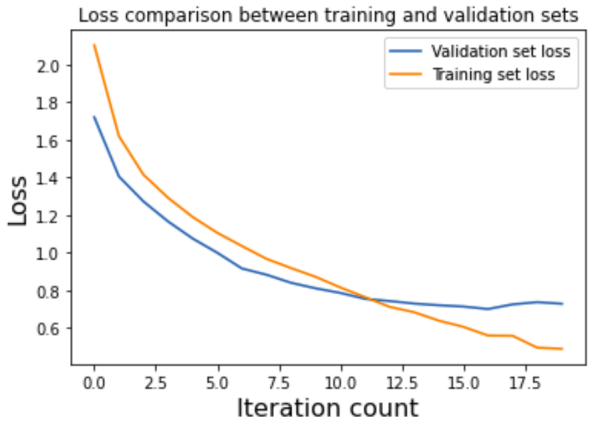

Figure 6: Graph for Question 1.3.5, plotting the loss of our training model, dropout = 0.5

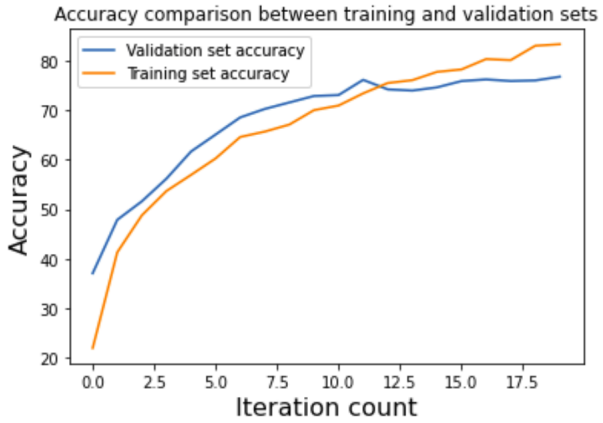

Figure 7: Graph for Question 1.3.5, plotting the accuracy of our training model, dropout = 0.5

Comment on dropout of 0.5 vs 0.2 dropout: The dropout of 0.5 reduces the over fitting of the previous model(s) model drastically, leading to a better convergence of the validation data set loss and accuracy. Both, test set accuracy and validation set accuracy are clearly above 70.0%. Here, the final test accuracy was 74.04%.

Here the loss (Figure 8) and accuracy (Figure 9) for the model with a dropout of 0.8:

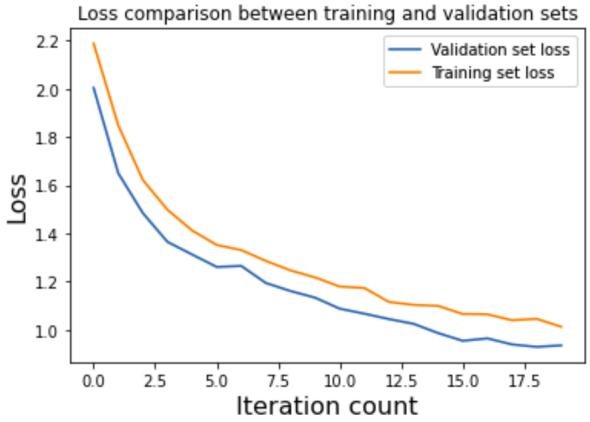

Figure 8: Graph for Question 1.3.5, plotting the loss of our training model, dropout = 0.8

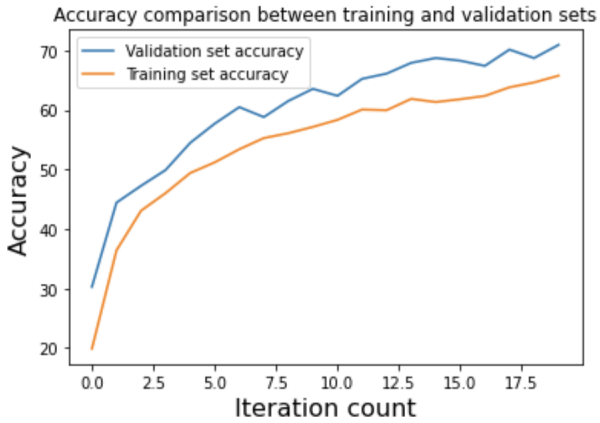

Figure 9: Graph for Question 1.3.5, plotting the accuracy of our training model, dropout = 0.8

Comment on dropout of 0.8 vs 0.5 dropout: The dropout of 0.8 remains with no over fitting in the model. However, here, the dropout rate is slightly too high which reduces the accuracy of the model for both test and validation set compared to a dropout of 0.5. Here, the final test accuracy was 66.72%.

1.3.6 Question 1.3.6

The best model achieved in 1.3.5 was the model with a dropout of 0.5. The final test set accuracy was 74.04% and the highest validation accuracy was achieved in epoch 17 of 75.5%.

I have split the following examination into two picture sets, one set not-normalized and one set normalized to depict the difference it has on image classification. For each of the tables it has been marked green, if the classificaiton is correct, or red, if the classification is false.
Note: The images selected of the test set are pseudo random, thus, iterated through in consecutive order.

The images of the not normalized validation set are as follows:

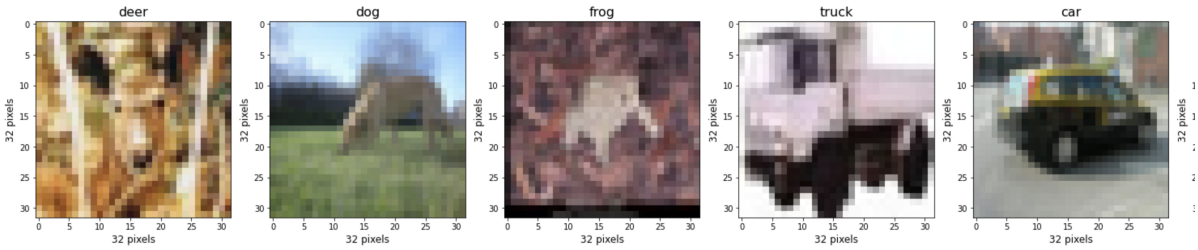

Figure 10: Question 1.3.6, plotting images 1-5 (out of 10) of the CIFAR10 test-dataset

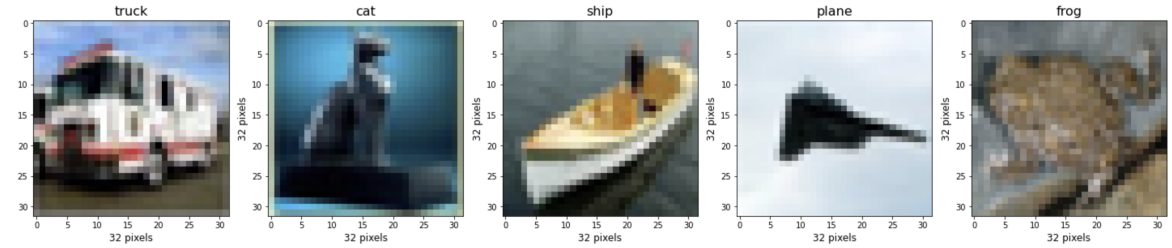

Figure 11: Question 1.3.6, plotting images 6-10 (out of 10) of the CIFAR10 test-dataset

The above images have been given the following classification by the highest output probability:

Images	Plane	Car	Bird	Cat	Deer	Dog	Frog	Horse	Ship	Truck
Image1 - Deer:	0.	0.	0.014	0.209	0.613	0.022	0.142	0.	0.	0.
Image2 - Dog:	0.075	0.	0.365	0.145	0.103	0.057	0.05	0.003	0.2	0.001
Image3 - Frog:	0.006	0.	0.087	0.333	0.406	0.109	0.055	0.002	0.003	0.
Image4 - Truck:	0.013	0.	0.023	0.531	0.325	0.066	0.027	0.002	0.011	0.002
Image5 - Car:	0.039	0.001	0.048	0.471	0.118	0.092	0.207	0.002	0.022	0.001
Image6 - Truck:	0.01	0.001	0.01	0.29	0.03	0.057	0.081	0.004	0.482	0.036
Image7 - Cat:	0.116	0.	0.177	0.167	0.141	0.255	0.093	0.013	0.035	0.001
Image8 - Ship:	0.006	0.	0.048	0.072	0.002	0.002	0.044	0.	0.826	0.
Image9 - Plane:	0.617	0.001	0.122	0.043	0.008	0.011	0.076	0.001	0.118	0.003
Image10 - Frog:	0.01	0.	0.22	0.281	0.086	0.094	0.304	0.001	0.005	0.

Here, the accuracy of classification is at 40%.

The images of the normalized validation set are as follows:

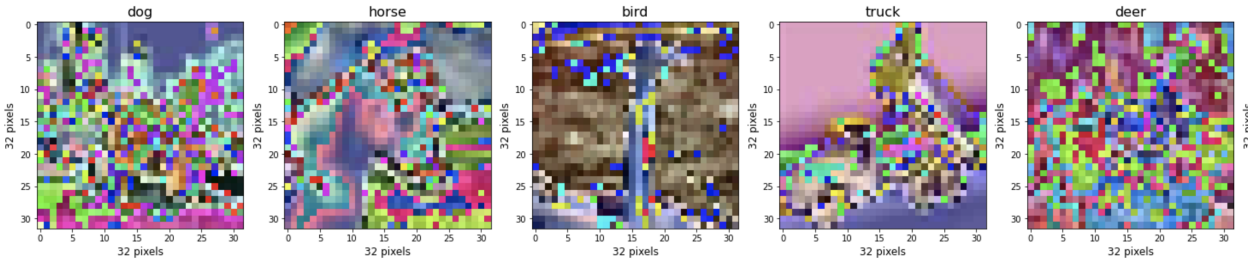

Figure 12: Question 1.3.6, plotting images 1-5 (out of 10) of the CIFAR10 test-dataset

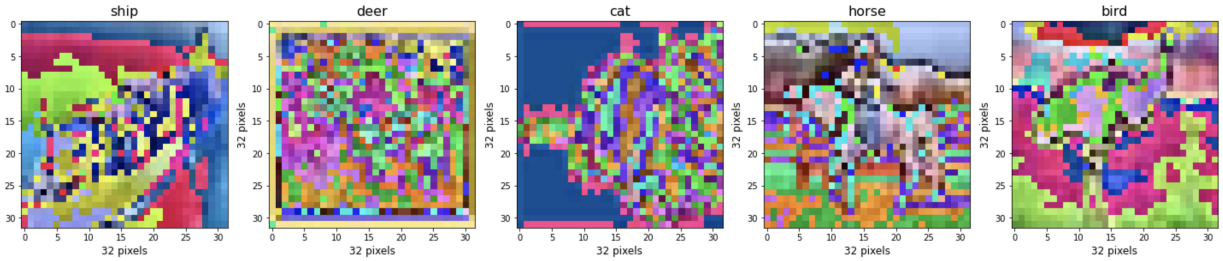

Figure 13: Question 1.3.6, plotting images 6-10 (out of 10) of the CIFAR10 test-dataset

The above images have been given the following classification by the highest output probability:

Images	**Plane**	**Car**	**Bird**	**Cat**	**Deer**	**Dog**	**Frog**	**Horse**	**Ship**	**Truck**
Image1 - Dog:	0.	0.	0.002	0.038	0.79	0.149	0.018	0.003	0.	0.
Image2 - Horse:	0.	0.	0.	0.	0.001	0.	0.	0.999	0.	0.
Image3 - Bird:	0.	0.	0.998	0.002	0.	0.	0.	0.	0.	0.
Image4 - Truck:	0.031	0.012	0.053	0.017	0.003	0.029	0.002	0.715	0.007	0.13
Image5 - Deer:	0.	0.	0.	0.008	0.986	0.004	0.	0.003	0.	0.
Image6 - Ship:	0.029	0.017	0.012	0.008	0.035	0.002	0.016	0.001	0.831	0.048
Image7 - Deer:	0.	0.	0.	0.006	0.834	0.01	0.003	0.148	0.	0.
Image8 - Car:	0.	0.	0.007	0.122	0.008	0.754	0.056	0.049	0.	0.004
Image9 - Horse:	0.	0.	0.	0.	0.	0.	0.	1.	0.	0.
Image10 - Bird:	0.	0.	0.715	0.014	0.219	0.009	0.001	0.043	0.	0.

Here, the accuracy of classification is at 70%.

1.3.7 Question 1.3.7

I have achieved a total validation accuracy of 80.3% in the last epoch **and a corresponding test accuracy of 80.15%**. Furthermore, out model achieved its best accuracy in epoch 13 at 81.6%. This has been done by adjusting the following parameters and code sections:

1. learning rate adjusted to 0.01 (previously at 0.001)
2. Momentum adjusted to 0.85 as I observed that the model oscillates slightly under 80% meaning that it was overshooting the local minima.
3. Epoch has not been adjusted (thus remained at 20 epochs).
4. Dropout has been slightly increased to 0.55
5. Instead of using relu we are now using gelu.

Here, the following losses (Figure 14) and accuracies (Figure 15) are depicted as follows:

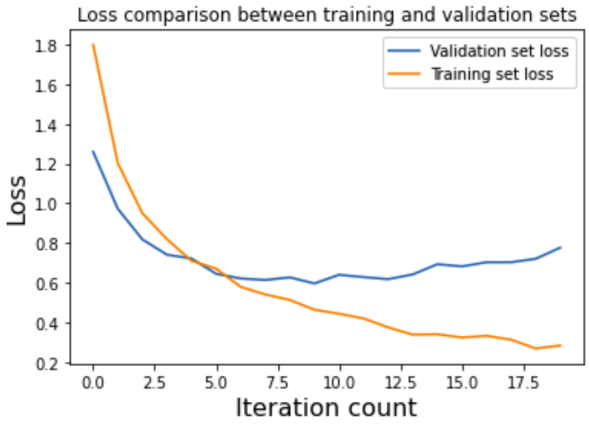

Figure 14: Graph for Question 1.3.7, plotting the Loss of our training model

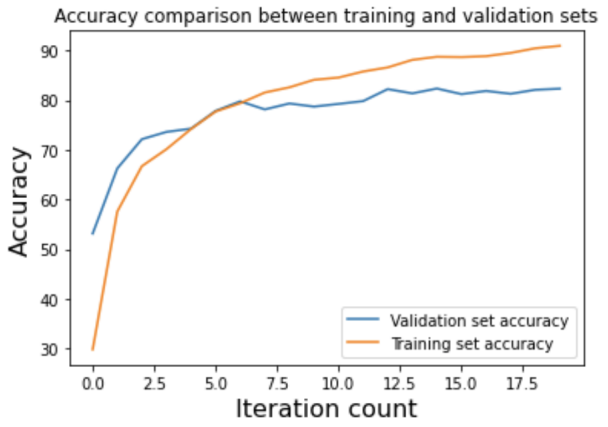

Figure 15: Graph for Question 1.3.7, plotting the accuracy of our training model

1.4 Question 1.4

1.4.1 Question 1.4.1

The softmax function is used as an activation function when using neural network models to normalize the outputs as a probability.

1.4.2 Question 1.4.2

1.4.3 Question 1.4.3

One example type for 1D convolution is a time series.

1.4.4 Question 1.4.4

During training time weights get set to 0 based on the argument p of the dropout. However, during test time the the weights remain unchanged.