DeepLearning Lab **2022/2023**

Student: LUCA PERNIGO StudentID:19-993-658

Assignment 2 Due date: 13 November 2022, 10:00 PM

---

1. Image Classification using ConvNets

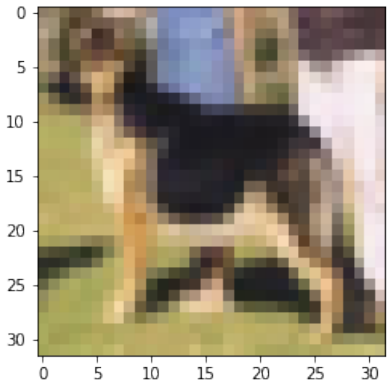
1.1. Dataset

1.1.1.

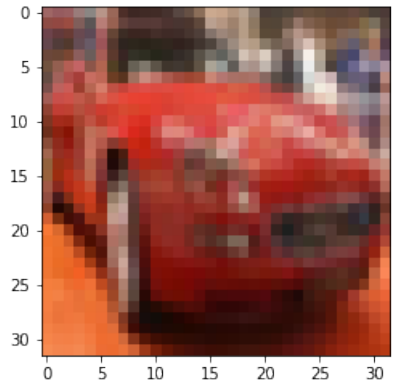
This report analyzes how convolutional neural networks perform in classifying images of the CIFAR-10 dataset. The above mentioned dataset can be loaded simply by importing torchvision and then calling torchvision.datasets.CIFAR10(), since this package already includes the CIFAR-10 dataset.

To access the information regarding our dataset call print(train_set) and print(test_set); by doing so the computer reports that the training set is composed of 50000 images while the test set is composed of 10000 images, therefore the CIFAR-10 dataset is made up of 60000 images.

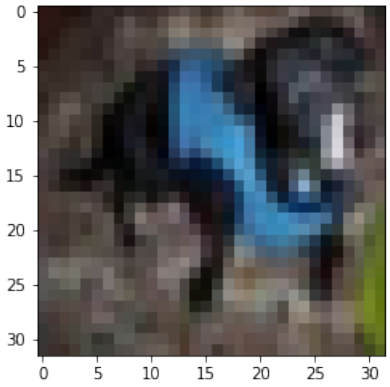
To give an example of the data contained in the CIFAR-10 dataset, four randomly images are reported hereafter Figure 1. Note, that the function imshow expects the images in the format [32, 32, 3], however when imported the images are in the format [3, 32, 32]. Thus, in order to set the images in the right format for plotting, it is necessary to call the transpose function in this way .transpose(1,2,0) so that the number of channels becomes the last dimension.



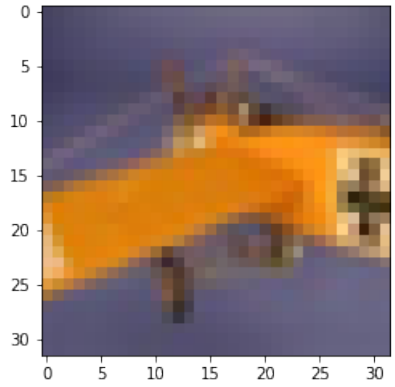
.: A german shepherd



.: A red car



.: A black dog



.: An yellow plane

Figure 1: Four random images from training set.

1.1.2.

In order to normalize the data, the normalization transform is added to the transform argument of torchvision.datasets.CIFAR10. This is done by passing to the transform argument of torchvision the element transorm.Composed made up of the ToTensor() and the Normalize() transforms.
To carry out our normalization, it is necessary to provide to the Normalize() transform the mean and standard deviations of each channel across the images in the training data. In this case the values to add are:
means=(0.4914, 0.4822, 0.4465) and
stds=(0.247, 0.243, 0.261).

1.1.3.

This subsection verifies that the means and standard deviations of the previous section are indeed the correct ones; just take for each channel all the pixels of all images and apply np.mean() and np.std(). Note that in this case the pixels are in the range from 0 to 255, therefore to convert their values to the 0-1 scale it is required to divide the means and standard deviations by 255. Doing so, the following values are obtained:
means=(0.49139967861519607, 0.48215840839460783, 0.44653091444546567) and
stds=(0.2470322324632819, 0.24348512800005573, 0.26158784172796434).

1.1.4.

Next, the 50000 images of the original training set were split into a validation set and into an effective training set. The validation contains 1000 images while the training consists of 49000 images.

Two different types of dataloaders have been implemented for this report for both the training and the validation set.
The first type is a dataloader with the WeightedRandomSampler sampler; in this case equal weights have been specificied and the replacement argument has been set to False so that images are drawn without replacement.
The second type is a dataloader with the SubsetRandomSampler sampler; in this case our sampler selects elements randomly from a list of given indices, without replacement.

1.2. Model

1.2.1.

Our analysis will concern the following convolutional neural network:

(a) Convolutional layer 1: 32 filters, 3×3.
(b) Convolutional layer 2: 32 filters, 3×3.
(c) Max-pooling layer 1: 2×2 windows.
(d) Convolutional layer 3: 64 filters, 3×3.
(e) Convolutional layer 4: 64 filters, 3×3.
(f) Max-pooling layer 2: 2×2 windows.
(g) Fully connected layer 1: 512 units.
(h) Output layer.

In this model there is no padding and the stride=1 for the convolutional layers, while stride=(2,2) for the max-pooling layers.
The activation function for the output layer is already embedded within nn.CrossEntropyLoss(), that is the reason why this last activation function has not to be explicitly specified in the model implementation.

1.3. Training

1.3.1.

After having built the model, the training pipeline has been implemented in such a way that the training loss and the accuracy on the training set are printed every 200 steps while the validation accuracy is printed at every epoch. Furthermore, the valuation accuracies for each epoch are stored in a dictionary, so that it is possible to keep track of the best validation accuracy and its corresponding epoch

1.3.2.

These are the hyperparameters used to train the model:

(a) SGD optimizer with learning rate 0.001 and momentum 0.9.
(b) A batch size of 32.
(c) 20 epochs.
(d) The loss adpoted was the cross entropy loss.

1.3.3.

After having trained the model, the final validation accuracy was 72.3% which is above the 70% threshold. Additionally, the best validation accuracy was 73.6% and it was achieved at the twelfth epoch.

1.3.4.

Hereafter, two plots are reported. In Figure 2 it is visualized how the training and validation losses behave as the number of epochs increase. Figure 3 shows how training and validation accuracy evolve over epochs.

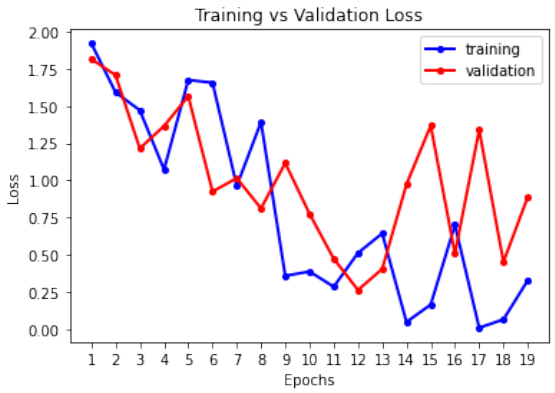


Figure 2: Training and Validation Loss over epochs

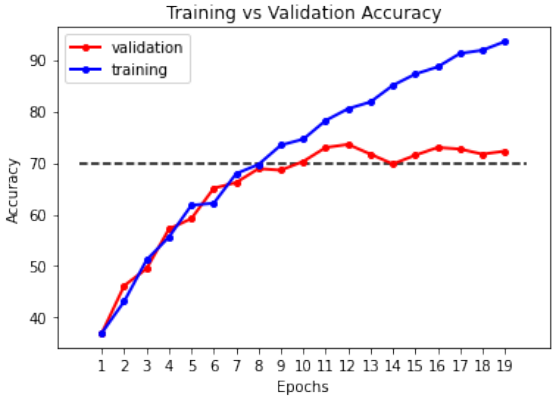


Figure 3: Training and Validation accuracy over ecpochs

The training loss constantly decreases as more epochs are run. Conversely, the validation loss decreases until epoch thirteen; from that epoch onward the validation loss starts fluctuating without significant improvements over the following epochs.

Note, that the accuracy on the validation set is above the 70% threshold. In addition, as the number of epochs increase the model accuracy on the training set tends to 100% while the model accuracy on the validation set fluctuates around 72%. This indicates that our simple model can learn the training set perfectly but when generalizing to the validation set, it achieves only a bit more than 70% in terms of accuracy.

1.3.5.

Dropout is an effective technique for regularizing our neural networks. The idea behind dropout is to randomly zero some of the elements of the input tensor with probability p. Therefore p can thought as the probability of a Bernoulli distribution where each element is zeroed with probability p and remains unchanged with probability 1-p. In this study, a dropout layer has been added after each max-pooling layer. In the following pages the training loss, validation loss, training accuracy and validation accuracy are reported for different dropout values.
p=0.2 (Figure 4 Figure 5), p=0.5 (Figure 6 Figure 7), p=0.8 (Figure 8 Figure 9)

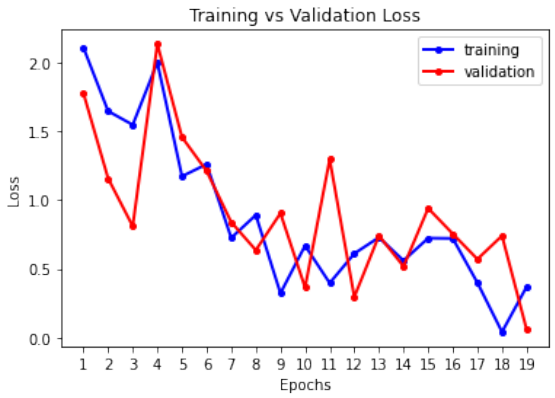


Figure 4: Training and Validation Loss over epochs, dropout=0.2

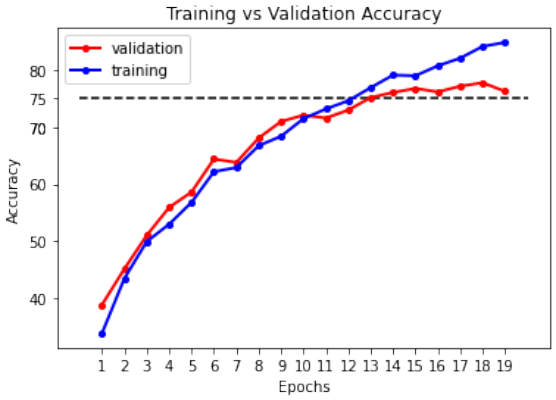


Figure 5: Training and Validation accuracy over epochs, dropout=0.2

With p=0.2 the best validation accuracy was 77.8% and it was achieved at the eighteenth epoch. With dropout p=0.2, the accuracy of the validation set gradually increases over epochs without fluctuating after a certain number of epochs. Additionally, by looking at the graph, the losses with dropout p=0.2 decrease faster than the losses without dropout.

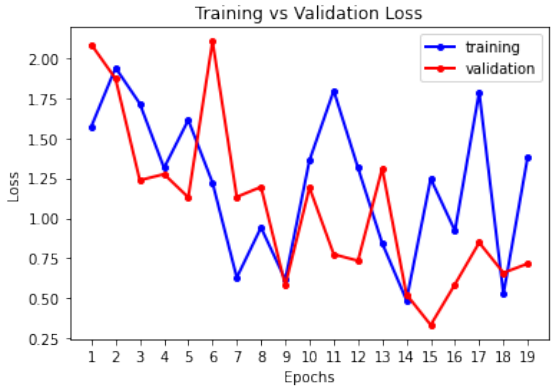


Figure 6: Training and Validation Loss over epochs, dropout=0.5

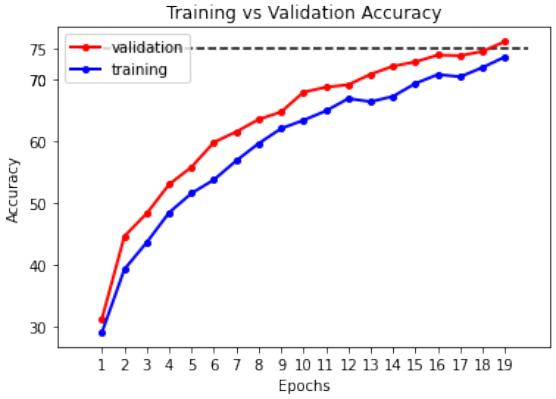


Figure 7: Training and Validation accuracy over epochs, dropout=0.5

Using p=0.5 the best validation accuracy was 76.2% and it was achieved at the nineteenth epoch. However, with p=0.5, the increase in the training accuracy is slower compared to the one without dropout.

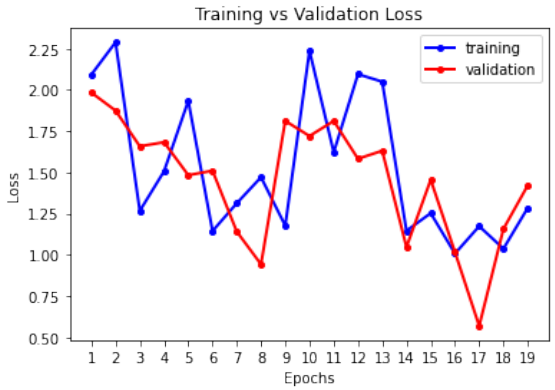


Figure 8: Training and Validation Loss over epochs, dropout=0.8

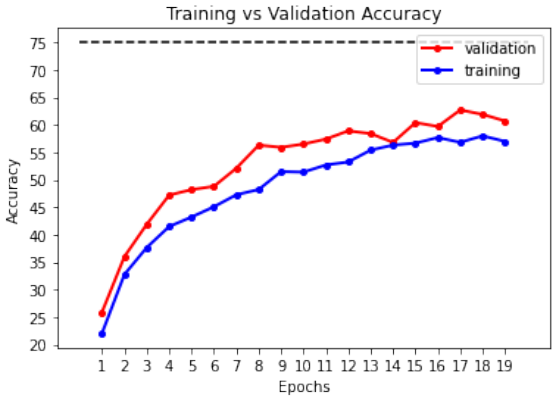
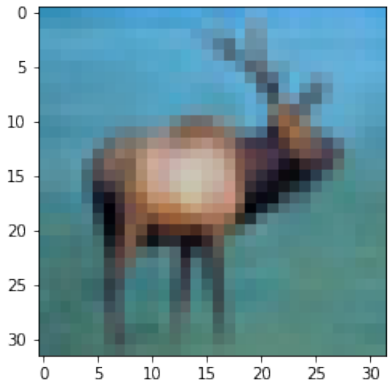


Figure 9: Training and Validation accuracy over epochs, dropout=0.8

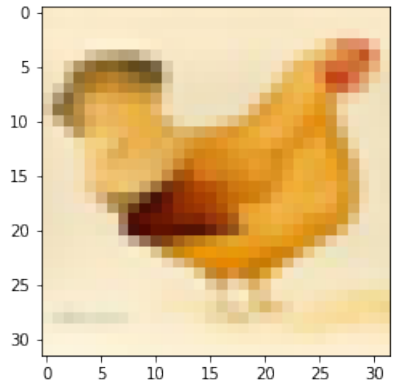
With p=0.8 the best validation accuracy was 62.7% and it was achieved at the seventeenth epoch. In this case the dropout p is clearly too high. It follows that the neural network does not learn well, since the non zero information provided to it is too small.
In order to see whether increasing the number of epochs could lead to a significant improvement in the validation accuracy, I tried setting the epochs to 60. However, the increase in validation accuracy achieved (5%) was not worth the time to run 60 epochs. Hence, a dropout rate p=0.8 is too much is this case.

1.3.6.

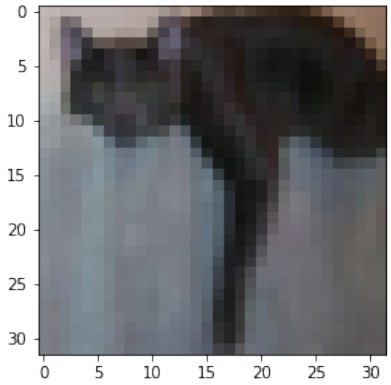
After all the above experimentations, the conclusion is that the best model for the CIFAR-10 image classification is the one with a dropuout p= 0.2.
This model correctly classified the test set images with an accuracy of 74.95%.
Hereafter, in Figure 10 are reported four random images from the validation set with their respective output probability distributions. Moreover, each of the greatest probability has been highlighted. To interpret these probability distributions please note the order of the labels ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck'). Note that the outputs of the model are logit unnormalized log probabilities α_i; to obtain probabilities they have to be transformed in the following way $\mathbf{p_i} = \frac{e^{\alpha_i}}{\sum_{i=1}^{10} e^{\alpha_i}}$



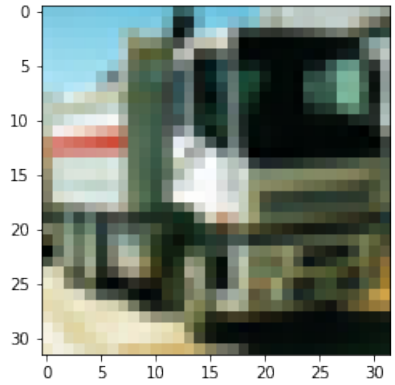
.: prob=[4.3255e-06, 1.1539e-12, 3.7746e-05, 1.1022e-07, **9.9995e-01**, 8.3442e-08, 3.9254e-08, 6.6496e-06, 7.3651e-07, 3.3007e-10]



.: prob=[7.6881e-03, 5.4616e-04, **7.8631e-01**, 7.2537e-02, 2.1252e-03, 7.9929e-02, 5.0146e-02, 3.5176e-04, 2.2084e-04, 1.4221e-04]



.: prob=[4.0985e-02, 8.5301e-06, 9.1121e-02, **5.2815e-01**, 4.4323e-02, 2.1026e-01, 1.2182e-03, 8.3865e-02, 1.7741e-05, 4.9916e-05]



.: prob=[3.6042e-08, 3.8036e-07, 5.8340e-11, 2.2159e-11, 2.2545e-11, 9.9071e-12, 4.9027e-13, 6.7130e-10, 7.7921e-10, **1.0000e+00**]

Figure 10: Four random images from validation set with corresponding probability distributions.

The prediction are fairly reasonable. Furthermore, note that the model finds easy classifying certain classes more than others.

1.3.7.

In order to achieve a test accuracy above 80% the following changes were made.

(i) The number of epochs was set to 25

(ii) The learning rate was set to 0.05

(iii) The dropout p was set to 0.2

(iv) More convolutional and max-pooling layers were added.

The architecture now is:

(a) Convolutional layer 1: 32 filters, 3 × 3, padding(1,1).
(b) Convolutional layer 2: 32 filters, 3 × 3, padding(1,1).
(c) Max-pooling layer 1: 2 × 2 windows, stride(2,2).
(d) Dropout layer 1, p=0.2
(e) Convolutional layer 3: 64 filters, 3 × 3, padding(1,1).
(f) Convolutional layer 4: 64 filters, 3 × 3, padding(1,1).
(g) Max-pooling layer 2: 2 × 2 windows, stride(2,2).
(h) Dropout layer 2, p=0.2
(i) Convolutional layer 5: 128 filters, 3 × 3, padding(1,1).
(j) Convolutional layer 6: 128 filters, 3 × 3, padding(1,1).
(k) Max-pooling layer 3: 2 × 2 windows, stride(2,2).
(l) Dropout layer 3, p=0.2
(m) Convolutional layer 7: 256 filters, 3 × 3, padding(1,1).
(n) Convolutional layer 8: 256 filters, 3 × 3, padding(1,1).
(o) Max-pooling layer 4: 2 × 2 windows, stride(2,2).
(p) Dropout layer 4, p=0.2
(q) Fully connected layer 1: 512 units.
(r) Output layer.

This model achieved a test accuracy of 81.96%.

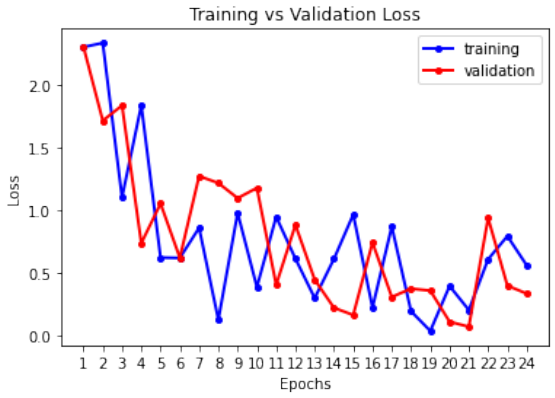


Figure 11: Training and Validation Loss over epochs

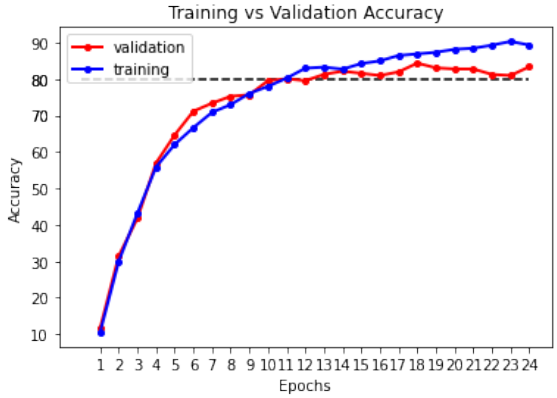


Figure 12: Training and Validation accuracy over epochs

1.4. Questions

1.4.1.

The softmax activation function is used to model the probability that a certain input belongs to a certain label; hence softmax comes in handy in classification tasks.

1.4.2.

The quantity the optimizer accumulates with a decay factor is the velocity $b_t = \mu b_{t-1} + (1 - \tau)g_t$, where μ is the momentum value.

1.4.3.

When it comes to signal processing we may want to use 1D convolution instead of transforming our data to 2D and then applying a 2D convolution.

1.4.4.

Dropout is used during the training phase to regularize our network. Therefore, during the testing phase dropout is deactivated.