CONVOLUTIONAL NEURAL NETWORKS

MILAN KRESOVIĆ – S266915 ERASMUS STUDENT

INTRODUCTION

In this homework, the topic of different types of **convolutional neural networks (CNN)** will be processed and discussed. CNN is the class of **deep neural networks** (neural networks with more than three layers, including input and output layers). The CNNs contain **convolutional layers** – a layer that compute a dot product between two matrices, where one is the set of learnable parameters, also known as the kernel, and the other is the restricted portion of the receptive field.

For purpose of this homework, the training and testing was done on the CIFAR 100 dataset. This dataset contains 100 classes of RGB images with a resolution of 32x32 and is already included in the library. The first part is training **fully connected neural network (FCNN)**, that is, network in which each neuron is connected to every neuron in the previous layer.

The second part is training of our CNN with different number of convolutional filters, with or without batch normalization (normalizing the inputs of each layer), but also trying wider fully connected layer or/and dropout on the fully connected layer. Beside this, different schemes of data augmentation (random horizontal flipping and random crop) were also tried.

At the end, state-of-the-art pre-trained model called **ResNet18** was trained with the best data augmentation to compare the results with our CNN.

The output of each running were two plots. The first is **training loss** - summation of the errors made for each example and the second is **accuracy** – percentage of true prediction in the testing set.

FULLY CONNECTED NEURAL NETWORK

In this homework, FCNN was made as the class that had two methods: one for initialization - where input, hidden and output layers were defined, and one method for forward pass, whereas backpropagation pass was implemented with Pytorch. FCNN had two hidden layers and one fully connected (FC) layer with each having 4096 neurons.

Batch size was set to 256. This parameter defines the size of the portions in which training samples will be divided and propagated through the network in one epoch. There were 20 epochs, with image resolution of 32x32 and learning rate 0.0001.

Loss behavior and accuracy are the following:

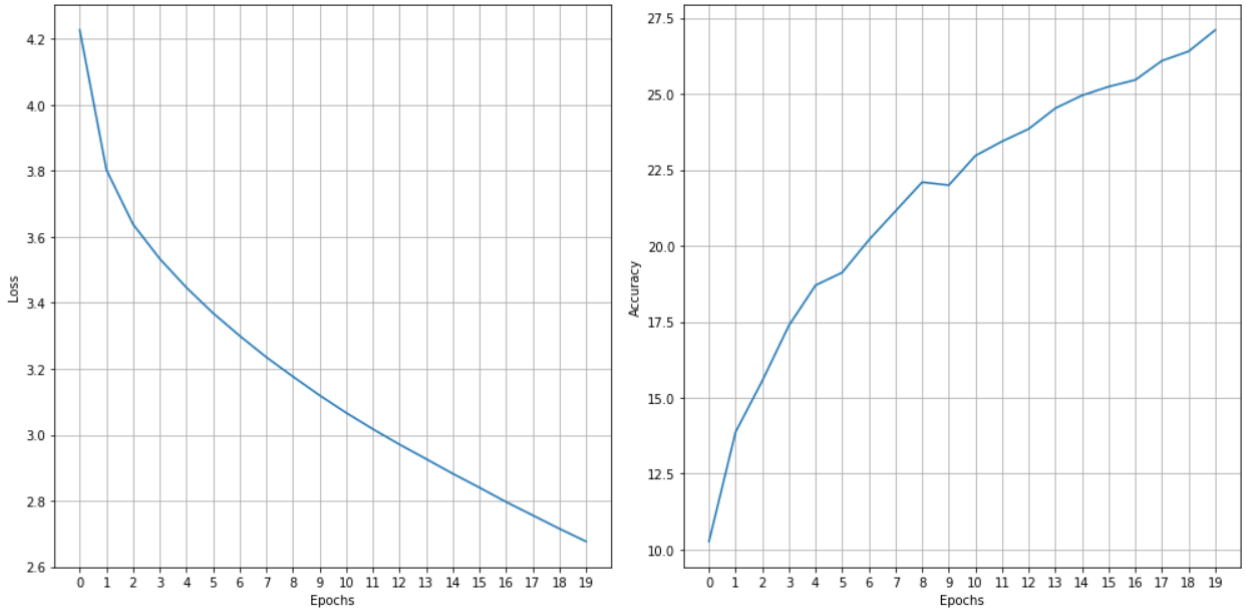

Figure 1. Loss behavior and accuracy over the epochs (FCNN)

As it can be seen on this graphs, loss is decreasing but not in the fast manner. Its drop wouldn't result in better accuracy if more epochs would've been used, only in overfitting the data. On the other hand, the accuracy is really low and doesn't seem as improving much. It will be shown that using old NN like this can't reciprocate the same results as CNN.

CONVOLUTIONAL NEURAL NETWORK

As it has been said, CNN can be differentiated from the standard NN, because they contain convolutional layer. This layer performs convolution, an linear operation that involves the multiplication of a set of weights (called a filter or a kernel) with the input. The result of usage of the filters is the detection of a specific type of feature in the input image. As filter is applied throughout the whole input, output is called **feature map**.

SIMPLE CNN

In the second part of the homework, the simple CNN was used. That is, a network composed of multiple convolutional layers, one pooling layer and one fully connected layer.

Pooling layers are used in CNN for reduction of the dimensionality. There are several non-linear functions to implement pooling, but in this homework max pooling was used that takes only the maximum value. For the activation fuction it was used ReLU instead of sigmoid that was used in the first part of the homework.

Graphs of loss and accuracy for model with number of filters in each layer being 32/32/32/64 are following:

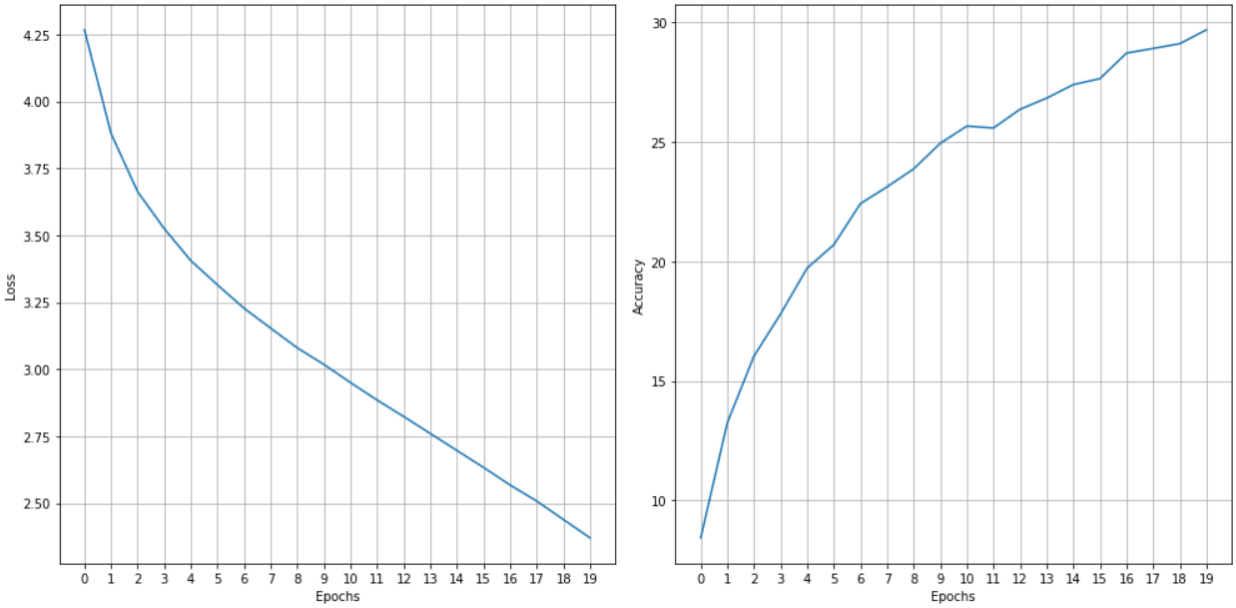

Figure 2. Loss behavior and accuracy over the epochs (CNN 32/32/32/64)

If we compare these results with results given by FCNN it can be noticed that the results haven't improved that much. Loss behavior is still not approaching 0 fast enough, and accuracy is only better by few percent. To improve results more kernels can be used.

Loss and accuracy over the epochs for CNN with kernel number 128/128/128/256:

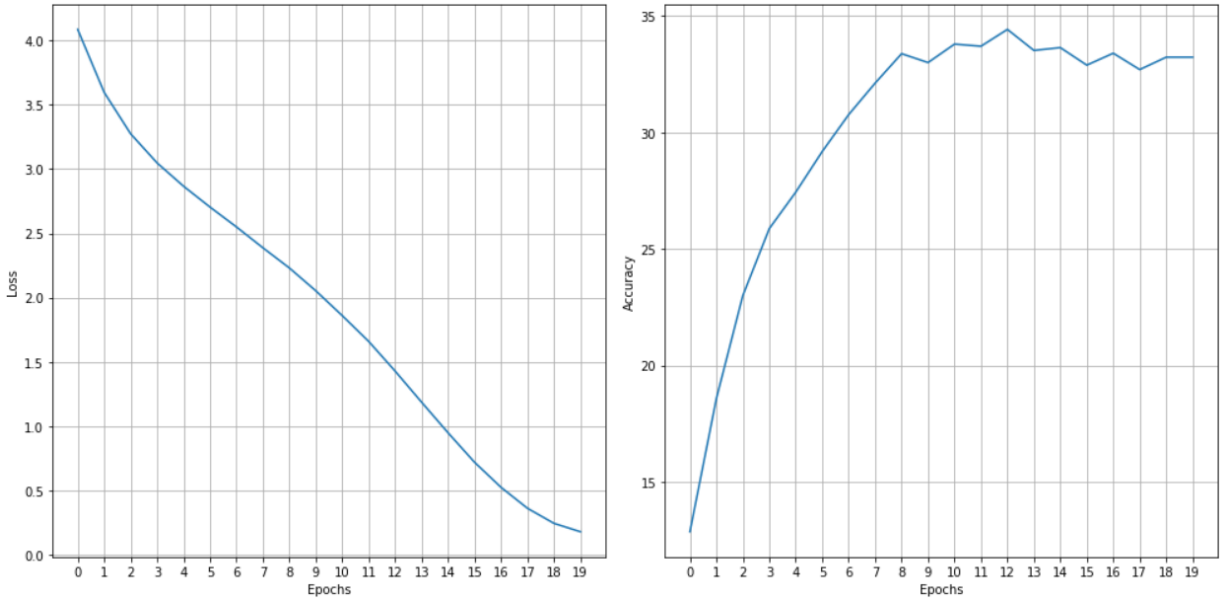

Figure 3. Loss behavior and accuracy over the epochs (CNN 128/128/128/256)

In figure 3, it can be seen that for the first time our model is converging, that is, loss is approaching 0. Accuracy is also up by few percent and it starts to oscillate around 34%. This can indicate possible overfitting in our model. The best accuracy is reached in 13th epoch. It's also important to notice that time needed for training is prolonged.

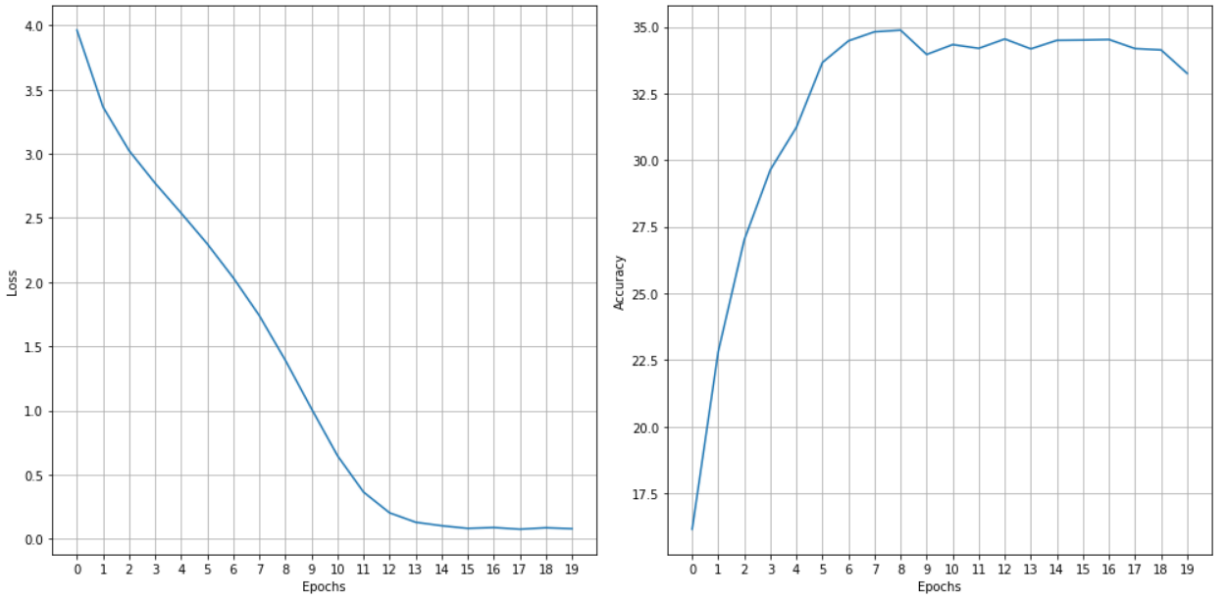

Figure 4. Loss behavior and accuracy over the epochs (CNN 256/256/256/512)

Figure 4 shows us the behavior of model with number of kernels 256/256/256/512. This CNN performs much worse time-training wise. On the other hand, this CNN reaches its peak accuracy value much faster, and its loss behavior is faster in approaching zero. Though, accuracy isn't changed as much. As in the previous model, signs of the overfitting can be seen.

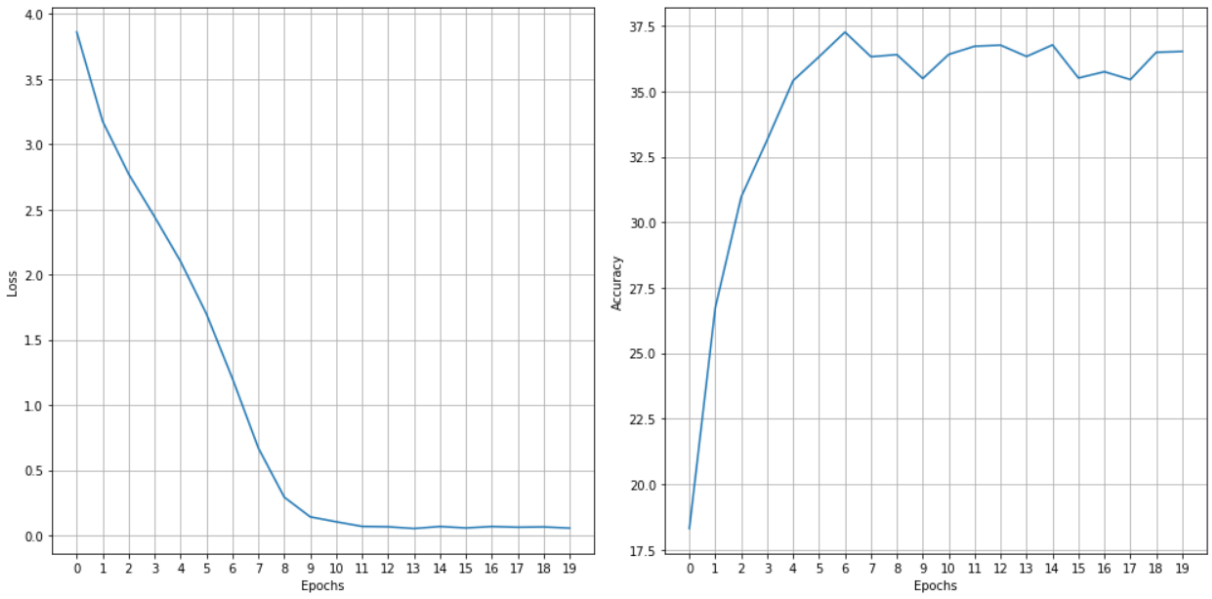

Figure 5. Loss behavior and accuracy over the epochs (CNN 512/512/512/1024)

CNN with number of kernels in each layer 512/512/512/1024 has the slowest training time of all the previous CNN. The accuracy is approved, but when the training time is taken into the consideration it can be seen that the accuracy gained is not of great significance. As in the previous CNNs, the overfitting can be observed.

The conclusion for this part is that using more filters results in a more powerful and therefore more time consuming model, but also we risk overfitting due to increased parameter count. When we over fit, we are actually fitting the noise on the training set. To get rid of this, some approaches like batch normalization, dropout and data augmentation can help.

CNN WITH BATCH NORMALIZATION AND DROPOUT

In CNN its common to normalize all the data so that it resembles a normal distribution. This can be done to prevent the early saturation of non-linear activation functions but also so that all input data is in the same range of values. The problem that arises is that if we only normalize input data, in the intermediate layers the distribution of the activations is constantly changing. This slows the training because every iteration layers have to adapt themselves to a new distribution. The name for this problem is **internal covariate shift**.

To solve this, **batch normalization** is usually applied. It's a method that is used to normalize the inputs of each layer. If we use batch normalization onto our CNN model with kernel number 128/128/128/256, the results are following:

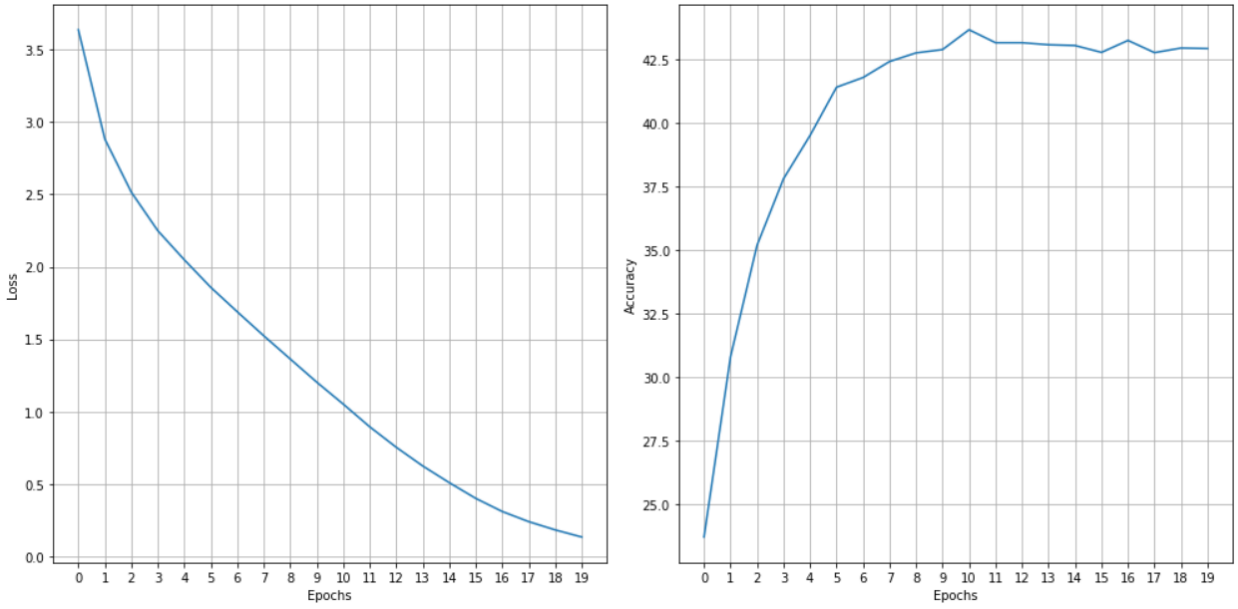

Figure 6. Loss behavior and accuracy over the epochs (CNN with Batch normalization)

In the figure 6, it can be seen that compared to the results from figure 3, accuracy is much higher and the overfitting of the data is not so pronounced.

In the next example, batch normalization with the wider first fully connected layer will be applied to the same CNN:

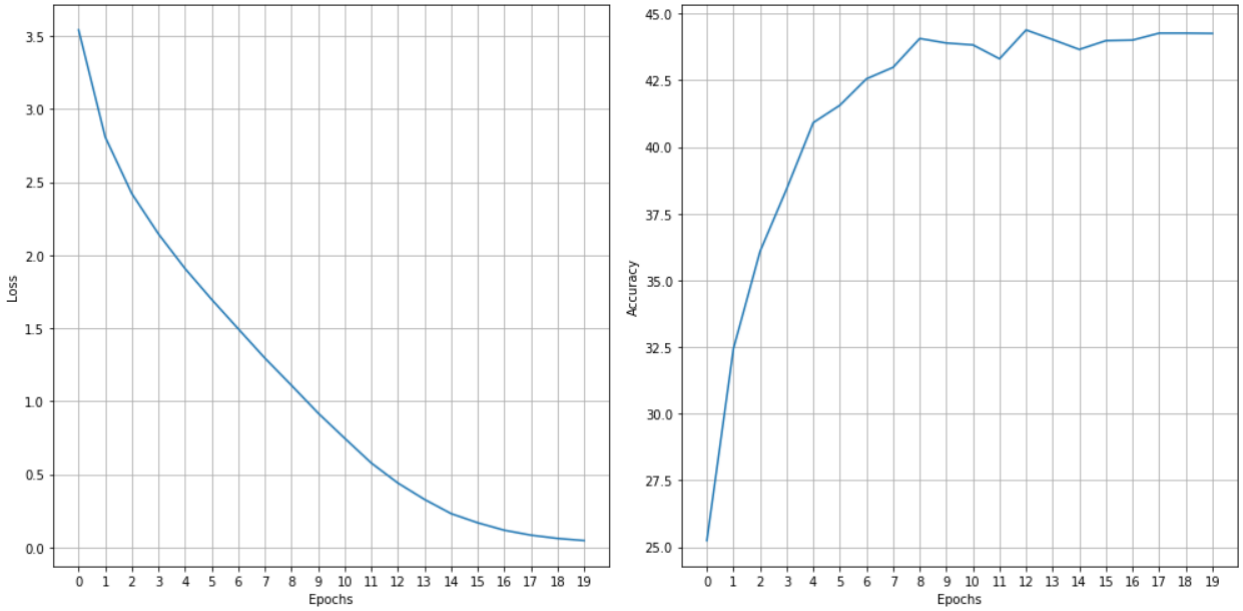

Figure 7. Loss behavior and accuracy over the epochs (CNN with Batch normalization + FC1 wider)

In this case, the first fully connected layer was widened. That is, the number of neurons was increased to 8192. The accuracy is higher, but not by a big margin.

The last model in this part is with a dropout in the first fully connected layer.

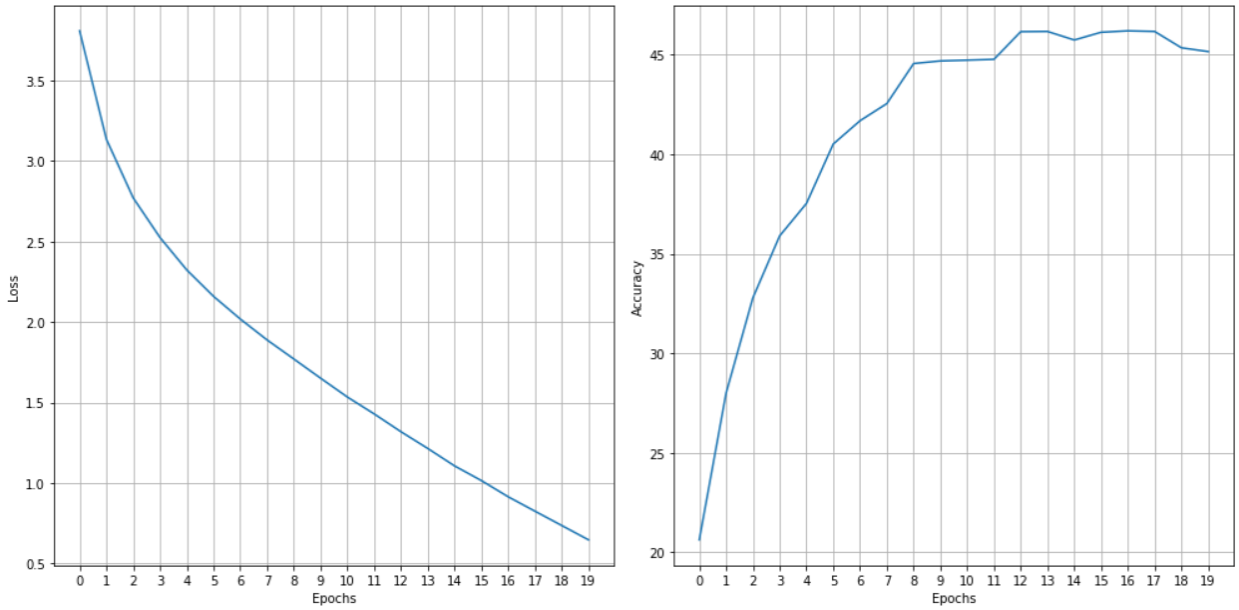

Figure 8. Loss behavior and accuracy over the epochs (CNN with Batch normalization + Dropout 0.5)

Dropout is used to prevent overfitting. During training time at each iteration, one random neuron is temporarily disabled with probability p. This method works because it prevents the network to be dependent on small number of neurons and forces network to adapt and operate independently. Dropout is only applied on the training set, so that we get even better result on the testing set.

Not surprisingly, model with batch normalization and dropout had the best performance with the respect to the accuracy. On the other hand, it didn't increase accuracy that much.

CNN WITH DATA AUGUMENTATION

In order to maximize usefulness of our training set **data augmentation** can be applied. This technique augments the training data by generating new examples via random transformation of the training test. These transformations can be: rotation, shifting, resizing, cropping etc. In this homework only model (CNN 128/128/128/256) was exposed to random horizontal flipping and random crop.

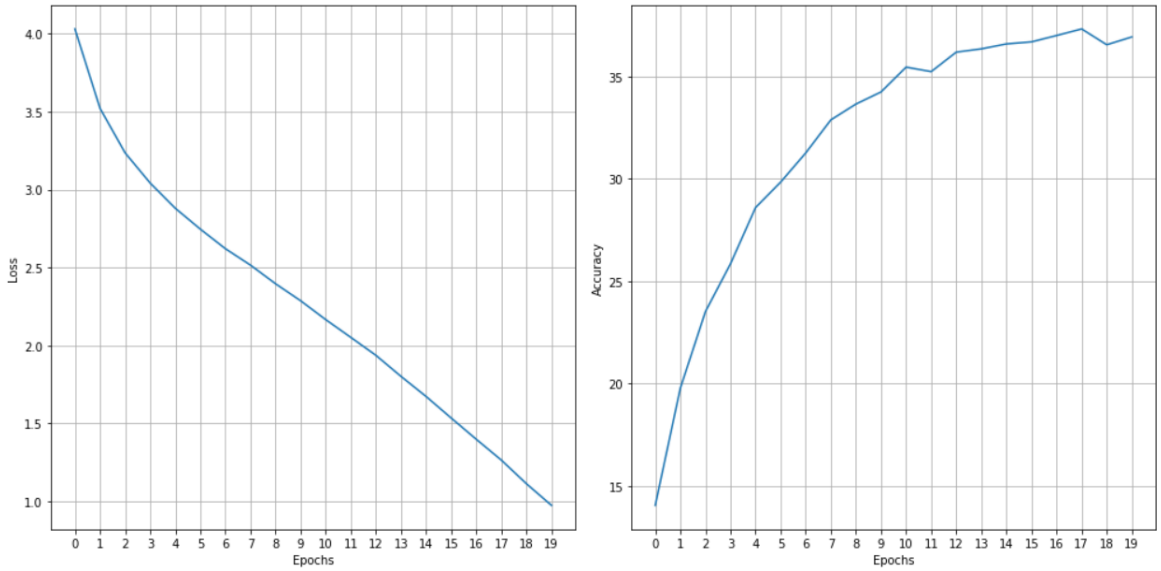

Figure 10. Loss behavior and accuracy over the epochs (Random horizontal flipping)

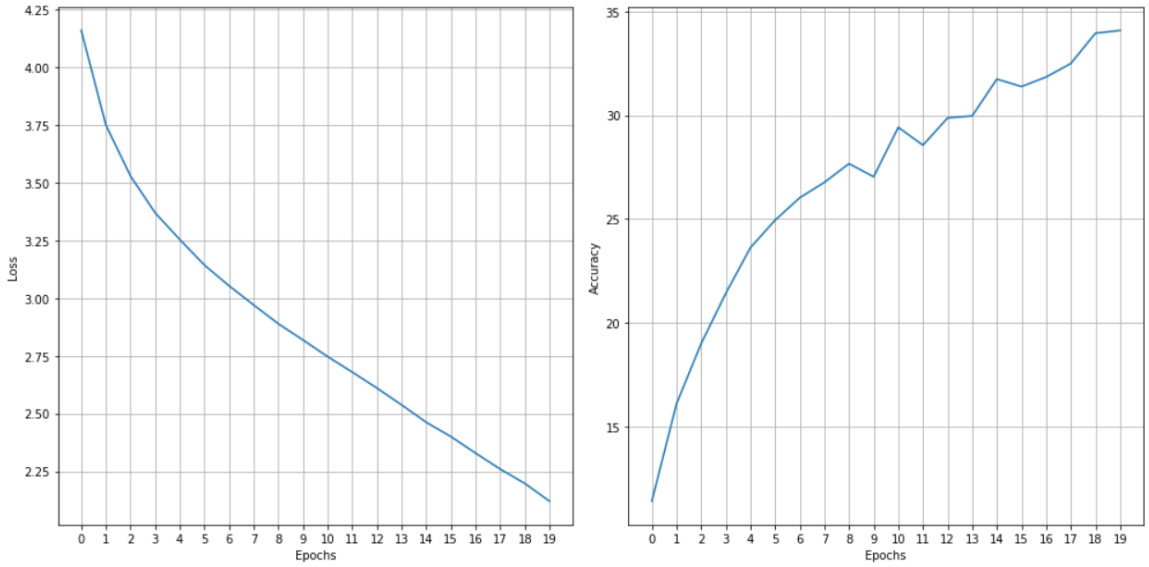

Figure 9. Loss behavior and accuracy over the epochs (Random crop)

As it can be seen on the figures 9 and 10, the accuracy has increased more with random horizontal flipping. The idea behind random horizontal flipping is that an object should be equally recognizable as its mirror image, so maybe this way model overfits less. That's why this method will be used in the last part of this homework. This way our state-of-the-art CNN will perform better.

RESNET18

In the last part of this homework pretrained state-of-the-art network that was used is called ResNet18. This network was trained on ImageNet dataset and by using it we are utilizing something called **transfer learning**. This method is usually used because having very large dataset is really rare, so using already made networks with good performance is strongly recommended. These kind of networks are either used as an initialization or a fixed feature extractor for a specific task. By training pretrained networks with our data we are doing the **fine-tuning**.

Parameters used for this part of the homework are the following: batch size of 128, 10 epochs, 224x224 resolution and Adam solver with learning rate of 0.0001. Additionally, data augmentation method was applied, specifically: random horizontal flipping.

The results of this network:

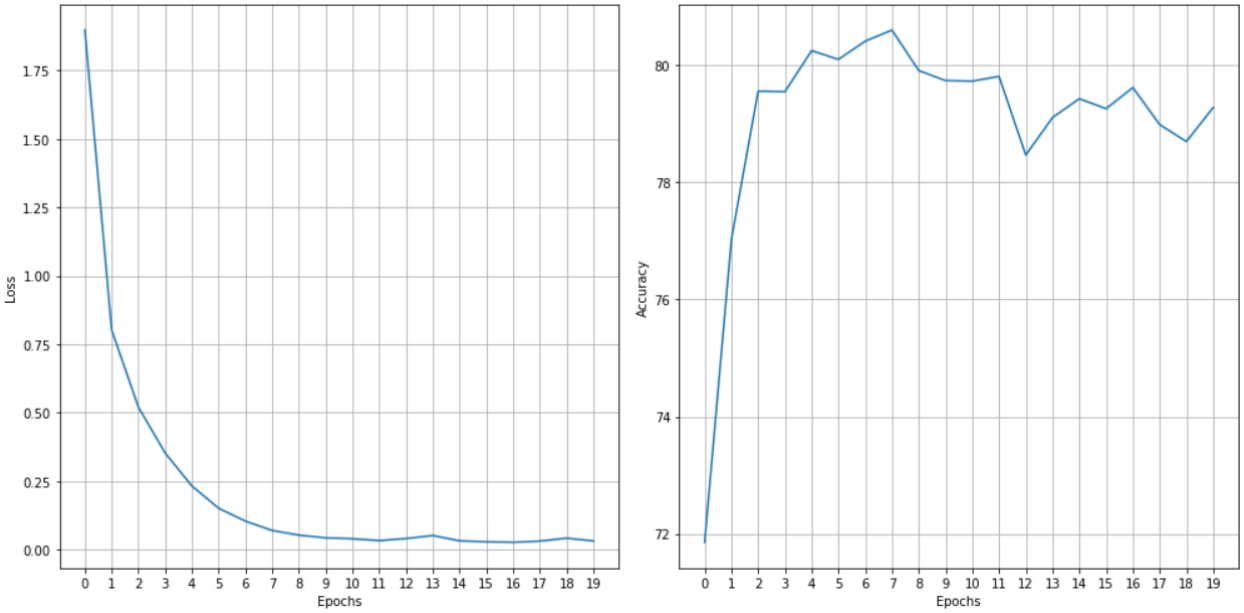

Figure 11. Loss behavior and accuracy over the epochs (ResNet18 with random horizontal flipping)

The network's loss value is converging to 0 after only 9-10 epochs, and it's giving extraordinary results with respect to accuracy compared with all the previous networks. The accuracy is around 79% with the oscillations around just one percent.

The conclusion is that for ordinary problems it is always better to train our dataset onto already made and pretrained networks instead of making our own from the scratch, because the results are usually better.