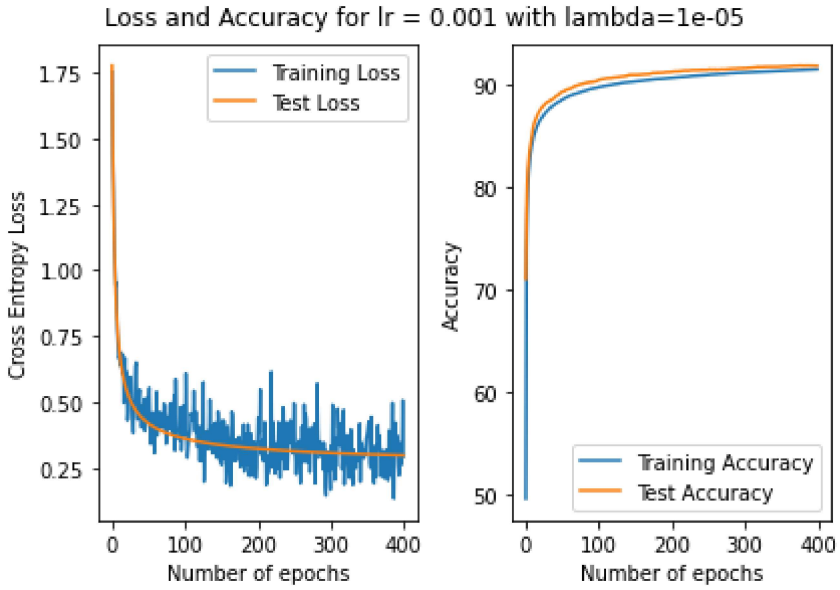
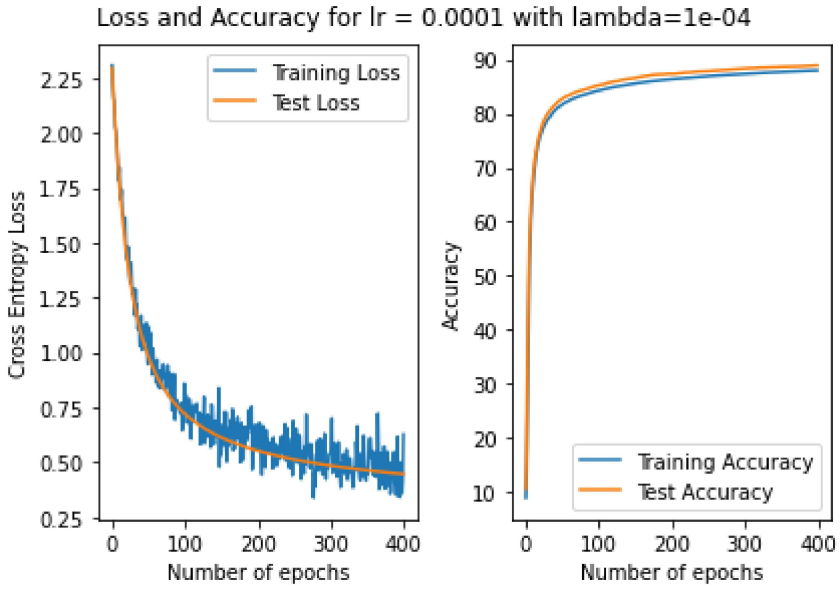
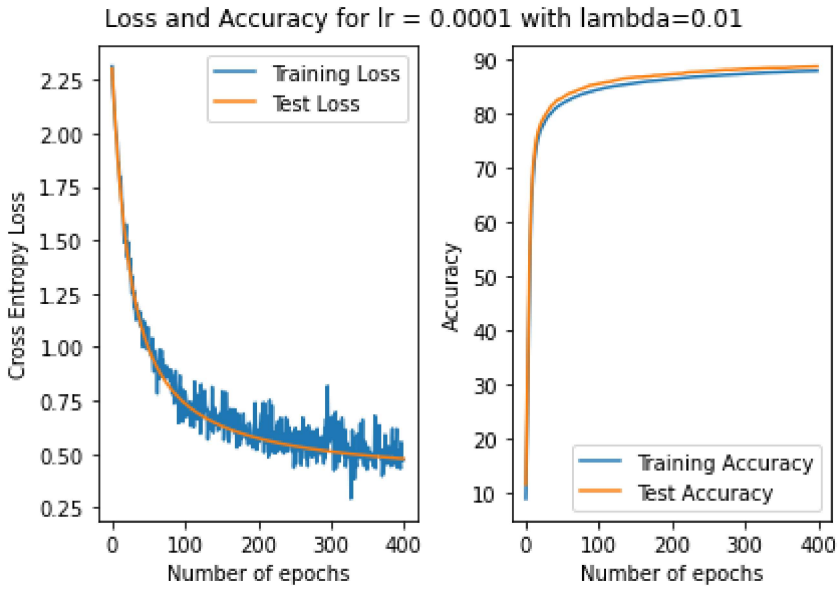
Name: Venkata Meghana Achanta

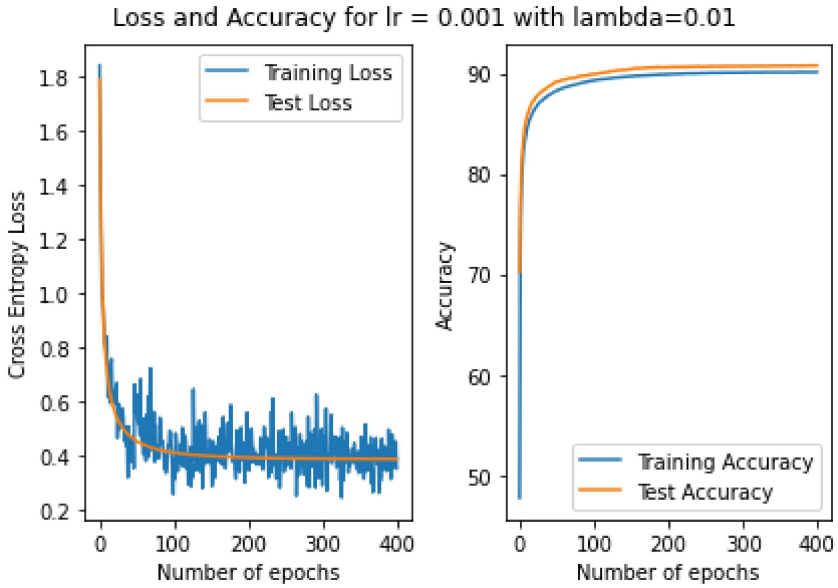
USC ID: 2578990261

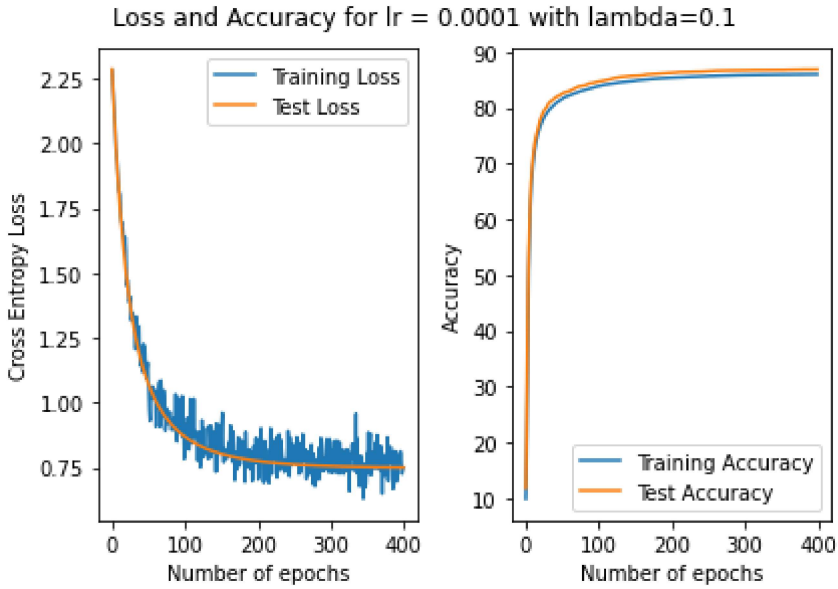
Q1. To design a single fully connected Neural Network using nn.Sequential, I used 2 different learning rates and 4 different weight-decay parameters to decide which will be the best learning rate.

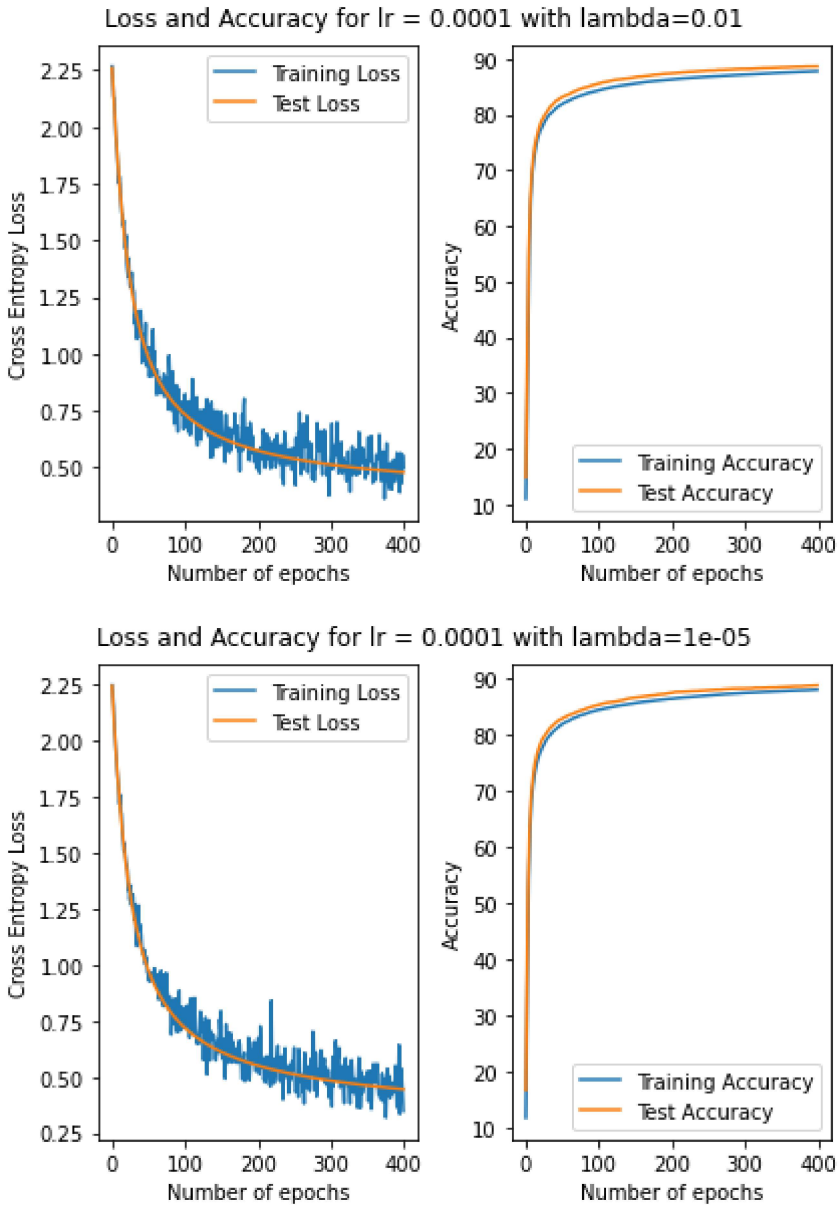




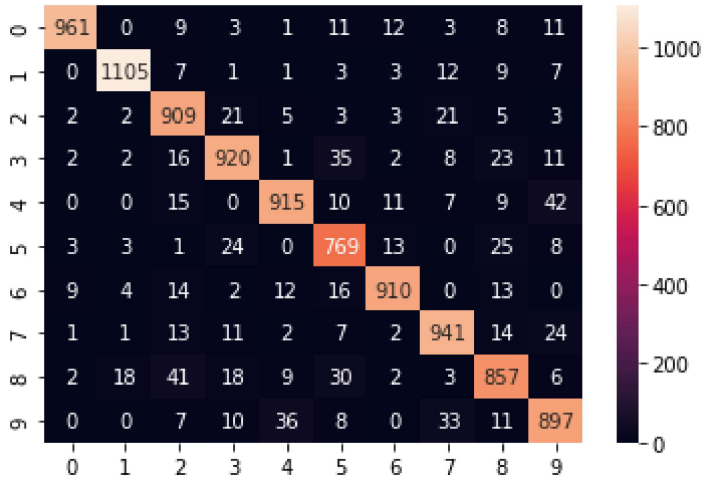

Loss and Accuracy for lr = 0.0001 with lambda=0.01
Training Loss
Test Loss
Cross Entropy Loss
Number of epochs
Training Accuracy
Test Accuracy
Accuracy
Number of epochs


Loss and Accuracy for lr = 0.001 with lambda=0.01
Training Loss
Test Loss
Cross Entropy Loss
Number of epochs
Training Accuracy
Test Accuracy
Accuracy
Number of epochs

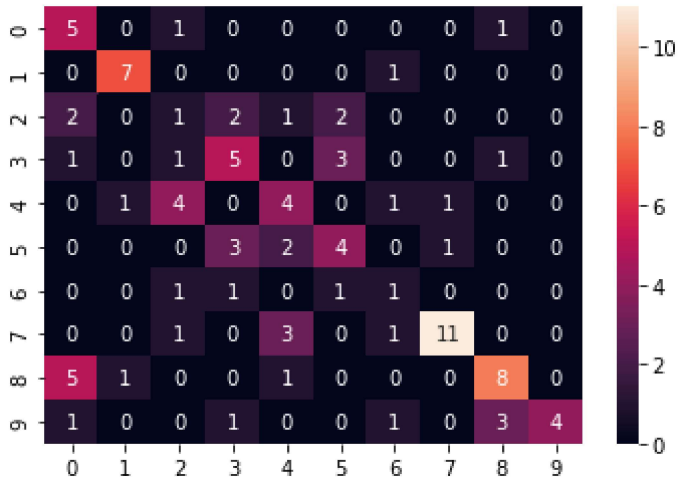

Loss and Accuracy for lr = 0.0001 with lambda=0.1
Training Loss
Test Loss
Cross Entropy Loss
Number of epochs
Training Accuracy
Test Accuracy
Accuracy
Number of epochs



After observing the graphs, I chose the final learning rate as 0.001 with weight decay as 1e-05. The logloss curve for training set fluctuates a bit as the Neural Network is just a fully connected layer. The training and test set accuracy for this network is about 91.63%. The heatmap of the confusion matrix obtained for the network is as follows:



Q2. i. The heat map of the confusion matrix for CIFAR10 dataset is shown below with a learning rate of 0.01, weight decay of 0.0001 and 30% dropout:



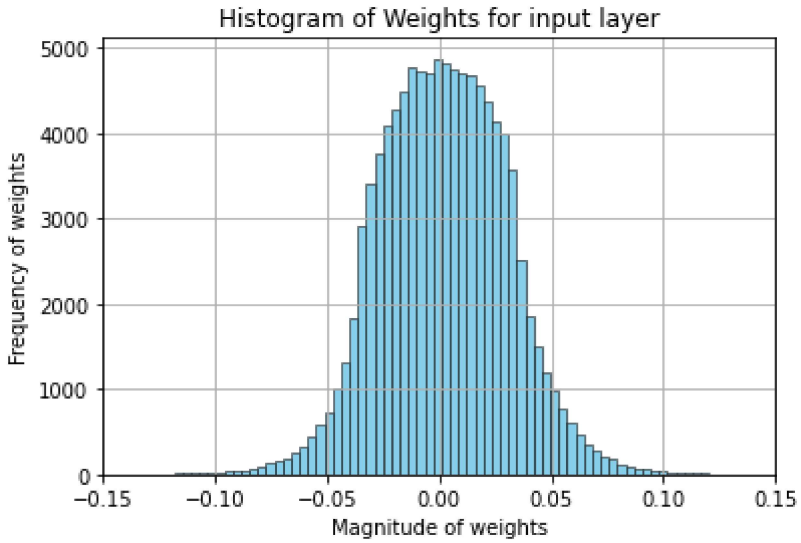
ii. Based on the confusion matrix, I observed the following about each of the classes of CIFAR10 dataset.

1. Class 0- Airplanes-- The class airplane got most confused with class bird and class ship.
2. Class 1- Cars-- The class Car got most confused with class frog.
3. Class 2- Birds-- The class Bird got frequently confused with class airplane, cats or dogs.
4. Class 3- Cats-- The class Cat got most confused with class Dog.
5. Class 4- Deer-- The class Deer got most confused with class Bird.
6. Class 5- Dogs-- The class Dog got most confused with class Cat and class deer.
7. Class 6- Frogs-- The class Frog got confused with class Dog, Cat and Bird.
8. Class 7- Horses-- The class Horse got confused with class Deer.
9. Class 8- Ships-- The class Ships got confused with class airplanes.
10. Class 9- Trucks-- The class Trucks got confused with class ships.

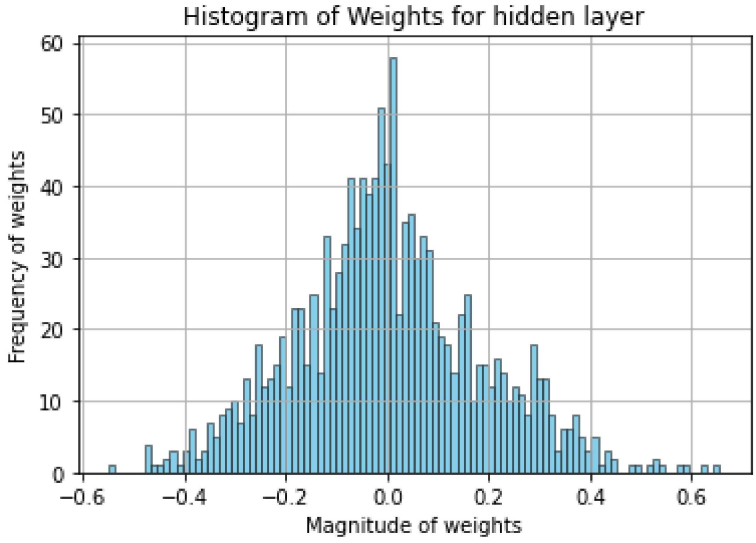
iii. The two classes that got confused overall are class deer and class dog.

Q3. The plots of histogram for the first scenario wherein we used 128 nodes and no dropout and no regularization.

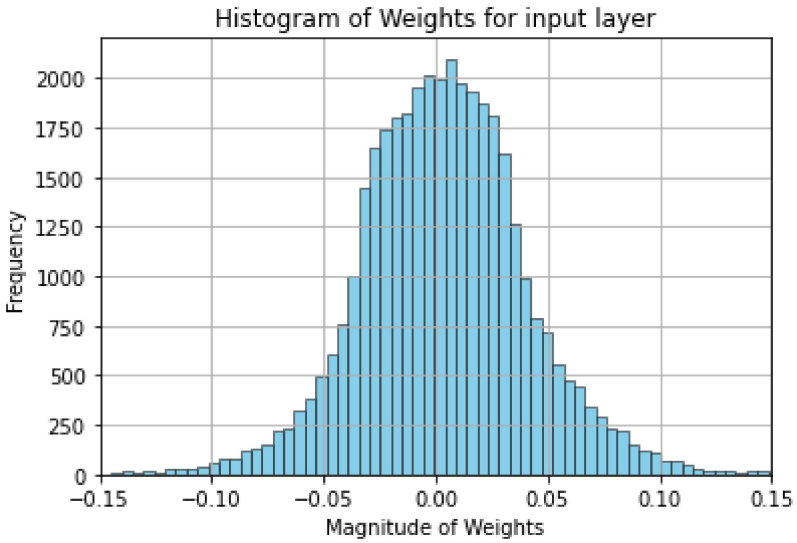
Histogram of Weights of input layer:



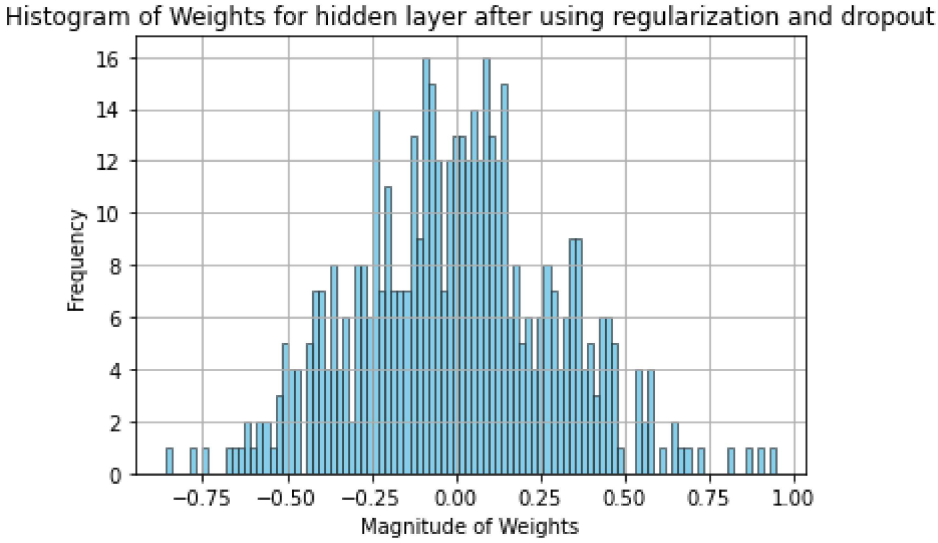
Histogram of Weights of the hidden layer(ReLU activation with no dropout or no regularization).



Histogram of Weights of input layer:



Histogram of Weights of Hidden Layer after using dropout of 0.2 and using weight decay of 0.0001 for L2 regularization.



After observing the histograms for both the models, the magnitude of weights obtained in the case where dropout and regularization were used have a higher magnitude when compared with the magnitude of weights obtained with no dropout or regularization.

Adding to that, the accuracy obtained for the first case with 128 nodes is almost similar to the accuracy obtained in the second case with 48 nodes.

Appendix:

```
#Q1

import torch
import torch.nn as nn
```

```
import torchvision
import torchvision.transforms as transforms
import torch.nn.functional as F
import matplotlib.pyplot as plt
import numpy as np

from pathlib import Path

import h5py
import torch
from torch.utils import data

class HDF5Dataset(data.Dataset):
  """Abstract HDF5 dataset

  Usage:
    from hdf5_dataset import HDF5Dataset
    train_set = hdf5_dataset.HDF5Dataset(
      file_path = f"{PATH_TO_HDF5}", data_name = 'xdata', label_name = 'ydata')
    train_loader = torch.utils.data.DataLoader(train_set, batch_size=100, shuffle=True)

    ** NOTE: labels is 1-hot encoding, not target label number (as in PyTorch MNIST dataset)
    ** keep in mind when comparing model output to target labels

  Input params:
    file_path: Path to the folder containing the dataset
    data_name: name of hd5_file "x" dataset
    data_name: name of hd5_file "y"
  """
  def __init__(self, file_path, data_name, label_name):
    super().__init__()
    self.data = {}

    self.data_name = data_name
    self.label_name = label_name

    h5dataset_fp = Path(file_path)
    assert(h5dataset_fp.is_file())

    with h5py.File(file_path) as h5_file:
      # iterate datasets
      for dname, ds in h5_file.items():
        self.data[dname] = ds[()]


  def __getitem__(self, index):
    # get data
    x = self.data[self.data_name][index]
    x = torch.from_numpy(x)

    # get label
```

```
        y = self.data[self.label_name][index]
        y = torch.from_numpy(y)
        return (x, y)


      def __len__(self):
        return len(self.data[self.data_name])

    #Defining the train_loader and test_loader

    train_set = HDF5Dataset(file_path = "/content/mnist_traindata.hdf5", data_name = 'xdata', lab
    train_loader = torch.utils.data.DataLoader(train_set, batch_size=100, shuffle=True)

    test_set = HDF5Dataset(file_path = "/content/mnist_testdata.hdf5", data_name = 'xdata', label
    test_loader = torch.utils.data.DataLoader(test_set, batch_size = test_set.__len__())

    #Defining the Model

    num_pixels = 28*28

    model = torch.nn.Sequential(nn.Linear(in_features=num_pixels, out_features=10))
    loss_fn = nn.CrossEntropyLoss()
    optimiser = torch.optim.SGD(model.parameters(), lr= 0.001, weight_decay = 1e-05)

    num_epochs = 400

    #Training the model

    count = 0
    accuracy_train_list = []
    accuracy_test_list = []
    loss_train_list = []
    loss_test_list = []
    y_train = []
    y_true = []
    for epoch in range(num_epochs):
      correct = 0
      for x_batch,y_batch in train_loader:
        count += 1
        x = x_batch

        y_hat = model(x)
        loss_train = loss_fn(y_hat,y_batch)

        optimiser.zero_grad()
        loss_train.backward()

        optimiser.step()

        predictions = torch.max(y_hat, 1)[1]
        predictions_batch = torch.max(y_batch, 1)[1]
        correct += (predictions_batch == predictions).sum().numpy()
```

```
      y_train.append(y_hat)
      y_true.append(y_batch)
    with torch.no_grad():
      total_test = 0
      correct_test = 0

      for x_test,y_test in test_loader:
        model.eval()
        output = model(x_test)
        prediction = torch.max(output, 1)[1]
        prediction_test = torch.max(y_test, 1)[1]
        loss_test = loss_fn(output,y_test)
        correct_test += (prediction == prediction_test).sum().numpy()
        total_test += len(y_test)

      accuracy_test = correct_test*100/total_test


    loss_train_list.append(loss_train.data)
    loss_test_list.append(loss_test.data)
    accuracy_train_list.append(100*correct/len(train_loader.dataset))
    accuracy_test_list.append(accuracy_test)


    print(f'Epoch: {epoch+1:02d}, Iteration: {count: 5d}, Loss: {loss_train.data:.4f}, ' + f'Ac

  print('Completed')


  #Plotting the curves for training and test set for logloss and accuracy

  plt.subplot(1,2,1)
  plt.plot(loss_train_list)
  plt.plot(loss_test_list)

  plt.xlabel('Number of epochs')
  plt.ylabel('Cross Entropy Loss')
  #plt.title('Number of epochs vs Cross Entropy Loss')
  plt.legend(['Training Loss','Test Loss'])

  plt.subplot(1,2,2)
  plt.plot(accuracy_train_list)
  plt.plot(accuracy_test_list)

  plt.xlabel('Number of epochs')
  plt.ylabel('Accuracy')
  #plt.title('Number of epochs vs Accuracy')
  plt.legend(['Training Accuracy','Test Accuracy'])

  plt.tight_layout(1)

  plt.suptitle('Loss and Accuracy for ln = 0.001 with lambda=1e-05', y = 1.01, va = 'center')
```

```
plt.suptitle('Loss and Accuracy for lr = 0.001 with lambda=1e-05 ', y = 1.01, va = 'center')
plt.show()

#Calculating the confusion matrix and printing the heat map

for x_test,y_test in test_loader:
      model.eval()
      output2 = model(x_test)
      prediction2 = torch.max(output2, 1)[1]
      prediction_test2 = torch.max(y_test, 1)[1]

from sklearn.metrics import confusion_matrix
import seaborn as sn

cf_matrix = confusion_matrix(prediction2, prediction_test2)
print(cf_matrix)

sn.heatmap(cf_matrix, fmt='0',annot=True)


#Q2.

import torch
import torch.nn as nn
import torchvision
import torchvision.transforms as transforms
import torch.nn.functional as F
import matplotlib.pyplot as plt
import numpy as np

#Importing Dataset

train_set = torchvision.datasets.CIFAR10(root = "./data", train = True, download = True, tran
test_set = torchvision.datasets.CIFAR10(root = "./data", train = False, download = True, tran

train_loader = torch.utils.data.DataLoader(train_set, batch_size = 100, shuffle = True)
test_loader = torch.utils.data.DataLoader(test_set, batch_size = 100, shuffle = False)

#Defining the model
model = torch.nn.Sequential(nn.Linear(in_features = 3*32*32 , out_features = 256),
                            nn.ReLU(),
                            nn.Dropout(0.3),
                            nn.Linear(in_features = 256, out_features = 128),
                            nn.ReLU(),
                            nn.Dropout(0.3)
                            )

#Defining loss function and optimiser
lossfn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr = 0.01, weight_decay = 0.0001)

num_epochs = 100
```

```
#Training the model
count = 0
accuracy_train_list = []
accuracy_test_list = []
loss_train_list = []
loss_test_list = []
l1_train = []
l2_train = []
y_train = []
y_true = []
for epoch in range(num_epochs):
  correct = 0
  for x_batch,y_batch in train_loader:
    count += 1
    #print(x_batch.shape)
    x = x_batch.reshape(100,3072)
    #print(x.shape)
    y_hat = model(x)
    # l1_train.append(model[0].weight)
    # l2_train.append(model[2].weight)
    loss_train = lossfn(y_hat,y_batch)

    optimizer.zero_grad()
    loss_train.backward()

    optimizer.step()
    #print(y_batch.shape)
    predictions = torch.max(y_hat, 1)[1]
    #predictions_batch = torch.max(y_batch, 1)[1]
    correct += (y_batch == predictions).sum().numpy()
    # y_train.append(y_hat)
    # y_true.append(y_batch)
  with torch.no_grad():
    total_test = 0
    correct_test = 0

    for x_test,y_test in test_loader:
      model.eval()
      output = model(x_test.reshape(100,3072))
      prediction = torch.max(output, 1)[1]
      #prediction_test = torch.max(y_test, 1)[1]
      loss_test = lossfn(output,y_test)
      correct_test += (prediction == y_test).sum().numpy()
      total_test += len(y_test)

    accuracy_test = correct_test*100/total_test


  loss_train_list.append(loss_train.data)
  loss_test_list.append(loss_test.data)
  accuracy_train_list.append(100*correct/len(train_loader.dataset))
```

```
      accuracy_test_list.append(accuracy_test)


      print(f'Epoch: {epoch+1:02d}, Iteration: {count: 5d}, Loss: {loss_train.data:.4f}, ' + f'Ac

    print('Completed')

    #Computing the confusion matrix
    for x_test,y_test in test_loader:
          model.eval()
          output2 = model(x_test.reshape(-1,3072))
          prediction2 = torch.max(output2, 1)[1]

    from sklearn.metrics import confusion_matrix
    import seaborn as sn

    cf_matrix = confusion_matrix(prediction2, y_test)
    print(cf_matrix)

    #Generating heat map of confusion matrix
    sn.heatmap(cf_matrix, fmt='0',annot=True)


    #Q3i

    import torch
    import torch.nn as nn
    import torchvision
    import torchvision.transforms as transforms
    import torch.nn.functional as F
    import matplotlib.pyplot as plt
    import numpy as np

    #importing the dataset

    train_set = torchvision.datasets.FashionMNIST(root = "./data", train = True, download = True,
    test_set = torchvision.datasets.FashionMNIST(root = "./data", train = False, download = True,

    train_loader = torch.utils.data.DataLoader(train_set, batch_size = 100, shuffle = True)
    test_loader = torch.utils.data.DataLoader(test_set, batch_size = 100, shuffle = False)

    #defining the model
    model = torch.nn.Sequential(nn.Linear(in_features = 28*28 , out_features = 128),
                                nn.ReLU(),
                                nn.Linear(in_features = 128, out_features = 10)
                                )
    #Defining the loss function and the optimiser
    lossfn = nn.CrossEntropyLoss()
    optimizer = torch.optim.SGD(model.parameters(), lr = 0.01)

    num_epochs = 40
```

```
#Training the model

count = 0
accuracy_train_list = []
accuracy_test_list = []
loss_train_list = []
loss_test_list = []
l1_train = []
l2_train = []
y_train = []
y_true = []
for epoch in range(num_epochs):
  correct = 0
  for x_batch,y_batch in train_loader:
    count += 1
    x = x_batch.reshape(100,784)
    #print(x.shape)
    y_hat = model(x)
    l1_train.append(model[0].weight)
    l2_train.append(model[2].weight)
    loss_train = lossfn(y_hat,y_batch)

    optimizer.zero_grad()
    loss_train.backward()

    optimizer.step()
    #print(y_batch.shape)
    predictions = torch.max(y_hat, 1)[1]
    #predictions_batch = torch.max(y_batch, 1)[1]
    correct += (y_batch == predictions).sum().numpy()
    # y_train.append(y_hat)
    # y_true.append(y_batch)
  with torch.no_grad():
    total_test = 0
    correct_test = 0

    for x_test,y_test in test_loader:
      model.eval()
      output = model(x_test.reshape(100,784))
      prediction = torch.max(output, 1)[1]
      #prediction_test = torch.max(y_test, 1)[1]
      loss_test = lossfn(output,y_test)
      correct_test += (prediction == y_test).sum().numpy()
      total_test += len(y_test)

    accuracy_test = correct_test*100/total_test


  loss_train_list.append(loss_train.data)
  loss_test_list.append(loss_test.data)
```

```
    accuracy_train_list.append(100*correct/len(train_loader.dataset))
    accuracy_test_list.append(accuracy_test)


    print(f'Epoch: {epoch+1:02d}, Iteration: {count: 5d}, Loss: {loss_train.data:.4f}, ' + f'Ac

print('Completed')

#Plotting the Weights for Input Layer

plt.hist(model[0].weight.detach().numpy().reshape(-1,1), bins = 100, color = 'skyblue', ec =
plt.grid()
plt.xlim([-0.15,0.15])
plt.xlabel('Magnitude of weights')
plt.ylabel('Frequency of weights')
plt.title('Histogram of Weights for input layer')

#Plotting the Weights for Hidden Layer

plt.hist(model[2].weight.detach().numpy().reshape(-1,1), bins = 100, color = 'skyblue', ec =
#plt.xlim([-0.15,0.15])
plt.grid()
plt.xlabel('Magnitude of weights')
plt.ylabel('Frequency of weights')
plt.title('Histogram of Weights for hidden layer')

#Q3.ii

import torch
import torch.nn as nn
import torchvision
import torchvision.transforms as transforms
import torch.nn.functional as F
import matplotlib.pyplot as plt
import numpy as np

#Importing Dataset

train_set = torchvision.datasets.FashionMNIST(root = "./data", train = True, download = True,
test_set = torchvision.datasets.FashionMNIST(root = "./data", train = False, download = True,

train_loader = torch.utils.data.DataLoader(train_set, batch_size = 100, shuffle = True)
test_loader = torch.utils.data.DataLoader(test_set, batch_size = 100, shuffle = False)

#Designing the model with dropout

model = torch.nn.Sequential(nn.Linear(in_features = 28*28 , out_features = 48),
                            nn.ReLU(),
                            nn.Dropout(0.2),
                            nn.Linear(in_features = 48, out_features = 10)
                            )
```

```
#Defining the Loss function and optimiser using weight_decay and learning rate
lossfn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr = 0.01, weight_decay = 0.0001)

num_epochs = 40

#Training the model

count = 0
accuracy_train_list = []
accuracy_test_list = []
loss_train_list = []
loss_test_list = []
l1_train = []
l2_train = []
y_train = []
y_true = []
for epoch in range(num_epochs):
  correct = 0
  for x_batch,y_batch in train_loader:
    count += 1
    x = x_batch.reshape(100,784)
    #print(x.shape)
    y_hat = model(x)
    # l1_train.append(model[0].weight)
    # l2_train.append(model[2].weight)
    loss_train = lossfn(y_hat,y_batch)

    optimizer.zero_grad()
    loss_train.backward()

    optimizer.step()
    #print(y_batch.shape)
    predictions = torch.max(y_hat, 1)[1]
    #predictions_batch = torch.max(y_batch, 1)[1]
    correct += (y_batch == predictions).sum().numpy()
    # y_train.append(y_hat)
    # y_true.append(y_batch)
  with torch.no_grad():
    total_test = 0
    correct_test = 0

    for x_test,y_test in test_loader:
      model.eval()
      output = model(x_test.reshape(100,784))
      prediction = torch.max(output, 1)[1]
      #prediction_test = torch.max(y_test, 1)[1]
      loss_test = lossfn(output,y_test)
      correct_test += (prediction == y_test).sum().numpy()
      total_test += len(y_test)
```

```
      accuracy_test = correct_test*100/total_test


    loss_train_list.append(loss_train.data)
    loss_test_list.append(loss_test.data)
    accuracy_train_list.append(100*correct/len(train_loader.dataset))
    accuracy_test_list.append(accuracy_test)


    print(f'Epoch: {epoch+1:02d}, Iteration: {count: 5d}, Loss: {loss_train.data:.4f}, ' + f'Ac

print('Completed')

#Plotting the Histogram of Weights for Input Layer

plt.hist(model[0].weight.detach().numpy().reshape(-1,1), bins = 100, color = 'skyblue', ec =
plt.xlim([-0.15,0.15])
plt.grid()
plt.xlabel('Magnitude of Weights')
plt.ylabel('Frequency')
plt.title('Histogram of Weights for input layer')

#Plotting the Histogram of Weights for Hidden Layer with Dropout = 0.2 and Regularization = 0

plt.hist(model[3].weight.detach().numpy().reshape(-1,1), bins = 100, color = 'skyblue', ec =
plt.grid()
plt.xlabel('Magnitude of Weights')
plt.ylabel('Frequency')
plt.title('Histogram of Weights for hidden layer after using regularization and dropout')
#plt.xlim([-0.15,0.15])
```

Colab paid products - Cancel contracts here