**Fahad Fiaz – (**303141**) – G2**

**System Info:**

|  |  |
| --- | --- |
| Processor | i7-5500U , 2.40GHz |
| Cores | 4 |
| Operating system | Windows 64 Bit |
| Ram | 8GB |
| Programming Language | Python 3.7.7 |

**Exercise1:**

First I have setup up some parameters for neural network and loaded the dataset. Neural network specifications are described below

# load datset  
olivetti = datasets.fetch\_olivetti\_faces*()*# Hyper-parameters   
input\_size = 4096 # 28x28  
hidden\_size = 100  
num\_classes = 40  
num\_epochs = 100  
batch\_size = 10  
learning\_rate = 0.01

Then I wrote a class “OlivettifacesDataset“ to do preprocessing on dataset and convert it to tensors.

class OlivettifacesDataset*(*Dataset*)*:  
  
 def \_\_init\_\_*(*self*)*:  
 # Initialize data, download, etc.  
 self.n\_samples = olivetti.data.shape*[*0*]* self.x\_data = torch.from\_numpy*(*olivetti.data*)* # size [n\_samples, n\_features]  
 self.y\_data = torch.from\_numpy*(*olivetti.target*)* # size [n\_samples, 1]  
 self.y\_data = self.y\_data.type*(*torch.LongTensor*)* # support indexing such that dataset[i] can be used to get i-th sample  
 def \_\_getitem\_\_*(*self, index*)*:  
 return self.x\_data*[*index*]*, self.y\_data*[*index*]* # we can call len(dataset) to return the size  
 def \_\_len\_\_*(*self*)*:  
 return self.n\_samples

Then I split the dataset into train dataset and test dataset. Then I used the Data loader function to load the dataset. This function will automatically shuffle and load the data in batches.

dataset = OlivettifacesDataset*()*train\_size = int*(*0.9 \* len*(*dataset*))*test\_size = len*(*dataset*)* - train\_size  
train\_dataset, test\_dataset = torch.utils.data.random\_split*(*dataset, *[*train\_size, test\_size*])*# Load whole dataset with DataLoader  
# shuffle: shuffle data, good for training  
train\_loader = DataLoader*(*dataset=train\_dataset,  
 batch\_size=batch\_size,  
 shuffle=True*)*test\_loader = DataLoader*(*dataset=test\_dataset,  
 batch\_size=batch\_size,  
 shuffle=False*)*

Then I created Neural Net according to specifications mentioned in exercise. Here I used 1 hidden layer with 100 neurons. Also I have used relu activation function on hidden layer. ”Input size” is number of features in 1 training example.

class NeuralNet*(*nn.Module*)*:  
 def \_\_init\_\_*(*self, input\_size, hidden\_size, num\_classes*)*:  
 super*(*NeuralNet, self*)*.\_\_init\_\_*()* self.input\_size = input\_size  
 self.l1 = nn.Linear*(*input\_size, hidden\_size*)* self.relu = nn.ReLU*()* self.l2 = nn.Linear*(*hidden\_size, num\_classes*)* def forward*(*self, x*)*:  
 out = self.l1*(*x*)* out = self.relu*(*out*)* out = self.l2*(*out*)* # no activation and no softmax at the end  
 return out

Function to count trainable parameters

def count\_parameters*(*model*)*:  
 return sum*(*p.numel*()* for p in model.parameters*()* if p.requires\_grad*)*

Here I used cross entropy loss. Also I used SGD optimizer to update my weights in backpropogation.

model = NeuralNet*(*input\_size, hidden\_size, num\_classes*)*writer = SummaryWriter*()*# Loss and optimizer  
criterion = nn.CrossEntropyLoss*()*optimizer = torch.optim.SGD*(*model.parameters*()*, lr=learning\_rate*)*

Below first data in loaded in batch. Batch size is defined above. Then do the forward pass, backward pass, calculate percentage prediction accuracy by dividing the correct classified prediction with total predictions done in specific batch. Here training loss is also calculated. The loss is return from the cross entropy loss function and added to get total training loss. Also some things are added in tensor board for graphs, histogram and distributions

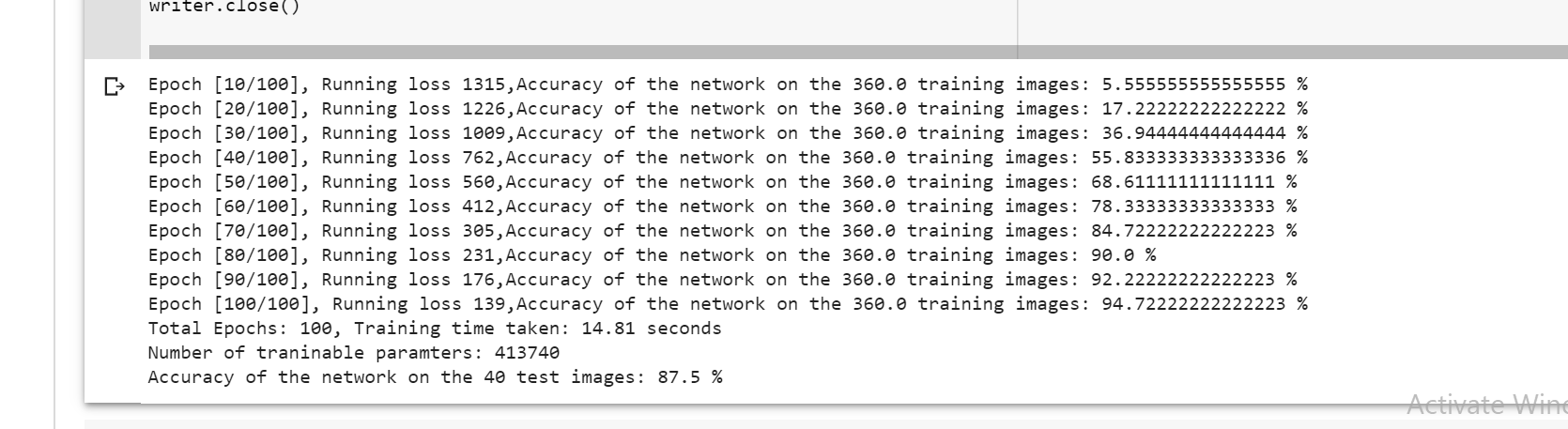
for epoch in range*(*num\_epochs*)*:  
 n\_correct = 0.0  
 n\_samples = 0.0  
 for i, *(*images, labels*)* in enumerate*(*train\_loader*)*:  
 # Forward pass  
 outputs = model*(*images*)* loss = criterion*(*outputs, labels*)* # Backward and optimize  
 optimizer.zero\_grad*()* loss.backward*()* optimizer.step*()* # Prediction and calculatng accuracy  
  
 \_, predicted = torch.max*(*outputs.data, 1*)* n\_samples += labels.size*(*0*)* n\_correct += *(*predicted == labels*)*.sum*()*.item*()* running\_loss += loss.item*()* if *(*epoch + 1*)* % 1 == 0:  
 writer.add\_histogram*(*'HiddenLayer.bias', model.l1.bias, epoch + 1*)* writer.add\_histogram*(*'HiddenLayer.weight', model.l1.weight, epoch + 1*)* if *(*epoch + 1*)* % 10 == 0:  
 acc = 100.0 \* n\_correct / n\_samples  
 print*(* f'Epoch [*{*epoch + 1*}*/*{*num\_epochs*}*], Running loss *{*int*(*running\_loss*)}*,Accuracy of the network on the *{*n\_samples*}* training images: *{*acc*}* %'*)* ############## TENSORBOARD ########################  
 writer.add\_scalar*(*'training loss', running\_loss, epoch + 1*)* writer.add\_scalar*(*'training accuracy', acc, epoch + 1*)* running\_loss = 0.0  
 ###################################################

The following code calculates accuracy of model on test set. The process of calculating accuracy is same as defined above.

with torch.no\_grad*()*:  
 n\_correct = 0  
 n\_samples = 0  
 running\_correct = 0.0  
 running\_sample = 0.0  
  
 for i, *(*images, labels*)* in enumerate*(*test\_loader*)*:  
 outputs = model*(*images*)* # max returns (value ,index)  
 \_, predicted = torch.max*(*outputs.data, 1*)* n\_samples += labels.size*(*0*)* n\_correct += *(*predicted == labels*)*.sum*()*.item*()* running\_sample += labels.size*(*0*)* running\_correct += *(*predicted == labels*)*.sum*()*.item*()* running\_loss += loss.item*()* if *(*i + 1*)* % 1 == 0:  
 acc = 100.0 \* running\_correct / running\_sample  
 ############## TENSORBOARD ########################  
 writer.add\_scalar*(*'Prediction accuracy', acc, i + 1*)* running\_correct = 0.0  
 running\_sample = 0.0  
 ###################################################  
  
 acc = 100.0 \* n\_correct / n\_samples  
 print*(*f'Accuracy of the network on the *{*n\_samples*}* test images: *{*acc*}* %'*)*

**Outputs:**

Neural network trained on 100 epochs with batch size 10



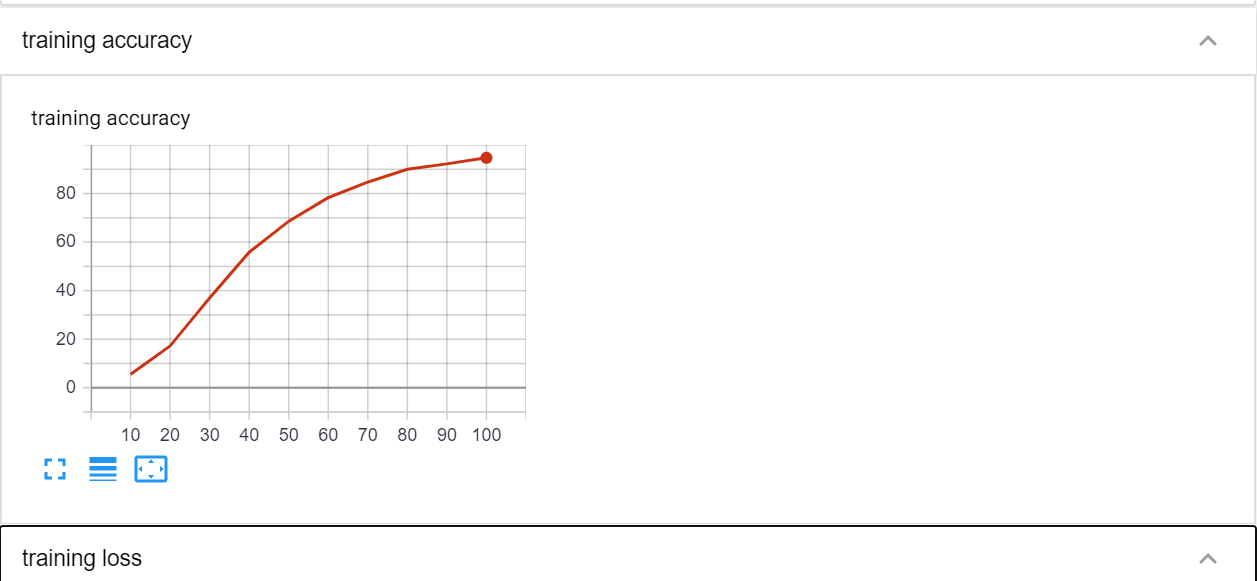
**Training time:** 14.81 seconds

**Number of Trainable parameter’s if 100 epochs run:** 413740

**Accuracy on test images:** 87.5%

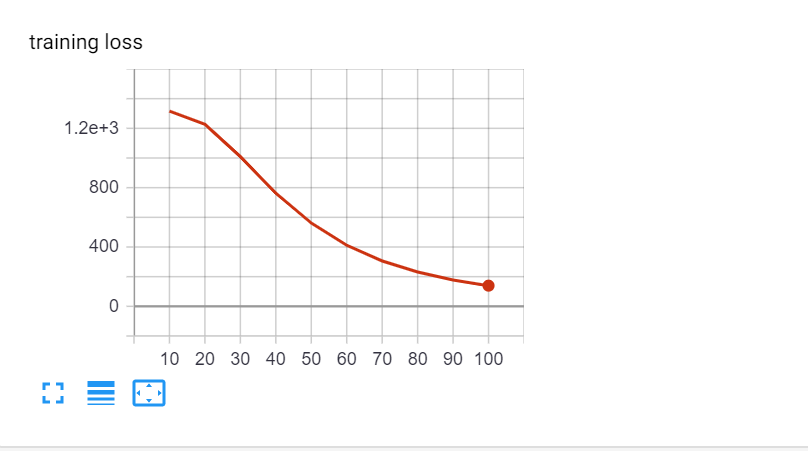
**Training accuracy:**

Here x-axis is number of epochs and y-axis is accuracy in that epoch. This graphs shows accuracy is increasing with each passing epoch



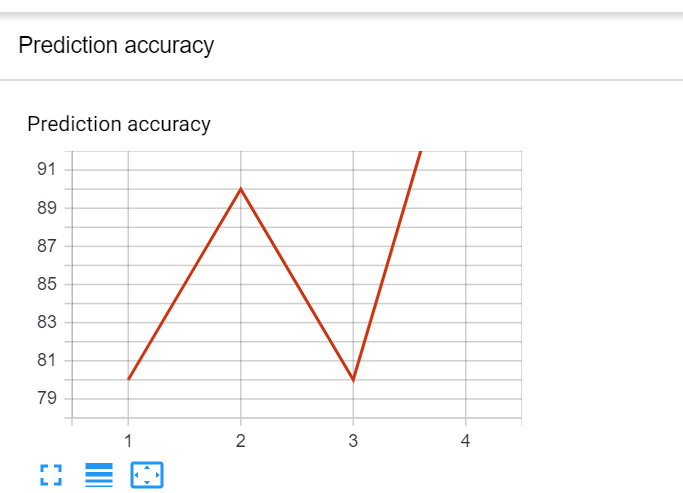
**Training loss:**

Here x-axis is number of epochs and y-axis is loss in that epoch. This graphs shows loss is decreasing with each passing epoch.

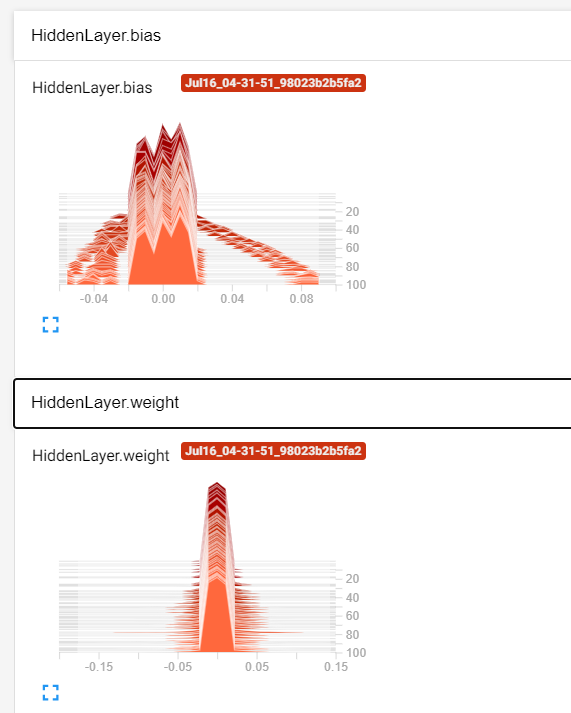


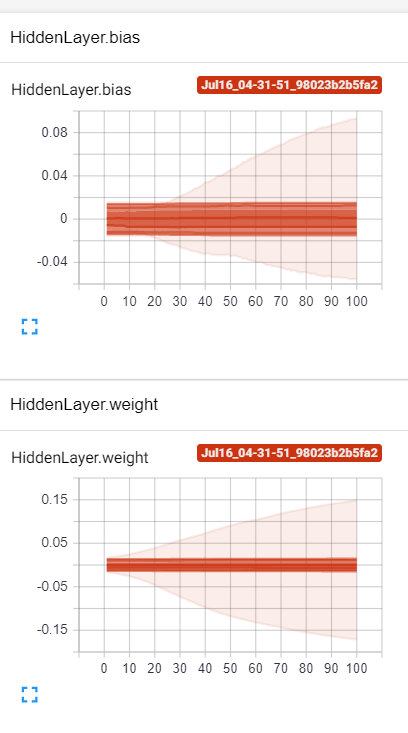
**Testing Accuracy:**

Here x-axis is batch number from total batch of images in test dataset and y-axis is accuracy in that batch.

****

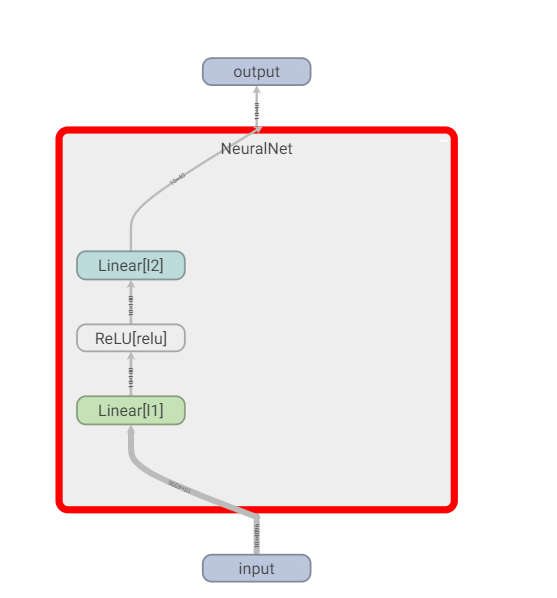
**Histogram and distribution of weights and bias:**

****

****

**Graph:**

Input Layer – relu(Hidden Layer)– Output Layer

****

**Exercise2:**

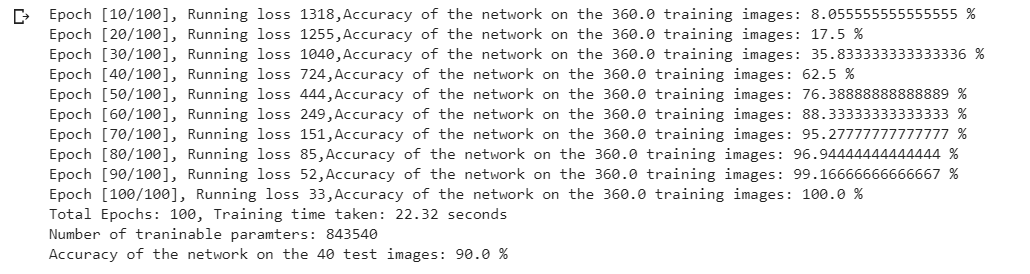
Here the code is almost same. I have already described the code above. I have only added a hidden layer in neural network.

**Neural network specification:** hidden\_layer1\_size = 200, hidden\_laye2 = 100, batch size = 10, learning rate = 0.01

**Neural network structure:** Input Layer – relu(Hidden Layer 1) –relu(Hidden Layer 2) - Output Layer

**Outputs:**

Neural network trained on 100 epochs with batch size 10



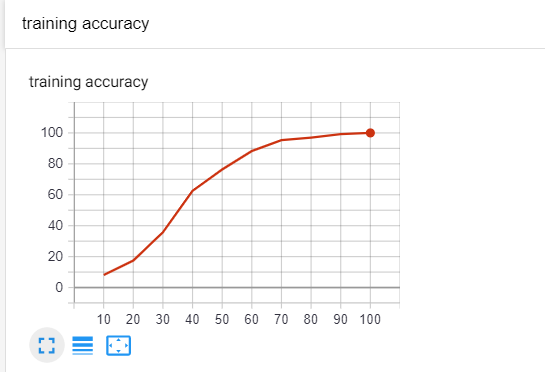
**Training time:** 22.32 seconds

**Number of Trainable parameter’s if 100 epochs run:** 843540

**Accuracy on test images:** 90%

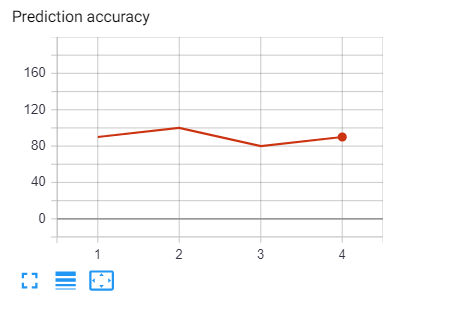
**Training accuracy:**

Here x-axis is number of epochs and y-axis is accuracy in that epoch. This graphs shows accuracy is increasing with each passing epoch



**Testing Accuracy:**

Here x-axis is batch number from total batch of images in test dataset and y-axis is accuracy in that batch.

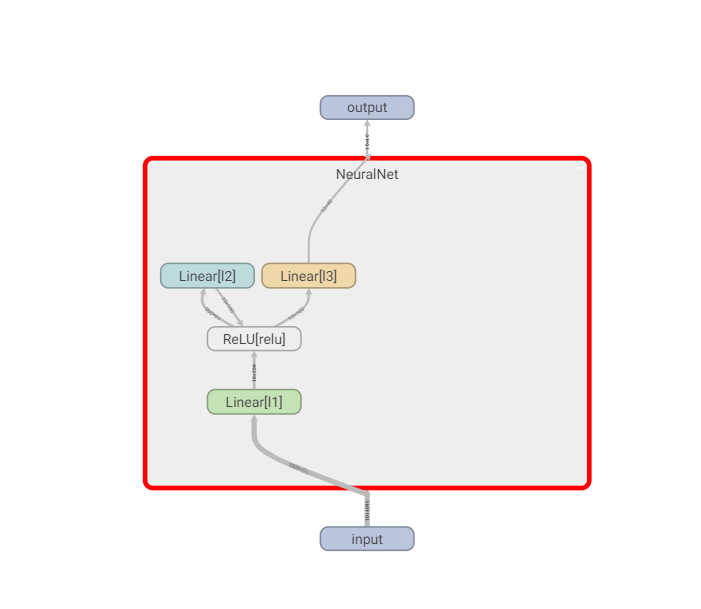
****

**Distribution and histogramof weights and bias:**

** **

** **

**Graph:**

****

**Comparison of Exercise 2 with Exercise 1:**

|  |  |  |
| --- | --- | --- |
|  | Exercise 1 | Exercise 2 |
| Time taken (seconds) | 14.81 | 22.32 |
| Trainable Parameters | 413740 | 843540 |
| Percentage accuracy | 87.5 | 90 |

**Exercise3:**

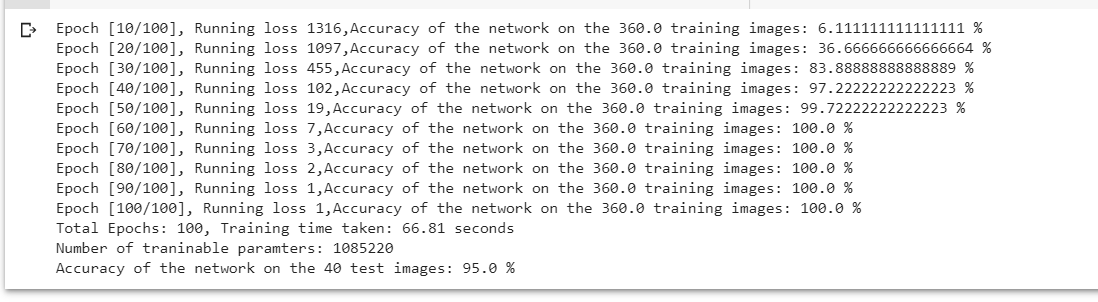
Here the code is almost same. I have already described the code above. But here the structure of neural network is changed. I have just made changes in “NeuralNet” class to get the required structure.

**Neural network structure:** Input Layer - Convolutional Layer - Max Pooling Layer - Fully Connected Layer - Output Layer

**Neural network specification:** fully connected layer size = 120, output layer size=40, batch size = 10, learning rate = 0.01

**Outputs:**

Neural network trained on 100 epochs with batch size 10



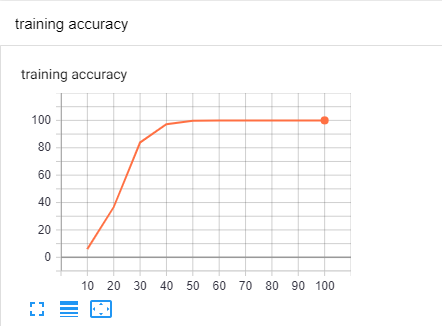
**Training time:** 66.81 seconds

**Number of Trainable parameter’s if 100 epochs run:** 1085220

**Accuracy on test images:** 95%

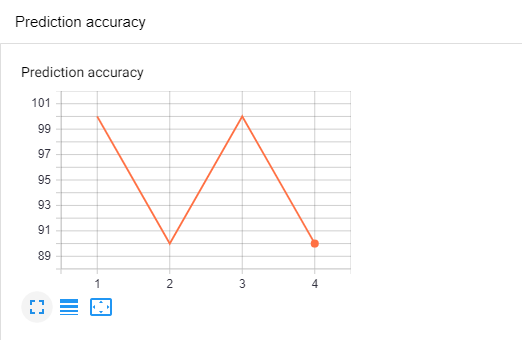
**Training accuracy:**

Here x-axis is number of epochs and y-axis is accuracy in that epoch. This graphs shows accuracy is increasing with each passing epoch

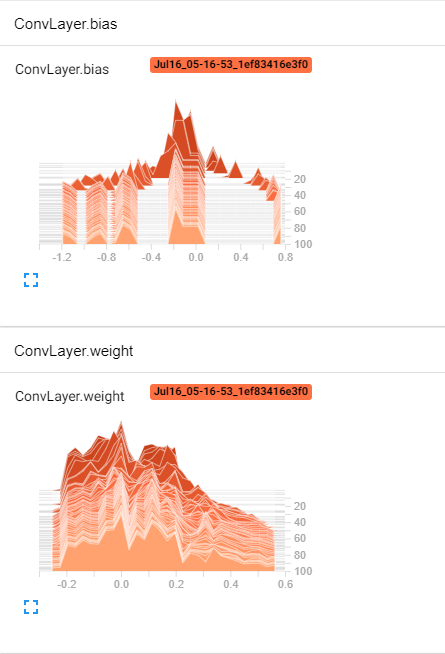
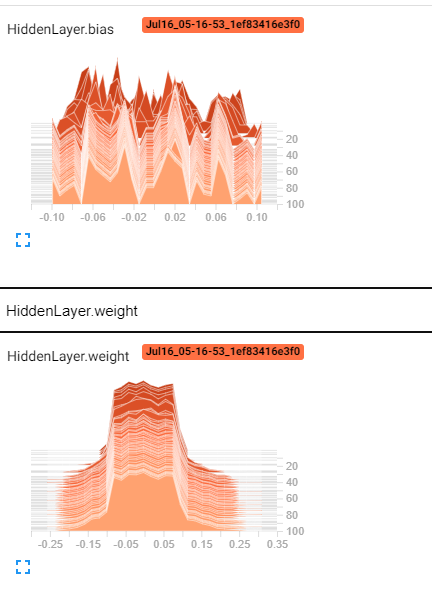


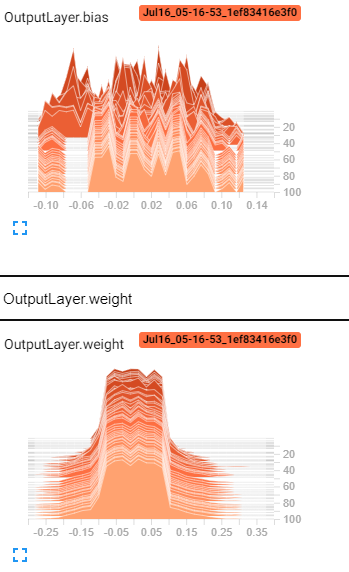
**Testing Accuracy:**

Here x-axis is batch number from total batch of images in test dataset and y-axis is accuracy in that batch.

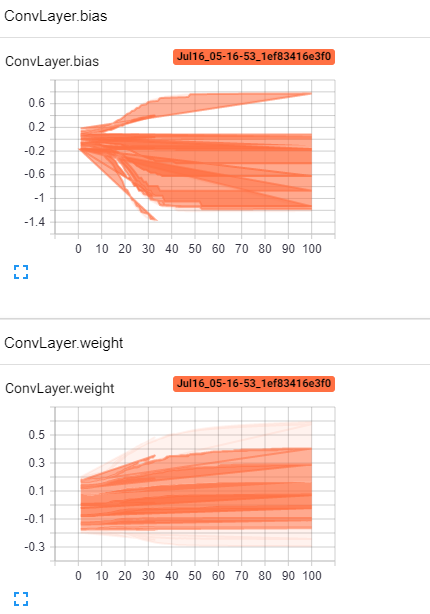
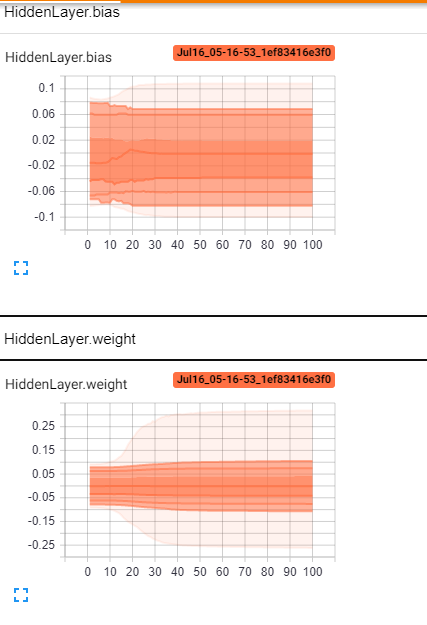
****

**Histogram of weights and bias:**

** **

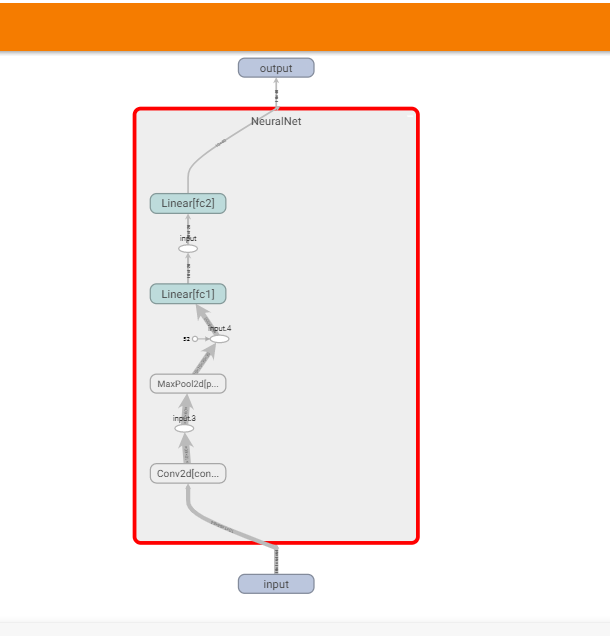
****

**Distribution of weights and bias:**

** **

****

**Graph:**

****

**Comparison of Exercise 3 with Exercise 2:**

|  |  |  |
| --- | --- | --- |
|  | Exercise 2 | Exercise 3 |
| Time taken (seconds) | 22.32 | 66.81 |
| Trainable Parameters | 843540 | 1085220 |
| Percentage accuracy | 90% | 95% |

**Exercise4:**

Here first I have set up some parameters:

# Hyper-parameters   
input\_size = 784 # 28x28  
num\_classes = 10  
num\_epochs = 10  
batch\_size = 100  
learning\_rate = 0.01

Then I loaded the Mnist dataset train dataset and test dataset. Then used data loader function to shuffle and load data in batches.

Then I wrote Multi Task Model class. In neural network I have used

1 convolutional layer – 1 Max Pooling Layer- 1 fully connected layer for classification (size: 120), 1 fully connected layer for regression (size:125) – 1 output layer for classification head, 1 output layer for regression head.

class MultiTaskModel*(*nn.Module*)*:  
 def \_\_init\_\_*(*self*)*:  
 super*(*MultiTaskModel, self*)*.\_\_init\_\_*()* self.conv1 = nn.Conv2d*(*1, 10, 5*)* # input channel size, output channel size ,kernal size  
 self.pool = nn.MaxPool2d*(*2, 2*)* self.fc1 = nn.Linear*(*10 \* 12 \* 12, 120*)* self.fc2 = nn.Linear*(*120, 10*)* self.fr1 = nn.Linear*(*10 \* 12 \* 12, 125*)* self.fr2 = nn.Linear*(*125, 100*)* def forward*(*self, input*)*:  
 x = self.pool*(*F.relu*(*self.conv1*(*input*)))* x = x.view*(*-1, 10 \* 12 \* 12*)* fc\_classifier = F.relu*(*self.fc1*(*x*))* classifier\_out = self.fc2*(*fc\_classifier*)* fc\_regression = F.relu*(*self.fr1*(*x*))* regression\_out = self.fr2*(*fc\_regression*)* outputs = *[*classifier\_out, regression\_out*]* return outputs

The following class take “Multi Task Model” model object and loss functions as input.

Then in forward pass it run the Multitask Model on input image which return 2 output (1st from regression head, other from classification head). Then it calculates classification loss and regression loss separately using loss function MSE for regression and Cross entropy for classification head. Then we sum the loss and return all calculated values.

class MultiTaskLoss*(*nn.Module*)*:  
 def \_\_init\_\_*(*self, model, loss\_fn*)*:  
 super*(*MultiTaskLoss, self*)*.\_\_init\_\_*()* self.model = model  
 self.loss\_fn = loss\_fn  
  
 def forward*(*self, input, targets*)*:  
 outputs = self.model*(*input*)* classification\_loss = self.loss\_fn*[*0*](*outputs*[*0*]*, targets*[*0*])* regression\_loss = self.loss\_fn*[*1*](*outputs*[*1*]*, targets*[*1*])* total\_loss = classification\_loss + regression\_loss  
 classification\_prediction = outputs*[*0*]* return classification\_loss, regression\_loss, total\_loss, classification\_prediction

Then I calculated the loss for regression, loss of classification and the total loss on training dataset using the following code.

for i, *(*images, labels*)* in enumerate*(*train\_loader*)*:  
 # Forward pass  
  
 images = images.to*(*device*)* # shifting data to gpu if available  
 labels = labels.to*(*device*)* labels2 = labels.to*(*device*)* loss\_cl, loss\_reg, total\_loss, \_ = mtl*(*images, *[*labels, labels*])* running\_loss += total\_loss.item*()* running\_classification\_loss += loss\_cl.item*()* running\_regression\_loss += loss\_reg.item*()*

if *(*epoch + 1*)* % 1 == 0:  
 print*(* f"Epoch [*{*epoch + 1*}*/*{*num\_epochs*}*], Total\_loss:*{*running\_loss:.2f*}*, Classification\_loss:*{*running\_classification\_loss:.2f*}*, Regression\_loss:*{*running\_regression\_loss:.2f*}*"*)*   
 running\_loss = 0.0  
 running\_classification\_loss = 0.0  
 running\_regression\_loss = 0.0

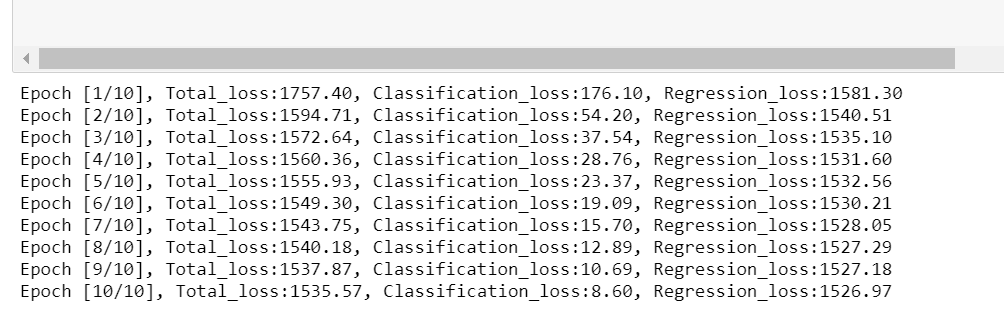
Following code calculate accuracy for classification on test dataset.

n\_correct = 0  
n\_samples = 0  
for i, *(*images, labels*)* in enumerate*(*test\_loader*)*:  
 outputs = mtl*(*images, *[*labels, labels*])* \_, \_, \_, predictions = mtl*(*images, *[*labels, labels*])* # max returns (value ,index)  
 values, predicted = torch.max*(*predictions.data, 1*)* n\_samples += labels.size*(*0*)* n\_correct += *(*predicted == labels*)*.sum*()*.item*()*acc = 100.0 \* n\_correct / n\_samples  
print*(*f'Accuracy of the classification network on the 10000 test images: *{*acc*}* %'*)*

**Outputs:**

I have run only 10 epochs to plot graph because it was taking a lot of time for more epochs.

**On Training dataset:**

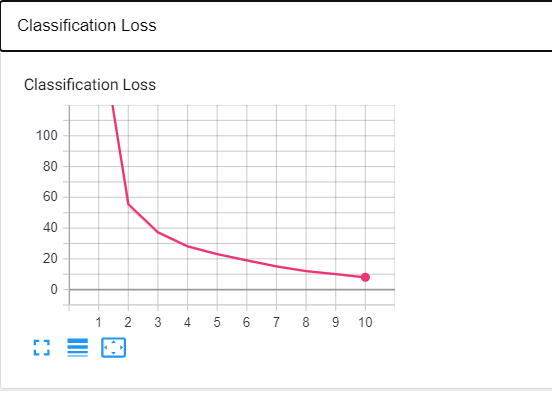


**On Testing dataset using classification head:**

**E:\DDA\Exercise_9\Final_Code\2.PNG**

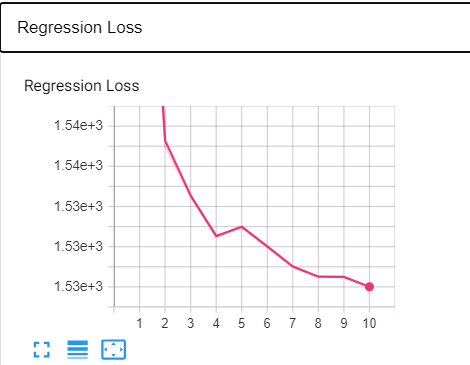
**loss of classification on training dataset:**

Here x-axis is number of epochs and y-axis is classification loss in that epoch. This graphs shows loss is decreasing with each passing epoch.

****

**loss for regression on training dataset:**

Here x-axis is number of epochs and y-axis is regression loss in that epoch. This graphs shows there is no significant decrease in loss with each passing epoch.

****

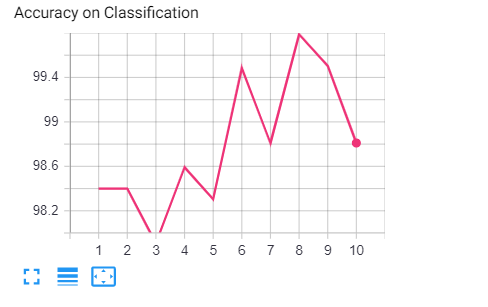
**Total loss on training dataset:**

Here x-axis is number of epochs and y-axis is total loss in that epoch. This graphs shows there is no significant decrease in total loss. This is due to the fact that the total loss is sum of classification loss and regression loss. Classification loss was decreasing but regression loss was not decreasing and regression loss is very big so there is not significant decrease in total loss. Better approach may be to calculate total loss by weighing multiple loss functions by considering the homoscedastic uncertainty of each task.

****

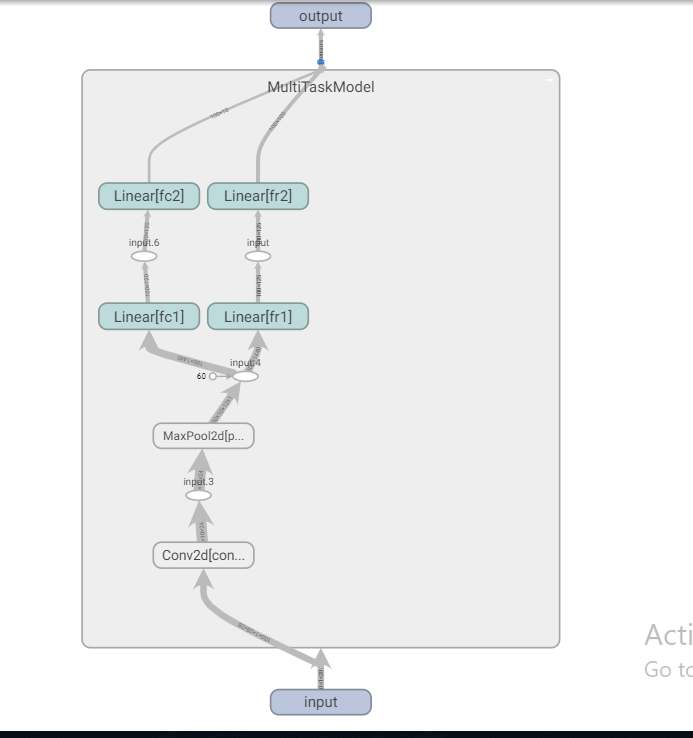
**Accuracy for classification:**

Here x-axis is batch number from total batch of images in test dataset and y-axis is accuracy in that batch.

****

**Final NN graph:**

1 convolutional layer – 1 Max Pooling Layer- 1 fully connected layer for classification (size: 120), 1 fully connected layer for regression (size:125) – 1 output layer for classification head, 1 output layer for regression head- final output.

****