Fashion MNIST Classification

The libraries are imported and data is loaded in the following cell.

In [1]:
```
import torch
import torchvision
import torchvision.transforms as transforms
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.autograd import Variable
from torch.utils import data as D
import time

import matplotlib.pyplot as plt
import numpy as np

torch.manual_seed(0)

train_set = torchvision.datasets.FashionMNIST(root = ".", train = True ,downl
oad = True , transform = transforms.ToTensor())
test_set = torchvision.datasets.FashionMNIST(root = ".", train = False ,downl
oad = True , transform = transforms.ToTensor())

def test_train_validation_split(train_set,test_set,train_ratio=0.7):
    """
    Train and Validation split
    if train_ratio is 0.7 it will split the train_set into 70% training data,
30% Validation Data.
    * this function is not used after a calrification on forum post
    params: train_set, test_set, train_ratio
    returns: training_loader,valid_loader,test_loader
    """
    train_len = int(train_ratio*len(train_set))
    valid_len = len(train_set) - train_len
    train, valid = D.random_split(train_set, lengths=[train_len, valid_len])
    training_loader = torch.utils.data.DataLoader(train, batch_size=32,shuffl
e = False)
    valid_loader = torch.utils.data.DataLoader(valid, batch_size=32,shuffle =
False)
    test_loader = torch.utils.data.DataLoader(test_set, batch_size=32,shuffle
= False)
    print("Train data len: ", len(training_loader),"Valid data len: ",len(val
id_loader),"Test data len: ",len(test_loader))
    return training_loader,valid_loader,test_loader
```

In the following cells, the data is analysed and simple visualisation of the data is done.

In [2]:
```
training_loader,valid_loader,test_loader = test_train_validation_split(train_
set,test_set,train_ratio=1)
```

```
Train data len:  1875 Valid data len:  0 Test data len:  313
```

In [3]:
```
len(training_loader),len(valid_loader),len(test_loader)
```

Out[3]:
```
(1875, 0, 313)
```

In [4]:
```
dataiter = iter(training_loader)
images, labels = dataiter.next()
print(type(images))
print(images.shape)
print(labels.shape)
```

```
<class 'torch.Tensor'>
torch.Size([32, 1, 28, 28])
torch.Size([32])
```

In [5]:
```
labels_map = {0 : 'T-Shirt', 1 : 'Trouser', 2 : 'Pullover', 3 : 'Dress', 4 :
'Coat', 5 : 'Sandal', 6 : 'Shirt',
              7 : 'Sneaker', 8 : 'Bag', 9 : 'Ankle Boot'};
fig = plt.figure(figsize=(8,8));
columns = 3;
rows = 3;
for i in range(1, columns*rows +1):
    img_xy = np.random.randint(len(train_set));
    img = train_set[img_xy][0][0,:,:]
    fig.add_subplot(rows, columns, i)
    plt.title(labels_map[train_set[img_xy][1]])
    plt.axis('off')
    plt.imshow(img, cmap='gray')
```

From the above we analysed that there are 10 classes of Fashion accessories, thus we would be building a Multiclass classifier.

In the following code, the CNN network is built with following specifications

```
Network(
  (conv1): Conv2d(1, 32, kernel_size=(5, 5), stride=(1, 1))
  (conv2): Conv2d(32, 64, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=1024, out_features=256, bias=True)
  (dropout1): Dropout(p=0.8, inplace=False)
  (out): Linear(in_features=256, out_features=10, bias=True)
)
```

Each layer is Xavier normalised.

In [6]:
```
# Build the neural network, expand on top of nn.Module
class Network(nn.Module):
    def __init__(self,num_classes=10,activation="relu",dropout_rate=0,trainin
g_flag=True):
        super(Network, self).__init__()
        self.num_classes = num_classes # 10 here
        self.activation = activation # which activation function to employ
        self.dropout_rate = dropout_rate # if dropout then  how much
        self.training_flag = training_flag # if dropout then trainable or not

        self.conv1 = nn.Conv2d(in_channels=1, out_channels=32, kernel_size=5)
        nn.init.xavier_normal_(self.conv1.weight)
        self.conv2 = nn.Conv2d(in_channels=32, out_channels=64, kernel_size=
5)
        nn.init.xavier_normal_(self.conv2.weight)
        self.fc1 = nn.Linear(in_features=1024, out_features=256)
        nn.init.xavier_normal_(self.fc1.weight)
        self.dropout1 = nn.Dropout(p=dropout_rate,inplace=False)
        self.out = nn.Linear(in_features=256, out_features=num_classes)
        nn.init.xavier_normal_(self.out.weight)



    def forward(self, t):

        # conv 1
        t = self.conv1(t)
        if self.activation == "relu":
            t = torch.nn.functional.relu(t)
        if self.activation == "tanh":
            t = torch.tanh(t)
        if self.activation == "sigmoid":
            t = torch.sigmoid(t)
        if self.activation == "elu":
            t = torch.nn.functional.elu(t)
        t = F.max_pool2d(t, kernel_size=2, stride=2)

        # conv 2
        t = self.conv2(t)
        if self.activation == "relu":
            t = torch.nn.functional.relu(t)
        if self.activation == "tanh":
            t = torch.tanh(t)
        if self.activation == "sigmoid":
            t = torch.sigmoid(t)
        if self.activation == "elu":
            t = torch.nn.functional.elu(t)
        t = F.max_pool2d(t, kernel_size=2, stride=2)

        t = t.view(t.size(0), -1)
        # fc1
        t = self.fc1(t)
        if self.activation == "relu":
            t = torch.nn.functional.relu(t)
        if self.activation == "tanh":
            t = torch.tanh(t)
        if self.activation == "sigmoid":
            t = torch.sigmoid(t)
        if self.activation == "elu":
            t = torch.nn.functional.elu(t)

        # output
        t = self.out(t)
        if self.dropout_rate > 0.0 and self.training_flag==True:
            t = self.dropout1(t)
        elif self.dropout_rate > 0.0 and self.training_flag==False:
            t = torch.nn.functional.dropout(t, self.dropout_rate)
        else:
```

Below functions are invoked to perform training and evaluation of the model

In [7]:
```
def train(model,x,y,criterion,optimizer):
    """
    train Neural Network
    """
    # zero the parameter gradients
    optimizer.zero_grad()

    # forward + backward + optimize
    outputs = model(x)
    loss = criterion(outputs, y)
    loss.backward()
    optimizer.step()
    return loss.item()

def valid(model,x_valid,y_valid,criterion):
    """
    perform validation NN
    * This is not performed
    """
    model.eval()
    y_pred = model(x_valid)
    loss = criterion(y_pred, y_valid)
    return loss.item()

def evaluation(model,loader):
    """
    perform evalaution NN
    """
    correct = 0
    total = 0
    model.eval()
    with torch.no_grad():
        for i, (images,labels) in enumerate(loader):
            images, labels = images.to(device), labels.to(device)
            images = Variable(images.float())
            labels = Variable(labels)
            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    return correct / total if total > 0 else 0
```

Below function is invoked each time a different parameter is set. For example, if we want to use different epoch size, below function is used and different epoch size can be set in *params

In [8]:
```
def classify(device,model,data,criterion,optimizer,*params):
    """
    One function to perform calssification with different model parameters
    """
    num_epochs,num_classes,batch_size,learning_rate,activation = params
    training_loader,valid_loader,test_loader = data

    total_step = len(training_loader)
    train_losses = []
    valid_losses = []
    train_accuracy = []
    valid_accuracy = []
    test_accuracy = []
    tte = [] # time taken per epoch
    for epoch in range(num_epochs):
        loss_epoch = 0
        tte_start = time.time()

        for i, (images,labels) in enumerate(training_loader):
            model.train()
            images, labels = images.to(device), labels.to(device)
            images = Variable(images.float())
            labels = Variable(labels)
            loss = train(model,images,labels,criterion,optimizer)
            loss_epoch += loss
        train_losses.append(loss)

        if len(valid_loader)>0:
            for j, (images,labels) in enumerate(valid_loader):
                images, labels = images.to(device), labels.to(device)
                images = Variable(images.float())
                labels = Variable(labels)
                loss = valid(model,images,labels,criterion)
            valid_losses.append(loss)
        else:
            valid_losses.append(0)
        train_accuracy.append(evaluation(model,training_loader))
        valid_accuracy.append(evaluation(model,valid_loader))
        test_accuracy.append(evaluation(model,test_loader))
        tte_end = time.time()
        elapsed_time = tte_end-tte_start
        tte.append(elapsed_time)
        print ('Epoch [{}/{}] {:.2f} sec, loss_epoch: {:.2f} ,loss: {:.4f}, a
cc: {:.4f}, v_loss: {:.1f}, v_acc: {:.1f}, test_acc: {:.4f}'.format(epoch+1,
num_epochs, elapsed_time,loss_epoch,train_losses[-1], train_accuracy[-1], val
id_losses[-1], valid_accuracy[-1],test_accuracy[-1]))

    final_test_accuracy = evaluation(model,test_loader)
    print('Test Accuracy of the model: {:.4f} %'.format(100 * final_test_accu
racy))
    print('Total Time taken: {:.3f} sec'.format(sum(tte)))
    return train_losses, valid_losses, train_accuracy, valid_accuracy, test_a
ccuracy, final_test_accuracy, sum(tte)
```

Below Code is performing Q1 a and b, epoch size is 50, SGD optimiser, learning rate is 0.1 and CrossEntropyLoss.

In [9]:
```
result = []
num_epochs = 50
num_classes = 10
batch_size = 100
learning_rate = 0.1
activation = "relu"
dropout_rate = 0
training_flag = True
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
net = Network(num_classes,activation=activation,dropout_rate=dropout_rate,tra
ining_flag=training_flag).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=learning_rate)
data = test_train_validation_split(train_set,test_set,train_ratio=1)
[training_loader,valid_loader,test_loader] = data

t_loss,v_loss,t_acc,v_acc,test_acc,final_test_accuracy,exec_time = classify(d
evice,net,data,criterion,optimizer,num_epochs,num_classes,batch_size,learning
_rate,activation)
end = [learning_rate,activation,dropout_rate,num_epochs,t_loss,v_loss,t_acc,v
_acc,test_acc,final_test_accuracy,exec_time]
result.append(end)
```

```
Train data len:  1875 Valid data len:  0 Test data len:  313
Epoch [1/50] 12.78 sec, loss_epoch: 958.68 ,loss: 0.1080, acc: 0.8730, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8618
Epoch [2/50] 14.81 sec, loss_epoch: 607.30 ,loss: 0.0929, acc: 0.8929, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8780
Epoch [3/50] 14.23 sec, loss_epoch: 520.96 ,loss: 0.0868, acc: 0.9055, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8872
Epoch [4/50] 15.26 sec, loss_epoch: 463.09 ,loss: 0.0891, acc: 0.9130, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8924
Epoch [5/50] 14.58 sec, loss_epoch: 416.08 ,loss: 0.0574, acc: 0.9143, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8909
Epoch [6/50] 15.21 sec, loss_epoch: 373.60 ,loss: 0.0491, acc: 0.9165, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8888
Epoch [7/50] 14.47 sec, loss_epoch: 336.97 ,loss: 0.0321, acc: 0.9248, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8953
Epoch [8/50] 16.38 sec, loss_epoch: 304.04 ,loss: 0.0172, acc: 0.9286, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8941
Epoch [9/50] 14.76 sec, loss_epoch: 269.39 ,loss: 0.0157, acc: 0.9342, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8954
Epoch [10/50] 15.22 sec, loss_epoch: 240.81 ,loss: 0.0115, acc: 0.9393, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8961
Epoch [11/50] 14.24 sec, loss_epoch: 220.96 ,loss: 0.0125, acc: 0.9363, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8920
Epoch [12/50] 14.94 sec, loss_epoch: 194.31 ,loss: 0.0180, acc: 0.9460, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8954
Epoch [13/50] 15.77 sec, loss_epoch: 183.04 ,loss: 0.0078, acc: 0.9472, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8946
Epoch [14/50] 15.59 sec, loss_epoch: 182.71 ,loss: 0.0023, acc: 0.9453, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8959
Epoch [15/50] 15.64 sec, loss_epoch: 165.97 ,loss: 0.0295, acc: 0.9432, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8908
Epoch [16/50] 15.12 sec, loss_epoch: 162.15 ,loss: 0.0049, acc: 0.9445, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8911
Epoch [17/50] 15.62 sec, loss_epoch: 151.60 ,loss: 0.0054, acc: 0.9483, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8932
Epoch [18/50] 15.26 sec, loss_epoch: 139.35 ,loss: 0.0080, acc: 0.9573, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8973
Epoch [19/50] 14.80 sec, loss_epoch: 130.62 ,loss: 0.0028, acc: 0.9421, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8877
Epoch [20/50] 14.77 sec, loss_epoch: 130.23 ,loss: 0.0126, acc: 0.9566, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8933
Epoch [21/50] 14.84 sec, loss_epoch: 119.55 ,loss: 0.0044, acc: 0.9583, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8973
Epoch [22/50] 14.36 sec, loss_epoch: 106.67 ,loss: 0.1069, acc: 0.9600, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8983
Epoch [23/50] 14.54 sec, loss_epoch: 107.21 ,loss: 0.0032, acc: 0.9590, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8968
Epoch [24/50] 14.84 sec, loss_epoch: 102.16 ,loss: 0.0424, acc: 0.9567, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8966
Epoch [25/50] 15.01 sec, loss_epoch: 98.62 ,loss: 0.0000, acc: 0.9684, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.9017
Epoch [26/50] 15.30 sec, loss_epoch: 96.88 ,loss: 0.0014, acc: 0.9639, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8987
Epoch [27/50] 15.26 sec, loss_epoch: 93.45 ,loss: 0.0007, acc: 0.9719, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.9028
Epoch [28/50] 15.30 sec, loss_epoch: 86.91 ,loss: 0.0026, acc: 0.9512, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8888
Epoch [29/50] 14.23 sec, loss_epoch: 100.99 ,loss: 0.0144, acc: 0.9680, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8990
Epoch [30/50] 15.35 sec, loss_epoch: 79.68 ,loss: 0.0366, acc: 0.9674, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8936
Epoch [31/50] 15.30 sec, loss_epoch: 80.87 ,loss: 0.0010, acc: 0.9596, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8888
Epoch [32/50] 15.01 sec, loss_epoch: 81.48 ,loss: 0.0173, acc: 0.9679, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8956
Epoch [33/50] 15.26 sec, loss_epoch: 75.32 ,loss: 0.0006, acc: 0.9661, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8948
Epoch [34/50] 16.19 sec, loss_epoch: 75.09 ,loss: 0.0484, acc: 0.9726, v_los
```

Saving the model to 'model.ckpt' file.

```
In [10]: torch.save(net.state_dict(), 'model.ckpt')
```

Below code is to get results of different activation functions namely "tanh","sigmoid","elu".

In [12]:
```
for activation in ["tanh","sigmoid","elu"]:
    num_epochs = 50
    num_classes = 10
    batch_size = 100
    learning_rate = 0.1
    dropout_rate = 0
    # activations = ["tanh","sigmoid","elu"]
    print(activation)
    training_flag = True
    device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
    net = Network(num_classes,activation=activation,dropout_rate=dropout_rat
e,training_flag=training_flag).to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.SGD(net.parameters(), lr=learning_rate)
    data = test_train_validation_split(train_set,test_set,train_ratio=1)
    [training_loader,valid_loader,test_loader] = data

    t_loss,v_loss,t_acc,v_acc,test_acc,final_test_accuracy,exec_time = classi
fy(device,net,data,criterion,optimizer,num_epochs,num_classes,batch_size,lear
ning_rate,activation)
    end = [learning_rate,activation,dropout_rate,num_epochs,t_loss,v_loss,t_a
cc,v_acc,test_acc,final_test_accuracy,exec_time]
    result.append(end)
```

```
tanh
Train data len:  1875 Valid data len:  0 Test data len:  313
Epoch [1/50] 15.19 sec, loss_epoch: 875.42 ,loss: 0.2551, acc: 0.8765, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8661
Epoch [2/50] 15.07 sec, loss_epoch: 604.26 ,loss: 0.2159, acc: 0.8939, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8785
Epoch [3/50] 15.20 sec, loss_epoch: 511.64 ,loss: 0.2043, acc: 0.9086, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8873
Epoch [4/50] 15.50 sec, loss_epoch: 444.18 ,loss: 0.1764, acc: 0.9188, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8917
Epoch [5/50] 14.40 sec, loss_epoch: 388.43 ,loss: 0.1518, acc: 0.9263, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8942
Epoch [6/50] 15.16 sec, loss_epoch: 338.31 ,loss: 0.1389, acc: 0.9329, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8951
Epoch [7/50] 15.32 sec, loss_epoch: 291.01 ,loss: 0.1248, acc: 0.9396, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8955
Epoch [8/50] 14.22 sec, loss_epoch: 244.78 ,loss: 0.0987, acc: 0.9465, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8971
Epoch [9/50] 15.18 sec, loss_epoch: 201.55 ,loss: 0.0849, acc: 0.9514, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8995
Epoch [10/50] 15.36 sec, loss_epoch: 161.77 ,loss: 0.1091, acc: 0.9567, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8974
Epoch [11/50] 15.30 sec, loss_epoch: 127.25 ,loss: 0.1039, acc: 0.9583, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8970
Epoch [12/50] 16.07 sec, loss_epoch: 101.51 ,loss: 0.1132, acc: 0.9699, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.9008
Epoch [13/50] 13.64 sec, loss_epoch: 84.79 ,loss: 0.0914, acc: 0.9698, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.9001
Epoch [14/50] 13.72 sec, loss_epoch: 67.75 ,loss: 0.0630, acc: 0.9714, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.9010
Epoch [15/50] 14.67 sec, loss_epoch: 62.22 ,loss: 0.1130, acc: 0.9758, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.9043
Epoch [16/50] 15.34 sec, loss_epoch: 52.32 ,loss: 0.0642, acc: 0.9761, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.9022
Epoch [17/50] 16.03 sec, loss_epoch: 41.82 ,loss: 0.0279, acc: 0.9772, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.9015
Epoch [18/50] 15.24 sec, loss_epoch: 34.43 ,loss: 0.1604, acc: 0.9715, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8984
Epoch [19/50] 15.04 sec, loss_epoch: 33.00 ,loss: 0.0155, acc: 0.9795, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.9046
Epoch [20/50] 15.38 sec, loss_epoch: 32.95 ,loss: 0.0112, acc: 0.9805, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.9037
Epoch [21/50] 15.34 sec, loss_epoch: 24.42 ,loss: 0.0415, acc: 0.9805, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8989
Epoch [22/50] 15.63 sec, loss_epoch: 19.03 ,loss: 0.0145, acc: 0.9856, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.9058
Epoch [23/50] 15.00 sec, loss_epoch: 14.44 ,loss: 0.0136, acc: 0.9818, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.9020
Epoch [24/50] 15.18 sec, loss_epoch: 11.68 ,loss: 0.0044, acc: 0.9878, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.9063
Epoch [25/50] 15.72 sec, loss_epoch: 8.88 ,loss: 0.0081, acc: 0.9925, v_loss:
0.0, v_acc: 0.0, test_acc: 0.9065
Epoch [26/50] 14.96 sec, loss_epoch: 5.72 ,loss: 0.0023, acc: 0.9936, v_loss:
0.0, v_acc: 0.0, test_acc: 0.9099
Epoch [27/50] 15.23 sec, loss_epoch: 4.05 ,loss: 0.0018, acc: 0.9945, v_loss:
0.0, v_acc: 0.0, test_acc: 0.9088
Epoch [28/50] 14.27 sec, loss_epoch: 3.15 ,loss: 0.0017, acc: 0.9974, v_loss:
0.0, v_acc: 0.0, test_acc: 0.9107
Epoch [29/50] 15.62 sec, loss_epoch: 2.64 ,loss: 0.0014, acc: 0.9983, v_loss:
0.0, v_acc: 0.0, test_acc: 0.9105
Epoch [30/50] 15.42 sec, loss_epoch: 2.33 ,loss: 0.0012, acc: 0.9986, v_loss:
0.0, v_acc: 0.0, test_acc: 0.9107
Epoch [31/50] 15.87 sec, loss_epoch: 2.11 ,loss: 0.0010, acc: 0.9989, v_loss:
0.0, v_acc: 0.0, test_acc: 0.9111
Epoch [32/50] 13.82 sec, loss_epoch: 1.93 ,loss: 0.0008, acc: 0.9992, v_loss:
0.0, v_acc: 0.0, test_acc: 0.9113
Epoch [33/50] 15.87 sec, loss_epoch: 1.78 ,loss: 0.0007, acc: 0.9993, v_loss:
0.0, v_acc: 0.0, test_acc: 0.9116
```

Below code is to get reults of perfomace of model for different learning rates 0.001, 0.1, 0.5, 1, 10

In [13]:
```
for learning_rate in [0.001,0.1,0.5,1,10]:
    num_epochs = 50
    num_classes = 10
    batch_size = 100
#     learning_rate = 0.1
    activation = "relu"
    dropout_rate = 0
    print(learning_rate)
    training_flag = True
    device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
    net = Network(num_classes,activation=activation,dropout_rate=dropout_rat
e,training_flag=training_flag).to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.SGD(net.parameters(), lr=learning_rate)
    data = test_train_validation_split(train_set,test_set,train_ratio=1)
    [training_loader,valid_loader,test_loader] = data

    t_loss,v_loss,t_acc,v_acc,test_acc,final_test_accuracy,exec_time = classi
fy(device,net,data,criterion,optimizer,num_epochs,num_classes,batch_size,lear
ning_rate,activation)
    end = [learning_rate,activation,dropout_rate,num_epochs,t_loss,v_loss,t_a
cc,v_acc,test_acc,final_test_accuracy,exec_time]
    result.append(end)
```

```
0.001
Train data len:  1875 Valid data len:  0 Test data len:  313
Epoch [1/50] 14.70 sec, loss_epoch: 3414.23 ,loss: 1.1826, acc: 0.6673, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.6631
Epoch [2/50] 14.25 sec, loss_epoch: 1579.82 ,loss: 0.9128, acc: 0.7034, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.6967
Epoch [3/50] 13.38 sec, loss_epoch: 1358.64 ,loss: 0.8518, acc: 0.7199, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.7116
Epoch [4/50] 15.09 sec, loss_epoch: 1258.39 ,loss: 0.8174, acc: 0.7380, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.7292
Epoch [5/50] 15.33 sec, loss_epoch: 1186.39 ,loss: 0.8006, acc: 0.7547, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.7440
Epoch [6/50] 15.10 sec, loss_epoch: 1130.57 ,loss: 0.7882, acc: 0.7699, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.7588
Epoch [7/50] 15.40 sec, loss_epoch: 1084.88 ,loss: 0.7754, acc: 0.7822, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.7704
Epoch [8/50] 15.13 sec, loss_epoch: 1045.27 ,loss: 0.7620, acc: 0.7920, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.7814
Epoch [9/50] 15.05 sec, loss_epoch: 1010.47 ,loss: 0.7481, acc: 0.8002, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.7887
Epoch [10/50] 15.11 sec, loss_epoch: 979.75 ,loss: 0.7304, acc: 0.8064, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.7966
Epoch [11/50] 15.12 sec, loss_epoch: 952.05 ,loss: 0.7110, acc: 0.8131, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8015
Epoch [12/50] 15.19 sec, loss_epoch: 926.94 ,loss: 0.6903, acc: 0.8186, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8071
Epoch [13/50] 15.21 sec, loss_epoch: 904.51 ,loss: 0.6716, acc: 0.8227, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8124
Epoch [14/50] 15.49 sec, loss_epoch: 884.10 ,loss: 0.6523, acc: 0.8270, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8179
Epoch [15/50] 15.39 sec, loss_epoch: 865.50 ,loss: 0.6350, acc: 0.8308, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8236
Epoch [16/50] 15.62 sec, loss_epoch: 848.28 ,loss: 0.6183, acc: 0.8354, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8268
Epoch [17/50] 15.08 sec, loss_epoch: 832.43 ,loss: 0.6031, acc: 0.8385, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8314
Epoch [18/50] 15.01 sec, loss_epoch: 817.78 ,loss: 0.5884, acc: 0.8416, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8336
Epoch [19/50] 14.51 sec, loss_epoch: 804.21 ,loss: 0.5744, acc: 0.8451, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8369
Epoch [20/50] 14.20 sec, loss_epoch: 791.59 ,loss: 0.5606, acc: 0.8477, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8415
Epoch [21/50] 14.71 sec, loss_epoch: 779.74 ,loss: 0.5493, acc: 0.8508, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8438
Epoch [22/50] 15.03 sec, loss_epoch: 768.62 ,loss: 0.5416, acc: 0.8532, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8459
Epoch [23/50] 14.74 sec, loss_epoch: 758.18 ,loss: 0.5318, acc: 0.8557, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8476
Epoch [24/50] 13.65 sec, loss_epoch: 748.27 ,loss: 0.5224, acc: 0.8585, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8494
Epoch [25/50] 15.30 sec, loss_epoch: 738.90 ,loss: 0.5115, acc: 0.8601, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8515
Epoch [26/50] 15.89 sec, loss_epoch: 730.19 ,loss: 0.4990, acc: 0.8626, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8524
Epoch [27/50] 15.27 sec, loss_epoch: 721.91 ,loss: 0.4885, acc: 0.8645, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8533
Epoch [28/50] 15.05 sec, loss_epoch: 714.07 ,loss: 0.4811, acc: 0.8661, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8545
Epoch [29/50] 14.77 sec, loss_epoch: 706.67 ,loss: 0.4745, acc: 0.8674, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8561
Epoch [30/50] 14.51 sec, loss_epoch: 699.59 ,loss: 0.4679, acc: 0.8689, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8571
Epoch [31/50] 15.10 sec, loss_epoch: 692.90 ,loss: 0.4615, acc: 0.8700, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8588
Epoch [32/50] 15.11 sec, loss_epoch: 686.53 ,loss: 0.4549, acc: 0.8713, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8603
Epoch [33/50] 14.48 sec, loss_epoch: 680.48 ,loss: 0.4493, acc: 0.8723, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8610
```

Below code is to get results of performace of the model when a dropout layer is applied using `nn.Dropout()` layer

In [14]:
```
for dropout_rate in [0.3,0.1,0.8]:
    num_epochs = 50
    num_classes = 10
    batch_size = 100
    learning_rate = 0.1
    activation = "relu"
    # dropout_rate = [0.3,0.1,0.8]
    training_flag = True
    device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
    net = Network(num_classes,activation=activation,dropout_rate=dropout_rat
e,training_flag=training_flag).to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.SGD(net.parameters(), lr=learning_rate)
    data = test_train_validation_split(train_set,test_set,train_ratio=1)
    [training_loader,valid_loader,test_loader] = data

    t_loss,v_loss,t_acc,v_acc,test_acc,final_test_accuracy,exec_time = classi
fy(device,net,data,criterion,optimizer,num_epochs,num_classes,batch_size,lear
ning_rate,activation)
    end = [learning_rate,activation,dropout_rate,num_epochs,t_loss,v_loss,t_a
cc,v_acc,test_acc,final_test_accuracy,exec_time]
    result.append(end)
```

```
Train data len:  1875 Valid data len:  0 Test data len:  313
Epoch [1/50] 12.80 sec, loss_epoch: 1679.94 ,loss: 0.7928, acc: 0.8610, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8496
Epoch [2/50] 13.18 sec, loss_epoch: 1350.48 ,loss: 0.5051, acc: 0.8892, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8754
Epoch [3/50] 12.76 sec, loss_epoch: 1267.41 ,loss: 0.8889, acc: 0.8950, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8834
Epoch [4/50] 12.89 sec, loss_epoch: 1223.88 ,loss: 0.5983, acc: 0.9117, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8939
Epoch [5/50] 12.67 sec, loss_epoch: 1179.29 ,loss: 0.5657, acc: 0.9171, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8994
Epoch [6/50] 12.98 sec, loss_epoch: 1150.59 ,loss: 0.5902, acc: 0.9180, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8947
Epoch [7/50] 12.96 sec, loss_epoch: 1117.02 ,loss: 0.3244, acc: 0.9241, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.9022
Epoch [8/50] 12.92 sec, loss_epoch: 1088.53 ,loss: 0.5666, acc: 0.9294, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.9026
Epoch [9/50] 12.85 sec, loss_epoch: 1062.22 ,loss: 0.4697, acc: 0.9339, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8995
Epoch [10/50] 12.86 sec, loss_epoch: 1046.99 ,loss: 0.5121, acc: 0.9367, v_lo
ss: 0.0, v_acc: 0.0, test_acc: 0.9020
Epoch [11/50] 12.57 sec, loss_epoch: 1025.70 ,loss: 0.6021, acc: 0.9398, v_lo
ss: 0.0, v_acc: 0.0, test_acc: 0.9018
Epoch [12/50] 12.87 sec, loss_epoch: 996.11 ,loss: 0.7430, acc: 0.9388, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8989
Epoch [13/50] 12.87 sec, loss_epoch: 985.83 ,loss: 0.5630, acc: 0.9418, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8984
Epoch [14/50] 12.63 sec, loss_epoch: 957.83 ,loss: 0.3477, acc: 0.9466, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.9028
Epoch [15/50] 12.87 sec, loss_epoch: 941.97 ,loss: 0.4597, acc: 0.9497, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.9043
Epoch [16/50] 12.72 sec, loss_epoch: 925.27 ,loss: 0.5669, acc: 0.9500, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.9013
Epoch [17/50] 12.66 sec, loss_epoch: 919.88 ,loss: 0.4002, acc: 0.9481, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8988
Epoch [18/50] 12.69 sec, loss_epoch: 894.44 ,loss: 0.5864, acc: 0.9579, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.9037
Epoch [19/50] 13.05 sec, loss_epoch: 898.89 ,loss: 0.3294, acc: 0.9591, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.9025
Epoch [20/50] 13.34 sec, loss_epoch: 898.00 ,loss: 0.5099, acc: 0.9650, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.9050
Epoch [21/50] 12.87 sec, loss_epoch: 880.90 ,loss: 0.5722, acc: 0.9571, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.9005
Epoch [22/50] 12.77 sec, loss_epoch: 861.67 ,loss: 0.4365, acc: 0.9539, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8949
Epoch [23/50] 13.17 sec, loss_epoch: 850.06 ,loss: 0.4857, acc: 0.9623, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.9054
Epoch [24/50] 11.54 sec, loss_epoch: 860.03 ,loss: 0.3093, acc: 0.9598, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8987
Epoch [25/50] 11.49 sec, loss_epoch: 843.16 ,loss: 0.4627, acc: 0.9602, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8965
Epoch [26/50] 11.49 sec, loss_epoch: 841.99 ,loss: 0.6514, acc: 0.9709, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.9073
Epoch [27/50] 11.49 sec, loss_epoch: 841.19 ,loss: 0.2594, acc: 0.9689, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.9058
Epoch [28/50] 11.53 sec, loss_epoch: 839.17 ,loss: 0.5408, acc: 0.9709, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.9020
Epoch [29/50] 12.68 sec, loss_epoch: 843.03 ,loss: 0.6268, acc: 0.9680, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.9014
Epoch [30/50] 12.90 sec, loss_epoch: 831.60 ,loss: 0.5885, acc: 0.9735, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.9036
Epoch [31/50] 13.09 sec, loss_epoch: 826.71 ,loss: 0.5956, acc: 0.9735, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.9066
Epoch [32/50] 12.96 sec, loss_epoch: 822.03 ,loss: 0.5142, acc: 0.9648, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8972
Epoch [33/50] 12.96 sec, loss_epoch: 818.25 ,loss: 0.3658, acc: 0.9618, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.8963
Epoch [34/50] 12.85 sec, loss_epoch: 801.54 ,loss: 0.3110, acc: 0.9695, v_los
```

From the above we can see that dropout penalty is applied only during training and not during evalaution. This has been discussed here https://stackoverflow.com/questions/53419474/using-dropout-in-pytorch-nn-dropout-vs-f-dropout (https://stackoverflow.com/questions/53419474/using-dropout-in-pytorch-nn-dropout-vs-f-dropout)

The below code uses `torch.nn.functional.dropout()` to apply dropout to the layers, using this it penalises the network irerespective of its phase, training or evalaution. The results are conclusive.

In [15]:
```
for dropout_rate in [0.3,0.1,0.8]:
    num_epochs = 50
    num_classes = 10
    batch_size = 100
    learning_rate = 0.1
    activation = "relu"
    # dropout_rate = [0.3,0.1,0.8]
    training_flag = False
    device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
    net = Network(num_classes,activation=activation,dropout_rate=dropout_rat
e,training_flag=training_flag).to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.SGD(net.parameters(), lr=learning_rate)
    data = test_train_validation_split(train_set,test_set,train_ratio=1)
    [training_loader,valid_loader,test_loader] = data

    t_loss,v_loss,t_acc,v_acc,test_acc,final_test_accuracy,exec_time = classi
fy(device,net,data,criterion,optimizer,num_epochs,num_classes,batch_size,lear
ning_rate,activation)
    end = [learning_rate,activation,dropout_rate,num_epochs,t_loss,v_loss,t_a
cc,v_acc,test_acc,final_test_accuracy,exec_time]
    result.append(end)
```

```
Train data len:  1875 Valid data len:  0 Test data len:  313
Epoch [1/50] 15.22 sec, loss_epoch: 1720.79 ,loss: 1.1399, acc: 0.6835, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.6783
Epoch [2/50] 14.53 sec, loss_epoch: 1357.55 ,loss: 0.9485, acc: 0.7026, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.6904
Epoch [3/50] 15.23 sec, loss_epoch: 1274.73 ,loss: 0.8280, acc: 0.7095, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.7008
Epoch [4/50] 14.73 sec, loss_epoch: 1225.02 ,loss: 0.7898, acc: 0.7195, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.7130
Epoch [5/50] 15.07 sec, loss_epoch: 1186.62 ,loss: 0.3586, acc: 0.7264, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.7147
Epoch [6/50] 16.03 sec, loss_epoch: 1143.82 ,loss: 0.5810, acc: 0.7319, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.7231
Epoch [7/50] 15.31 sec, loss_epoch: 1134.06 ,loss: 0.6613, acc: 0.7400, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.7223
Epoch [8/50] 15.34 sec, loss_epoch: 1105.29 ,loss: 0.6032, acc: 0.7374, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.7250
Epoch [9/50] 15.39 sec, loss_epoch: 1081.08 ,loss: 0.4110, acc: 0.7396, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.7269
Epoch [10/50] 15.00 sec, loss_epoch: 1054.57 ,loss: 0.6689, acc: 0.7464, v_lo
ss: 0.0, v_acc: 0.0, test_acc: 0.7219
Epoch [11/50] 15.18 sec, loss_epoch: 1027.83 ,loss: 0.6174, acc: 0.7465, v_lo
ss: 0.0, v_acc: 0.0, test_acc: 0.7189
Epoch [12/50] 15.46 sec, loss_epoch: 1006.16 ,loss: 0.4045, acc: 0.7478, v_lo
ss: 0.0, v_acc: 0.0, test_acc: 0.7237
Epoch [13/50] 15.65 sec, loss_epoch: 988.95 ,loss: 0.6037, acc: 0.7479, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.7174
Epoch [14/50] 15.81 sec, loss_epoch: 980.96 ,loss: 0.7440, acc: 0.7542, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.7262
Epoch [15/50] 15.85 sec, loss_epoch: 972.68 ,loss: 0.5086, acc: 0.7522, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.7139
Epoch [16/50] 15.13 sec, loss_epoch: 948.14 ,loss: 0.4517, acc: 0.7572, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.7239
Epoch [17/50] 15.61 sec, loss_epoch: 927.50 ,loss: 0.5620, acc: 0.7586, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.7157
Epoch [18/50] 15.06 sec, loss_epoch: 925.17 ,loss: 0.5007, acc: 0.7600, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.7222
Epoch [19/50] 15.20 sec, loss_epoch: 911.47 ,loss: 0.3781, acc: 0.7578, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.7161
Epoch [20/50] 15.73 sec, loss_epoch: 899.77 ,loss: 0.5116, acc: 0.7604, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.7245
Epoch [21/50] 15.20 sec, loss_epoch: 897.52 ,loss: 0.5718, acc: 0.7607, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.7103
Epoch [22/50] 15.67 sec, loss_epoch: 885.98 ,loss: 0.4710, acc: 0.7622, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.7165
Epoch [23/50] 14.82 sec, loss_epoch: 872.27 ,loss: 0.4477, acc: 0.7634, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.7141
Epoch [24/50] 14.54 sec, loss_epoch: 878.53 ,loss: 0.5329, acc: 0.7650, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.7205
Epoch [25/50] 15.69 sec, loss_epoch: 872.95 ,loss: 0.5333, acc: 0.7578, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.7122
Epoch [26/50] 15.30 sec, loss_epoch: 840.50 ,loss: 0.4293, acc: 0.7598, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.7124
Epoch [27/50] 15.68 sec, loss_epoch: 833.29 ,loss: 0.4859, acc: 0.7605, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.7154
Epoch [28/50] 16.31 sec, loss_epoch: 846.66 ,loss: 0.4400, acc: 0.7723, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.7193
Epoch [29/50] 14.82 sec, loss_epoch: 834.93 ,loss: 0.5670, acc: 0.7695, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.7206
Epoch [30/50] 15.78 sec, loss_epoch: 828.57 ,loss: 0.2602, acc: 0.7658, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.7075
Epoch [31/50] 15.41 sec, loss_epoch: 823.10 ,loss: 0.4102, acc: 0.7701, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.7120
Epoch [32/50] 15.61 sec, loss_epoch: 839.06 ,loss: 0.4347, acc: 0.7720, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.7177
Epoch [33/50] 14.95 sec, loss_epoch: 825.20 ,loss: 0.5253, acc: 0.7775, v_los
s: 0.0, v_acc: 0.0, test_acc: 0.7169
Epoch [34/50] 15.05 sec, loss_epoch: 809.07 ,loss: 0.3498, acc: 0.7597, v_los
```

Saving and tranforming all the results to a usable format in pandas, dataframe.

In [56]:
```
import pandas as pd
df = pd.DataFrame(result, columns = ['learning_rate','activation','dropout_ra
te','num_epochs','t_loss','v_loss','t_acc','v_acc','test_acc','final_test_acc
uracy','exec_time'])
df['final_train_acc'] = df.apply(lambda x: x.t_acc[-1],axis = 1)
df.to_csv("results.csv",header=False)
```

In [57]:
```
df.head(3)
```

Out[57]:

	learning_rate	activation	dropout_rate	num_epochs	t_loss	v_loss	t_ac
0	0.1	relu	0.0	50	[0.1079825758934021, 0.09290650486946106, 0.08...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[0.8730166666666667 0.8929333333333334 0.905..
1	0.1	tanh	0.0	50	[0.25511041283607483, 0.21585991978645325, 0.2...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[0.8765, 0.8939 0.9085833333333333 0.9188166..
2	0.1	sigmoid	0.0	50	[0.6761802434921265, 0.5924240350723267, 0.586...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[0.7176333333333333 0.7651666666666667 0.798..

Q2 b results

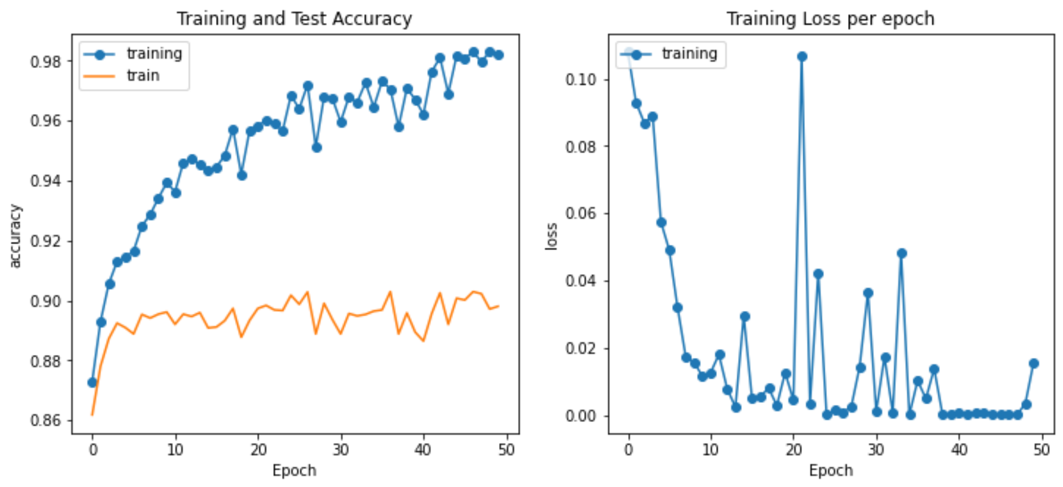
In [58]:
```
pd.DataFrame([[df.loc[0, ['final_train_acc']][0],df.loc[0, ['final_test_accur
acy']][0]]],columns=['Final Train Accuracy','Final Test Acurracy'])
```

Out[58]:

	Final Train Accuracy	Final Test Acurracy
0	0.9822	0.898

In [59]:
```
t_acc = df.loc[0, ['t_acc']][0]
t_loss = df.loc[0, ['t_loss']][0]
test_acc = df.loc[0, ['test_acc']][0]

fig = plt.figure(figsize=(12,5))
subfig = fig.add_subplot(121)
subfig.plot(t_acc, 'o-' ,label="training")
subfig.plot(test_acc, label="train")
subfig.set_title('Training and Test Accuracy')
subfig.set_xlabel('Epoch')
subfig.set_ylabel('accuracy')
subfig.legend(loc='upper left')
subfig = fig.add_subplot(122)
subfig.plot(t_loss, 'o-' ,label="training")
# subfig.plot(v_loss, label="validation")
subfig.set_title('Training Loss per epoch')
subfig.set_xlabel('Epoch')
subfig.set_ylabel('loss')
subfig.legend(loc='upper left')
```

Out[59]:
```
<matplotlib.legend.Legend at 0x7fdb6f7fbe80>
```



Q2 c Activation function

In [60]:
```
options = ['sigmoid','tanh','elu']
df[df['activation'].isin(options)].loc[:,['activation','final_train_acc','fin
al_test_accuracy']]
```

Out[60]:

	activation	final_train_acc	final_test_accuracy
1	tanh	1.000000	0.9117
2	sigmoid	0.939467	0.9024
3	elu	0.994233	0.9032

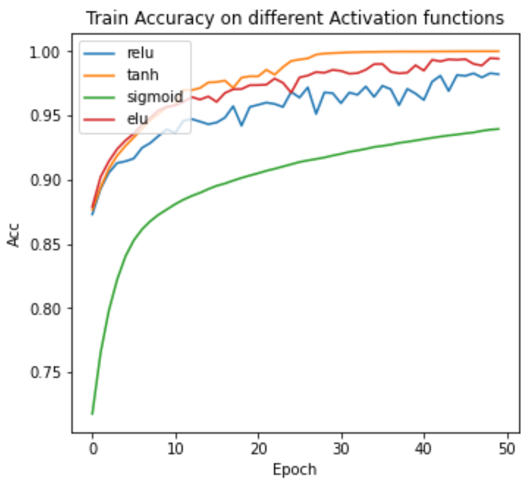
In [61]:
```
options = ['sigmoid','tanh','elu']
d_act_ = df[df['activation'].isin(options)].loc[:,['t_acc','test_acc','t_loss
']]
df[df['activation'].isin(options)].loc[:,['t_acc','test_acc','t_loss']]
```

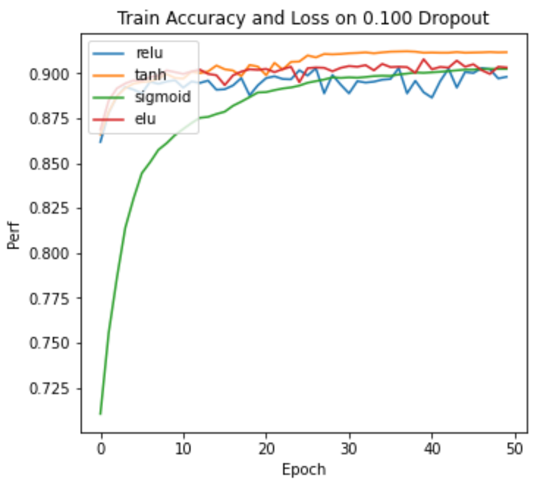
Out[61]:

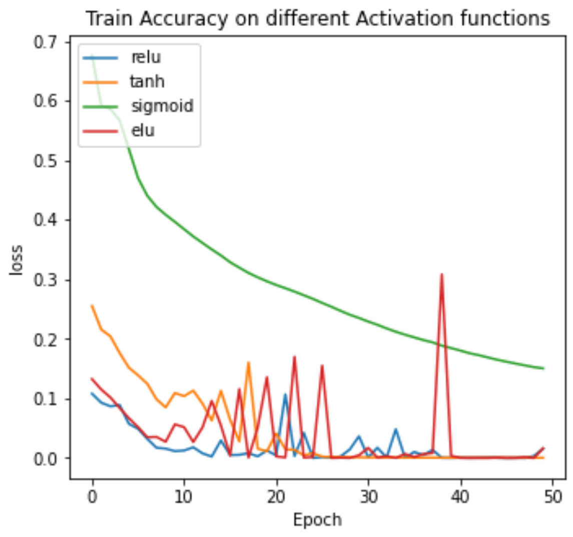
	t_acc	test_acc	t_loss
1	[0.8765, 0.8939, 0.9085833333333333, 0.9188166...	[0.8661, 0.8785, 0.8873, 0.8917, 0.8942, 0.895...	[0.25511041283607483, 0.21585991978645325, 0.2...
2	[0.7176333333333333, 0.7651666666666667, 0.798...	[0.7105, 0.7556, 0.7864, 0.814, 0.8303, 0.8444...	[0.6761802434921265, 0.5924240350723267, 0.586...
3	[0.8789333333333333, 0.9021666666666667, 0.914...	[0.8686, 0.8848, 0.8916, 0.8943, 0.896, 0.8958...	[0.13247402012348175, 0.1150437444448471, 0.10...

In [62]:
```
d_act_t_acc = d_act_['t_acc']
d_act_test_acc = d_act_['test_acc']
d_act_t_loss = d_act_['t_loss']

fig = plt.figure(figsize=(12,5))
subfig = fig.add_subplot(121)
subfig.plot(t_acc ,label="relu")
subfig.plot(d_act_t_acc[1] ,label="tanh")
subfig.plot(d_act_t_acc[2] ,label="sigmoid")
subfig.plot(d_act_t_acc[3] ,label="elu")
subfig.set_title('Train Accuracy on different Activation functions')
subfig.set_xlabel('Epoch')
subfig.set_ylabel('Acc')
subfig.legend(loc='upper left')
subfig = fig.add_subplot(122)
subfig.plot(test_acc ,label="relu")
subfig.plot(d_act_test_acc[1] ,label="tanh")
subfig.plot(d_act_test_acc[2] ,label="sigmoid")
subfig.plot(d_act_test_acc[3] ,label="elu")
subfig.set_title('Train Accuracy and Loss on 0.100 Dropout')
subfig.set_xlabel('Epoch')
subfig.set_ylabel('Perf')
subfig.legend(loc='upper left')

fig = plt.figure(figsize=(12,5))
subfig = fig.add_subplot(121)
subfig.plot(t_loss ,label="relu")
subfig.plot(d_act_t_loss[1] ,label="tanh")
subfig.plot(d_act_t_loss[2] ,label="sigmoid")
subfig.plot(d_act_t_loss[3] ,label="elu")
subfig.set_title('Train Accuracy on different Activation functions')
subfig.set_xlabel('Epoch')
subfig.set_ylabel('loss')
subfig.legend(loc='upper left')
```

Out[62]:
```
<matplotlib.legend.Legend at 0x7fdb753c00b8>
```







Q3 c learning rate

In [63]:
```
option1 = ['relu']
option2 = [0.0]
options3 = [0.001,0.1,0.5,1,10]
d = df.loc[df['activation'].isin(option1)]
d = d.loc[d['dropout_rate'].isin(option2)]
d[d['learning_rate'].isin(options3)].loc[:,['activation','learning_rate','fin
al_train_acc','final_test_accuracy']][1:6]
```

Out[63]:

	activation	learning_rate	final_train_acc	final_test_accuracy
4	relu	0.001	0.886917	0.8749
5	relu	0.100	0.975883	0.8960
6	relu	0.500	0.100000	0.1000
7	relu	1.000	0.100000	0.1000
8	relu	10.000	0.100000	0.1000

In [64]:
```
option1 = ['relu']
option2 = [0.0]
options3 = [0.001,0.1,0.5,1,10]
d = df.loc[df['activation'].isin(option1)]
d = d.loc[d['dropout_rate'].isin(option2)]
d = d[d['learning_rate'].isin(options3)].loc[:,['t_acc','t_loss']][1:6]
```

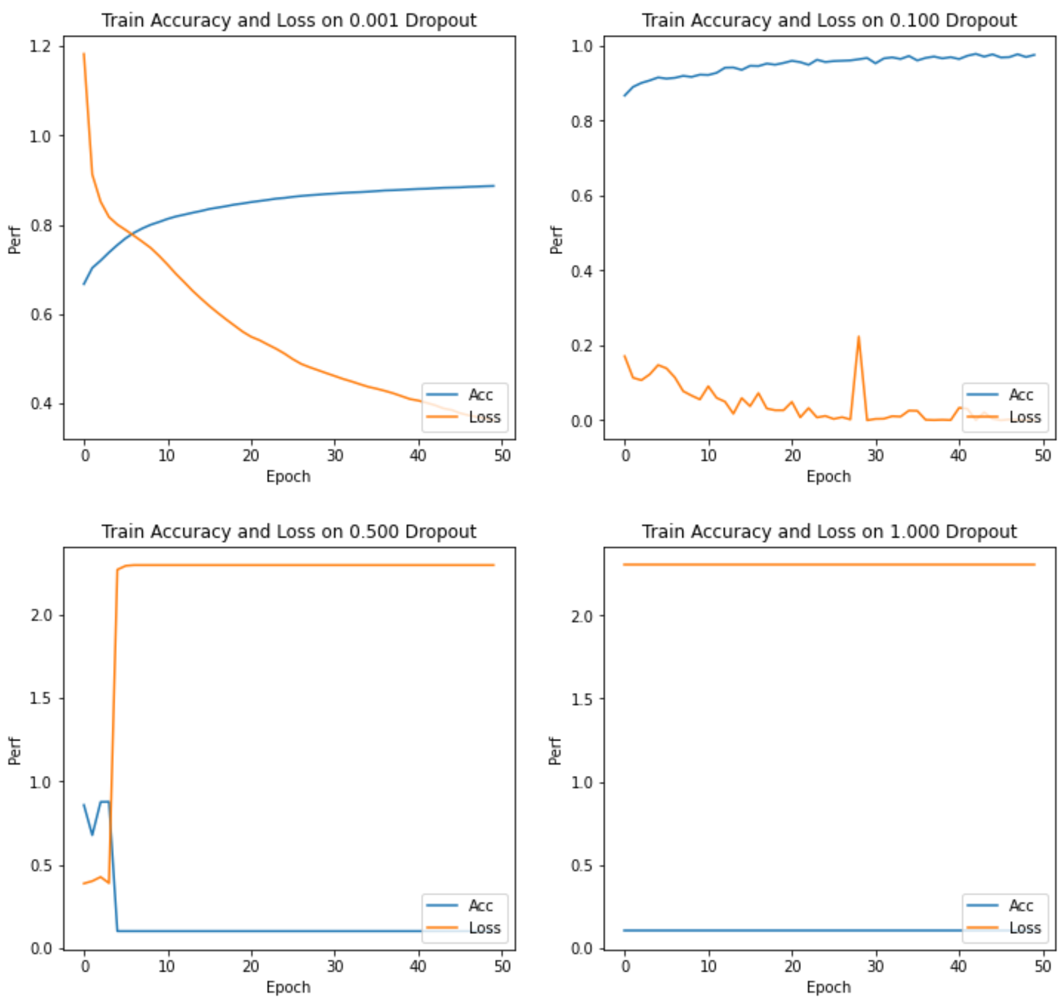
In [65]:
```
d_t_acc = d['t_acc']
d_t_loss = d['t_loss']
d
```

Out[65]:

	t_acc	t_loss
4	[0.66725, 0.7034, 0.7198666666666667, 0.73805,...	[1.1825708150863647, 0.9127765893936157, 0.851...
5	[0.86725, 0.8904, 0.9009666666666667, 0.907633...	[0.17129608988761902, 0.11376349627971649, 0.1...
6	[0.85845, 0.6772, 0.877, 0.8779333333333333, 0...	[0.3865751028060913, 0.4008912444114685, 0.425...
7	[0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, ...	[2.3038978576660156, 2.3038978576660156, 2.303...
8	[0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, ...	[nan, nan, nan, nan, nan, nan, nan, nan, nan, ...

In [66]:
```
fig = plt.figure(figsize=(12,5))
subfig = fig.add_subplot(121)
subfig.plot(d_t_acc[4] ,label="Acc")
subfig.plot(d_t_loss[4] ,label="Loss")
subfig.set_title('Train Accuracy and Loss on 0.001 Dropout')
subfig.set_xlabel('Epoch')
subfig.set_ylabel('Perf')
subfig.legend(loc='lower right')
subfig = fig.add_subplot(122)
subfig.plot(d_t_acc[5] ,label="Acc")
subfig.plot(d_t_loss[5] ,label="Loss")
subfig.set_title('Train Accuracy and Loss on 0.100 Dropout')
subfig.set_xlabel('Epoch')
subfig.set_ylabel('Perf')
subfig.legend(loc='lower right')

fig = plt.figure(figsize=(12,5))
subfig = fig.add_subplot(121)
subfig.plot(d_t_acc[6] ,label="Acc")
subfig.plot(d_t_loss[6] ,label="Loss")
subfig.set_title('Train Accuracy and Loss on 0.500 Dropout')
subfig.set_xlabel('Epoch')
subfig.set_ylabel('Perf')
subfig.legend(loc='lower right')
subfig = fig.add_subplot(122)
subfig.plot(d_t_acc[7] ,label="Acc")
subfig.plot(d_t_loss[7] ,label="Loss")
subfig.set_title('Train Accuracy and Loss on 1.000 Dropout')
subfig.set_xlabel('Epoch')
subfig.set_ylabel('Perf')
subfig.legend(loc='lower right')
```

```
Out[66]: <matplotlib.legend.Legend at 0x7fdb6f56bdd8>
```



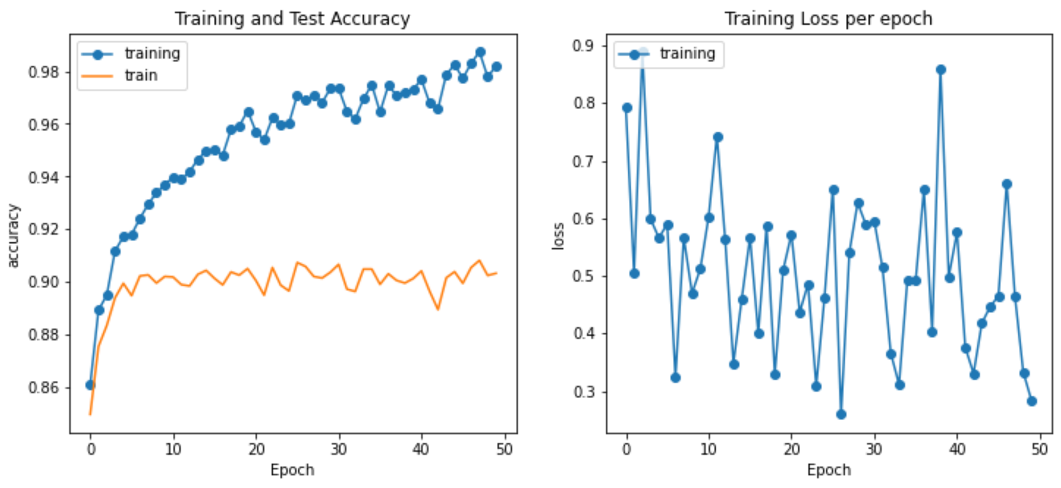
Q2 d, here we analyse the results obtained for different dropouts using different methods. The issue is decribed here https://stackoverflow.com/questions/53419474/using-dropout-in-pytorch-nn-dropout-vs-f-dropout (https://stackoverflow.com/questions/53419474/using-dropout-in-pytorch-nn-dropout-vs-f-dropout) and for this we have obtained different results.

```
In [36]: options = [0.1,0.3,0.8]
         df[df['dropout_rate'].isin(options)].loc[:,['dropout_rate','final_train_acc
         ','final_test_accuracy']]
Out[36]:
```

	dropout_rate	final_train_acc	final_test_accuracy
9	0.3	0.981950	0.9032
10	0.1	0.977067	0.9031
11	0.8	0.962300	0.9053
12	0.3	0.778033	0.7149
13	0.1	0.937100	0.8573
14	0.8	0.295700	0.2826

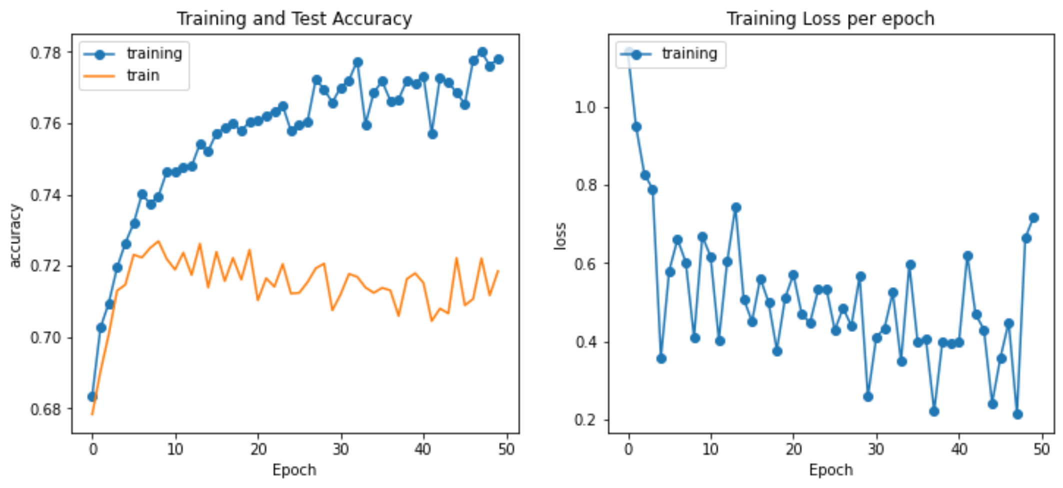
```
In [28]: t_acc = df.loc[9, ['t_acc']][0]
         test_acc = df.loc[9, ['test_acc']][0]
         t_loss = df.loc[9, ['t_loss']][0]
         fig = plt.figure(figsize=(12,5))
         subfig = fig.add_subplot(121)
         subfig.plot(t_acc, 'o-' ,label="training")
         subfig.plot(test_acc, label="train")
         subfig.set_title('Training and Test Accuracy')
         subfig.set_xlabel('Epoch')
         subfig.set_ylabel('accuracy')
         subfig.legend(loc='upper left')
         subfig = fig.add_subplot(122)
         subfig.plot(t_loss, 'o-' ,label="training")
         # subfig.plot(v_loss, label="validation")
         subfig.set_title('Training Loss per epoch')
         subfig.set_xlabel('Epoch')
         subfig.set_ylabel('loss')
         subfig.legend(loc='upper left')
```

```
Out[28]: <matplotlib.legend.Legend at 0x7fdb75782cc0>
```



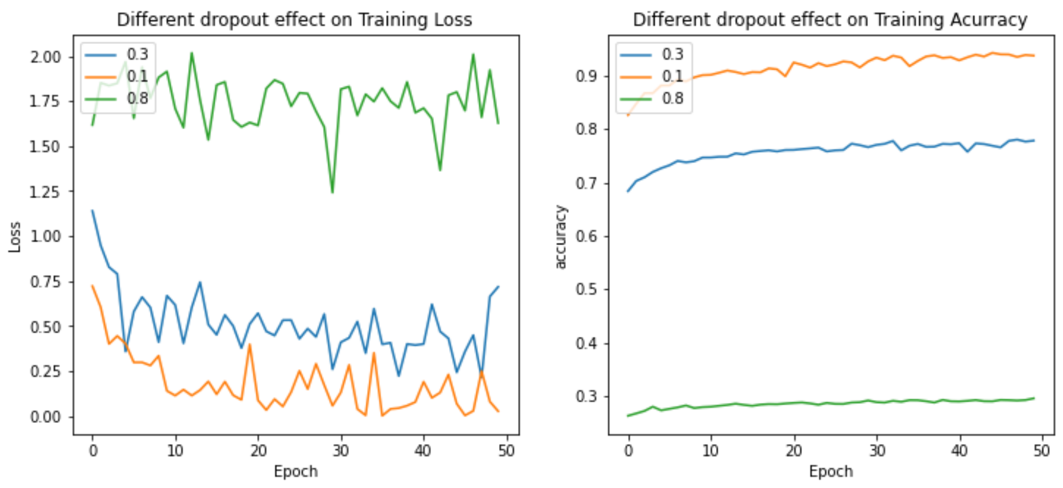
In [29]:
```
t_acc = df.loc[12, ['t_acc']][0]
test_acc = df.loc[12, ['test_acc']][0]
t_loss = df.loc[12, ['t_loss']][0]
fig = plt.figure(figsize=(12,5))
subfig = fig.add_subplot(121)
subfig.plot(t_acc, 'o-' ,label="training")
subfig.plot(test_acc, label="train")
subfig.set_title('Training and Test Accuracy')
subfig.set_xlabel('Epoch')
subfig.set_ylabel('accuracy')
subfig.legend(loc='upper left')
subfig = fig.add_subplot(122)
subfig.plot(t_loss, 'o-' ,label="training")
# subfig.plot(v_loss, label="validation")
subfig.set_title('Training Loss per epoch')
subfig.set_xlabel('Epoch')
subfig.set_ylabel('loss')
subfig.legend(loc='upper left')
```

Out[29]:
```
<matplotlib.legend.Legend at 0x7fdb7557ef98>
```



In [30]:
```
t_loss = df[-3:]['t_loss']
t_acc = df[-3:]['t_acc']

fig = plt.figure(figsize=(12,5))
subfig = fig.add_subplot(121)
subfig.plot(t_loss[12] ,label="0.3")
subfig.plot(t_loss[13] ,label="0.1")
subfig.plot(t_loss[14] ,label="0.8")
subfig.set_title('Different dropout effect on Training Loss')
subfig.set_xlabel('Epoch')
subfig.set_ylabel('Loss')
subfig.legend(loc='upper left')
subfig = fig.add_subplot(122)
subfig.plot(t_acc[12] ,label="0.3")
subfig.plot(t_acc[13] ,label="0.1")
subfig.plot(t_acc[14] ,label="0.8")
subfig.set_title('Different dropout effect on Training Acurracy')
subfig.set_xlabel('Epoch')
subfig.set_ylabel('accuracy')
subfig.legend(loc='upper left')
```

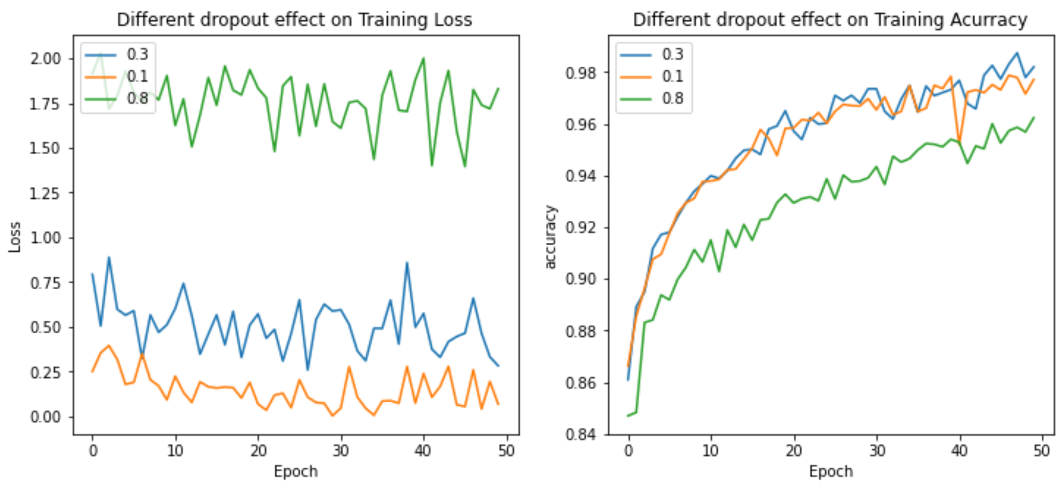
Out[30]:
```
<matplotlib.legend.Legend at 0x7fdb754b7b70>
```



In [32]:
```
t_loss = df[-6:-3]['t_loss']
t_acc = df[-6:-3]['t_acc']
```

In [33]:
```
fig = plt.figure(figsize=(12,5))
subfig = fig.add_subplot(121)
subfig.plot(t_loss[9] ,label="0.3")
subfig.plot(t_loss[10] ,label="0.1")
subfig.plot(t_loss[11] ,label="0.8")
subfig.set_title('Different dropout effect on Training Loss')
subfig.set_xlabel('Epoch')
subfig.set_ylabel('Loss')
subfig.legend(loc='upper left')
subfig = fig.add_subplot(122)
subfig.plot(t_acc[9] ,label="0.3")
subfig.plot(t_acc[10] ,label="0.1")
subfig.plot(t_acc[11] ,label="0.8")
subfig.set_title('Different dropout effect on Training Acurracy')
subfig.set_xlabel('Epoch')
subfig.set_ylabel('accuracy')
subfig.legend(loc='upper left')
```

Out[33]:
```
<matplotlib.legend.Legend at 0x7fdb753192b0>
```


Different dropout effect on Training Loss
Loss
Epoch
Different dropout effect on Training Acurracy
accuracy
Epoch