CSCI 5922 -Neural Networks and Deep Learning
Lab - 1 Solutions

Gowri Shankar Raju Kurapati
Student ID 110568555

September 11, 2022

1 Neural Network Hyperparameters

1.1 Dataset & Hyperparameters Used

- CIFAR 10 Dataset
  - Source : From *torchvision.datasets*
  - The CIFAR-10 dataset used consists of 50000 32x32 color images in 10 classes, with 5000 images per class.
  - Since these are color images, each image has the tensor dimension of 3 * 32 * 32 where the first dimension is RGB colors.
  - The 10 classes (labels) are $airplane(0), automobile(1), bird(2), cat(3), deer(4), dog(5), frog(6), horse(7), ship(8), truck(9)$
- Hyperparameters
  - training/test split is $70 : 30$
  - Number of neurons in each hidden layer is 1024 is held constant for all hidden layers used for the experiment.
  - The input size to the neural network is 3 * 32 * 32 image which is reshaped to a vector of length 3072 and is fed to the network.
  - As stated above, the output layer consists of 10 neurons to predict 10 classes of the CIFAR 10 dataset.
  - The learning rate is set to 0.001 with a batch size of 128 and is held constant for the experiment.
  - The models are trained for 30 epochs.
- Methods:
  - All neural networks are Vanilla Neural Networks (no CNNs Used) with input Layer, hidden Layer, and output Layer with softMax.
  - Loss function used for the experiment is the Cross Entropy.
  - Adam Optimizer with a learning rate of 0.001 is used.

  - As specified, I have considered three activation functions ReLU, Tanh and Sigmoid.
  - Also, I have iterated the experiment with 1,2& 3 hidden layers while also iterating with activation functions as mentioned above, which leaves us with nine neural network models.
    * The models are named as Lab1_P1 _HiddenL_<no of hidden layers>_Activation _<FunctionName> to capture the activation function and the number of hidden layers.

1.2 Results & Analysis

Reporting **Test Accuracies** for the NINE NN Models

ActivationF / Hidden Layers	ReLU	Tanh	Sigmoid
1	49.30	43.96	47.49
2	50.73	40.12	47.62
3	50.21	36.49	45.95

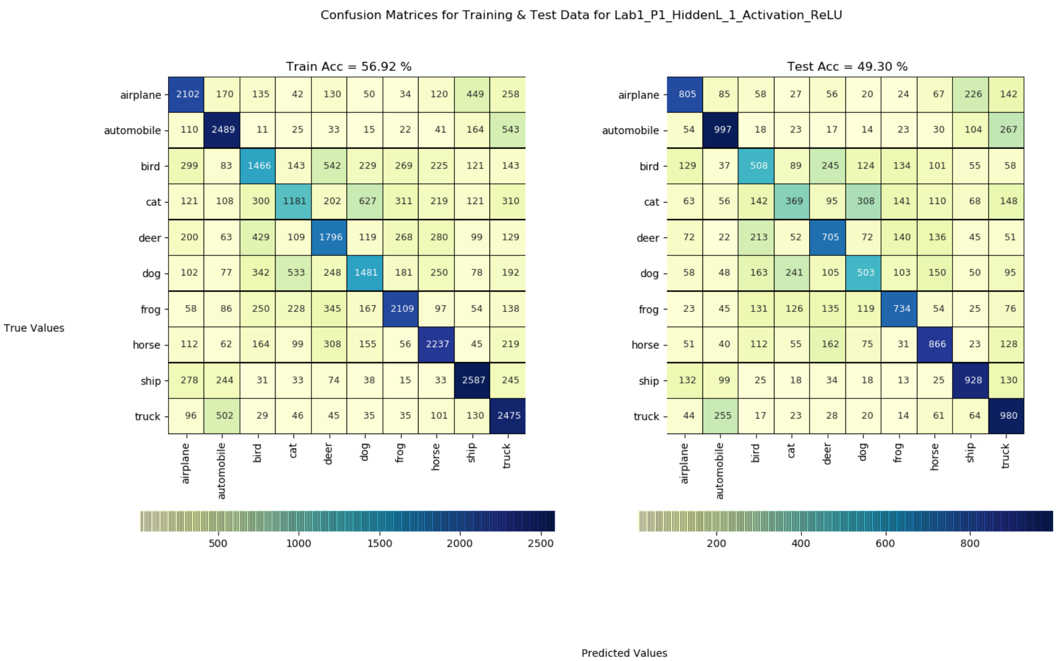
Reporting **Training Accuracies** for the NINE NN Models for comparison

ActivationF / Hidden Layers	ReLU	Tanh	Sigmoid
1	56.92	48.36	55.27
2	64.80	42.63	55.09
3	70.52	38.37	51.37

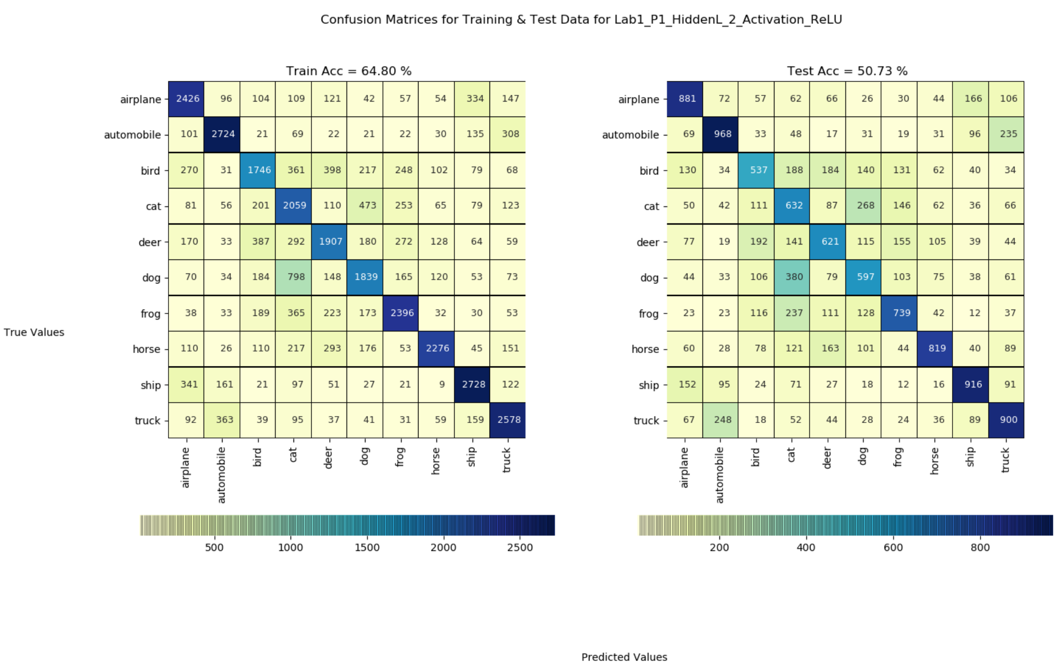
- From the tables above, we could infer that,
  - [**ReLU Performs Better**] From the test accuracies table, ReLU activation gives accuracies better than the Tanh and Sigmoid functions when compared against a configuration of hidden layers. Since it is an image dataset (which is reshaped) the model may be trying to pick up small features from the pixel (like tire, headlights etc) and its surroundings and then try to map them to a bigger feature (like a car). Since ReLU gives zero value before a threshold and a positive value after a threshold, at each neuron in the hidden layers, it might capture whether a small feature exists or not and if it exists, how persistent it is. This analogy is in perfect sync with the ReLU function. This can also be strengthed by the fact of poor accuracies when using continuous functions like tanh and Sigmoid, though sigmoid seems to do better than tanh.
  - [**Overfitting With More Hidden Layers**] When considering the better performing ReLU, we can see that as the number of hidden layers is increased, the training accuracy increases significantly(65 -> 71) but at the same time we can see the testing accuracy decreasing (50.73 -> 50.21). This shows that though the model is performing well on training data with a significant amount of accuracy increase, the test accuracy almost remains stagnant. This is a clear sign of the model trying to overfit the training data. Sigmoid, Tanh activation functions seem to struggle to capture the features from the image pixel data and even the increase in the hidden layers isn't helping the model to perform better, and sometimes the training accuracy itself is going down by underfitting the training data and thereby having an impact on test accuracy as well.

From the confusion matrices above, we could infer that

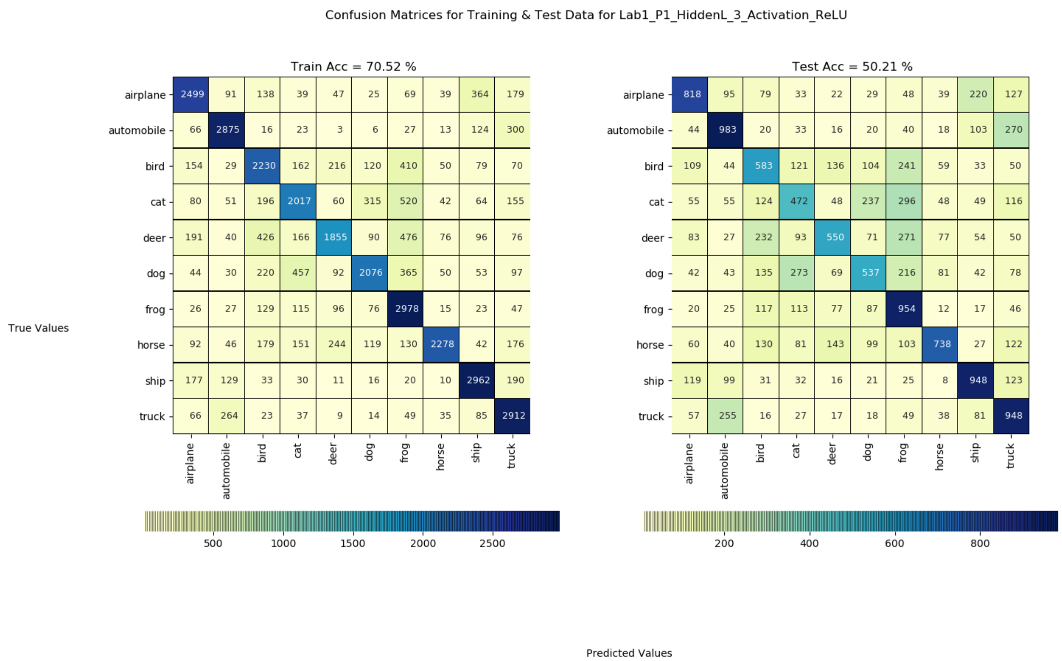
- [**Some Features are not learned**] One common thing to observe from all the 9 pairs of confusion matrices is that,



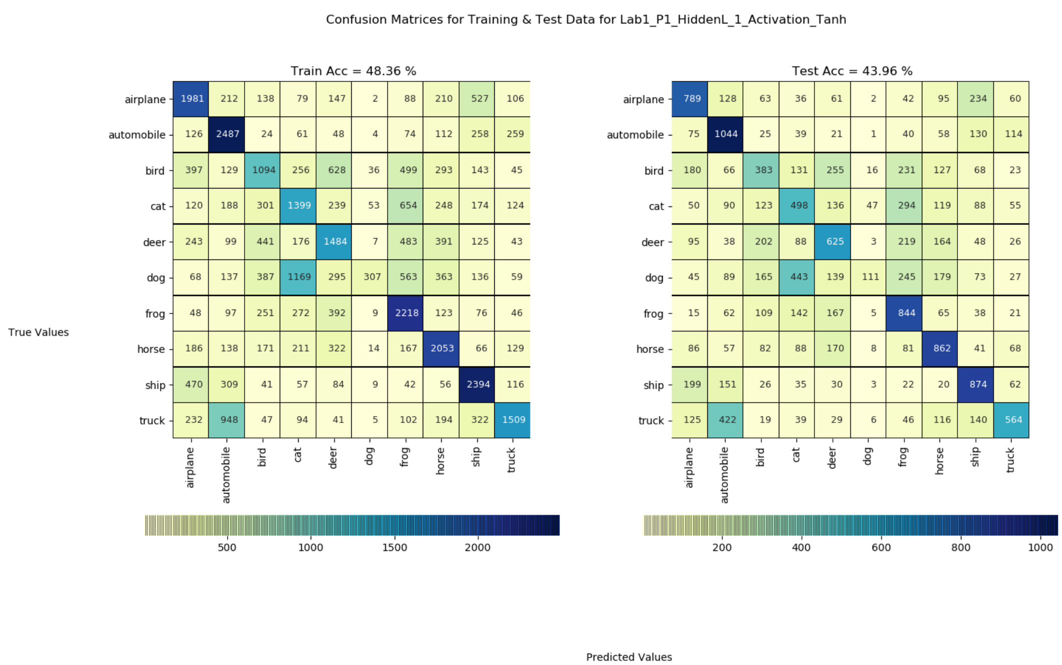
(a) 1 Hidden Layer with ReLU



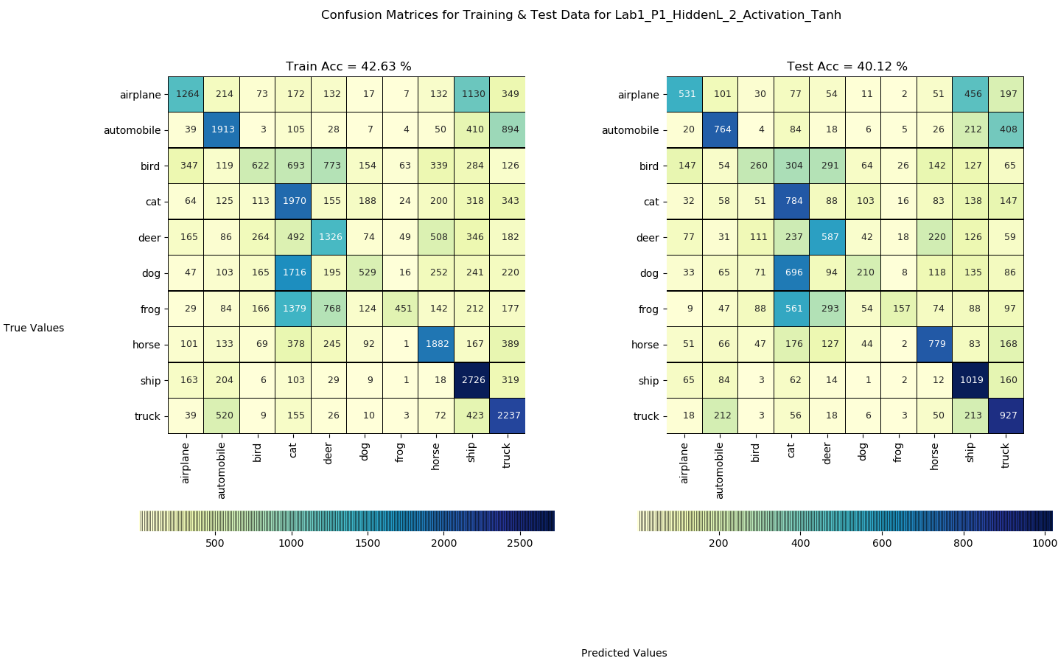
(b) 2 Hidden Layers with ReLU



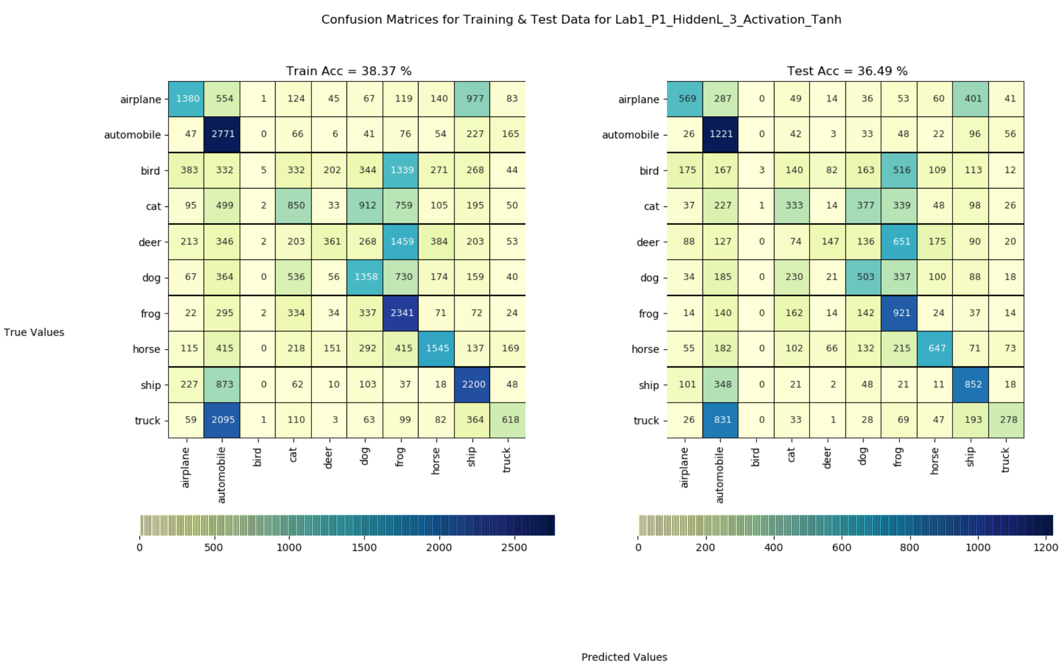
(a) 3 Hidden Layer with ReLU



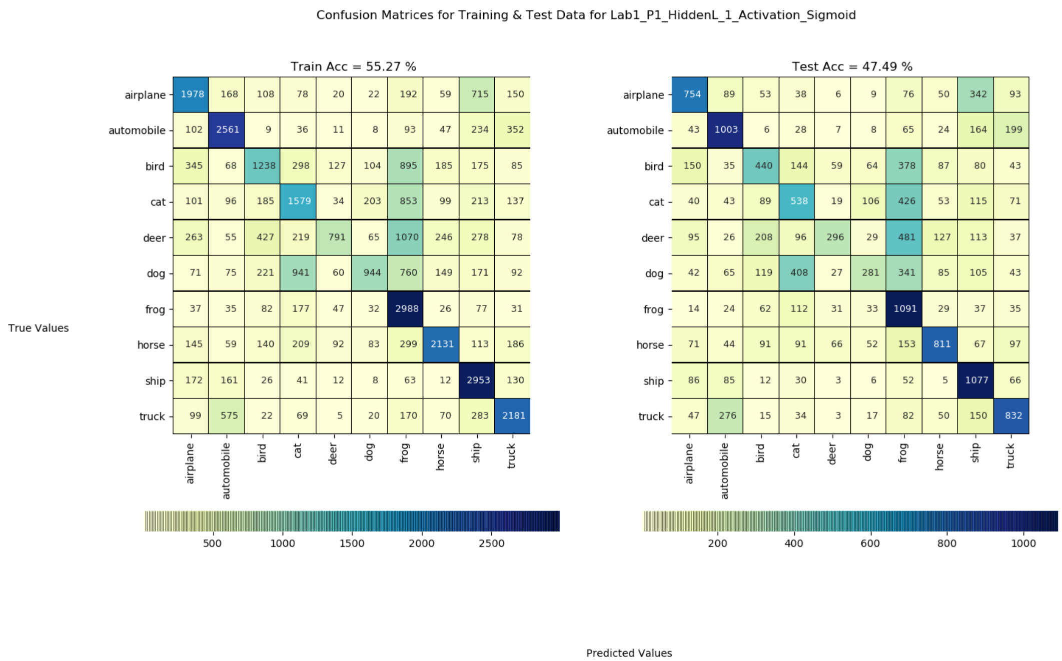
(b) 1 Hidden Layer with Tanh



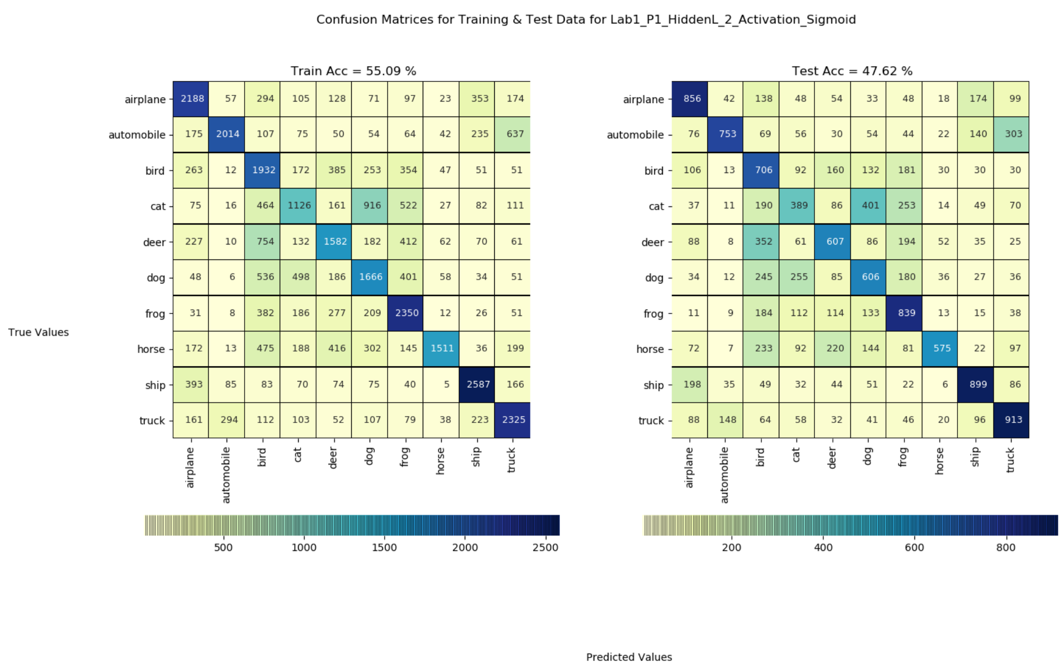
(a) 2 Hidden Layers with Tanh



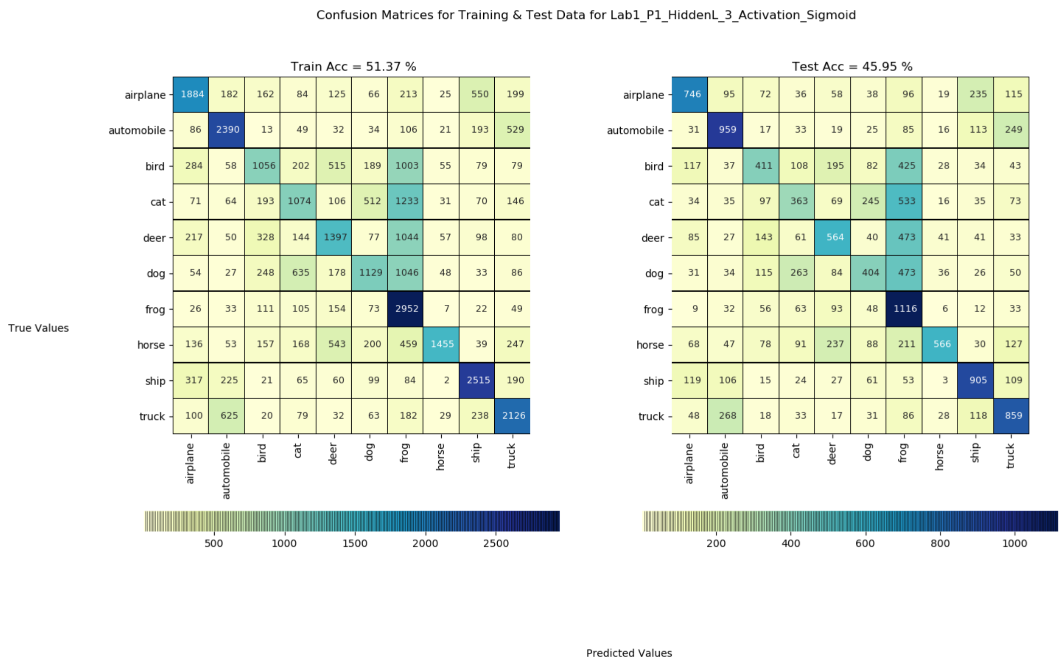
(b) 3 Hidden Layers with Tanh



(a) 1 Hidden Layer with Sigmoid



(b) 2 Hidden Layers with Sigmoid



(a) 3 Hidden Layers with Sigmoid

- When the color for one square cell in the training confusion matrix is compared with the counterpart of the testing-confusion matrix, the color schema almost remains the same. This tells us two things, one being that the random split function that PyTorch uses balances the training dataset and test dataset appropriately, and the density of the classes remain the same. Secondly, the model misclassifies training data points and in approximately the same ratio, it misclassifies the similar classes in the test dataset as well. This proves that the model is unable to learn few features at all despite more hidden layers. These features may be the important distinguishers for two classes and since the model is unable to learn those, the accuracy falls down. (for example, the model fails significantly to distinguish between cats and dogs, frogs and cats) Thus, we can infer that a few features cannot be learned just by increasing the number of hidden layers.
- Also, we can observe that the activation function plays an important role to learn certain features. When the same activation function is used, the model is unable to learn a few features that a model with other activation functions learns. For instance, the model with the tanh activation function strongly misclassifies truck as automobile whereas the models with ReLU & Sigmoid do just fine to distinguish the same classes.

2 Neural Network Hyperparameters

2.1 Dataset & Hyperparameters Used

- Fashion MNIST Dataset
  - Source : From *torchvision.datasets*
  - The Fashion MNIST dataset used consists of 60000 28x28 single layered images in 10 classes, with 6000 images per class.

- The 10 classes (labels) are $T{-}shirt/top(0), Trouser(1), Pullover(2), Dress(3), Coat(4), Sandal(5), Shirt(6),$ $Sneaker(7), Bag(8), Ankleboot(9)$

- Hyperparameters
  - training/test split is $70 : 30$
  - Number of neurons in each hidden layer is 256 is held constant for all hidden layers used for the experiment.
  - The input size to the neural network is 28 * 28 image which is reshaped to a vector of length 784 and is fed to the network.
  - As stated above, the output layer consists of 10 neurons to predict 10 classes of the Fashion MNIST dataset.
  - The learning rate is set to 0.001 with a batch size of 64 and is held constant for the experiment.
  - The models are trained for 30 epochs.

- Methods:
  - All neural networks are Vanilla Neural Networks (no CNNs Used) with input Layer, hidden Layer, and output Layer with softMax.
  - Loss function used for the experiment is the Cross Entropy.
  - Adam Optimizer with a learning rate of 0.001 is used.
  - As specified, I have considered three activation functions ReLU.
  - Also, I have iterated the experiment with 20%, 40%, 60%, 80%, and 100% of training data to train the network which leaves us with five neural network models.
    * The models are named as Lab1_P2 _TrainingRatioSplit _<TrRatioNumber> to capture the Training Ratio split used for the model.

2.2 Results & Analysis

- The number of training points for 20% of the training dataset is around 0.2 * (0.7 * 60000) = 8400, so for a single epoch, the model is trained on 8400 images. On top of it, the random split of PyTorch is used to create a sub-dataset of the training dataset, which balances the class ratio, trying to preserve the probability distribution of the parent dataset. 8400 images consisting of balanced classes is a good amount of images to train on and that is why accuracy is around 84.5% on the test data after epoch 1.
- For any particular training ratio, as the number of epochs increases, we can see that the test accuracy increases. As the model sees more and more meaningful data, it should be able to classify it better. This trend can be clearly observed in figure 1 of Section 2.
- If we compare the plots for different training ratios, we can see that the test accuracy begins to saturate irrespective of the increase in the number of epochs. Ideally, for training ratio '1', this saturation is supposed to be achieved first and then followed by a model with a 0.8 training ratio, and this descending follows. But this expected trend is not visible for the training dataset taken. This can be explained from the quality of the dataset that is split and number of training points for the training. Even for as low as 0.2 splits, we have as large as 8400 images(with balanced classes as the parent), the accuracy is as high as 84% and even the other large splits don't contribute that much to learning new features for the dataset taken.

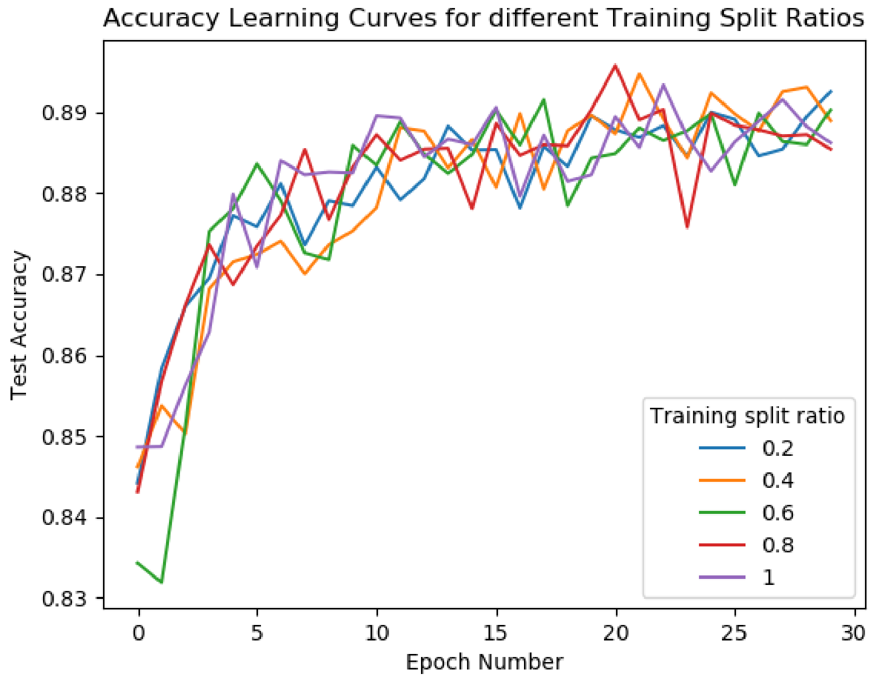


Figure 1: TEST Accuracy Learning Curves on Fashion MNIST Dataset for different Training Ratios

- Also we can see that the test accuracy saturates near 89% of irrespective seeing more and more data and also the number of epochs. This is because, from just the vanilla neural networks, the model is not able to learn any new features, the features it could learn became stagnant. This would call for trying out CNNs if they could learn features that this experiment NN couldn't.

In [ ]:
```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torch.utils.data import DataLoader
import torchvision.datasets as datasets
import torchvision.transforms as transforms
from tqdm import tqdm
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import seaborn as sn
import pandas as pd
import numpy as np
```

In [ ]:
```
#Using GPU if exists
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

In [ ]:
```
# Load Data
fashion_MNIST_dataset = datasets.CIFAR10(root='dataset/CIFAR10', train=True,
                                          transform=transforms.ToTensor(), download=True)
```

In [ ]:
```
# Create a Fully Connected Network
def linear_block(in_f, out_f , activation_function):
    return nn.Sequential(
             nn.Linear(in_f, out_f),
             activation_function
    )

class NN(nn.Module):  # Inheriting this class from nn.Module
    def __init__(self, input_size, output_size, no_of_hidden_layers,
                 no_of_neurons_hidden_layer, activation_function):
        super(NN, self).__init__()
        #inputLayer
        self.input_layer = linear_block(input_size, no_of_neurons_hidden_layer, activati
        self.isHiddenLayer = False
        #Hidden Layers
        if(no_of_hidden_layers != 0 ):
            self.isHiddenLayer = True
            self.hidden_layers = [linear_block(no_of_neurons_hidden_layer,
                                                 no_of_neurons_hidden_layer, activation_fu
                                  for i in range(no_of_hidden_layers)]
            self.hidden_layer = nn.Sequential(*(self.hidden_layers))
        #OutputLayer
        self.output_layer =  nn.Linear(no_of_neurons_hidden_layer, output_size)
        #SoftMax
        #self.softmaxOperation =

    def forward(self, x):
        x = self.input_layer(x)
        if(self.isHiddenLayer):
            x = self.hidden_layer(x)
        x = self.output_layer(x)
        #x = self.softmaxOperation(x)
        return x
```

In [ ]:
```
def split_dataset(dataset, ratio):
    subsetALength = (int)( len(dataset) * ratio)
    subsetA, subsetB = torch.utils.data.random_split(dataset,
                                                     [subsetALength, len(dataset)-subset
    subsetALoader = DataLoader(dataset=subsetA, batch_size=batch_size, shuffle=True)
    subsetBLoader =  DataLoader(dataset=subsetB, batch_size=batch_size, shuffle=True)
```

```python
    return(subsetALoader,subsetBLoader,subsetA)




# Train Network
def trainNetwork(train_loader, model):
    for epoch in range(num_epochs):
        losses = []
        loop = tqdm(enumerate(train_loader), total=len(train_loader), leave=False)
    #     if epoch % 3 == 0:
    #         checkpoint = {'state_dict': model.state_dict(), 'optimizer': optimizer.sta
    #         save_checkpoint(checkpoint)
        for batch_ids, (data, targets) in loop:
            data = data.to(device=device)
            #Flattening the image
            data = data.reshape(data.shape[0], -1)
            #print(data.shape)
            targets = targets.to(device=device)
            #print(data.shape)
            scores = model(data)
            loss = criterion(scores, targets)
            losses.append(loss.item())
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            #loop.set_description(f'Epoch [{epoch}/{num_epochs}]')
            #loop.set_postfix(loss=loss.item())
        mean_loss = sum(losses) / len(losses)
       # scheduler.step(mean_loss)
        print(f'Loss at epoch {epoch} was {mean_loss:.5f}')
```

In [ ]:

```python
# constant for classes
classes = ('airplane', 'automobile', 'bird', 'cat', 'deer',
        'dog', 'frog', 'horse', 'ship', 'truck')
def check_accuracy(loader, model):
    num_correct = 0
    num_samples = 0
    y_pred = torch.tensor([], device=device).float()
    y_true = torch.tensor([], device=device).float()
    model.eval()
    with torch.no_grad():
        for x, y in tqdm(loader):
            x = x.to(device=device)
            y = y.to(device=device)
            x = x.reshape(x.shape[0], -1)
            scores = model(x)
            _, predictions = scores.max(1)
            #print(type(y_true),type(y_pred))
            y_pred = torch.cat((y_pred, predictions.float()))
            y_true = torch.cat((y_true, y.float()),0)
            num_correct += (predictions == y).sum()
            num_samples += predictions.size(0)
        accuracy = float(num_correct) / float(num_samples)
        print(f'For Model: {model.name} : Got {num_correct}/{num_samples}
        with accuracy {float(num_correct) / float(num_samples) * 100:.2f}')
        return (accuracy, y_pred, y_true)

def getConfusionMatrixAndAccuracy(train_loader, test_loader, model):
        train_accuracy, y_train_pred, y_train_true = check_accuracy(train_loader,model)
        test_accuracy, y_test_pred, y_test_true = check_accuracy(test_loader,model)
        #Confusion matrix
        fig, (ax1, ax2) = plt.subplots(1, 2,  figsize=(16,9))
        fig.suptitle(f'Confusion Matrices for Training & Test Data for {model.name}')
        ax1.set_title(f' Train Acc = {train_accuracy*100:.2f} %')
```

```
        cbar_kws = {"orientation":"horizontal",
             "drawedges":True,
           }

        fig.text(0.5, 0.01, "Predicted Values", va='center')
        fig.text(0.01, 0.5, "True Values", va='center')
        plt.xlabel('Predicted Values')
        plt.ylabel('True Values')

        plotAlongGivenAxis(y_train_true, y_train_pred,ax1,cbar_kws)

        ax2.set_title(f' Test Acc = {test_accuracy*100:.2f} %')
        plotAlongGivenAxis(y_test_true, y_test_pred,ax2,cbar_kws)

        plt.savefig(f'{model.name}.png')
        return fig


def plotAlongGivenAxis(y_true, y_pred, axis, cbar_kws):
        cf_matrix = confusion_matrix(y_true.cpu(), y_pred.cpu())
        df_cm = pd.DataFrame(cf_matrix, index = [i for i in classes],columns = [i for i
        hm = sn.heatmap(df_cm, annot=True,annot_kws={"size": 9}, ax=axis,fmt="5d", squar
        return hm
```

In [ ]:

```
#Question 1

# Hyperparameters
tr = 0.7
no_hidden_layers_list = [0,1,2]
no_0f_neurons_hidden_layer = 1024
activation_functions = [nn.ReLU(), nn.Tanh(), nn.Sigmoid()]
input_size = 3072
output_size = 10
learning_rate = 0.001
batch_size = 128
num_epochs = 20
tr = 0.7

# Loss and optimizer
criterion = nn.CrossEntropyLoss()
model_names = []


#Splitting Dataset
trainL, testL,_ = split_dataset(fashion_MNIST_dataset, tr)

#creating models with zero, one and two hidden layers
for af in activation_functions:
    for no_hidden_layers in no_hidden_layers_list:
        #Create Model
        model = NN(input_size,output_size,no_hidden_layers,
                   no_0f_neurons_hidden_layer,af).to(device)
        model.name = "Lab1_P1_HiddenL_" + str(no_hidden_layers+1) + "_Activation_" + af.
        model_names.append(model.name)
        print(model)

        #Optimizer
        optimizer = optim.Adam(model.parameters(), lr=learning_rate)
        #scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, patience=5, factor=
        #train the model
        trainNetwork(trainL, model)
        torch.save(model, ""+model.name+ ".pth")
```

In [ ]:
```
#Get Results
for m in model_names:
    model = torch.load(""+m+ ".pth")
    getConfusionMatrixAndAccuracy(trainL, testL, model)
```

In [ ]:
```
#Question 2

# Hyperparameters
tr = 0.7
no_hidden_layers = 1
no_0f_neurons_hidden_layer = 256
af = nn.ReLU()
input_size = 784
output_size = 10
learning_rate = 0.001
batch_size = 64
num_epochs = 30
training_ratios=[0.2,0.4,0.6,0.8,1]


# Load Data
dataset = datasets.FashionMNIST(root='dataset/FashionMNIST', train=True,
                                transform=transforms.ToTensor(), download=True)
# Loss and optimizer
criterion = nn.CrossEntropyLoss()
model_names_Q2 = []
#Splitting Dataset to Training & Test
firstSplitLoader, testLoader, firstSplitDataset = split_dataset(dataset, tr)
#creating models with zero, one and two hidden layers x
accuracies_tr = []
for tr_split in training_ratios:
        requiredTrainLoader,_,_ = split_dataset(firstSplitDataset,tr_split)
        accuracies_epoch = []
        #Create Model
        model = NN(input_size,output_size,no_hidden_layers,
                   no_0f_neurons_hidden_layer,af).to(device)
        model.name = "Lab1_P2_TrainingRatioSplit_" + str(tr_split)
        model_names_Q2.append(model.name)
        #print(model)
        #Optimizer
        optimizer = optim.Adam(model.parameters(), lr=learning_rate)
        #train the model
        for epoch in range(num_epochs):
            losses = []
            loop = tqdm(enumerate(requiredTrainLoader),
                        total=len(requiredTrainLoader), leave=False)
            for batch_ids, (data, targets) in loop:
                data = data.to(device=device)
                #Flattening the image
                data = data.reshape(data.shape[0], -1)
                #print(data.shape)
                targets = targets.to(device=device)
                #print(data.shape)
                scores = model(data)
                loss = criterion(scores, targets)
                losses.append(loss.item())
                optimizer.zero_grad()
                loss.backward()
                optimizer.step()
                #loop.set_description(f'Epoch [{epoch}/{num_epochs}]')
                #loop.set_postfix(loss=loss.item())
            mean_loss = sum(losses) / len(losses)
            accuracy,_,_ = check_accuracy(testLoader,model)
            accuracies_epoch.append(accuracy)
            #print(accuracies_epoch)
```

```
            #Resume Training
            model.train()
           # scheduler.step(mean_loss)
            print(f'Loss at epoch {epoch} was {mean_loss:.5f}')
        accuracies_tr.append(accuracies_epoch)
        #print(accuracies_tr)
        torch.save(model, ""+model.name+ ".pth")
```

In [ ]:
```
for i in range(len(accuracies_tr)):
    plt.plot(range(len(accuracies_tr[i])),accuracies_tr[i],
             label = "tr :"+str( training_ratios[i]))
plt.legend(labels = training_ratios,title = "Training split ratio")
plt.xlabel("Epoch Number")
plt.ylabel("Test Accuracy")
plt.title("Accuracy Learning Curves for different Training Split Ratios")
plt.savefig('Lab1_P2.png')
plt.show()
```

In [ ]:
```
print(len(dataset))
```

In [ ]:
```
# Sanity Check for the Model
af = nn.ReLU()
model = NN(784,10,1,1024,af).to(device)
model.name = "dfafda"
torch.save(model, ""+model.name+ ".pth")
#Batch Input
x = torch.randn(128, 784).to(device)
y = model(x)

print("x[0]s output is: ", y[0])
print(torch.sum(y[0]))
```

In [ ]: