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

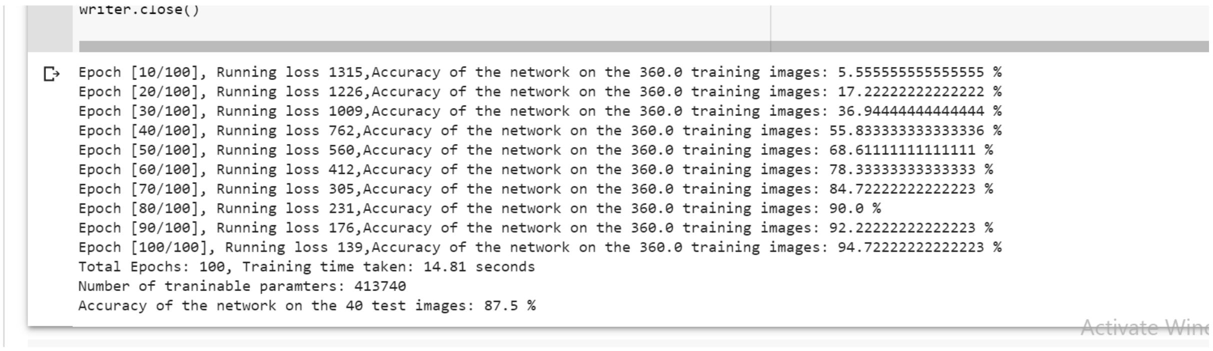
```
for epoch in range(num_epochs):
    n_correct = 0.0
    n_samples = 0.0
    for i, (images, labels) in enumerate(train_loader):
        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)

        # 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



```
writer.close()

Epoch [10/100], Running loss 1315,Accuracy of the network on the 360.0 training images: 5.555555555555555 %
Epoch [20/100], Running loss 1226,Accuracy of the network on the 360.0 training images: 17.2222222222222 %
Epoch [30/100], Running loss 1009,Accuracy of the network on the 360.0 training images: 36.94444444444444 %
Epoch [40/100], Running loss 762,Accuracy of the network on the 360.0 training images: 55.833333333333336 %
Epoch [50/100], Running loss 560,Accuracy of the network on the 360.0 training images: 68.6111111111111 %
Epoch [60/100], Running loss 412,Accuracy of the network on the 360.0 training images: 78.3333333333333 %
Epoch [70/100], Running loss 305,Accuracy of the network on the 360.0 training images: 84.7222222222223 %
Epoch [80/100], Running loss 231,Accuracy of the network on the 360.0 training images: 90.0 %
Epoch [90/100], Running loss 176,Accuracy of the network on the 360.0 training images: 92.2222222222223 %
Epoch [100/100], Running loss 139,Accuracy of the network on the 360.0 training images: 94.7222222222223 %
Total Epochs: 100, Training time taken: 14.81 seconds
Number of traninable paramters: 413740
Accuracy of the network on the 40 test images: 87.5 %
```

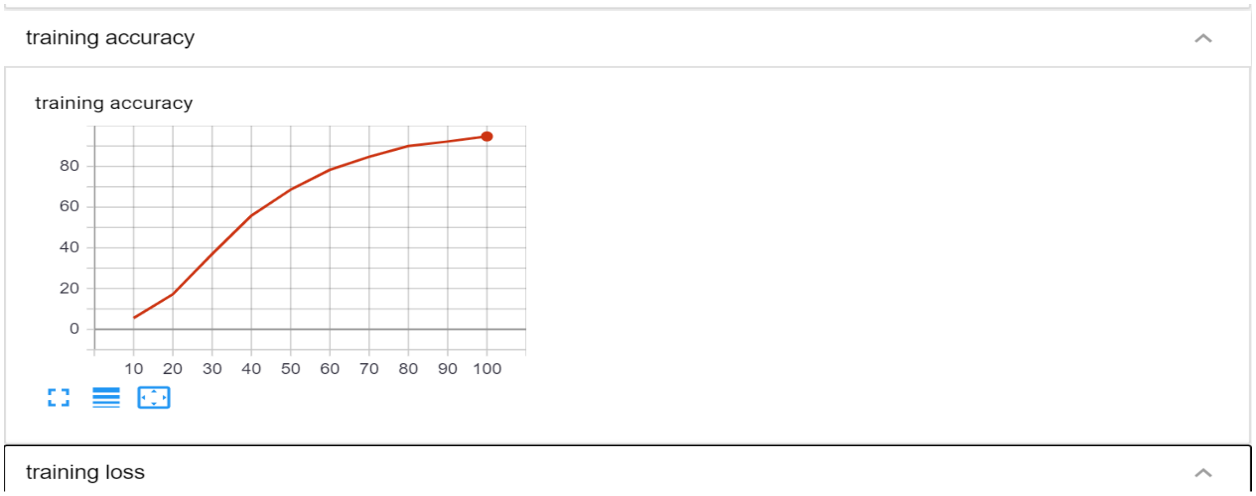
Training time: 14.81 seconds

Number of Trainable parameter's if 100 epochs run: 413740

Accuracy on test images: 87.5%

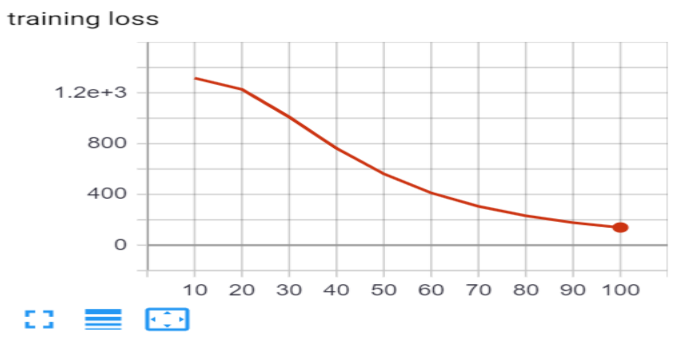
Training accuracy:

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



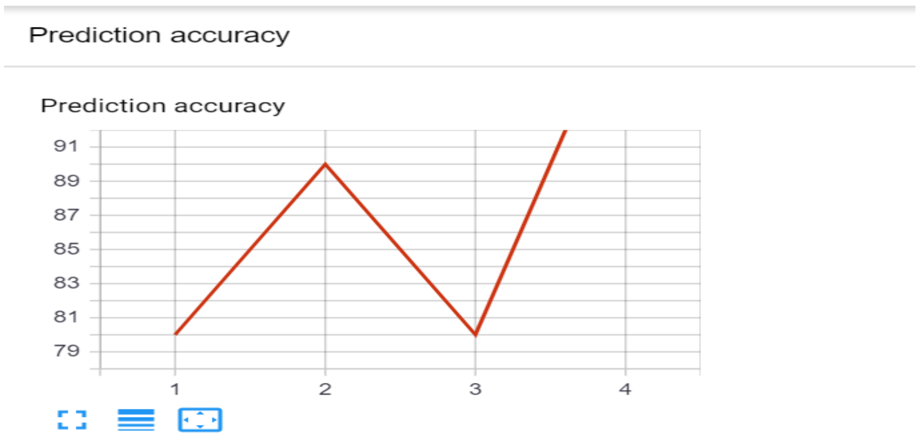
Training loss:

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.

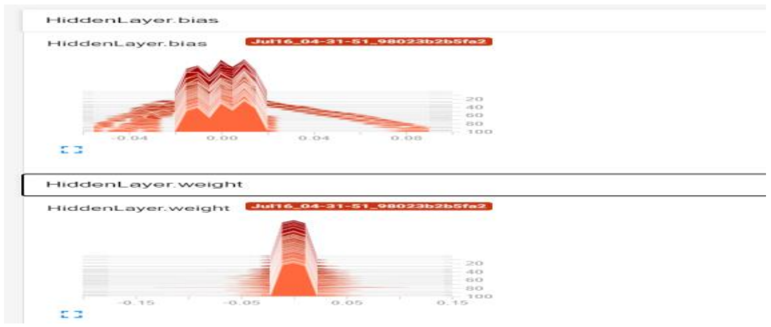


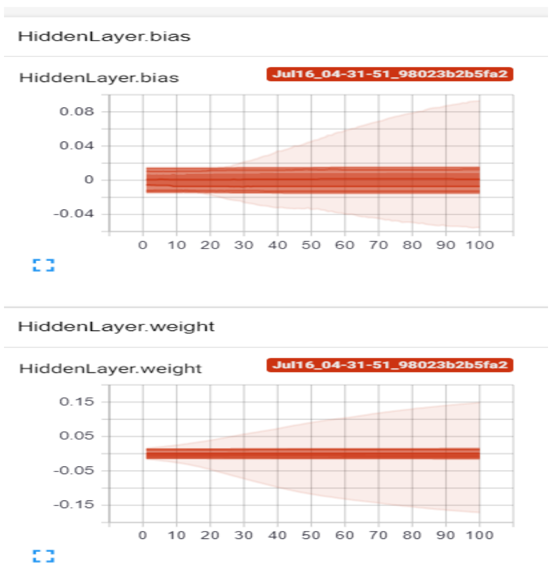
Testing Accuracy:

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



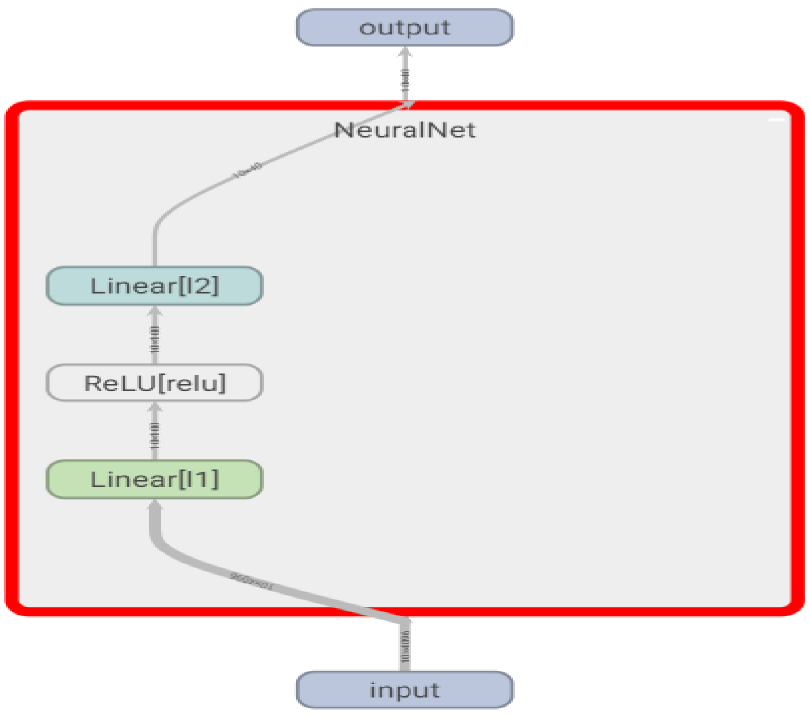
Histogram and distribution of weights and bias:





Graph:

Input Layer – relu(Hidden Layer)– Output Layer



Exercise2:

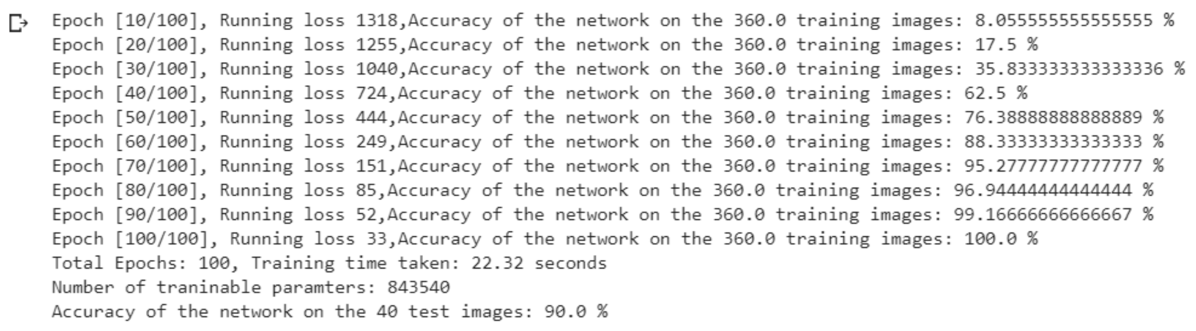
Here the code is almost same. I have already described the code above. I have only added a hidden layer in neural network.

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



```
Epoch [10/100], Running loss 1318,Accuracy of the network on the 360.0 training images: 8.055555555555555 %
Epoch [20/100], Running loss 1255,Accuracy of the network on the 360.0 training images: 17.5 %
Epoch [30/100], Running loss 1040,Accuracy of the network on the 360.0 training images: 35.833333333333336 %
Epoch [40/100], Running loss 724,Accuracy of the network on the 360.0 training images: 62.5 %
Epoch [50/100], Running loss 444,Accuracy of the network on the 360.0 training images: 76.38888888888889 %
Epoch [60/100], Running loss 249,Accuracy of the network on the 360.0 training images: 88.33333333333333 %
Epoch [70/100], Running loss 151,Accuracy of the network on the 360.0 training images: 95.27777777777777 %
Epoch [80/100], Running loss 85,Accuracy of the network on the 360.0 training images: 96.94444444444444 %
Epoch [90/100], Running loss 52,Accuracy of the network on the 360.0 training images: 99.16666666666667 %
Epoch [100/100], Running loss 33,Accuracy of the network on the 360.0 training images: 100.0 %
Total Epochs: 100, Training time taken: 22.32 seconds
Number of traninable paramters: 843540
Accuracy of the network on the 40 test images: 90.0 %
```

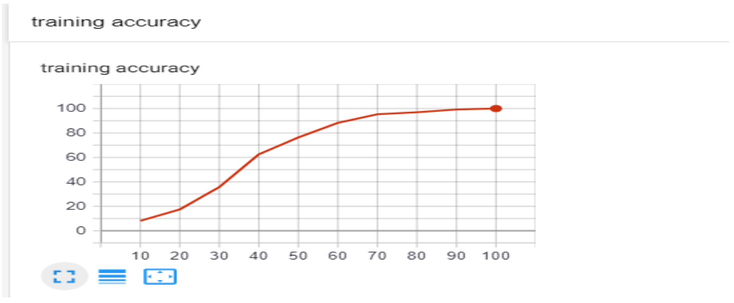
Training time: 22.32 seconds

Number of Trainable parameter's if 100 epochs run: 843540

Accuracy on test images: 90%

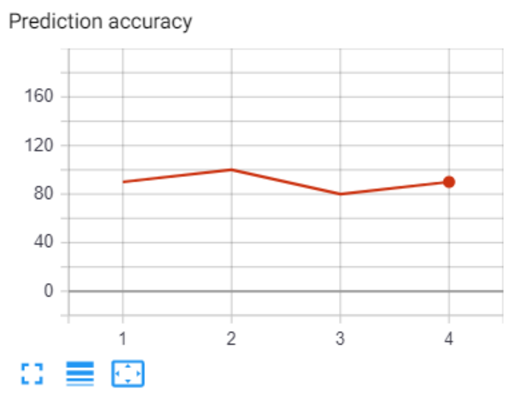
Training accuracy:

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

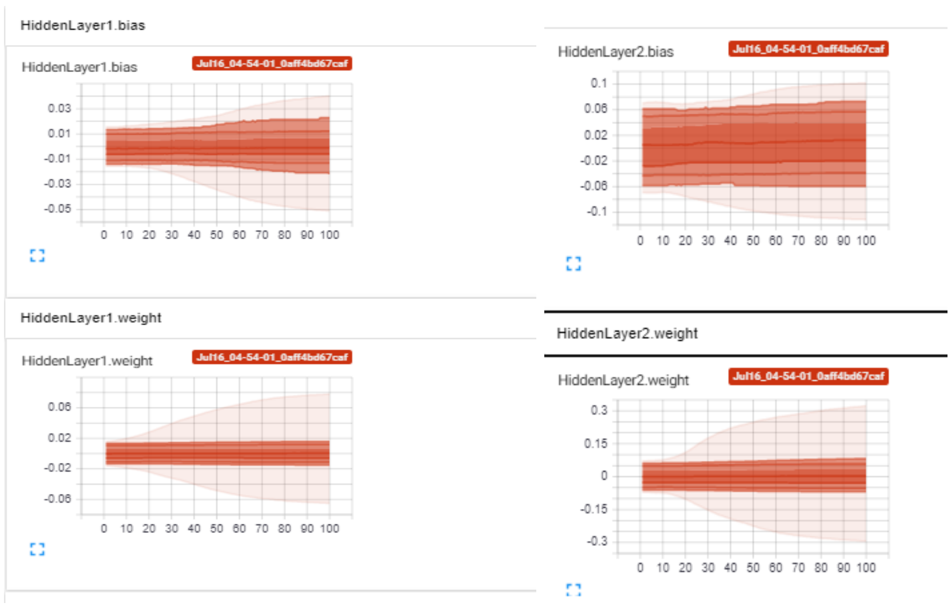


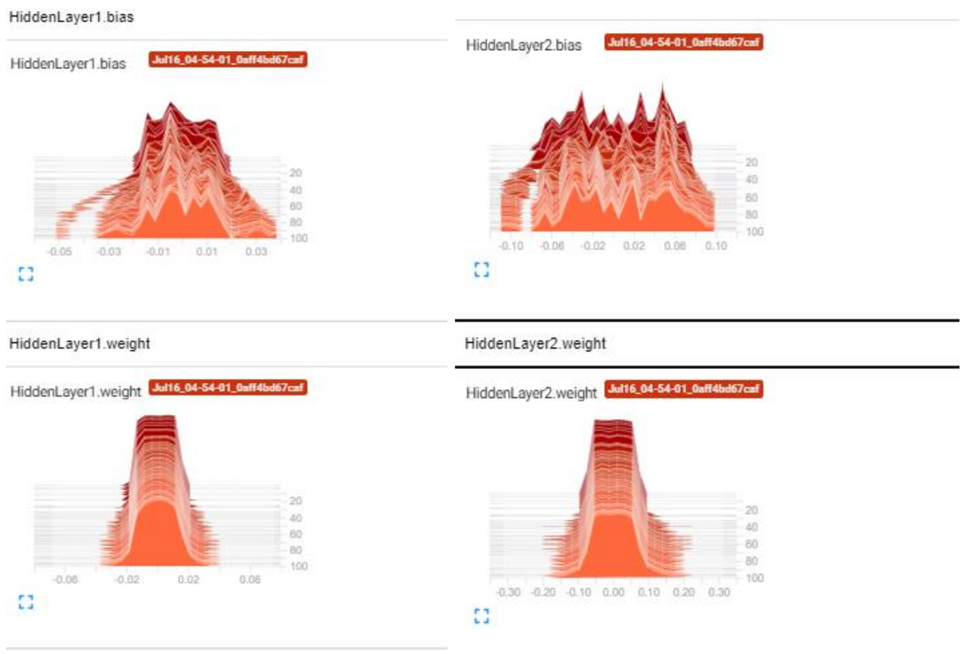
Testing Accuracy:

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

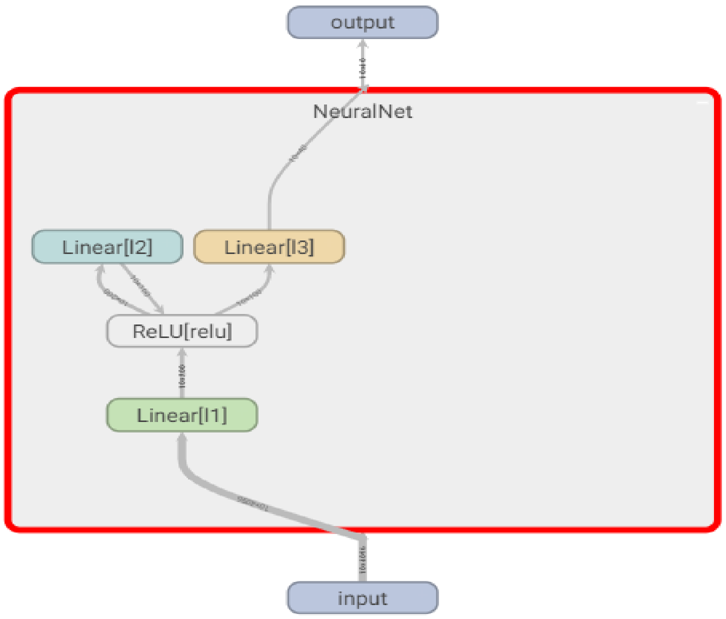


Distribution and histogramof weights and bias:




HiddenLayer1.bias
HiddenLayer1.bias
Jul16_04-54-01_0aff4bd67caf
HiddenLayer2.bias
Jul16_04-54-01_0aff4bd67caf
HiddenLayer1.weight
HiddenLayer1.weight
Jul16_04-54-01_0aff4bd67caf
HiddenLayer2.weight
HiddenLayer2.weight
Jul16_04-54-01_0aff4bd67caf

Graph:


output
NeuralNet
Linear[l2]
Linear[l3]
ReLU[relu]
Linear[l1]
input

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:

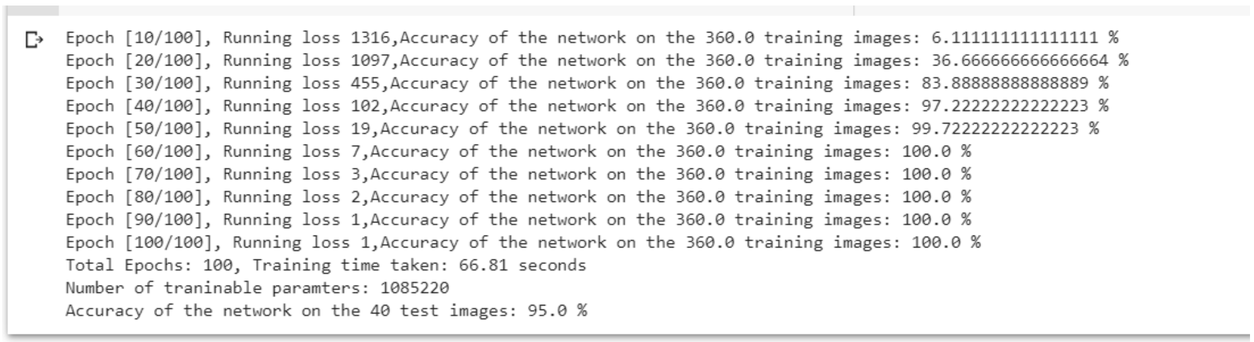
Here the code is almost same. I have already described the code above. But here the structure of neural network is changed. I have just made changes in “NeuralNet” class to get the required structure.

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



```
Epoch [10/100], Running loss 1316,Accuracy of the network on the 360.0 training images: 6.111111111111111 %
Epoch [20/100], Running loss 1097,Accuracy of the network on the 360.0 training images: 36.666666666666664 %
Epoch [30/100], Running loss 455,Accuracy of the network on the 360.0 training images: 83.88888888888889 %
Epoch [40/100], Running loss 102,Accuracy of the network on the 360.0 training images: 97.22222222222223 %
Epoch [50/100], Running loss 19,Accuracy of the network on the 360.0 training images: 99.72222222222223 %
Epoch [60/100], Running loss 7,Accuracy of the network on the 360.0 training images: 100.0 %
Epoch [70/100], Running loss 3,Accuracy of the network on the 360.0 training images: 100.0 %
Epoch [80/100], Running loss 2,Accuracy of the network on the 360.0 training images: 100.0 %
Epoch [90/100], Running loss 1,Accuracy of the network on the 360.0 training images: 100.0 %
Epoch [100/100], Running loss 1,Accuracy of the network on the 360.0 training images: 100.0 %
Total Epochs: 100, Training time taken: 66.81 seconds
Number of traninable paramters: 1085220
Accuracy of the network on the 40 test images: 95.0 %
```

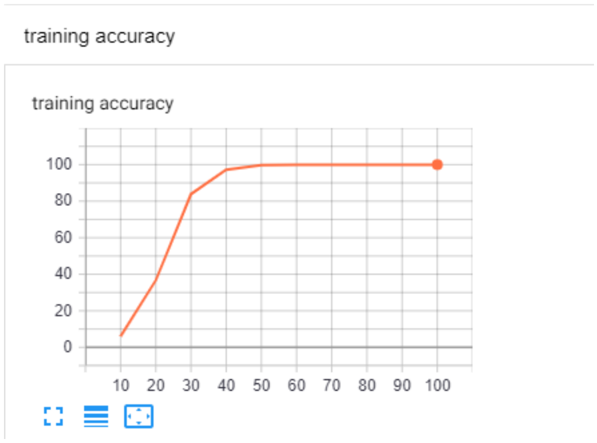
Training time: 66.81 seconds

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

Accuracy on test images: 95%

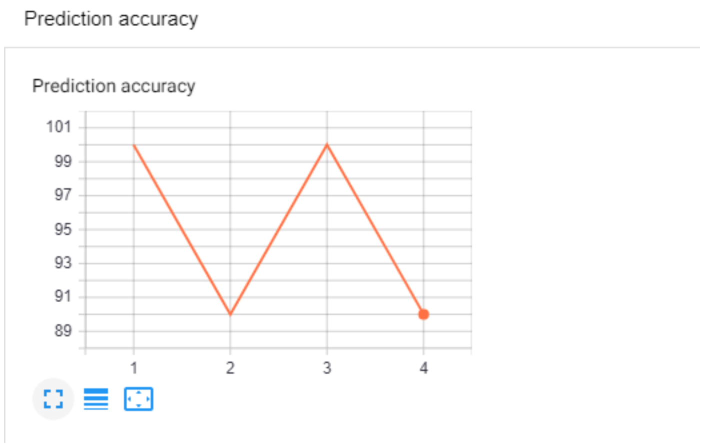
Training accuracy:

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

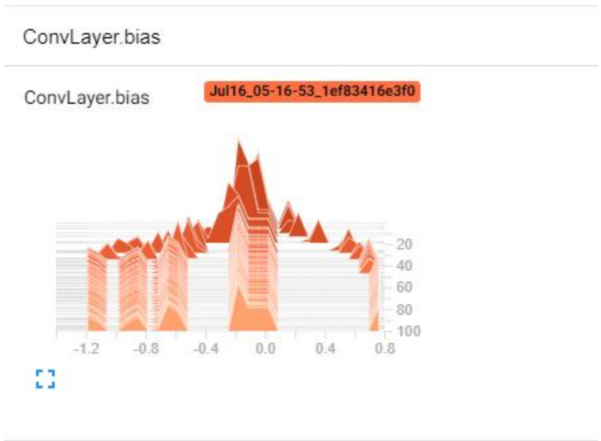


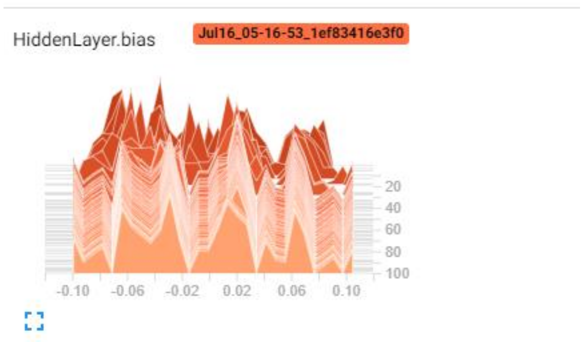
Testing Accuracy:

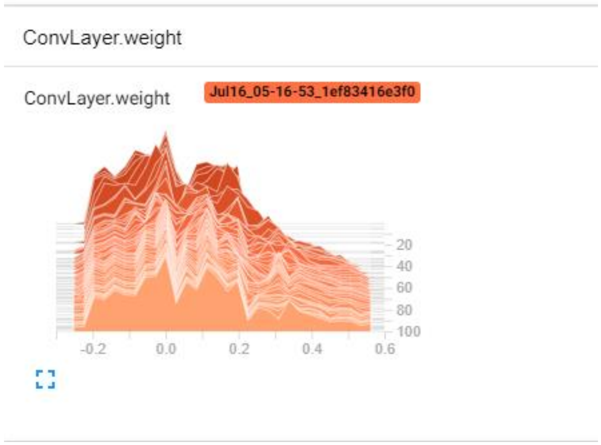
Here x-axis is batch number from total batch of images in test dataset and y-axis is accuracy in that batch.

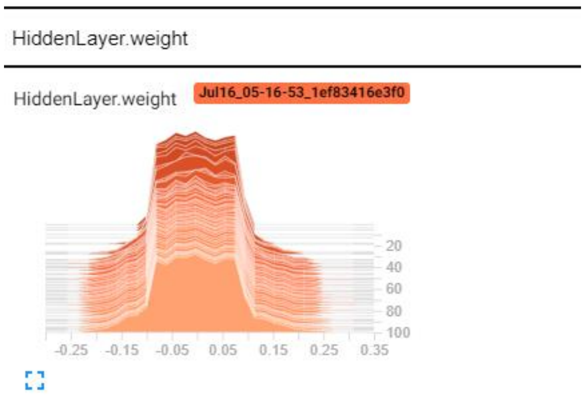


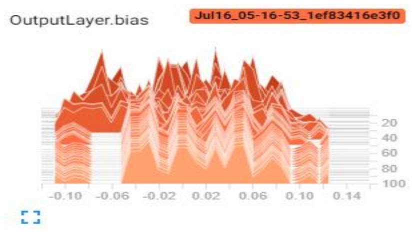
Histogram of weights and bias:

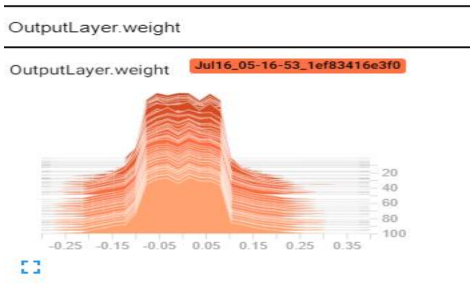




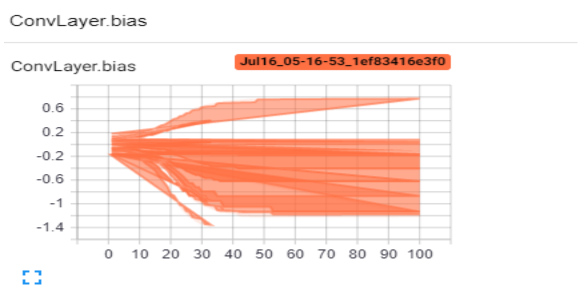


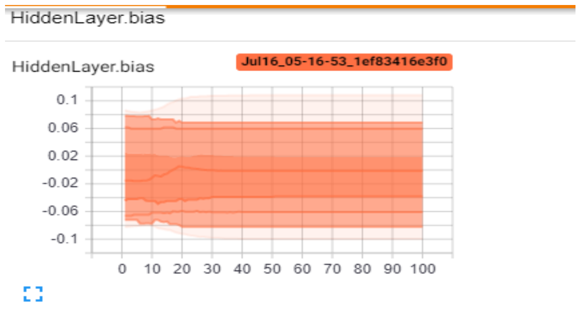


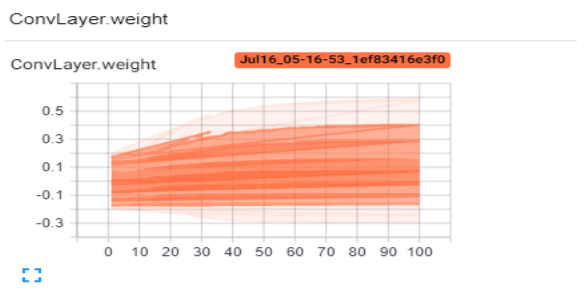


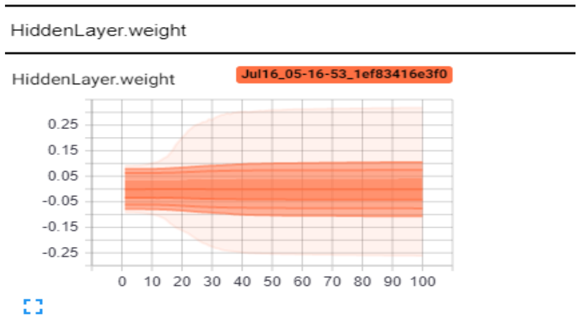


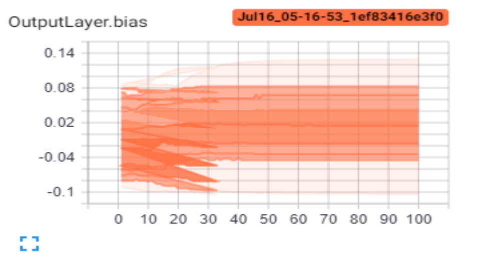
Distribution of weights and bias:

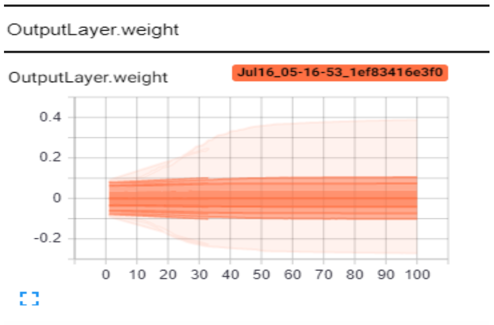




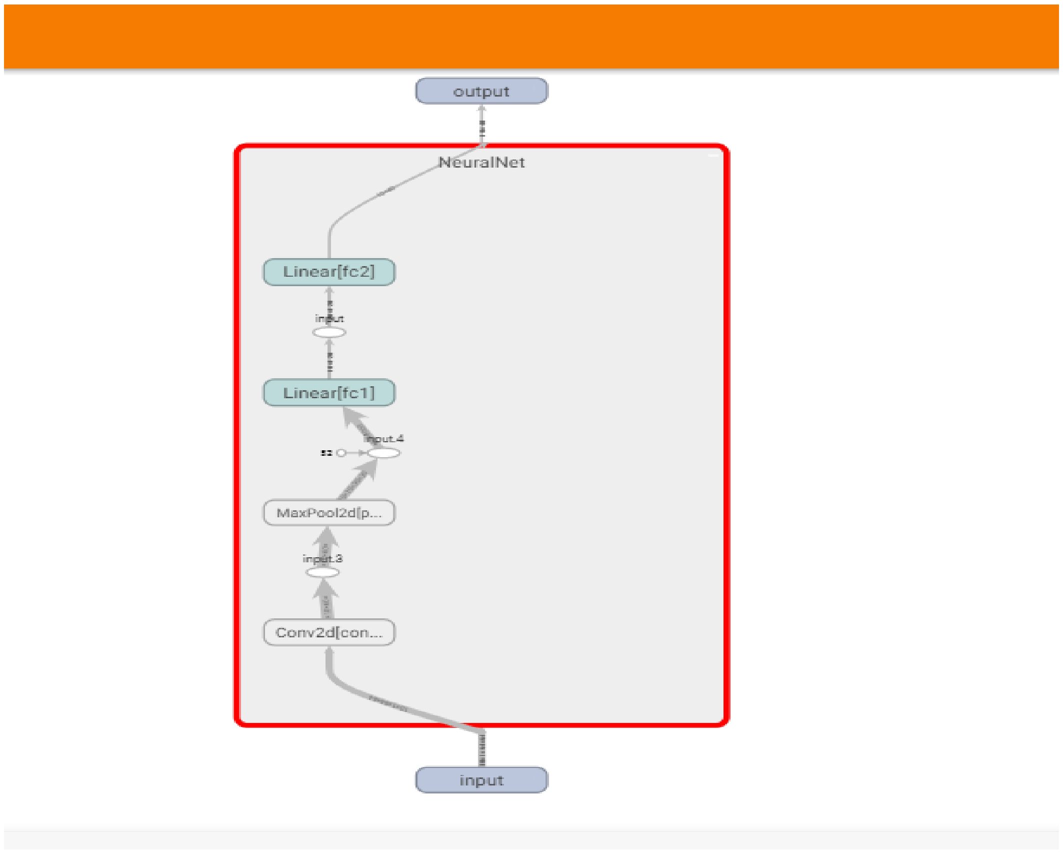








Graph:



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.

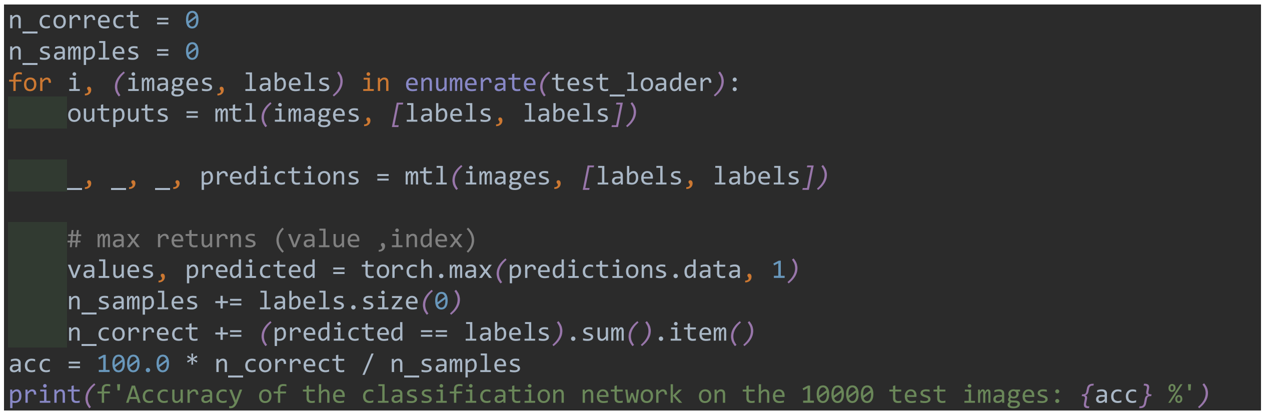
```
class MultiTaskLoss(nn.Module):
    def __init__(self, model, loss_fn):
        super(MultiTaskLoss, self).__init__()
        self.model = model
        self.loss_fn = loss_fn

    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.

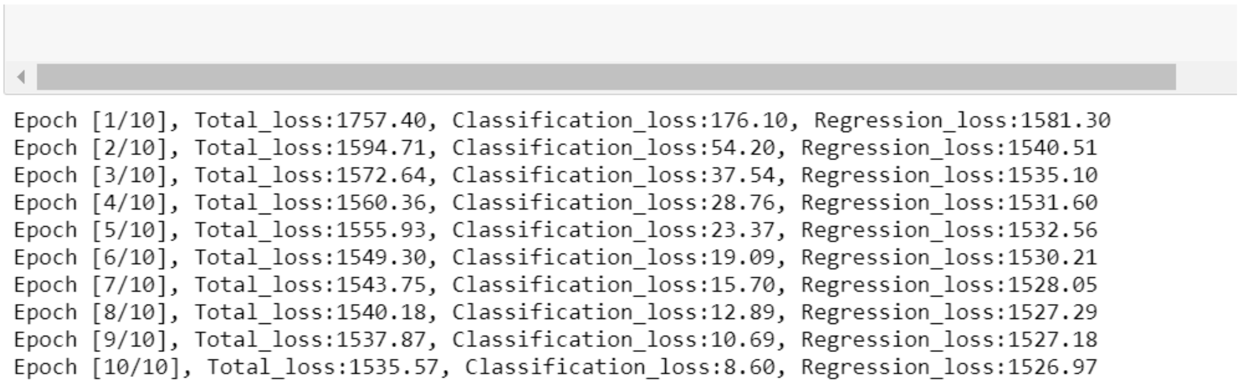


```python
n_correct = 0
n_samples = 0
for i, (images, labels) in enumerate(test_loader):
    outputs = mtl(images, [labels, labels])

    _, _, _, 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:

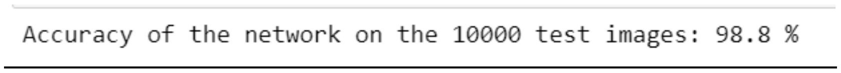
I have run only 10 epochs to plot graph because it was taking a lot of time for more epochs.

On Training dataset:



```
Epoch [1/10], Total_loss:1757.40, Classification_loss:176.10, Regression_loss:1581.30
Epoch [2/10], Total_loss:1594.71, Classification_loss:54.20, Regression_loss:1540.51
Epoch [3/10], Total_loss:1572.64, Classification_loss:37.54, Regression_loss:1535.10
Epoch [4/10], Total_loss:1560.36, Classification_loss:28.76, Regression_loss:1531.60
Epoch [5/10], Total_loss:1555.93, Classification_loss:23.37, Regression_loss:1532.56
Epoch [6/10], Total_loss:1549.30, Classification_loss:19.09, Regression_loss:1530.21
Epoch [7/10], Total_loss:1543.75, Classification_loss:15.70, Regression_loss:1528.05
Epoch [8/10], Total_loss:1540.18, Classification_loss:12.89, Regression_loss:1527.29
Epoch [9/10], Total_loss:1537.87, Classification_loss:10.69, Regression_loss:1527.18
Epoch [10/10], Total_loss:1535.57, Classification_loss:8.60, Regression_loss:1526.97
```

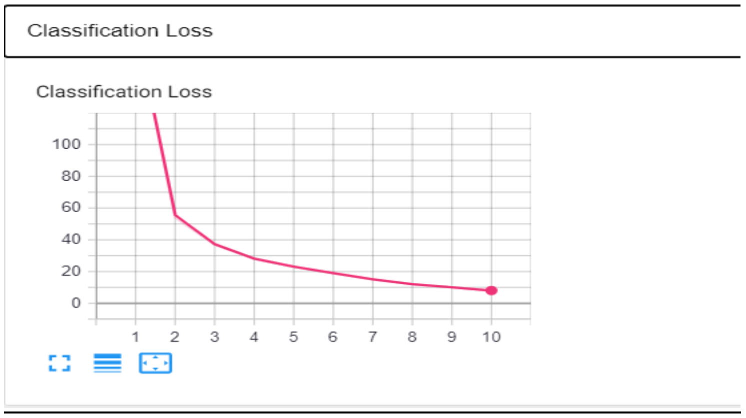
On Testing dataset using classification head:



```
Accuracy of the network on the 10000 test images: 98.8 %
```

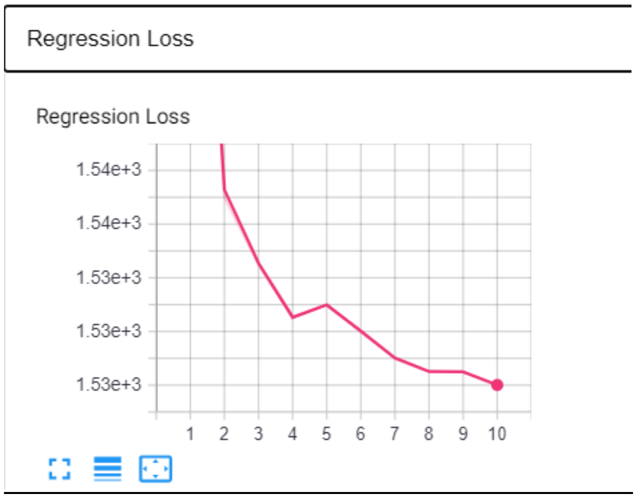
loss of classification on training dataset:

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.



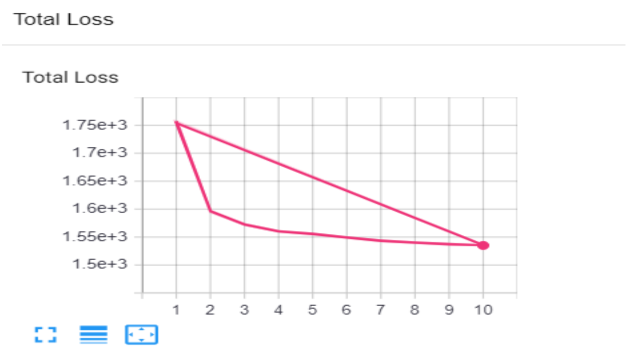
loss for regression on training dataset:

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.



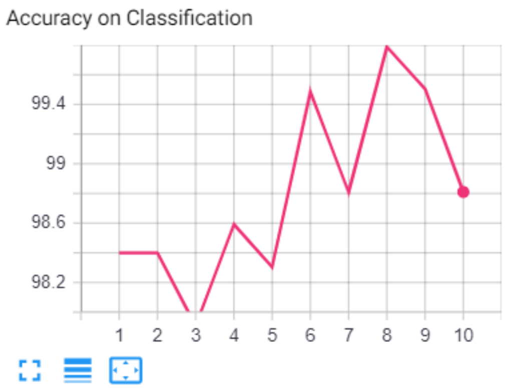
Total loss on training dataset:

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.



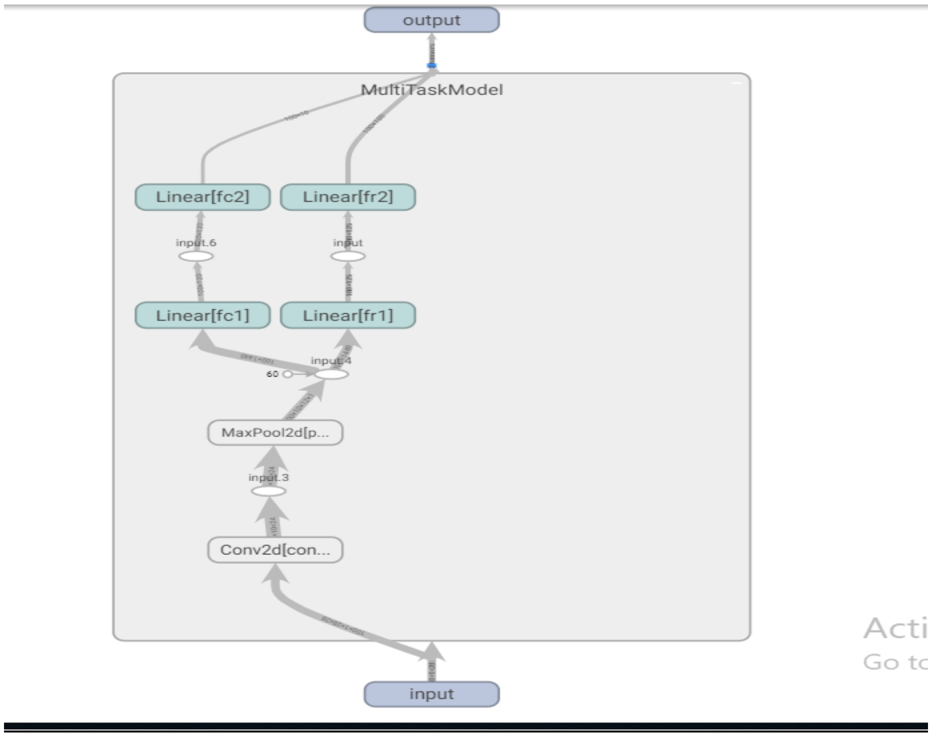
Accuracy for classification:

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.


output
MultiTaskModel
Linear[fc2]
Linear[fr2]
input.6
input
Linear[fc1]
Linear[fr1]
input.4
60
MaxPool2d[p...
input.3
Conv2d[con...
input
Acti
Go tc