Deep Learning Lab

Kaan Egemen Şen

Assignment 2

Year 2022-2023

1 Dataset

1.1 Exercise

Loaded Cifar-10 dataset and imshow 4 image from dataset can be visualize below 1 :

Figure 1: 4 image from database

1.2 Exercise

Normalized and transform images with given means and std values can be found in code

1.3 Exercise (Bonus)

See section under Exercise 3 Bonus markdown in the code submission calculated values are : 2

```
Mean of Red :0.49139968
 Mean of Green :0.48215827
 Mean of Blue :0.44653124
Std of Red :0.24703233
 Std of Green :0.24348505
 Std of Blue :0.26158768
```

Figure 2: Mean and std values of each color channel

We can verify that given mean/std values are correct

1.4 Exercise

With using SubsetRandomSampler we split our data into 2 part training and validation train and validation loader can be found also code submission under Exercise 4 Markdown split result is in Figure 3 :

```
Validation size :  1000
Training size :  49000
```

Figure 3: Splitting images into train and validation

2 Model

2.1 Exercise

Created first convolutional network with given layers in code 4 :

3 Training

3.1 Exercise

Implemented the training pipeline can be found in code as Section 3.1 :

3.2 Exercise

We use the hyperparameters as given in assignment paper can be found in code.

```
CNNModel(
  (conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1))
  (conv2): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
  (pool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
  (conv4): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1))
  (pool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc): Linear(in_features=1600, out_features=512, bias=True)
  (out): Linear(in_features=512, out_features=10, bias=True)
  (activation): ReLU(inplace=True)
)
```

Figure 4: Architecture of Model

3.3 Exercise

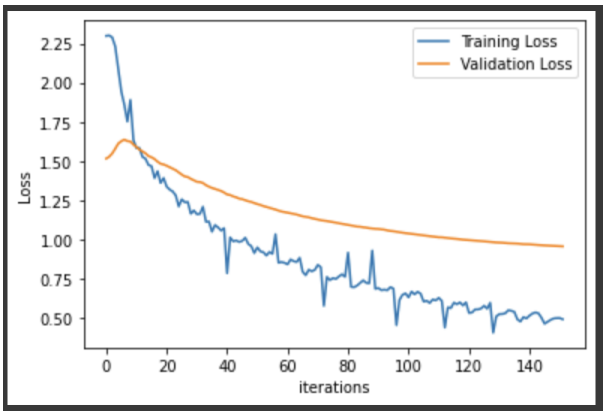
Trained our model and received validation accuracy as given below 9 :

```
epoch: 15, steps: 200, train_loss: 0.565, running_acc: 80.8 %
epoch: 15, steps: 400, train_loss: 0.561, running_acc: 80.7 %
epoch: 15, steps: 600, train_loss: 0.595, running_acc: 79.6 %
epoch: 15, steps: 800, train_loss: 0.586, running_acc: 80.0 %
epoch: 15, steps: 1000, train_loss: 0.597, running_acc: 79.5 %
epoch: 15, steps: 1200, train_loss: 0.578, running_acc: 79.9 %
epoch: 15, steps: 1400, train_loss: 0.598, running_acc: 78.9 %
Validation accuracy:  73.6 %
epoch: 16, steps: 0, train_loss: 0.530, running_acc: 78.1 %
epoch: 16, steps: 200, train_loss: 0.533, running_acc: 81.6 %
epoch: 16, steps: 400, train_loss: 0.552, running_acc: 80.6 %
epoch: 16, steps: 600, train_loss: 0.553, running_acc: 81.0 %
epoch: 16, steps: 800, train_loss: 0.559, running_acc: 80.2 %
epoch: 16, steps: 1000, train_loss: 0.576, running_acc: 80.0 %
epoch: 16, steps: 1200, train_loss: 0.557, running_acc: 81.0 %
epoch: 16, steps: 1400, train_loss: 0.595, running_acc: 79.4 %
Validation accuracy:  72.6 %
epoch: 17, steps: 0, train_loss: 0.404, running_acc: 81.2 %
epoch: 17, steps: 200, train_loss: 0.507, running_acc: 82.8 %
epoch: 17, steps: 400, train_loss: 0.523, running_acc: 81.6 %
epoch: 17, steps: 600, train_loss: 0.523, running_acc: 82.0 %
epoch: 17, steps: 800, train_loss: 0.528, running_acc: 81.8 %
epoch: 17, steps: 1000, train_loss: 0.547, running_acc: 81.5 %
epoch: 17, steps: 1200, train_loss: 0.544, running_acc: 81.4 %
epoch: 17, steps: 1400, train_loss: 0.537, running_acc: 81.4 %
Validation accuracy:  72.6 %
epoch: 18, steps: 0, train_loss: 0.489, running_acc: 81.2 %
epoch: 18, steps: 200, train_loss: 0.475, running_acc: 83.2 %
epoch: 18, steps: 400, train_loss: 0.504, running_acc: 82.8 %
epoch: 18, steps: 600, train_loss: 0.495, running_acc: 82.7 %
epoch: 18, steps: 800, train_loss: 0.513, running_acc: 82.5 %
epoch: 18, steps: 1000, train_loss: 0.527, running_acc: 81.9 %
epoch: 18, steps: 1200, train_loss: 0.533, running_acc: 81.6 %
epoch: 18, steps: 1400, train_loss: 0.528, running_acc: 81.8 %
Validation accuracy:  74.1 %
epoch: 19, steps: 0, train_loss: 0.498, running_acc: 71.9 %
epoch: 19, steps: 200, train_loss: 0.461, running_acc: 83.4 %
epoch: 19, steps: 400, train_loss: 0.474, running_acc: 83.3 %
epoch: 19, steps: 600, train_loss: 0.486, running_acc: 82.6 %
epoch: 19, steps: 800, train_loss: 0.495, running_acc: 82.8 %
epoch: 19, steps: 1000, train_loss: 0.497, running_acc: 82.6 %
epoch: 19, steps: 1200, train_loss: 0.498, running_acc: 82.4 %
epoch: 19, steps: 1400, train_loss: 0.490, running_acc: 83.1 %
Validation accuracy:  73.7 %
Finished Training
```

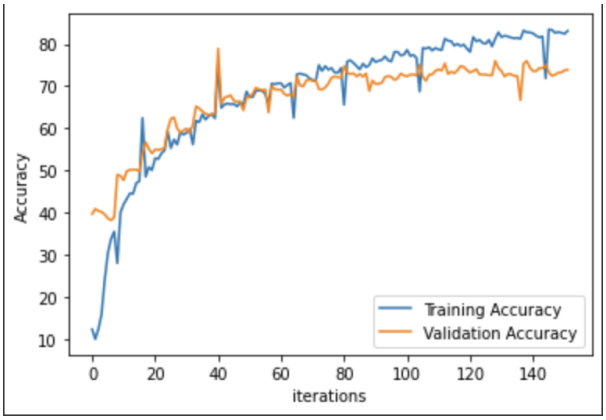
Figure 5: Output of First Model

3.4 Exercise

Now we will plot the training and validation loss 6a and training validation accuracy 6b



(a) Plot of Training/Validation Loss

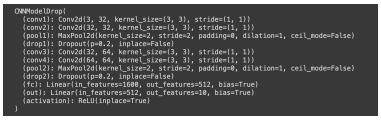


(b) Plot of Training/Validation accuracy

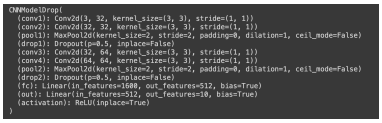
As we can see the loss of training is more than validation because we don't have dropout layer and we can't optimize our inputs at each mini-batch and also the same reason we have more accuracy at training than validation.

3.5 Exercice

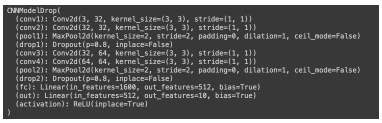
Dropout has a parameter of p which is probability of inactive neurons in corresponding layer.
We modified our model with two DropOut 3 different probability values 7a 7b 7c:



(a) DropOut model with p=20%

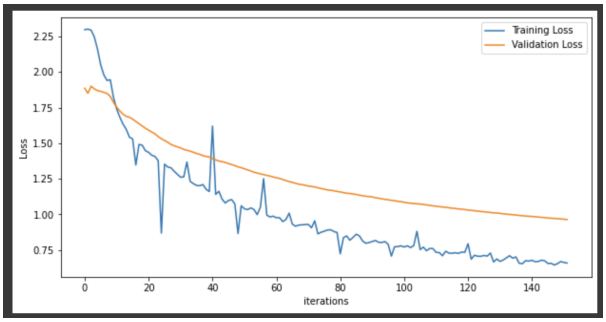


(b) DropOut model with p=50%

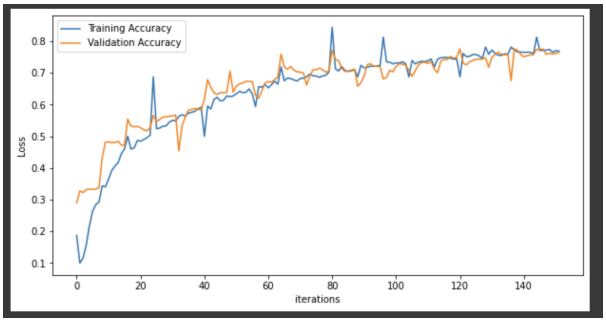


(c) DropOut model with p=80%

And also we can see different learning curves of each value of p below with p equals 0.2 loss 8a and accuracy 8b:



(a) Training/Validation Loss of p=20%



(b) Training/Validation Accuracy of p=20%

As we observe accuracy is not as good as we expected
We have a good validation accuracy for model with p=50% here is the output :

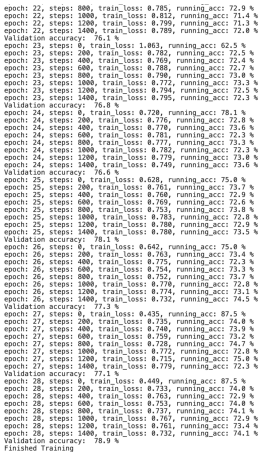
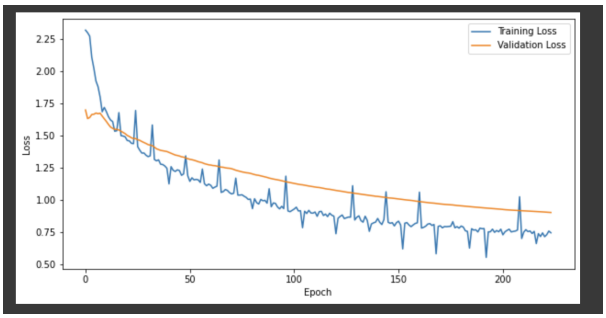


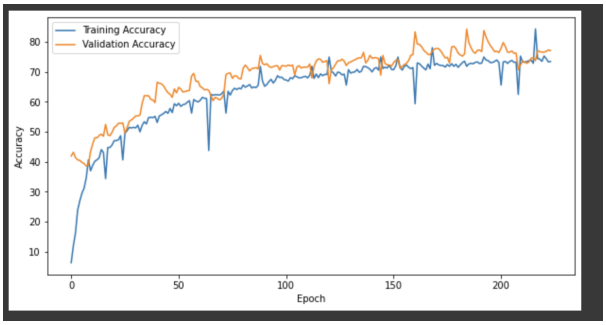
Figure 9: Output of Model with Dropout p=0.5

As we observe the validation accuracy is above 70%

Curves for p=0.5 is below with loss 10a and accuracy plot 10b:



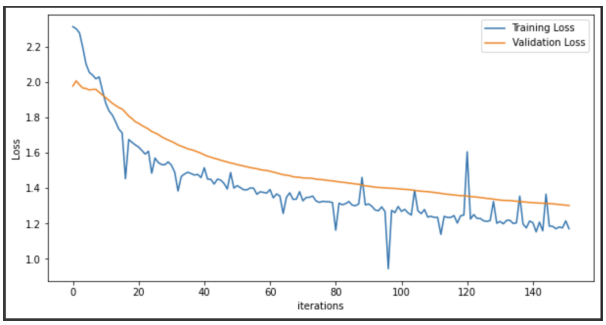
(a) Training/Validation Loss of Probability p=50%



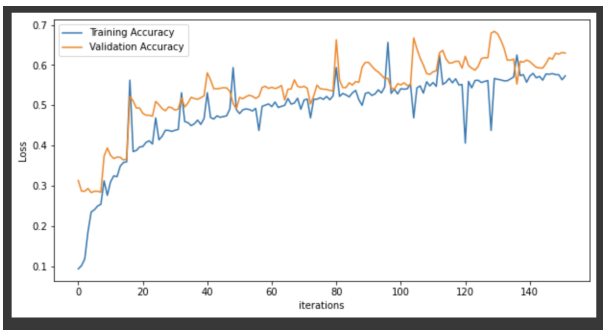
(b) Training/Validation Accuracy of Probability p=50%

Here we have a better curving but we needed to change epoch number to 29 improve convergence it differ from p=0.2 because it has better accuracy and better loss

Also we have two more plot for p=80% with loss 11a and accuracy 11b :



(a) Training/Validation Loss of Probability p=80%



(b) Training/Validation Accuracy of Probability p=80%

For this value of probability, we have a nice curve but the loss and accuracy value is not enough as we

wanted,the output of this model can be seen below 12 :

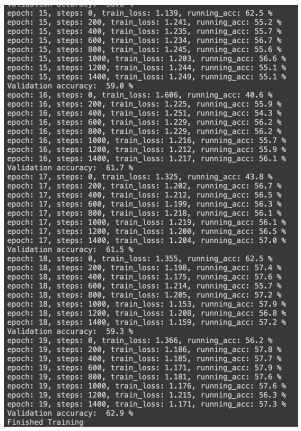
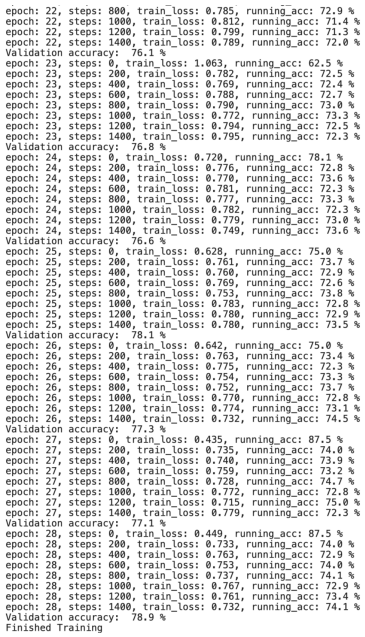


Figure 12: Output of Model with p=0.8

As we can observe the final validation of this model is 62.9 with 20 epoch.Even we add more epoch this not improve our model's accuracy much

3.6 Exercise

The best model we obtained is the model with p=50% also the final validation 13a accuracy and test accuracy 13b we obtained is shown :



(a) Output of model with Precision p=0.5

(b) Test Accuracy of model with p=0.5

With this trained model we randomly select 4 validation images 14 and calculate their probability distribution by classes with the help of softmax function

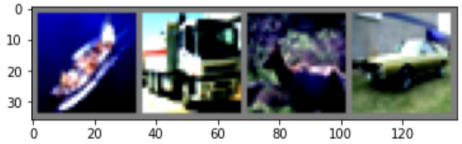


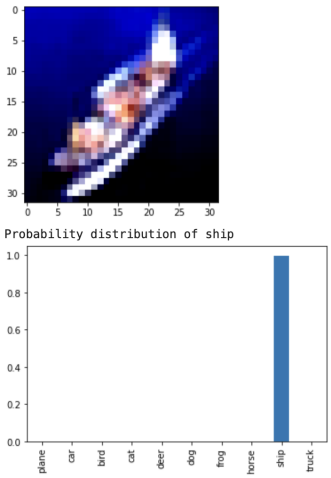
Figure 14: Randomly selected 4 images

To plot them better we turned this probabilities into DataFrame 15 by the help of pandas library

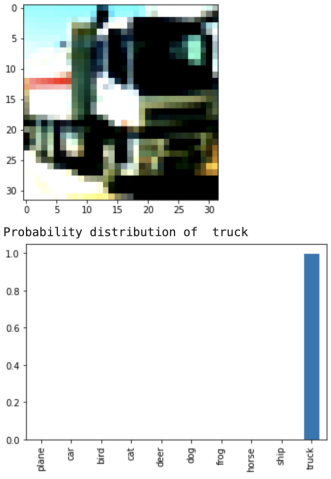
	plane	car	bird	cat	deer	dog	frog	horse	ship	truck
0	0.001143	0.000472	1.354979e-04	0.000021	8.251639e-07	1.044095e-05	5.280125e-04	0.000002	0.997679	0.000010
1	0.000023	0.001853	1.985228e-07	0.000004	8.409294e-07	3.978790e-07	5.445802e-08	0.000024	0.000035	0.998060
2	0.002731	0.010763	6.473011e-02	0.106039	4.277173e-01	1.047198e-02	3.322676e-01	0.015964	0.026970	0.002345
3	0.063451	0.856819	4.779393e-04	0.000252	5.494499e-04	9.854383e-05	1.390731e-03	0.000054	0.001209	0.075699

Figure 15: DataFrame of 4 images

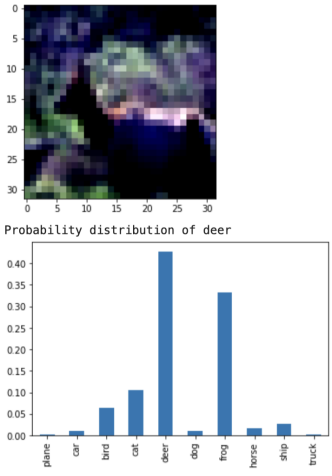
And we have 4 images with their probabilities 16a , 16b , 16c and 16d:



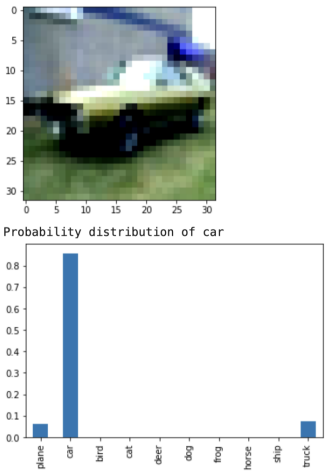
(a) Probability Distribution of Ship



(b) Probability Distribution of Truck



(c) Probability Distribution of Deer



(d) Probability Distribution of Car

With that small example we can say that predictions of this model is reasonable because to be correct it should give a strong probability of each image but as we can see the image of deer the probabilities of the other classes are not less.

3.7 Exercise (Bonus)

Improved model architecture shown 17 :

```
CNNImp(
  (conv_layer): Sequential(
    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.01, inplace=True)
    (3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
    (4): LeakyReLU(negative_slope=0.01, inplace=True)
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Dropout2d(p=0.05, inplace=False)
    (7): Conv2d(64, 256, kernel_size=(3, 3), stride=(1, 1))
    (8): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (9): LeakyReLU(negative_slope=0.01, inplace=True)
    (10): Conv2d(256, 128, kernel_size=(3, 3), stride=(1, 1))
    (11): LeakyReLU(negative_slope=0.01, inplace=True)
    (12): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (13): Dropout2d(p=0.05, inplace=False)
  )
  (fc_layer): Sequential(
    (0): Linear(in_features=3200, out_features=1024, bias=True)
    (1): LeakyReLU(negative_slope=0.01, inplace=True)
    (2): Linear(in_features=1024, out_features=512, bias=True)
    (3): LeakyReLU(negative_slope=0.01, inplace=True)
    (4): Linear(in_features=512, out_features=10, bias=True)
  )
)
```

Figure 17: Architecture of Best Model

Added some batch normalization for preventing overfitting and used LeakyRelu activation instead of Relu also used probability as p=50% for dropout layer also decrease epoch number to 15 to prevent overtrain our model the output of training available as below 18:

```
Validation accuracy:  78.2 %
epoch: 8, steps: 0, train_loss: 0.602, running_acc: 68.8 %
epoch: 8, steps: 200, train_loss: 0.475, running_acc: 83.6 %
epoch: 8, steps: 400, train_loss: 0.494, running_acc: 82.7 %
epoch: 8, steps: 600, train_loss: 0.487, running_acc: 83.1 %
epoch: 8, steps: 800, train_loss: 0.484, running_acc: 83.3 %
epoch: 8, steps: 1000, train_loss: 0.492, running_acc: 82.8 %
epoch: 8, steps: 1200, train_loss: 0.475, running_acc: 83.2 %
epoch: 8, steps: 1400, train_loss: 0.514, running_acc: 82.5 %
Validation accuracy:  79.6 %
epoch: 9, steps: 0, train_loss: 0.235, running_acc: 93.8 %
epoch: 9, steps: 200, train_loss: 0.416, running_acc: 86.0 %
epoch: 9, steps: 400, train_loss: 0.424, running_acc: 85.6 %
epoch: 9, steps: 600, train_loss: 0.410, running_acc: 85.6 %
epoch: 9, steps: 800, train_loss: 0.413, running_acc: 85.2 %
epoch: 9, steps: 1000, train_loss: 0.437, running_acc: 84.4 %
epoch: 9, steps: 1200, train_loss: 0.423, running_acc: 84.6 %
epoch: 9, steps: 1400, train_loss: 0.429, running_acc: 85.0 %
Validation accuracy:  81.8 %
epoch: 10, steps: 0, train_loss: 0.429, running_acc: 87.5 %
epoch: 10, steps: 200, train_loss: 0.353, running_acc: 88.1 %
epoch: 10, steps: 400, train_loss: 0.354, running_acc: 87.7 %
epoch: 10, steps: 600, train_loss: 0.356, running_acc: 87.1 %
epoch: 10, steps: 800, train_loss: 0.360, running_acc: 87.7 %
epoch: 10, steps: 1000, train_loss: 0.379, running_acc: 86.5 %
epoch: 10, steps: 1200, train_loss: 0.351, running_acc: 87.9 %
epoch: 10, steps: 1400, train_loss: 0.369, running_acc: 87.1 %
Validation accuracy:  81.5 %
epoch: 11, steps: 0, train_loss: 0.281, running_acc: 96.9 %
epoch: 11, steps: 200, train_loss: 0.290, running_acc: 90.2 %
epoch: 11, steps: 400, train_loss: 0.302, running_acc: 89.4 %
epoch: 11, steps: 600, train_loss: 0.298, running_acc: 89.7 %
epoch: 11, steps: 800, train_loss: 0.301, running_acc: 89.9 %
epoch: 11, steps: 1000, train_loss: 0.328, running_acc: 88.5 %
epoch: 11, steps: 1200, train_loss: 0.295, running_acc: 89.2 %
epoch: 11, steps: 1400, train_loss: 0.315, running_acc: 89.0 %
Validation accuracy:  80.0 %
epoch: 12, steps: 0, train_loss: 0.287, running_acc: 93.8 %
epoch: 12, steps: 200, train_loss: 0.224, running_acc: 92.7 %
epoch: 12, steps: 400, train_loss: 0.246, running_acc: 91.4 %
epoch: 12, steps: 600, train_loss: 0.238, running_acc: 91.8 %
epoch: 12, steps: 800, train_loss: 0.248, running_acc: 91.1 %
epoch: 12, steps: 1000, train_loss: 0.254, running_acc: 91.1 %
epoch: 12, steps: 1200, train_loss: 0.254, running_acc: 91.4 %
epoch: 12, steps: 1400, train_loss: 0.269, running_acc: 90.7 %
Validation accuracy:  80.0 %
epoch: 13, steps: 0, train_loss: 0.247, running_acc: 90.6 %
epoch: 13, steps: 200, train_loss: 0.183, running_acc: 93.9 %
epoch: 13, steps: 400, train_loss: 0.185, running_acc: 94.0 %
epoch: 13, steps: 600, train_loss: 0.217, running_acc: 92.4 %
epoch: 13, steps: 800, train_loss: 0.196, running_acc: 93.7 %
epoch: 13, steps: 1000, train_loss: 0.212, running_acc: 92.7 %
epoch: 13, steps: 1200, train_loss: 0.212, running_acc: 92.5 %
epoch: 13, steps: 1400, train_loss: 0.204, running_acc: 92.5 %
Validation accuracy:  80.9 %
epoch: 14, steps: 0, train_loss: 0.052, running_acc: 100.0 %
epoch: 14, steps: 200, train_loss: 0.143, running_acc: 95.1 %
epoch: 14, steps: 400, train_loss: 0.152, running_acc: 94.8 %
epoch: 14, steps: 600, train_loss: 0.167, running_acc: 94.1 %
epoch: 14, steps: 800, train_loss: 0.169, running_acc: 94.1 %
epoch: 14, steps: 1000, train_loss: 0.161, running_acc: 94.5 %
epoch: 14, steps: 1200, train_loss: 0.161, running_acc: 94.7 %
epoch: 14, steps: 1400, train_loss: 0.174, running_acc: 94.2 %
Validation accuracy:  82.1 %
Finished Training
```

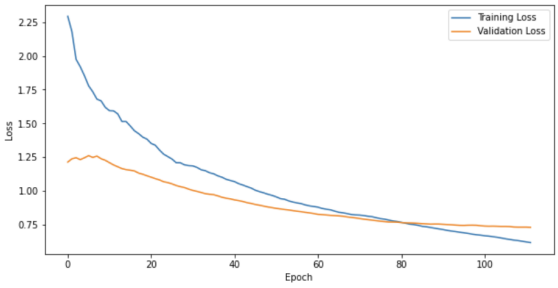
Figure 18: Output of Best Model

We also have a satisfying test accuracy value 19:

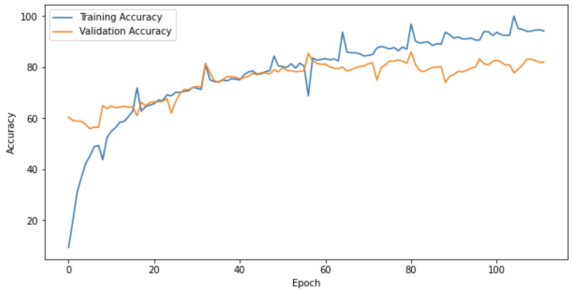
```
Test accuracy:  80.2 %
```

Figure 19: Test Accuracy of Best Model

The learning curves of this model can be observe too 20a and 19 :



(a) Training/Validation Loss Of Best Model



(b) Training/Validation Accuracy Of Best Model

4 Questions

Question 1

When (for what purpose) do you use softmax activation?

For multi-class classification problems where class membership is required on more than two class labels,because softmax returns values between 0-1 that we can differ each class with one value.

Question 2

With non-zero Momentum value we include velocity v_t value to our training.

Question 3

Name one example type of data for which you may want to use 1D convolution.

1D convolution happens when the kernel can only move in one direction across the data. One example can be ECG signals data.

Question 4

What is test and training time behavior of Dropout ?

During training time, dropout randomly sets node values to zero.So dropout randomly kills node values with “dropout probability” .

During Test time, dropout does not kill node values, but all the weights in the layer were multiplied by probability value of dropout.