Deep Learning Assignment 2

Hamza Görgülü Department of Computer Science and Engineering
Koç University
İstanbul
hgorgulu22@ku.edu.tr

Abstract

This report is prepared for COMP541 Deep Learning assignment.

1 Convolutional Neural Networks

1.1 Convolutional Filter Receptive Field

Look to the table for a visual answer. The formulation for the output size with a feature map size of N, and filter size of F is as follows: $1 + (N - F)/S$. So, the first convolutional layer with a filter size of 5x5 will produce Z1 as $1 + (A - 5) = A - 4$. Then Z_1 with filter size 3x3 will create the next feature map as $1 + (A - 4 - 3) = A - 6$. Now we should find the receptive field of a single node in Z_2 over the first convolutional layer. As we go deeper, we will extract more information with a smaller size. The formulation for a receptive field with L layers and KxK filters is as follows: $1 + L * (K - 1)$. Since the filter sizes are different, I will calculate the receptive field one by one. Now, we could first calculate the receptive field on Z_1 for a single node on the Z_2 feature map as follows: $1 + 1 * (3 - 1) = 3$. Then, I applied the same formula to a node in the corner of the Z_1 feature map. Its receptive field is as follows: $1 + 1 * (5 - 1)$. This will add extra 2 sizes to the previous receptive field that we found, and the receptive field on the first convolutional layer will become 7x7 size. The visual explanation is illustrated in Fig. 1

1.2 Run the Pytorch ConvNet

- There are 2 convolutional layers, 2 pooling layers, and 2 linear layers.
- ReLU is used on the hidden nodes.
- CrossEntropyLoss is used to train the neural network.
- The Adam optimizer is used to minimize the loss. We put the model parameters and the learning rate into the optimizer, and it updates the parameters based on the calculated loss.

What is the training accuracy for your network after training? What is the validation accuracy? What do these two numbers tell you about what your network is doing?
-After 50 epochs, the training accuracy was 0.6583, and the validation accuracy was 0.5274. It seems that the loss is decreasing, and the accuracy is increasing for both the train and the validation set over the epochs. In the first 10 epochs, the accuracy for both is increasing dramatically, and I know this is a good sign. The accuracy and the loss graphs for both validation and accuracy are stated in Figure 2.

1.3 Add Pooling Layers

After you applied max pooling, what happened to your results? How did the training accuracy vs. validation accuracy change? What does that tell you about the effect of max pooling on your network?

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

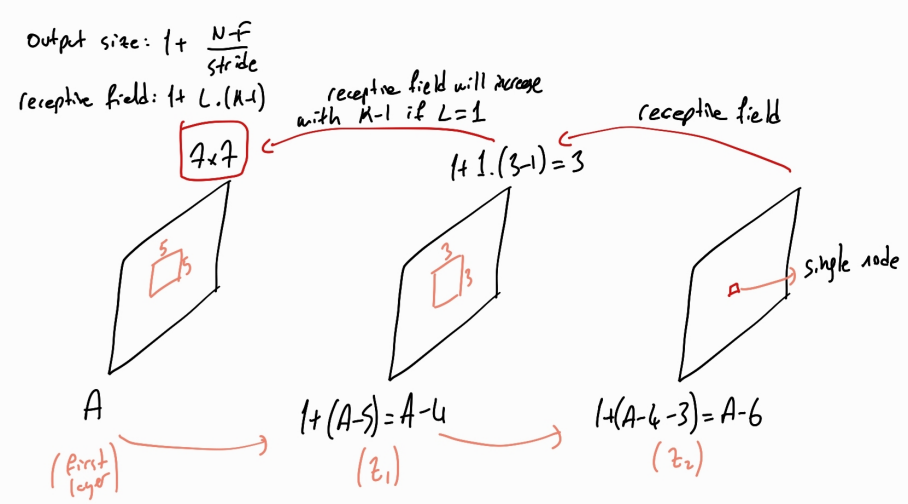

Figure 1: Visual solution for section 1.1

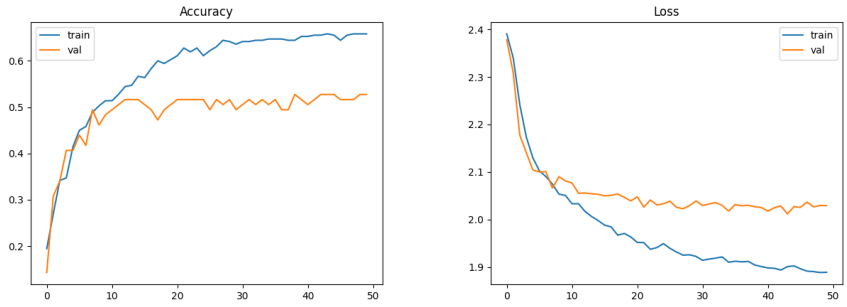

Figure 2: Accuracy and loss graphs for training and validation set in section 1.2

After adding two pooling layers with kernel size 2x2 and stride 2, the accuracies for both training and validation remained the same. Normally, we expect the pooling layers to have a regularization effect on the performances; however, I think using seed for Torch and NumPy caused this similarity.

1.4 Regularize Your Network!

Dropout: Firstly, I applied a dropout after the first ReLU with a 0.5 probability. However, it decreased the performance significantly. Then I implemented additional dropouts after other ReLU's with a different number of probabilities. The optimal condition is satisfied with 0.2 probabilities after all ReLU's with 0.67 training accuracy and 0.62 validation accuracy. It is clear that dropout gave rise to the performance of our model and decreased the overfitting that we observed in the previous sections. The gap between the training accuracy was about 13% in the previous section, and it became 5% in this section with the help of dropouts I implemented. The graphs for accuracy and loss after implemented dropout method is stated in Fig. 3

Lr scheduler: I implemented ReduceLROnPlateau, STepLR, MultiStepLR, and CossineAnnealingLR as the learning rate scheduler. They did not have a significant performance difference. However, ReduceLROnPlateau performed better with a tiny gap. I am aware that there may be better solutions since there are many conditions and I am not able to implement all of them because of the time limitation.
I used ReduceLROnPlateau as a learning rate scheduler with a different number of parameters. In the optimal scenario, I first increased my existing learning rate from 1e-4 to 5e-4, in order to speed up the learning in the first epochs. Then I set up the patience as 20 and the factor as 0.8. Thus, the learning rate would decrease with a coefficient of 0.8 in every 20 iterations that there was no progress.

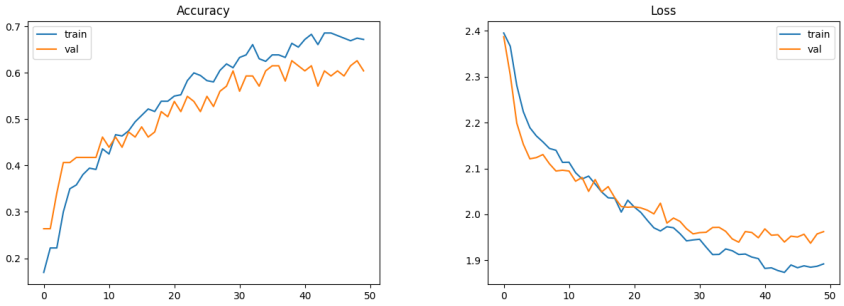

Figure 3: Accuracy and loss graphs after using dropout in section 1.4

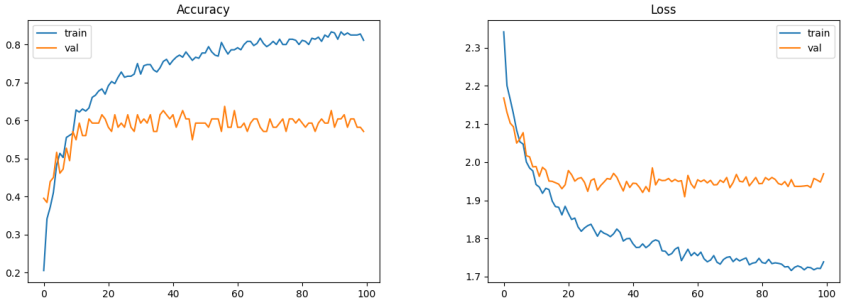

Figure 4: Accuracy and loss graphs after using ReduceLROnPlateau learning rate scheduler in section 1.4

I was able to receive 0.64 accuracy for validation, and 0.79 accuracy for the training set. There was a 2% increase in the validation accuracy and a 12% increase in the training accuracy. It seems that overfitting is an issue again. So, I will apply weight regularization next. The figures for accuracy and loss is showed in Fig. 4

Optimizer and weight decay: I applied Adam, SGD, RMSprop, Adadelta, and Adagrad optimizers in order to optimizer the performance. Firstly, I should state that Adadelta has performed the worst, and SGD and Adagrad performed badly when compared to the following optimizers. Adam and RMSprop's performance was similar, and they were the best. I chose the Adam as an optimizer and applied a weight decay, and found the value of 1e-5 as the most optimal value for it. The accuracy for validation was 0.65 and 0.80 for the training. The accuracy and the loss curves are represented in Fig. 5

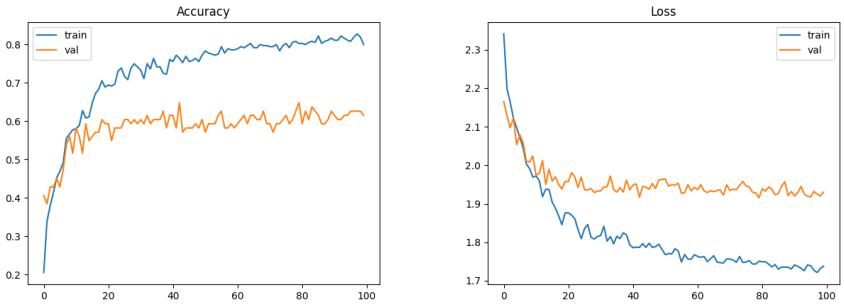

Figure 5: Accuracy and loss graphs after using Adam optimizer with 1e-5 weight decay in section 1.4

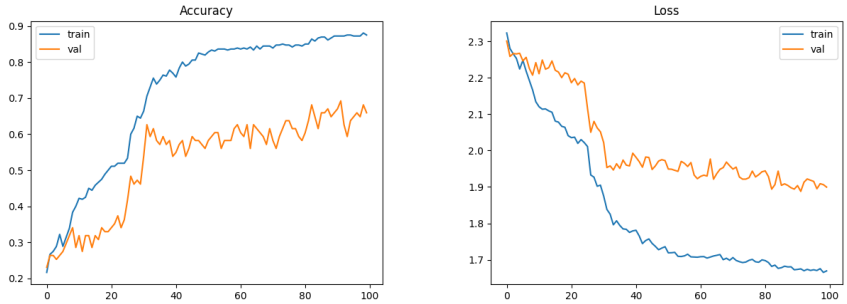

Figure 6: Accuracy and loss graphs after playing with kernel size and stride in section 1.5

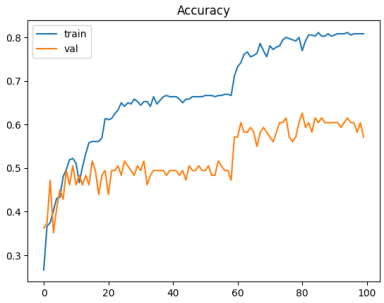

Figure 7: A spike in accuracy after a continuous smooth line

I have also implemented early stopping. However, since I provided the best results I got, it did not affect the results by just finishing the training earlier. I tried to stop it just after the part where I got the best result.

Overall, I observed that the most effective regularization was the dropout method. It has drastically increased the validation accuracy from 52% to 62%. Therefore, using ReduceLROnPlateau as a learning rate scheduler was an impact of 2% on the validation performance. Lastly, using the Adam optimizer with a weight decay increased the performance by 1%.

1.5 Experiment with Your Architecture

Using larger kernel size and stride are both a tradeoff. A large kernel size and a small stride gave me the best result; however, it took much more time and computation. For example, with a 7x7 kernel size and 1 stride, I was able to get 0.69 accuracy over the validation, and 0.87 accuracy over the training set. Even though the results were better, I decided to go with the previous hyperparameters since 7x7 kernel and 1 stride takes a very long time to train. The accuracy and loss graphs are illustrated in Fig. 6

During the trials, I realized a sudden increase in accuracy in the middle of a random trial. When I saw it, I remembered the speech from Andrew Ng that is about early stopping and orthogonalization. That is why I wanted to include it in Fig. 7

An additional convolutional layer with a pooling layer also decreases the size of the input of the linear layer at the end. However, adding an extra layer reduces the accuracy of the validation set in this model. However, I could not achieve better performance by adding another layer.
In convolutional neural networks, the height and width of the model decrease and the depth of the model increases as you go to the fully connected layer. But what causes this to happen? The answer is the filters that are applied to the convolutional layers. As you apply many filters to the input image, the height and width will decrease thanks to the filter size, and the depth will increase thanks to the number of filters. As you have more layers, the model will be deeper. Of course, the role of the pooling layer is also important in reducing the dimensions of the given channels.

Table 1: Accuracy values over the sections

Section Name	Training acc.	Validation acc.
Section 1.2	0.65	0.52
Section 1.3	0.65	0.52
Section 1.4(dropout)	0.67	0.62
Section 1.4(lr scheduler)	0.79	0.64
Section 1.4(weight decay)	0.80	0.65
Section 1.4(weight decay)	0.80	0.65
Section 1.5(not selected)	0.87	0.69
Section 1.5.1	0.83	0.66
Section 1.6	0.83	0.66

1.5.1 Optimize Your Architecture

I implemented Local Response Normalization, Batch Normalization, and Weight Normalization. However, they did not impact the performance as I expected. I was able to receive a slight increase in validation accuracy. I tried to use data augmentation techniques such as Grayscale, ColorJitter, RandomCrop, RandomRotation etc; however, they were always decreasing the validation performance. For this reason, I did not use it. The best performance I was able to achieve in this section was 66% validation accuracy. In the previous section, I got a 69% validation performance, but I decided not to use it because of computation limitations.

1.6 Test Your Final Architecture on Variations of the Data

I used data augmentation in order to test the model's overall performance. The following sentences represent my observations:
-The model is translation invariant. I applied RandomAffine only on the validation set with (0.1,0.1) translate value and did not see a difference in the performance.
-I used ColorJitter in order to see the effect of brightness, contrast, and hues. The brightness with 0.5 value has decreased the model's performance drastically to 0.35 accuracy. The contrast with the 0.5 value has also caused a reduction in the performance with 0.35 accuracy. The changes in saturation and hue showed similar performances. To sum it up, the model is not invariant to the changes in ColorJitter features such as brightness, contrast etc.
-Converting images with the Grayscale method in validation did not impact the performance. The model is invariant to Grayscale. However, inverting with RandomInvert reduced the performance.

2 Transfer Learning with Deep Network

2.1 Train a Multilayer Perceptron

First, I have been using batch size 12 and never got a result better than 42% accuracy on the test set. Then I changed the batch size to 32, and I am now able to get better than 55% accuracy. I believe batch size is a very important hyperparameter.
I tried different learning rates with a sort of frequency between [1e-2] and [1e-5]. The best one I observed was [1e-3] with the Adam optimizer. However, the loss graph was giving a signal of a high learning rate even though I got the best result with a learning rate of [1e-3]. Then I decided to use a learning rate scheduler with 6 patients and a factor of 0.1. This led to a tiny increase in test performance.
Moreover, I implemented a dropout layer with 0.5 probability in order to prevent this overfitting I observed. This helped the model to decrease the training accuracy and increase the validation accuracy. The test performance was an accuracy of 0.67.
Therefore, I tried the most known optimizers, which are Adam, SGD, SGD with momentum, Nesterov, Adagrad, and RMSprop, to increase the accuracy. The Adam optimizer was the best among them in terms of accuracy and loss. The visualization of the accuracy and the loss for both training and validation is stated in Fig. 8 Lastly, the test performance was an accuracy of 0.67.

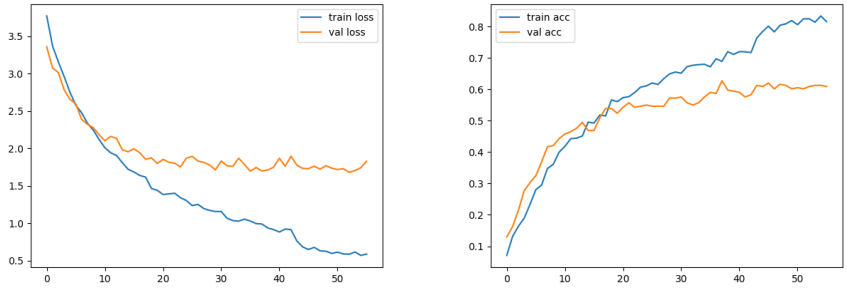

Figure 8: Accuracy and loss graphs for training and validation set in section 2.1

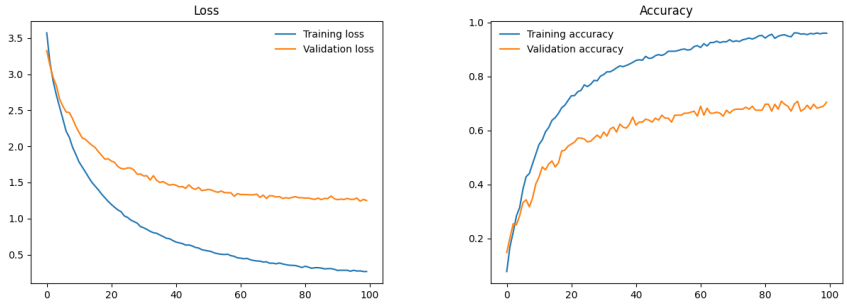

Figure 9: Accuracy and loss graphs for training and validation set in section 2.2

2.2 AlexNet as a Fixed Feature Extractor

Since I used a pre-trained model, I had to preprocess the data in the same way the model does for its training. So here, I preprocessed the input images with 224 crops and normalize them with the mean and standard deviation of the ImageNet dataset to get the input tensor. Then I created the input batch by using unsqueeze(0) method and tested if the model was working the way I expected.
Then, I created a FeatureExtractor class that takes the 4th conv layer and its ReLU activation from AlexNet architecture. Then I defined the Classifier class that flattens the upcoming input from FeatureExtractor and pushes it into a Linear layer. Since we have a total number of 40 actors and actresses the output size of this linear layer is 40.
Therefore, I concatenated these two classes in an another class. To sum it up, I used the 4th conv layer of AlexNet as feature extractor and appended a fully connected layer at the end of it. The model is ready to optimize.

The model had first a severe overfitting problem. Then I insert a dropout after the activation of convolutional layer. Therefore, I also added a pooling layer in order to reduce the computation time by reducing the number of nodes that Classifier takes as input. With these changes, I was able to increase the test accuracy from 53% to 72%. Therefore, the model performed 70% on the test set. However, the model overfits even though I use dropout and weight decay. The loss and accuracy graphs of this model is illustrated in Fig. 9

After optimizing the model architecture, I state the model parameters as follows: CrossEntropyLoss as loss function, Adam as optimizer, 5e-5 as learning rate, and 32 as batch size.

2.3 Visualize Weights

2.4 Finetuning AlexNet