Project 03

Ece 763 Computer Vision

Professor: Tianfu Wu

Author: Adarsh Puri
(apuri3@ncsu.edu)

Aim:

- Use Neural Network including FeedForward NN and LeNet5, to do face and nonface images classification
- Babysitting the Process and tuning the Hyperparameters

Platform: Google Colabs (GPU), Jupiter Notebook

Data Preparation: Load the dataset in the Google Drive to a folder named “Data”.
This folder should contain your “face” folder and “nonface” folder having images for training and testing.
After Mounting the Google Drive in Google Colab’s Jupiter Notebook,
give the path of the “Data” folder to the Variable named as “ Train_PATH”.
I have taken 90% of Data as the Training set and the rest 10% as the test set.

Code and Results:

Code and Results:

MOUNT THE DRIVE

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.a

Enter your authorization code:
··········
Mounted at /content/drive
```

```
from __future__ import print_function

import os
import torch
from PIL import Image

import matplotlib.image as mpimg
import matplotlib.pyplot as plt


import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

import torch.utils.data as data
import torchvision.transforms as transforms
import torchvision.models as models
import torch.utils.data
import torchvision.datasets

from torch.autograd import Variable
```

```
import copy
import time
import numpy as np
import os
```

LOAD IMAGES

```
TRAIN_PATH = '/content/drive/My Drive/Data'
loader1=transforms.ToTensor()# if not normalize then in range[0,1]
#normalization for simple feedforward neural network
loader2=transforms.Compose(
    [transforms.ToTensor(),#convert an image to tensor
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

#without normalization for LeNet5
loader3=transforms.Compose(
        [transforms.Resize((32,32)),
         transforms.ToTensor()])

#normalization for LeNet5
loader4=transforms.Compose(
        [transforms.Resize((32,32)),
         transforms.ToTensor(),
         transforms.Normalize((0.5,0.5,0.5),(0.5,0.5,0.5))])

def load_images_flow(batch_size,which_trans):
    if(which_trans == 1):
         transform = loader1
    if(which_trans == 2):
         transform = loader2
    if(which_trans == 3):
         transform = loader3
    if(which_trans == 4):
         transform = loader4

    train_set = torchvision.datasets.ImageFolder(root= TRAIN_PATH, transform=transform)
```

```
    train_set = torchvision.datasets.ImageFolder(root= TRAIN_PATH, transform=transform)
    VALID_RATIO = 0.9 # 90%- Train, 10%- Validation
    n_train_examples = int(len(train_set) * VALID_RATIO)
    print("Training Examples - ",n_train_examples)
    n_valid_examples = len(train_set) - n_train_examples
    print("Validation Examples - ",n_valid_examples)
    train_data, valid_data = data.random_split(train_set,
                                               [n_train_examples, n_valid_examples])
    valid_data = copy.deepcopy(valid_data)
    valid_data.dataset.transform = transform

    train_loader = torch.utils.data.DataLoader(train_data, batch_size=batch_size, shuffle=True, num_workers=2)
    valid_loader = torch.utils.data.DataLoader(valid_data, batch_size=batch_size, shuffle=True, num_workers=2)
    return train_loader,valid_loader
```

OPTIMIZER

```
def createLossAndOptimizer(net, learning_rate = 0.001, weight_decay = 0, loss_method = "SGD"):
    #weight_decay: tuning parameter of L2 term
     #Loss function
    loss = torch.nn.CrossEntropyLoss()

    #Optimizer
    if loss_method == "SGD":
        optimizer = optim.SGD(net.parameters(), lr=learning_rate, weight_decay=weight_decay)
    if loss_method == "Adam":
        optimizer = optim.Adam(net.parameters(), lr=learning_rate, weight_decay=weight_decay)
    return(loss, optimizer)
```

Feedforward Neural Network and LeNet5

```
input_size = 3*60*60; hidden_size = 50; output_size = 2
class Two_Layers_NN(torch.nn.Module):

    def __init__(self):
        super(Two_Layers_NN,self).__init__()
```

```
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.fc2 = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        out = self.fc1(x)
        out = self.fc2(out)
        return out




class LeNet(torch.nn.Module):

    def __init__(self):
        super(LeNet,self).__init__()

        self.conv1 = torch.nn.Conv2d(3, 6, kernel_size=(5,5), stride=1)#input 3 channels, output 6 channels

        self.maxpool = nn.MaxPool2d(kernel_size=(2,2),stride=2)
        # torch.nn.AdaptiveAvgPool2d() can avoid overfitting
        self.conv2 = torch.nn.Conv2d(6, 16, kernel_size=(5,5), stride=1)#output may not be the times of input


        self.fc1 = torch.nn.Linear(5*5*16,120)

        self.fc2 = torch.nn.Linear(120,84)

        self.fc3 = torch.nn.Linear(84,2)


    def forward(self, x):
        out = self.conv1(x) #input 32*32*3 output 28*28*6

        #x = self.batchnorm1(x)

        out = self.maxpool(out) #output 14*14*6
```

```
        out = self.conv2(out) #output 10*10*16

        out = self.maxpool(out) #output 5*5*16

        out = out.view(-1, 5 * 5 * 16)#flatten

        out = self.fc1(out)

        out = self.fc2(out)

        out = self.fc3(out)
        return(out)
```

TRAINING FUNCTION

```
def train_net(net, loss_method, which_model, whether_norm, batch_size, n_epochs, learning_rate,
              weight_decay, print_train_process = True, print_test_process= True, validation = True):
    assert(which_model == "NN" or which_model == "LeNet")# the input should be reasonable

    #choose right transformation
    if (whether_norm == False and which_model == "NN"):
        which_trans = 1
    elif (whether_norm == True and which_model == "NN"):
        which_trans = 2
    elif (whether_norm == False and which_model == "LeNet"):
        which_trans = 3
    elif (whether_norm == True and which_model == "LeNet"):
        which_trans = 4

    print(which_trans)

    #Get training data and test data
    train_loader,test_loader = load_images_flow(batch_size,which_trans)


    n_batches = len(train_loader)
```

```
    loss, optimizer =  createLossAndOptimizer(net, learning_rate = learning_rate,
                                              weight_decay = weight_decay,loss_method=loss_method)

    #Time for printing
    start_time = time.time()
    print_every = n_batches // 10

    for epoch in range(n_epochs):
        #epoch = 0
        running_loss = 0
        running_correct_num = 0

        total_train_loss = 0
        total_train_num = 0
        total_correct_train_num = 0

        for i, data in enumerate(train_loader): # handle every batch_size pictures

            (inputs,labels) = data

            if(which_model == "NN"):
                 inputs = inputs.view(inputs.size()[0],3*60*60)#flatten
                 #inputs = inputs.view(batch_size,3*60*60) not batch size since

            inputs, labels = Variable(inputs), Variable(labels)

            optimizer.zero_grad() # whether zero setting is okay ?
            #print(inputs.size())
            #Forward pass, backward pass, optimize
            outputs = net(inputs) #why ? same as forward

            m, predicted = torch.max(outputs.data,1)
            total_train_num += labels.size(0)
            running_correct_num += (predicted == labels).sum().item()
            total_correct_train_num += (predicted == labels).sum().item()

            loss_size = loss(outputs, labels)
            if np.isnan(loss_size.data):
```

```
                raise ValueError("loss explode due to large regularization or learning rate")

            loss_size.backward()
            optimizer.step()

            #print statistics
            running_loss += loss_size.data
            total_train_loss += loss_size.data

            #print every 10th batch
            if(print_train_process == True):
                  if (i+1) % (print_every) == 0:
                        print("Epoch {}, {:d}% \t train_loss: {:.4f} train_accuracy:{:d}% took: {:.2f}s".format(
                             epoch+1, int(100*(i+1)/n_batches), running_loss/print_every/batch_size, int(100 * running_correct_num /p
                             time.time()-start_time)) # loss for currect running_loss and running_correct_num not accumulated ones
                        #reset running loss and time
                        running_loss = 0.0
                        running_correct_num = 0.0
                        start_time = time.time()


        #For validation
        if(validation == True):
            total_test_loss = 0
            total_test_num = 0
            correct_test_num = 0
            for inputs, labels in test_loader:
                if (which_model == "NN"):
                    inputs = inputs.view(inputs.size()[0], 3*60*60)
                    inputs, labels = Variable(inputs), Variable(labels)

                #Forward pass
                test_outputs = net(inputs)

                #test accuracy rate
                m, predicted = torch.max(test_outputs.data, 1)
                #print(predicted)
                total_test_num += labels.size(0)
```

```
                total_test_num += labels.size(0)

                correct_test_num += (predicted == labels).sum().item()
                test_loss_size=loss(test_outputs, labels)
                total_test_loss += test_loss_size.data

            if(print_test_process == True):
                print("Test loss = {:.4f} Test Accuracy = {:d}%".format(total_test_loss / len(test_loader),
                      int(100 * correct_test_num / total_test_num)))
        #print('Accuracy of the network on the 10000 test images: %d %%' % (100 * correct_test_num / total_test_num))
            elif(epoch == n_epochs-1):
                 print("Test loss = {:.4f} Test Accuracy = {:d}%".format(total_test_loss / len(test_loader),
                      int(100 * correct_test_num / total_test_num)))

            #print("Training finished, took {:.2f}s".format(time.time() - start_time))

    if(not print_train_process == True):
          print("train_loss: {:.8f} train_accuracy:{:d}% learning rate:{:.8f} regularization:{:.8f} running time:{:.4f}" .format(runn
           learning_rate, weight_decay, time.time()-start_time))
```

First We do for FEED Forward Neural Network and then LeNet5

Compare the Results with and without normalization

```
weight_decay = 0; learning_rate = 0.001
print("weight decay:{:.4f} learning rate:{:.4f}".format(weight_decay, learning_rate))
NN = Two_Layers_NN()
train_net(NN, which_model = "NN", whether_norm = False, batch_size=5, n_epochs=1, learning_rate = learning_rate,
             weight_decay = weight_decay, print_train_process = True, print_test_process = False, validation = False,loss_method = "
del NN
```

```
weight decay:0.0000 learning rate:0.0010
1
Training Examples -  3003
Validation Examples -  334
Epoch 1, 9%      train_loss: 0.1263 train_accuracy:66% took: 0.50s
Epoch 1, 19%     train_loss: 0.1034 train_accuracy:77% took: 0.45s
Epoch 1, 29%     train_loss: 0.1013 train_accuracy:78% took: 0.43s
Epoch 1, 39%     train_loss: 0.0852 train_accuracy:84% took: 0.42s
Epoch 1, 49%     train_loss: 0.0790 train_accuracy:86% took: 0.44s
Epoch 1, 59%     train_loss: 0.0855 train_accuracy:83% took: 0.43s
Epoch 1, 69%     train_loss: 0.0730 train_accuracy:89% took: 0.42s
Epoch 1, 79%     train_loss: 0.0818 train_accuracy:83% took: 0.44s
Epoch 1, 89%     train_loss: 0.0748 train_accuracy:85% took: 0.43s
Epoch 1, 99%     train_loss: 0.0690 train_accuracy:86% took: 0.43s
```

Without normalization, optimization converge slow.

**Sanity Check : **

Varying Regularisation and keeping learning rate same

```
import random

weight_decay = 0; learning_rate = 0.001
print("weight decay:{:.4f} learning rate:{:.4f}".format(weight_decay, learning_rate))
NN = Two_Layers_NN()
train_net(NN, which_model = "NN", whether_norm = True, batch_size=5, n_epochs=1, learning_rate = learning_rate,
              weight_decay = weight_decay, print_train_process = True, print_test_process = False, validation = False,loss_method = "

del NN


weight_decay = 1000; learning_rate = 0.001
print("weight decay:{:.4f} learning rate:{:.4f}".format(weight_decay, learning_rate))
NN = Two_Layers_NN()
train_net(NN, which_model = "NN", whether_norm = True, batch_size=5, n_epochs=1, learning_rate = learning_rate,
              weight_decay = weight_decay, print_train_process = True, print_test_process = False, validation = False, loss_method =
```

```
del NN

weight_decay = 10000; learning_rate = 0.001
print("weight decay:{:.4f} learning rate:{:.4f}".format(weight_decay, learning_rate))
NN = Two_Layers_NN()
train_net(NN, which_model = "NN", whether_norm = True, batch_size=5, n_epochs=1, learning_rate = learning_rate,
              weight_decay = weight_decay, print_train_process = True, print_test_process = False, validation = False, loss_method =

del NN
#comments:  when regularization increase, loss goes up and when regularization is too large, loss explode
```

```
weight decay:0.0000 learning rate:0.0010
2
Training Examples -  3003
Validation Examples -  334
Epoch 1, 9%      train_loss: 0.1070 train_accuracy:74% took: 0.53s
Epoch 1, 19%     train_loss: 0.0989 train_accuracy:78% took: 0.46s
Epoch 1, 29%     train_loss: 0.0840 train_accuracy:82% took: 0.47s
Epoch 1, 39%     train_loss: 0.0940 train_accuracy:80% took: 0.48s
Epoch 1, 49%     train_loss: 0.0765 train_accuracy:84% took: 0.46s
Epoch 1, 59%     train_loss: 0.0869 train_accuracy:79% took: 0.47s
Epoch 1, 69%     train_loss: 0.0802 train_accuracy:81% took: 0.45s
Epoch 1, 79%     train_loss: 0.0847 train_accuracy:80% took: 0.46s
Epoch 1, 89%     train_loss: 0.0743 train_accuracy:84% took: 0.47s
Epoch 1, 99%     train_loss: 0.0767 train_accuracy:81% took: 0.49s
weight decay:1000.0000 learning rate:0.0010
2
Training Examples -  3003
Validation Examples -  334
Epoch 1, 9%      train_loss: 0.1387 train_accuracy:51% took: 0.59s
Epoch 1, 19%     train_loss: 0.1386 train_accuracy:51% took: 0.45s
Epoch 1, 29%     train_loss: 0.1386 train_accuracy:44% took: 0.45s
Epoch 1, 39%     train_loss: 0.1386 train_accuracy:49% took: 0.45s
Epoch 1, 49%     train_loss: 0.1386 train_accuracy:48% took: 0.47s
Epoch 1, 59%     train_loss: 0.1386 train_accuracy:50% took: 0.47s
Epoch 1, 69%     train_loss: 0.1386 train_accuracy:44% took: 0.45s
Epoch 1, 79%     train_loss: 0.1386 train_accuracy:52% took: 0.49s
Epoch 1, 89%     train_loss: 0.1386 train_accuracy:52% took: 0.55s
Epoch 1, 99%     train_loss: 0.1386 train_accuracy:51% took: 0.51s
weight decay:10000.0000 learning rate:0.0010
2
Training Examples -  3003
Validation Examples -  334
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-67-d241aabf54ac> in <module>()
     22 NN = Two_Layers_NN()
     23 train_net(NN, which_model = "NN", whether_norm = True, batch_size=5, n_epochs=1, learning_rate = learning_rate,
---> 24              weight_decay = weight_decay, print_train_process = True, print_test_process = False, validation = False, l
     25
     26 del NN

<ipython-input-65-6a79802d62e0> in train_net(net, loss_method, which_model, whether_norm, batch_size, n_epochs, learning_rate, w
```

```
     56             loss_size = loss(outputs, labels)
     57             if np.isnan(loss_size.data):
---> 58                 raise ValueError("loss explode due to large regularization or learning rate")
     59
     60             loss_size.backward()

ValueError: loss explode due to large regularization or learning rate
```

SEARCH STACK OVERFLOW

Case 1: As seen, when we increase the weight decay / regularization from 0 to 1000, Loss also increases.

Case 2: when Regularization is 10000, Loss EXPLODES.

Now, Varying the Learning Rate and keeping the Regularisation constant at 0.001.

```
weight_decay = 0.001; learning_rate = 0.000001
print("weight decay:{:.4f} learning rate:{:.8f}".format(weight_decay, learning_rate))
NN = Two_Layers_NN()
train_net(NN, which_model = "NN", whether_norm = True, batch_size=5, n_epochs=1, learning_rate = learning_rate,
              weight_decay = weight_decay, print_train_process = True, print_test_process = False, validation = False, loss_method =
del NN

weight_decay = 0.001; learning_rate = 0.001
print("weight decay:{:.4f} learning rate:{:.8f}".format(weight_decay, learning_rate))
NN = Two_Layers_NN()
train_net(NN, which_model = "NN", whether_norm = True, batch_size=5, n_epochs=1, learning_rate = learning_rate,
              weight_decay = weight_decay, print_train_process = True, print_test_process = False, validation = False, loss_method =

del NN

weight_decay = 0.001; learning_rate = 0.1
print("weight decay:{:.4f} learning rate:{:.8f}".format(weight_decay, learning_rate))
NN = Two_Layers_NN()
train_net(NN, which_model = "NN", whether_norm = True, batch_size=5, n_epochs=1, learning_rate = learning_rate,
```

```
                weight_decay = weight_decay, print_train_process = True, print_test_process = False, validation = False, loss_method =

del NN
```

```
weight decay:0.0010 learning rate:0.00000100
2
Training Examples -  3003
Validation Examples -  334
Epoch 1, 9%      train_loss: 0.1407 train_accuracy:49% took: 0.50s
Epoch 1, 19%     train_loss: 0.1408 train_accuracy:50% took: 0.46s
Epoch 1, 29%     train_loss: 0.1406 train_accuracy:48% took: 0.46s
Epoch 1, 39%     train_loss: 0.1407 train_accuracy:49% took: 0.47s
Epoch 1, 49%     train_loss: 0.1386 train_accuracy:52% took: 0.44s
Epoch 1, 59%     train_loss: 0.1368 train_accuracy:56% took: 0.48s
Epoch 1, 69%     train_loss: 0.1381 train_accuracy:52% took: 0.46s
Epoch 1, 79%     train_loss: 0.1385 train_accuracy:52% took: 0.46s
Epoch 1, 89%     train_loss: 0.1373 train_accuracy:55% took: 0.47s
Epoch 1, 99%     train_loss: 0.1336 train_accuracy:60% took: 0.45s
weight decay:0.0010 learning rate:0.00100000
2
Training Examples -  3003
Validation Examples -  334
Epoch 1, 9%      train_loss: 0.1035 train_accuracy:77% took: 0.49s
Epoch 1, 19%     train_loss: 0.0921 train_accuracy:79% took: 0.43s
Epoch 1, 29%     train_loss: 0.0851 train_accuracy:81% took: 0.44s
Epoch 1, 39%     train_loss: 0.0778 train_accuracy:83% took: 0.47s
Epoch 1, 49%     train_loss: 0.0889 train_accuracy:79% took: 0.50s
Epoch 1, 59%     train_loss: 0.0711 train_accuracy:86% took: 0.51s
Epoch 1, 69%     train_loss: 0.0656 train_accuracy:86% took: 0.47s
Epoch 1, 79%     train_loss: 0.0731 train_accuracy:84% took: 0.44s
Epoch 1, 89%     train_loss: 0.0653 train_accuracy:85% took: 0.48s
Epoch 1, 99%     train_loss: 0.0629 train_accuracy:86% took: 0.45s
weight decay:0.0010 learning rate:0.10000000
2
Training Examples -  3003
Validation Examples -  334
Epoch 1, 9%      train_loss: 460695857490942305969925617457561 6.0000 train_accuracy:44% took: 0.47s
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-68-64bd5c402aad> in <module>()
     18 NN = Two_Layers_NN()
     19 train_net(NN, which_model = "NN", whether_norm = True, batch_size=5, n_epochs=1, learning_rate = learning_rate,
---> 20              weight_decay = weight_decay, print_train_process = True, print_test_process = False, validation = False, l
     21
     22 del NN
```

```
<ipython-input-65-6a79802d62e0> in train_net(net, loss_method, which_model, whether_norm, batch_size, n_epochs, learning_rate, w
     56             loss_size = loss(outputs, labels)
     57             if np.isnan(loss_size.data):
---> 58                 raise ValueError("loss explode due to large regularization or learning rate")
     59
     60             loss_size.backward()

ValueError: loss explode due to large regularization or learning rate
```

SEARCH STACK OVERFLOW

Case 1: When learning rate is 0.000001(too small), Loss barely changes.

Case 2: When learning rate is 0.001, Loss changes reasonably.

Case 3: When learning rate is 0.1, Loss EXPLODES.

**After this, we will do **

Hyperparameter optimization (random search)

```
for count in range(10):
    learning_rate = 10 ** random.uniform(-6,-1)
    weight_decay = 10 ** random.uniform(-6,0)

    NN = Two_Layers_NN()

    try:
        train_net(NN, which_model = "NN", whether_norm = True, batch_size=5, n_epochs=5, learning_rate = learning_rate,
              weight_decay = weight_decay, print_train_process = False, print_test_process = False, validation = True,loss_method = "
    except:
        print("loss explodes. learning rate:{:.8f} regularization:{:.8f}".format(learning_rate, weight_decay))

    del NN
```

```
2
Training Examples -  3003
Validation Examples -  334
Test loss = 0.5192 Test Accuracy = 80%
train_loss: 0.10345407 train_accuracy:79% learning rate:0.00000521 regularization:0.00275003 running time:25.9373
2
Training Examples -  3003
Validation Examples -  334
Test loss = 0.2479 Test Accuracy = 90%
train_loss: 0.05355413 train_accuracy:89% learning rate:0.00564785 regularization:0.21250154 running time:26.1331
2
Training Examples -  3003
Validation Examples -  334
Test loss = 0.1921 Test Accuracy = 93%
train_loss: 0.03042419 train_accuracy:94% learning rate:0.00264795 regularization:0.00263606 running time:25.3011
2
Training Examples -  3003
Validation Examples -  334
Test loss = 0.1610 Test Accuracy = 93%
train_loss: 0.02487139 train_accuracy:95% learning rate:0.00553499 regularization:0.00002068 running time:25.4065
2
Training Examples -  3003
Validation Examples -  334
Test loss = 0.5057 Test Accuracy = 78%
train_loss: 0.10146198 train_accuracy:79% learning rate:0.00000875 regularization:0.03200406 running time:25.8074
2
Training Examples -  3003
Validation Examples -  334
Test loss = 0.1772 Test Accuracy = 91%
train_loss: 0.03561469 train_accuracy:93% learning rate:0.00478636 regularization:0.06362139 running time:25.3221
2
Training Examples -  3003
Validation Examples -  334
Test loss = 0.2202 Test Accuracy = 90%
train_loss: 0.03931881 train_accuracy:93% learning rate:0.00145240 regularization:0.00028463 running time:25.2512
2
Training Examples -  3003
Validation Examples -  334
Test loss = 0.2494 Test Accuracy = 88%
train_loss: 0.02351312 train_accuracy:95% learning rate:0.02027592 regularization:0.00003782 running time:25.4330
2
Training Examples -  3003
```

```
Validation Examples -  334
Test loss = 0.1624 Test Accuracy = 95%
train_loss: 0.03653247 train_accuracy:94% learning rate:0.03080337 regularization:0.00443945 running time:25.4879
2
Training Examples -  3003
Validation Examples -  334
Test loss = 0.3335 Test Accuracy = 86%
train_loss: 0.05732502 train_accuracy:88% learning rate:0.00033360 regularization:0.00914354 running time:25.7065
```

When Learning Rate: 0.03080337 and Regularization:0.00443945,

then train loss is the smallest and accuracy is largest

```
learning_rate = 0.03080337; weight_decay =0.00443945
NN = Two_Layers_NN()
train_net(NN, which_model = "NN", whether_norm = True, batch_size=5, n_epochs=10, learning_rate = learning_rate,
              weight_decay = weight_decay, print_train_process = True, print_test_process = True, validation = True,loss_method = "SG
del NN
```

```
2
Training Examples -  3003
Validation Examples -  334
Epoch 1, 9%      train_loss: 0.1209 train_accuracy:71% took: 0.54s
Epoch 1, 19%     train_loss: 0.0883 train_accuracy:80% took: 0.45s
Epoch 1, 29%     train_loss: 0.0822 train_accuracy:82% took: 0.47s
Epoch 1, 39%     train_loss: 0.0725 train_accuracy:85% took: 0.50s
Epoch 1, 49%     train_loss: 0.0618 train_accuracy:86% took: 0.48s
Epoch 1, 59%     train_loss: 0.0592 train_accuracy:87% took: 0.47s
Epoch 1, 69%     train_loss: 0.0637 train_accuracy:87% took: 0.46s
Epoch 1, 79%     train_loss: 0.0506 train_accuracy:91% took: 0.46s
Epoch 1, 89%     train_loss: 0.0358 train_accuracy:93% took: 0.46s
Epoch 1, 99%     train_loss: 0.0477 train_accuracy:89% took: 0.48s
Test loss = 0.1688 Test Accuracy = 94%
Epoch 2, 9%      train_loss: 0.0412 train_accuracy:92% took: 0.99s
Epoch 2, 19%     train_loss: 0.0345 train_accuracy:92% took: 0.44s
Epoch 2, 29%     train_loss: 0.0540 train_accuracy:90% took: 0.48s
Epoch 2, 39%     train_loss: 0.0391 train_accuracy:94% took: 0.48s
Epoch 2, 49%     train_loss: 0.0532 train_accuracy:91% took: 0.47s
Epoch 2, 59%     train_loss: 0.0432 train_accuracy:92% took: 0.47s
Epoch 2, 69%     train_loss: 0.0732 train_accuracy:88% took: 0.47s
Epoch 2, 79%     train_loss: 0.0443 train_accuracy:91% took: 0.47s
Epoch 2, 89%     train_loss: 0.0421 train_accuracy:93% took: 0.48s
Epoch 2, 99%     train_loss: 0.0372 train_accuracy:92% took: 0.47s
Test loss = 0.1781 Test Accuracy = 93%
Epoch 3, 9%      train_loss: 0.0291 train_accuracy:94% took: 1.00s
Epoch 3, 19%     train_loss: 0.0320 train_accuracy:93% took: 0.50s
Epoch 3, 29%     train_loss: 0.0666 train_accuracy:89% took: 0.46s
Epoch 3, 39%     train_loss: 0.0515 train_accuracy:89% took: 0.45s
Epoch 3, 49%     train_loss: 0.0240 train_accuracy:96% took: 0.47s
Epoch 3, 59%     train_loss: 0.0887 train_accuracy:87% took: 0.47s
Epoch 3, 69%     train_loss: 0.0601 train_accuracy:89% took: 0.46s
Epoch 3, 79%     train_loss: 0.0325 train_accuracy:95% took: 0.52s
Epoch 3, 89%     train_loss: 0.0356 train_accuracy:93% took: 0.53s
Epoch 3, 99%     train_loss: 0.0234 train_accuracy:95% took: 0.47s
Test loss = 0.1506 Test Accuracy = 94%
Epoch 4, 9%      train_loss: 0.0257 train_accuracy:95% took: 1.00s
Epoch 4, 19%     train_loss: 0.0213 train_accuracy:95% took: 0.46s
Epoch 4, 29%     train_loss: 0.0313 train_accuracy:93% took: 0.46s
Epoch 4, 39%     train_loss: 0.0234 train_accuracy:95% took: 0.48s
Epoch 4, 49%     train_loss: 0.3182 train_accuracy:84% took: 0.49s
Epoch 4, 59%     train_loss: 0.0260 train_accuracy:94% took: 0.52s
```

```
Epoch 4, 69%     train_loss: 0.0306 train_accuracy:94% took: 0.49s
Epoch 4, 79%     train_loss: 0.1254 train_accuracy:85% took: 0.45s
Epoch 4, 89%     train_loss: 0.0299 train_accuracy:93% took: 0.48s
Epoch 4, 99%     train_loss: 0.0273 train_accuracy:95% took: 0.47s
Test loss = 0.1659 Test Accuracy = 93%
Epoch 5, 9%      train_loss: 0.0226 train_accuracy:95% took: 0.98s
Epoch 5, 19%     train_loss: 0.0202 train_accuracy:95% took: 0.45s
Epoch 5, 29%     train_loss: 0.0290 train_accuracy:94% took: 0.45s
Epoch 5, 39%     train_loss: 0.0260 train_accuracy:94% took: 0.46s
Epoch 5, 49%     train_loss: 0.0335 train_accuracy:94% took: 0.49s
Epoch 5, 59%     train_loss: 0.0256 train_accuracy:94% took: 0.50s
Epoch 5, 69%     train_loss: 0.0307 train_accuracy:93% took: 0.47s
Epoch 5, 79%     train_loss: 0.0171 train_accuracy:98% took: 0.46s
Epoch 5, 89%     train_loss: 0.0291 train_accuracy:95% took: 0.48s
Epoch 5, 99%     train_loss: 0.0291 train_accuracy:94% took: 0.45s
Test loss = 0.1645 Test Accuracy = 95%
Epoch 6, 9%      train_loss: 0.0209 train_accuracy:96% took: 0.97s
Epoch 6, 19%     train_loss: 0.0223 train_accuracy:96% took: 0.43s
Epoch 6, 29%     train_loss: 0.0262 train_accuracy:94% took: 0.48s
Epoch 6, 39%     train_loss: 0.0262 train_accuracy:95% took: 0.46s
Epoch 6, 49%     train_loss: 0.0248 train_accuracy:95% took: 0.46s
Epoch 6, 59%     train_loss: 0.0341 train_accuracy:94% took: 0.45s
Epoch 6, 69%     train_loss: 0.0260 train_accuracy:94% took: 0.48s
Epoch 6, 79%     train_loss: 0.0486 train_accuracy:90% took: 0.48s
Epoch 6, 89%     train_loss: 0.0270 train_accuracy:95% took: 0.50s
Epoch 6, 99%     train_loss: 0.0209 train_accuracy:96% took: 0.50s
Test loss = 0.1476 Test Accuracy = 95%
Epoch 7, 9%      train_loss: 0.0147 train_accuracy:98% took: 0.96s
Epoch 7, 19%     train_loss: 0.0155 train_accuracy:97% took: 0.47s
Epoch 7, 29%     train_loss: 0.0284 train_accuracy:93% took: 0.47s
Epoch 7, 39%     train_loss: 0.0362 train_accuracy:91% took: 0.46s
Epoch 7, 49%     train_loss: 0.0187 train_accuracy:97% took: 0.47s
Epoch 7, 59%     train_loss: 0.0163 train_accuracy:96% took: 0.47s
Epoch 7, 69%     train_loss: 0.0297 train_accuracy:95% took: 0.52s
Epoch 7, 79%     train_loss: 0.0268 train_accuracy:95% took: 0.46s
Epoch 7, 89%     train_loss: 0.0184 train_accuracy:96% took: 0.52s
Epoch 7, 99%     train_loss: 0.0362 train_accuracy:93% took: 0.47s
Test loss = 0.1555 Test Accuracy = 94%
Epoch 8, 9%      train_loss: 0.0231 train_accuracy:96% took: 0.98s
Epoch 8, 19%     train_loss: 0.0229 train_accuracy:96% took: 0.44s
Epoch 8, 29%     train_loss: 0.0281 train_accuracy:94% took: 0.47s
Epoch 8, 39%     train_loss: 0.0159 train_accuracy:97% took: 0.54s
```

```
Epoch 8, 49%     train_loss: 0.0247 train_accuracy:94% took: 0.55s
Epoch 8, 59%     train_loss: 0.0255 train_accuracy:96% took: 0.51s
Epoch 8, 69%     train_loss: 0.0197 train_accuracy:95% took: 0.50s
Epoch 8, 79%     train_loss: 0.0181 train_accuracy:96% took: 0.46s
Epoch 8, 89%     train_loss: 0.0315 train_accuracy:93% took: 0.45s
Epoch 8, 99%     train_loss: 0.0163 train_accuracy:97% took: 0.48s
Test loss = 0.1663 Test Accuracy = 95%
Epoch 9, 9%      train_loss: 0.0161 train_accuracy:96% took: 0.97s
Epoch 9, 19%     train_loss: 0.0227 train_accuracy:96% took: 0.47s
Epoch 9, 29%     train_loss: 0.0183 train_accuracy:97% took: 0.46s
Epoch 9, 39%     train_loss: 0.0164 train_accuracy:96% took: 0.47s
Epoch 9, 49%     train_loss: 0.0269 train_accuracy:95% took: 0.48s
Epoch 9, 59%     train_loss: 0.0217 train_accuracy:94% took: 0.47s
Epoch 9, 69%     train_loss: 0.0222 train_accuracy:96% took: 0.50s
Epoch 9, 79%     train_loss: 0.0332 train_accuracy:94% took: 0.50s
Epoch 9, 89%     train_loss: 0.0310 train_accuracy:93% took: 0.47s
Epoch 9, 99%     train_loss: 0.0176 train_accuracy:96% took: 0.48s
Test loss = 0.1589 Test Accuracy = 95%
Epoch 10, 9%     train_loss: 0.0209 train_accuracy:95% took: 1.03s
Epoch 10, 19%    train_loss: 0.0295 train_accuracy:94% took: 0.47s
Epoch 10, 29%    train_loss: 0.0178 train_accuracy:96% took: 0.45s
Epoch 10, 39%    train_loss: 0.0214 train_accuracy:95% took: 0.48s
Epoch 10, 49%    train_loss: 0.0146 train_accuracy:97% took: 0.46s
Epoch 10, 59%    train_loss: 0.0161 train_accuracy:97% took: 0.50s
Epoch 10, 69%    train_loss: 0.0193 train_accuracy:97% took: 0.47s
Epoch 10, 79%    train_loss: 0.0156 train_accuracy:96% took: 0.47s
Epoch 10, 89%    train_loss: 0.0302 train_accuracy:96% took: 0.47s
Epoch 10, 99%    train_loss: 0.0274 train_accuracy:95% took: 0.47s
Test loss = 0.1383 Test Accuracy = 95%
```

Sanity Check to LENET5

```
#sanity check
weight_decay = 0; learning_rate = 0.001
print("weight decay:{:.4f} learning rate:{:.8f}".format(weight_decay, learning_rate))
LN = LeNet()
train_net(LN, which_model = "LeNet", whether_norm = True, batch_size=5, n_epochs=1, learning_rate = learning_rate,
              weight_decay = weight_decay, print_train_process = True, print_test_process = False, validation = False,loss_method = "
```

```
del LN

weight_decay = 0.001; learning_rate = 0.001
print("weight decay:{:.4f} learning rate:{:.8f}".format(weight_decay, learning_rate))
LN = LeNet()
train_net(LN, which_model = "LeNet", whether_norm = True, batch_size=5, n_epochs=1, learning_rate = learning_rate,
              weight_decay = weight_decay, print_train_process = True, print_test_process = False, validation = False,loss_method = "

del LN

weight_decay = 10000; learning_rate = 0.001
print("weight decay:{:.4f} learning rate:{:.8f}".format(weight_decay, learning_rate))
LN = LeNet()
train_net(LN, which_model = "LeNet", whether_norm = True, batch_size=5, n_epochs=1, learning_rate = learning_rate,
              weight_decay = weight_decay, print_train_process = True, print_test_process = False, validation = False,loss_method = "

del LN
```

```
weight decay:0.0000 learning rate:0.00100000
4
Training Examples -  3003
Validation Examples -  334
Epoch 1, 9%      train_loss: 0.1388 train_accuracy:46% took: 0.62s
Epoch 1, 19%     train_loss: 0.1375 train_accuracy:56% took: 0.55s
Epoch 1, 29%     train_loss: 0.1363 train_accuracy:78% took: 0.53s
Epoch 1, 39%     train_loss: 0.1345 train_accuracy:82% took: 0.53s
Epoch 1, 49%     train_loss: 0.1332 train_accuracy:80% took: 0.53s
Epoch 1, 59%     train_loss: 0.1303 train_accuracy:83% took: 0.54s
Epoch 1, 69%     train_loss: 0.1286 train_accuracy:80% took: 0.54s
Epoch 1, 79%     train_loss: 0.1263 train_accuracy:78% took: 0.56s
Epoch 1, 89%     train_loss: 0.1194 train_accuracy:80% took: 0.56s
Epoch 1, 99%     train_loss: 0.1137 train_accuracy:78% took: 0.63s
weight decay:0.0010 learning rate:0.00100000
4
Training Examples -  3003
Validation Examples -  334
Epoch 1, 9%      train_loss: 0.1397 train_accuracy:53% took: 0.60s
Epoch 1, 19%     train_loss: 0.1391 train_accuracy:49% took: 0.52s
Epoch 1, 29%     train_loss: 0.1374 train_accuracy:55% took: 0.54s
Epoch 1, 39%     train_loss: 0.1368 train_accuracy:59% took: 0.56s
Epoch 1, 49%     train_loss: 0.1355 train_accuracy:72% took: 0.58s
Epoch 1, 59%     train_loss: 0.1339 train_accuracy:75% took: 0.57s
Epoch 1, 69%     train_loss: 0.1310 train_accuracy:82% took: 0.57s
Epoch 1, 79%     train_loss: 0.1308 train_accuracy:77% took: 0.53s
Epoch 1, 89%     train_loss: 0.1282 train_accuracy:76% took: 0.53s
Epoch 1, 99%     train_loss: 0.1264 train_accuracy:74% took: 0.53s
weight decay:10000.0000 learning rate:0.00100000
4
Training Examples -  3003
Validation Examples -  334
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-71-891dac2a2eb1> in <module>()
     19 LN = LeNet()
     20 train_net(LN, which_model = "LeNet", whether_norm = True, batch_size=5, n_epochs=1, learning_rate = learning_rate,
---> 21              weight_decay = weight_decay, print_train_process = True, print_test_process = False, validation = False,lo
     22
     23 del LN

<ipython-input-65-6a79802d62e0> in train_net(net, loss_method, which_model, whether_norm, batch_size, n_epochs, learning_rate, w
```

```
     56             loss_size = loss(outputs, labels)
     57             if np.isnan(loss_size.data):
---> 58                 raise ValueError("loss explode due to large regularization or learning rate")
     59
     60             loss_size.backward()

ValueError: loss explode due to large regularization or learning rate
```

SEARCH STACK OVERFLOW

Case 1: As Regularization increases, loss also increases. Case 2: When regularization is too large i.e. 10000, LOSS EXPLODES.

```
weight_decay = 0.001; learning_rate = 0.00001
print("weight decay:{:.4f} learning rate:{:.8f}".format(weight_decay, learning_rate))
LN = LeNet()
train_net(LN, which_model = "LeNet", whether_norm = True, batch_size=5, n_epochs=1, learning_rate = learning_rate,
              weight_decay = weight_decay, print_train_process = True, print_test_process = False, validation = False,loss_method = "

del LN

weight_decay = 0.001; learning_rate = 0.001
print("weight decay:{:.4f} learning rate:{:.8f}".format(weight_decay, learning_rate))
LN = LeNet()
train_net(LN, which_model = "LeNet", whether_norm = True, batch_size=5, n_epochs=1, learning_rate = learning_rate,
              weight_decay = weight_decay, print_train_process = True, print_test_process = False, validation = False,loss_method = "

del LN

weight_decay = 0.001; learning_rate = 0.125
print("weight decay:{:.4f} learning rate:{:.8f}".format(weight_decay, learning_rate))
LN = LeNet()
train_net(LN, which_model = "LeNet", whether_norm = True, batch_size=5, n_epochs=1, learning_rate = learning_rate,
              weight_decay = weight_decay, print_train_process = True, print_test_process = False, validation = False,loss_method = "

del LN
```

```
weight decay:0.0010 learning rate:0.00001000
4
Training Examples -  3003
Validation Examples -  334
Epoch 1, 9%      train_loss: 0.1389 train_accuracy:52% took: 0.53s
Epoch 1, 19%     train_loss: 0.1407 train_accuracy:47% took: 0.51s
Epoch 1, 29%     train_loss: 0.1385 train_accuracy:54% took: 0.50s
Epoch 1, 39%     train_loss: 0.1398 train_accuracy:50% took: 0.52s
Epoch 1, 49%     train_loss: 0.1407 train_accuracy:48% took: 0.56s
Epoch 1, 59%     train_loss: 0.1392 train_accuracy:51% took: 0.54s
Epoch 1, 69%     train_loss: 0.1388 train_accuracy:53% took: 0.51s
Epoch 1, 79%     train_loss: 0.1406 train_accuracy:48% took: 0.54s
Epoch 1, 89%     train_loss: 0.1412 train_accuracy:46% took: 0.52s
Epoch 1, 99%     train_loss: 0.1404 train_accuracy:49% took: 0.54s
weight decay:0.0010 learning rate:0.00100000
4
Training Examples -  3003
Validation Examples -  334
Epoch 1, 9%      train_loss: 0.1400 train_accuracy:47% took: 0.54s
Epoch 1, 19%     train_loss: 0.1381 train_accuracy:52% took: 0.52s
Epoch 1, 29%     train_loss: 0.1375 train_accuracy:44% took: 0.50s
Epoch 1, 39%     train_loss: 0.1358 train_accuracy:52% took: 0.50s
Epoch 1, 49%     train_loss: 0.1336 train_accuracy:64% took: 0.52s
Epoch 1, 59%     train_loss: 0.1314 train_accuracy:65% took: 0.55s
Epoch 1, 69%     train_loss: 0.1297 train_accuracy:70% took: 0.54s
Epoch 1, 79%     train_loss: 0.1273 train_accuracy:70% took: 0.52s
Epoch 1, 89%     train_loss: 0.1223 train_accuracy:77% took: 0.50s
Epoch 1, 99%     train_loss: 0.1188 train_accuracy:74% took: 0.50s
weight decay:0.0010 learning rate:0.12500000
4
Training Examples -  3003
Validation Examples -  334
Epoch 1, 9%      train_loss: 0.1257 train_accuracy:68% took: 0.53s
Epoch 1, 19%     train_loss: 0.0971 train_accuracy:82% took: 0.46s
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-73-53ee84d8bd14> in <module>()
     19 LN = LeNet()
     20 train_net(LN, which_model = "LeNet", whether_norm = True, batch_size=5, n_epochs=1, learning_rate = learning_rate,
---> 21               weight_decay = weight_decay, print_train_process = True, print_test_process = False, validation = False,lo
     22
     23 del LN
```

```
<ipython-input-65-6a79802d62e0> in train_net(net, loss_method, which_model, whether_norm, batch_size, n_epochs, learning_rate, w
     56             loss_size = loss(outputs, labels)
     57             if np.isnan(loss_size.data):
---> 58                 raise ValueError("loss explode due to large regularization or learning rate")
     59
     60             loss_size.backward()

ValueError: loss explode due to large regularization or learning rate
```

SEARCH STACK OVERFLOW

Case 1: When learning rate is too small (0.00001), Loss barely changes.

Case 2: When learning rate is 0.001, Loss changes reasonably.

Case 3: When learning rate is too large 0.125 here, loss explodes.

Hyperparameter Optimization(random search) LENET5

```
for count in range(10):
    learning_rate = 10 ** random.uniform(-5,-2)
    weight_decay = 10 ** random.uniform(-6,-1)

    LN = LeNet()

    try:
        train_net(LN, which_model = "LeNet", whether_norm = True, batch_size=5, n_epochs=5, learning_rate = learning_rate,
              weight_decay = weight_decay, print_train_process = False, print_test_process = False, validation = True,loss_method = "
    except:
        print("loss explodes. learning rate:{:.8f} regularization:{:.8f}".format(learning_rate, weight_decay))

    del LN
```

```
4
Training Examples -  3003
Validation Examples -  334
Test loss = 0.1174 Test Accuracy = 96%
train_loss: 0.02429818 train_accuracy:95% learning rate:0.00515260 regularization:0.00002367 running time:28.5694
4
Training Examples -  3003
Validation Examples -  334
Test loss = 0.6808 Test Accuracy = 54%
train_loss: 0.13652195 train_accuracy:55% learning rate:0.00003117 regularization:0.00000141 running time:28.2623
4
Training Examples -  3003
Validation Examples -  334
Test loss = 0.6186 Test Accuracy = 80%
train_loss: 0.12761182 train_accuracy:77% learning rate:0.00010018 regularization:0.00087610 running time:28.6384
4
Training Examples -  3003
Validation Examples -  334
Test loss = 0.6745 Test Accuracy = 74%
train_loss: 0.13559210 train_accuracy:73% learning rate:0.00003850 regularization:0.00799900 running time:28.4200
4
Training Examples -  3003
Validation Examples -  334
Test loss = 0.6765 Test Accuracy = 48%
train_loss: 0.13595977 train_accuracy:47% learning rate:0.00001101 regularization:0.00000662 running time:28.7284
4
Training Examples -  3003
Validation Examples -  334
Test loss = 0.5562 Test Accuracy = 76%
train_loss: 0.10825501 train_accuracy:79% learning rate:0.00017152 regularization:0.00157937 running time:28.4043
4
Training Examples -  3003
Validation Examples -  334
Test loss = 0.6897 Test Accuracy = 53%
train_loss: 0.13809760 train_accuracy:55% learning rate:0.00001328 regularization:0.01522049 running time:28.3302
4
Training Examples -  3003
Validation Examples -  334
Test loss = 0.3677 Test Accuracy = 90%
train_loss: 0.07652792 train_accuracy:86% learning rate:0.00086848 regularization:0.00003098 running time:29.6632
4
Training Examples -  3003
```

```
Validation Examples -  334
Test loss = 0.1491 Test Accuracy = 94%
train_loss: 0.03695062 train_accuracy:94% learning rate:0.00601909 regularization:0.09616404 running time:28.2185
4
Training Examples -  3003
Validation Examples -  334
Test loss = 0.1310 Test Accuracy = 94%
train_loss: 0.02450075 train_accuracy:95% learning rate:0.00654510 regularization:0.00009792 running time:28.4715
```

Choose the pair with highest accuracy rate and smallest loss on LENET5

Learning Rate : 0.00515260 and Regularization : 0.00002367.

```
learning_rate = 0.00515260; weight_decay = 0.00002367
LN = LeNet()
train_net(LN, which_model = "LeNet", whether_norm = True, batch_size=5, n_epochs=10, learning_rate = learning_rate,
              weight_decay = weight_decay, print_train_process = True, print_test_process = True, validation = True,loss_method = "SG
del LN
```

```
4
Training Examples -  3003
Validation Examples -  334
Epoch 1, 9%      train_loss: 0.1338 train_accuracy:64% took: 0.54s
Epoch 1, 19%     train_loss: 0.1149 train_accuracy:80% took: 0.54s
Epoch 1, 29%     train_loss: 0.1029 train_accuracy:80% took: 0.51s
Epoch 1, 39%     train_loss: 0.0821 train_accuracy:83% took: 0.52s
Epoch 1, 49%     train_loss: 0.0943 train_accuracy:80% took: 0.50s
Epoch 1, 59%     train_loss: 0.0734 train_accuracy:86% took: 0.52s
Epoch 1, 69%     train_loss: 0.0794 train_accuracy:86% took: 0.53s
Epoch 1, 79%     train_loss: 0.0713 train_accuracy:88% took: 0.53s
Epoch 1, 89%     train_loss: 0.0659 train_accuracy:88% took: 0.53s
Epoch 1, 99%     train_loss: 0.0668 train_accuracy:91% took: 0.55s
Test loss = 0.2463 Test Accuracy = 93%
Epoch 2, 9%      train_loss: 0.0546 train_accuracy:91% took: 1.07s
Epoch 2, 19%     train_loss: 0.0524 train_accuracy:91% took: 0.48s
Epoch 2, 29%     train_loss: 0.0440 train_accuracy:92% took: 0.52s
Epoch 2, 39%     train_loss: 0.0361 train_accuracy:94% took: 0.53s
Epoch 2, 49%     train_loss: 0.0281 train_accuracy:97% took: 0.55s
Epoch 2, 59%     train_loss: 0.0391 train_accuracy:94% took: 0.59s
Epoch 2, 69%     train_loss: 0.0476 train_accuracy:91% took: 0.52s
Epoch 2, 79%     train_loss: 0.0399 train_accuracy:92% took: 0.52s
Epoch 2, 89%     train_loss: 0.0363 train_accuracy:93% took: 0.52s
Epoch 2, 99%     train_loss: 0.0265 train_accuracy:95% took: 0.52s
Test loss = 0.1349 Test Accuracy = 94%
Epoch 3, 9%      train_loss: 0.0374 train_accuracy:94% took: 1.13s
Epoch 3, 19%     train_loss: 0.0400 train_accuracy:93% took: 0.49s
Epoch 3, 29%     train_loss: 0.0309 train_accuracy:95% took: 0.49s
Epoch 3, 39%     train_loss: 0.0288 train_accuracy:95% took: 0.53s
Epoch 3, 49%     train_loss: 0.0379 train_accuracy:94% took: 0.55s
Epoch 3, 59%     train_loss: 0.0414 train_accuracy:93% took: 0.52s
Epoch 3, 69%     train_loss: 0.0353 train_accuracy:94% took: 0.54s
Epoch 3, 79%     train_loss: 0.0281 train_accuracy:94% took: 0.53s
Epoch 3, 89%     train_loss: 0.0235 train_accuracy:96% took: 0.54s
Epoch 3, 99%     train_loss: 0.0221 train_accuracy:95% took: 0.52s
Test loss = 0.1109 Test Accuracy = 95%
Epoch 4, 9%      train_loss: 0.0194 train_accuracy:97% took: 1.07s
Epoch 4, 19%     train_loss: 0.0367 train_accuracy:95% took: 0.47s
Epoch 4, 29%     train_loss: 0.0264 train_accuracy:93% took: 0.54s
Epoch 4, 39%     train_loss: 0.0331 train_accuracy:95% took: 0.52s
Epoch 4, 49%     train_loss: 0.0386 train_accuracy:94% took: 0.53s
Epoch 4, 59%     train_loss: 0.0223 train_accuracy:95% took: 0.53s
```

```
Epoch 4, 69%     train_loss: 0.0211 train_accuracy:95% took: 0.53s
Epoch 4, 79%     train_loss: 0.0419 train_accuracy:93% took: 0.52s
Epoch 4, 89%     train_loss: 0.0247 train_accuracy:96% took: 0.51s
Epoch 4, 99%     train_loss: 0.0272 train_accuracy:95% took: 0.53s
Test loss = 0.1057 Test Accuracy = 95%
Epoch 5, 9%      train_loss: 0.0257 train_accuracy:96% took: 1.07s
Epoch 5, 19%     train_loss: 0.0333 train_accuracy:94% took: 0.50s
Epoch 5, 29%     train_loss: 0.0350 train_accuracy:95% took: 0.51s
Epoch 5, 39%     train_loss: 0.0271 train_accuracy:95% took: 0.54s
Epoch 5, 49%     train_loss: 0.0214 train_accuracy:96% took: 0.51s
Epoch 5, 59%     train_loss: 0.0215 train_accuracy:96% took: 0.52s
Epoch 5, 69%     train_loss: 0.0303 train_accuracy:94% took: 0.52s
Epoch 5, 79%     train_loss: 0.0219 train_accuracy:95% took: 0.54s
Epoch 5, 89%     train_loss: 0.0186 train_accuracy:95% took: 0.52s
Epoch 5, 99%     train_loss: 0.0203 train_accuracy:96% took: 0.53s
Test loss = 0.0951 Test Accuracy = 96%
Epoch 6, 9%      train_loss: 0.0202 train_accuracy:94% took: 1.08s
Epoch 6, 19%     train_loss: 0.0285 train_accuracy:95% took: 0.53s
Epoch 6, 29%     train_loss: 0.0272 train_accuracy:95% took: 0.51s
Epoch 6, 39%     train_loss: 0.0182 train_accuracy:97% took: 0.52s
Epoch 6, 49%     train_loss: 0.0212 train_accuracy:97% took: 0.55s
Epoch 6, 59%     train_loss: 0.0213 train_accuracy:96% took: 0.52s
Epoch 6, 69%     train_loss: 0.0231 train_accuracy:96% took: 0.53s
Epoch 6, 79%     train_loss: 0.0311 train_accuracy:95% took: 0.53s
Epoch 6, 89%     train_loss: 0.0258 train_accuracy:94% took: 0.54s
Epoch 6, 99%     train_loss: 0.0223 train_accuracy:96% took: 0.51s
Test loss = 0.0962 Test Accuracy = 96%
Epoch 7, 9%      train_loss: 0.0183 train_accuracy:95% took: 1.08s
Epoch 7, 19%     train_loss: 0.0193 train_accuracy:96% took: 0.53s
Epoch 7, 29%     train_loss: 0.0210 train_accuracy:96% took: 0.48s
Epoch 7, 39%     train_loss: 0.0275 train_accuracy:95% took: 0.54s
Epoch 7, 49%     train_loss: 0.0161 train_accuracy:98% took: 0.52s
Epoch 7, 59%     train_loss: 0.0151 train_accuracy:97% took: 0.53s
Epoch 7, 69%     train_loss: 0.0222 train_accuracy:95% took: 0.56s
Epoch 7, 79%     train_loss: 0.0195 train_accuracy:96% took: 0.56s
Epoch 7, 89%     train_loss: 0.0289 train_accuracy:94% took: 0.52s
Epoch 7, 99%     train_loss: 0.0241 train_accuracy:96% took: 0.51s
Test loss = 0.0884 Test Accuracy = 97%
Epoch 8, 9%      train_loss: 0.0199 train_accuracy:96% took: 1.08s
Epoch 8, 19%     train_loss: 0.0288 train_accuracy:95% took: 0.51s
Epoch 8, 29%     train_loss: 0.0226 train_accuracy:96% took: 0.51s
Epoch 8, 39%     train_loss: 0.0181 train_accuracy:96% took: 0.52s
```

```
Epoch 8, 49%     train_loss: 0.0171 train_accuracy:97% took: 0.51s
Epoch 8, 59%     train_loss: 0.0162 train_accuracy:98% took: 0.52s
Epoch 8, 69%     train_loss: 0.0174 train_accuracy:96% took: 0.53s
Epoch 8, 79%     train_loss: 0.0165 train_accuracy:97% took: 0.56s
Epoch 8, 89%     train_loss: 0.0179 train_accuracy:97% took: 0.52s
Epoch 8, 99%     train_loss: 0.0244 train_accuracy:97% took: 0.55s
Test loss = 0.0719 Test Accuracy = 96%
Epoch 9, 9%      train_loss: 0.0246 train_accuracy:96% took: 1.14s
Epoch 9, 19%     train_loss: 0.0193 train_accuracy:96% took: 0.49s
Epoch 9, 29%     train_loss: 0.0174 train_accuracy:96% took: 0.52s
Epoch 9, 39%     train_loss: 0.0238 train_accuracy:97% took: 0.52s
Epoch 9, 49%     train_loss: 0.0136 train_accuracy:98% took: 0.55s
Epoch 9, 59%     train_loss: 0.0187 train_accuracy:96% took: 0.52s
Epoch 9, 69%     train_loss: 0.0131 train_accuracy:97% took: 0.51s
Epoch 9, 79%     train_loss: 0.0100 train_accuracy:97% took: 0.51s
Epoch 9, 89%     train_loss: 0.0262 train_accuracy:95% took: 0.53s
Epoch 9, 99%     train_loss: 0.0145 train_accuracy:97% took: 0.52s
Test loss = 0.0672 Test Accuracy = 97%
Epoch 10, 9%     train_loss: 0.0153 train_accuracy:97% took: 1.05s
Epoch 10, 19%    train_loss: 0.0111 train_accuracy:98% took: 0.53s
Epoch 10, 29%    train_loss: 0.0196 train_accuracy:97% took: 0.51s
Epoch 10, 39%    train_loss: 0.0205 train_accuracy:96% took: 0.53s
Epoch 10, 49%    train_loss: 0.0209 train_accuracy:97% took: 0.56s
Epoch 10, 59%    train_loss: 0.0116 train_accuracy:97% took: 0.56s
Epoch 10, 69%    train_loss: 0.0182 train_accuracy:96% took: 0.57s
Epoch 10, 79%    train_loss: 0.0168 train_accuracy:96% took: 0.55s
Epoch 10, 89%    train_loss: 0.0205 train_accuracy:96% took: 0.55s
Epoch 10, 99%    train_loss: 0.0163 train_accuracy:97% took: 0.56s
Test loss = 0.0696 Test Accuracy = 97%
```