# Project 2 ECE 763 Computer Vision

## Author:

Yiming Wang, ywang225@ncsu.edu

## Aim:

1. Use Neural Network including Feedforward NN and LeNet5, a special CNN, to do face and nonface images classification
2. Babysitting the process and tune parameters.

## Data Preparation:

1. Download Face images here (http://vis-www.cs.umass.edu/fddb/) and crop the face on my own and divide into Train and Test data set
2. Crop nonface images from the background of the images and divide into Train and Test data set.

## Reference:

1. Babysitting process
   231n Convolutional Neural Network (http://cs231n.github.io)
   Code Example (https://medium.com/udacity-pytorch-challengers/ideas-on-how-to-fine-tune-a-pre-trained-model-in-pytorch-184c47185a20)
2. Convolutional Neural Networks
   Code Example (https://blog.algorithmia.com/convolutional-neural-nets-in-pytorch/)
   Tutorial (https://pytorch.org/docs/stable/optim.html)
   Image classification (https://medium.com/datadriveninvestor/creating-a-pytorch-image-classifier-da9db139ba80)
   CNN Image classification (https://medium.com/@vivekvscool/image-classification-cnn-with-pytorch-5b2cb9ef9476)
3. LeNet5
   Tutorial (https://engmrk.com/lenet-5-a-classic-cnn-architecture/)
4. Jupyter Notebook
   Tutorial
   (https://jupyternotebook.readthedocs.io/en/stable/examples/Notebook/Working%20With%20Markdown%

In [2]:
```
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 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 and prepocessing

In [ ]:
```
#os.getcwd()
#os.chdir("Documents/ncsu course/ncsu 2019 spring/ECE/Project 2/")
#os.chdir("..")


assert(os.getcwd()=="/Users/wangyiming/Documents/ncsu course/ncsu 2019 spring/ECE/Project 2")
resolution = 60


Train_root = "resolution"+str(resolution)+"by"+str(resolution)+"/extracted_pics/Train/"
Test_root = "resolution"+str(resolution)+"by"+str(resolution)+"/extracted_pics/Test/"
Train = os.listdir(Train_root)
Test = os.listdir(Test_root)
```

```
#without normalization for simple feedfordward neural network
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,root,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=root, transform=
transform)
    train_loader = torch.utils.data.DataLoader(train_set, batch_size=b
atch_size, shuffle=True, num_workers=2)

    return train_loader
```

Define Optimizer

In [ ]:
```
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, weig
ht_decay=weight_decay)
    if loss_method == "Adam":
        optimizer = optim.Adam(net.parameters(), lr=learning_rate, wei
ght_decay=weight_decay)
    return(loss, optimizer)
```

Define Feedforward Neural Network and LeNet5

In [ ]:
```
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)
```

Define training function

In [ ]:

```
def train_net(net, loss_method, which_model, whether_norm, batch_size,
n_epochs, learning_rate, weight_decay, print_train_process = True, pri
nt_test_process= True, validation = True):
    assert(which_model == "NN" or which_model == "LeNet")# the input s
hould 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 = load_images_flow(batch_size, Train_root, which_tran
s)
    test_loader = load_images_flow(batch_size, Test_root, which_trans)


    n_batches = len(train_loader)
    loss, optimizer =  createLossAndOptimizer(net, learning_rate = lea
rning_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_s
ize pictures

            (inputs,labels) = data

            if(which_model == "NN"):
                inputs = inputs.view(inputs.size()[0],3*60*60)#flatte
n
                #inputs = inputs.view(batch_size,3*60*60) not batch s
ize 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().ite
m()

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

```
                raise ValueError("loss explode due to large regulariza
tion 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), runnin
g_loss/print_every/batch_size, int(100 * running_correct_num /print_ev
ery/batch_size),
                              time.time()-start_time)) # loss for curre
ct 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)

                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}%".forma
t(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}%".form
at(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(running_los
s/total_train_num, int(100 * total_correct_train_num / total_train_num
),
           learning_rate, weight_decay, time.time()-start_time))
```

Compare the results with and without normalization

In [19]:
```
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 = "SGD")
del NN
```

```
weight decay:0.0000 learning rate:0.0010
Epoch 1, 10%     train_loss: 0.1167 train_accuracy:73% took: 0.62s
Epoch 1, 20%     train_loss: 0.0894 train_accuracy:81% took: 0.53s
Epoch 1, 30%     train_loss: 0.0842 train_accuracy:82% took: 0.55s
Epoch 1, 40%     train_loss: 0.0770 train_accuracy:85% took: 0.54s
Epoch 1, 50%     train_loss: 0.0704 train_accuracy:86% took: 0.51s
Epoch 1, 60%     train_loss: 0.0728 train_accuracy:86% took: 0.50s
Epoch 1, 70%     train_loss: 0.0635 train_accuracy:89% took: 0.52s
Epoch 1, 80%     train_loss: 0.0587 train_accuracy:90% took: 0.51s
Epoch 1, 90%     train_loss: 0.0585 train_accuracy:92% took: 0.50s
Epoch 1, 100%    train_loss: 0.0558 train_accuracy:92% took: 0.51s
```

Without normalization, optimization converge slow.

Sanity check

In [4]:
```
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 = "SGD")

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 = "SGD")

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 = "SGD")

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

```
weight decay:0.0000 learning rate:0.0010
Epoch 1, 10%     train_loss: 0.1054 train_accuracy:72% took: 0.66s
Epoch 1, 20%     train_loss: 0.0902 train_accuracy:78% took: 0.55s
Epoch 1, 30%     train_loss: 0.0783 train_accuracy:85% took: 0.64s
Epoch 1, 40%     train_loss: 0.0741 train_accuracy:84% took: 0.54s
Epoch 1, 50%     train_loss: 0.0733 train_accuracy:85% took: 0.64s
Epoch 1, 60%     train_loss: 0.0706 train_accuracy:86% took: 0.58s
Epoch 1, 70%     train_loss: 0.0624 train_accuracy:89% took: 0.54s
Epoch 1, 80%     train_loss: 0.0659 train_accuracy:87% took: 0.51s
Epoch 1, 90%     train_loss: 0.0594 train_accuracy:89% took: 0.52s
Epoch 1, 100%    train_loss: 0.0610 train_accuracy:90% took: 0.53s
weight decay:1000.0000 learning rate:0.0010
Epoch 1, 10%     train_loss: 0.1386 train_accuracy:49% took: 0.96s
Epoch 1, 20%     train_loss: 0.1386 train_accuracy:51% took: 0.54s
Epoch 1, 30%     train_loss: 0.1386 train_accuracy:50% took: 0.56s
Epoch 1, 40%     train_loss: 0.1386 train_accuracy:51% took: 0.55s
Epoch 1, 50%     train_loss: 0.1386 train_accuracy:50% took: 0.52s
Epoch 1, 60%     train_loss: 0.1386 train_accuracy:52% took: 0.52s
Epoch 1, 70%     train_loss: 0.1386 train_accuracy:52% took: 0.56s
Epoch 1, 80%     train_loss: 0.1386 train_accuracy:47% took: 0.66s
Epoch 1, 90%     train_loss: 0.1386 train_accuracy:49% took: 0.54s
Epoch 1, 100%    train_loss: 0.1386 train_accuracy:53% took: 0.54s
weight decay:10000.0000 learning rate:0.0010
```

```
---------------------------------------------------------------------------
-------
ValueError                                Traceback (most recent cal
l last)
<ipython-input-4-21d314271fff> 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_proce
ss = True, print_test_process = False, validation = False, loss_meth
od = "SGD")
     25
     26 del NN

<ipython-input-2-373f73c14a16> in train_net(net, loss_method, which_
model, whether_norm, batch_size, n_epochs, learning_rate, weight_dec
ay, print_train_process, print_test_process, validation)
    207             loss_size = loss(outputs, labels)
    208             if np.isnan(loss_size.data):
--> 209                 raise ValueError("loss explode due to large
regularization or learning rate")
    210
    211             loss_size.backward()

ValueError: loss explode due to large regularization or learning rat
e
```

When regularization increases from 0 to 1000, loss also increases. And when regularization is 10000, loss explodes.

In [6]:
```
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 = "SGD")
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 = "SGD")

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 = "SGD")

del NN
```

```
weight decay:0.0010 learning rate:0.00000100
Epoch 1, 10%     train_loss: 0.1412 train_accuracy:48% took: 0.63s
Epoch 1, 20%     train_loss: 0.1392 train_accuracy:50% took: 0.56s
Epoch 1, 30%     train_loss: 0.1405 train_accuracy:49% took: 0.69s
Epoch 1, 40%     train_loss: 0.1384 train_accuracy:52% took: 0.68s
Epoch 1, 50%     train_loss: 0.1382 train_accuracy:55% took: 0.54s
Epoch 1, 60%     train_loss: 0.1377 train_accuracy:53% took: 0.56s
Epoch 1, 70%     train_loss: 0.1372 train_accuracy:56% took: 0.55s
Epoch 1, 80%     train_loss: 0.1374 train_accuracy:55% took: 0.57s
Epoch 1, 90%     train_loss: 0.1355 train_accuracy:58% took: 0.55s
Epoch 1, 100%    train_loss: 0.1353 train_accuracy:58% took: 0.55s
weight decay:0.0010 learning rate:0.00100000
Epoch 1, 10%     train_loss: 0.1112 train_accuracy:68% took: 0.59s
Epoch 1, 20%     train_loss: 0.0876 train_accuracy:77% took: 0.70s
Epoch 1, 30%     train_loss: 0.0839 train_accuracy:79% took: 0.51s
Epoch 1, 40%     train_loss: 0.0801 train_accuracy:81% took: 0.52s
Epoch 1, 50%     train_loss: 0.0776 train_accuracy:81% took: 0.74s
Epoch 1, 60%     train_loss: 0.0740 train_accuracy:83% took: 0.57s
Epoch 1, 70%     train_loss: 0.0715 train_accuracy:83% took: 0.61s
Epoch 1, 80%     train_loss: 0.0639 train_accuracy:87% took: 0.55s
Epoch 1, 90%     train_loss: 0.0629 train_accuracy:86% took: 0.56s
Epoch 1, 100%    train_loss: 0.0624 train_accuracy:87% took: 0.52s
weight decay:0.0010 learning rate:0.10000000
```

```
---------------------------------------------------------------------------
-------
ValueError                                Traceback (most recent cal
l last)
<ipython-input-6-3abc2a5cbdeb> in <module>
     19 NN = Two_Layers_NN()
     20 train_net(NN, which_model = "NN", whether_norm = True, batch
_size=5, n_epochs=1, learning_rate = learning_rate,
---> 21                  weight_decay = weight_decay, print_train_proce
ss = True, print_test_process = False, validation = False, loss_meth
od = "SGD")
     22
     23 del NN

<ipython-input-2-373f73c14a16> in train_net(net, loss_method, which_
model, whether_norm, batch_size, n_epochs, learning_rate, weight_dec
ay, print_train_process, print_test_process, validation)
    207             loss_size = loss(outputs, labels)
    208             if np.isnan(loss_size.data):
--> 209                 raise ValueError("loss explode due to large
regularization or learning rate")
    210
    211             loss_size.backward()

ValueError: loss explode due to large regularization or learning rat
e
```

When learning rate is 0.000001(too small), loss barely changes. When learning rate is 0.001, loss changes reasonably. When learning rate is 0.1, loss explodes.

Hyperparameter optimization (random search)

In [7]:
```
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 = "SGD")
    except:
        print("loss explodes. learning rate:{:.8f} regularization:{:.8f}".format(learning_rate, weight_decay))

    del NN
```

```
Test loss = 0.1346 Test Accuracy = 95%
train_loss: 0.03103231 train_accuracy:94% learning rate:0.01331377 regularization:0.00002323 running time:28.9505
Test loss = 0.1417 Test Accuracy = 95%
train_loss: 0.03743628 train_accuracy:93% learning rate:0.02483877 regularization:0.00006424 running time:28.2589
Test loss = 0.3235 Test Accuracy = 93%
train_loss: 0.07033894 train_accuracy:89% learning rate:0.00026566 regularization:0.41746897 running time:27.7455
Test loss = 0.4991 Test Accuracy = 71%
train_loss: 0.10152265 train_accuracy:72% learning rate:0.00000938 regularization:0.00000534 running time:27.8754
Test loss = 0.2857 Test Accuracy = 91%
train_loss: 0.06496082 train_accuracy:88% learning rate:0.00012364 regularization:0.00941348 running time:31.7366
Test loss = 0.5141 Test Accuracy = 76%
train_loss: 0.10385424 train_accuracy:75% learning rate:0.00000476 regularization:0.00000144 running time:30.7335
Test loss = 0.1700 Test Accuracy = 94%
train_loss: 0.05674806 train_accuracy:89% learning rate:0.02808510 regularization:0.02969595 running time:28.8332
Test loss = 0.2654 Test Accuracy = 93%
train_loss: 0.05979075 train_accuracy:91% learning rate:0.00014072 regularization:0.00003963 running time:29.9148
loss explodes. learning rate:0.07727178 regularization:0.00005829
Test loss = 0.1697 Test Accuracy = 95%
train_loss: 0.04097901 train_accuracy:94% learning rate:0.00052046 regularization:0.00830640 running time:27.9550
```

When learning rate:0.00052046 regularization:0.00830640, train loss is the smallest and accuracy is largest.

In [9]:
```
learning_rate = 0.00052046; weight_decay =0.00830640
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 = "SGD")
del NN
```

```
Epoch 1, 10%     train_loss: 0.1118 train_accuracy:69% took: 0.59s
Epoch 1, 20%     train_loss: 0.0951 train_accuracy:74% took: 0.55s
Epoch 1, 30%     train_loss: 0.0901 train_accuracy:76% took: 0.53s
Epoch 1, 40%     train_loss: 0.0888 train_accuracy:78% took: 0.52s
Epoch 1, 50%     train_loss: 0.0851 train_accuracy:80% took: 0.53s
Epoch 1, 60%     train_loss: 0.0826 train_accuracy:82% took: 0.53s
Epoch 1, 70%     train_loss: 0.0778 train_accuracy:82% took: 0.52s
Epoch 1, 80%     train_loss: 0.0739 train_accuracy:84% took: 0.52s
Epoch 1, 90%     train_loss: 0.0712 train_accuracy:86% took: 0.53s
Epoch 1, 100%    train_loss: 0.0681 train_accuracy:86% took: 0.53s
Test loss = 0.3191 Test Accuracy = 85%
Epoch 2, 10%     train_loss: 0.0695 train_accuracy:86% took: 0.73s
Epoch 2, 20%     train_loss: 0.0628 train_accuracy:88% took: 0.54s
Epoch 2, 30%     train_loss: 0.0626 train_accuracy:88% took: 0.53s
Epoch 2, 40%     train_loss: 0.0583 train_accuracy:89% took: 0.54s
Epoch 2, 50%     train_loss: 0.0656 train_accuracy:88% took: 0.52s
Epoch 2, 60%     train_loss: 0.0656 train_accuracy:87% took: 0.70s
Epoch 2, 70%     train_loss: 0.0615 train_accuracy:89% took: 0.62s
Epoch 2, 80%     train_loss: 0.0604 train_accuracy:90% took: 0.55s
Epoch 2, 90%     train_loss: 0.0587 train_accuracy:90% took: 0.67s
Epoch 2, 100%    train_loss: 0.0571 train_accuracy:90% took: 0.52s
Test loss = 0.2638 Test Accuracy = 91%
Epoch 3, 10%     train_loss: 0.0566 train_accuracy:91% took: 0.82s
Epoch 3, 20%     train_loss: 0.0553 train_accuracy:91% took: 0.53s
Epoch 3, 30%     train_loss: 0.0518 train_accuracy:92% took: 0.74s
Epoch 3, 40%     train_loss: 0.0536 train_accuracy:92% took: 0.53s
Epoch 3, 50%     train_loss: 0.0517 train_accuracy:92% took: 0.52s
Epoch 3, 60%     train_loss: 0.0507 train_accuracy:92% took: 0.54s
Epoch 3, 70%     train_loss: 0.0490 train_accuracy:93% took: 0.87s
Epoch 3, 80%     train_loss: 0.0480 train_accuracy:92% took: 0.54s
Epoch 3, 90%     train_loss: 0.0517 train_accuracy:92% took: 0.59s
Epoch 3, 100%    train_loss: 0.0523 train_accuracy:91% took: 0.79s
Test loss = 0.2137 Test Accuracy = 93%
Epoch 4, 10%     train_loss: 0.0434 train_accuracy:94% took: 0.77s
Epoch 4, 20%     train_loss: 0.0478 train_accuracy:91% took: 0.52s
Epoch 4, 30%     train_loss: 0.0501 train_accuracy:92% took: 0.72s
Epoch 4, 40%     train_loss: 0.0420 train_accuracy:94% took: 0.65s
Epoch 4, 50%     train_loss: 0.0452 train_accuracy:94% took: 0.53s
Epoch 4, 60%     train_loss: 0.0471 train_accuracy:93% took: 0.52s
Epoch 4, 70%     train_loss: 0.0444 train_accuracy:93% took: 0.52s
Epoch 4, 80%     train_loss: 0.0452 train_accuracy:94% took: 0.53s
```

```
Epoch 4, 90%     train_loss: 0.0471 train_accuracy:93% took: 0.71s
Epoch 4, 100%    train_loss: 0.0434 train_accuracy:94% took: 0.58s
Test loss = 0.1848 Test Accuracy = 95%
Epoch 5, 10%     train_loss: 0.0378 train_accuracy:96% took: 0.78s
Epoch 5, 20%     train_loss: 0.0388 train_accuracy:95% took: 0.52s
Epoch 5, 30%     train_loss: 0.0436 train_accuracy:94% took: 0.53s
Epoch 5, 40%     train_loss: 0.0413 train_accuracy:93% took: 0.54s
Epoch 5, 50%     train_loss: 0.0438 train_accuracy:94% took: 0.51s
Epoch 5, 60%     train_loss: 0.0405 train_accuracy:94% took: 0.51s
Epoch 5, 70%     train_loss: 0.0427 train_accuracy:93% took: 0.68s
Epoch 5, 80%     train_loss: 0.0434 train_accuracy:93% took: 0.52s
Epoch 5, 90%     train_loss: 0.0401 train_accuracy:94% took: 0.53s
Epoch 5, 100%    train_loss: 0.0400 train_accuracy:94% took: 0.52s
Test loss = 0.1827 Test Accuracy = 94%
Epoch 6, 10%     train_loss: 0.0348 train_accuracy:95% took: 0.75s
Epoch 6, 20%     train_loss: 0.0393 train_accuracy:94% took: 0.53s
Epoch 6, 30%     train_loss: 0.0387 train_accuracy:93% took: 0.52s
Epoch 6, 40%     train_loss: 0.0369 train_accuracy:95% took: 0.71s
Epoch 6, 50%     train_loss: 0.0395 train_accuracy:93% took: 0.52s
Epoch 6, 60%     train_loss: 0.0434 train_accuracy:93% took: 0.52s
Epoch 6, 70%     train_loss: 0.0360 train_accuracy:95% took: 0.52s
Epoch 6, 80%     train_loss: 0.0357 train_accuracy:96% took: 0.53s
Epoch 6, 90%     train_loss: 0.0348 train_accuracy:94% took: 0.52s
Epoch 6, 100%    train_loss: 0.0404 train_accuracy:93% took: 0.70s
Test loss = 0.1617 Test Accuracy = 95%
Epoch 7, 10%     train_loss: 0.0338 train_accuracy:94% took: 0.82s
Epoch 7, 20%     train_loss: 0.0384 train_accuracy:94% took: 0.63s
Epoch 7, 30%     train_loss: 0.0413 train_accuracy:93% took: 0.52s
Epoch 7, 40%     train_loss: 0.0340 train_accuracy:94% took: 0.58s
Epoch 7, 50%     train_loss: 0.0324 train_accuracy:95% took: 0.53s
Epoch 7, 60%     train_loss: 0.0343 train_accuracy:95% took: 0.52s
Epoch 7, 70%     train_loss: 0.0349 train_accuracy:95% took: 0.54s
Epoch 7, 80%     train_loss: 0.0372 train_accuracy:93% took: 0.54s
Epoch 7, 90%     train_loss: 0.0341 train_accuracy:95% took: 0.65s
Epoch 7, 100%    train_loss: 0.0366 train_accuracy:93% took: 1.04s
Test loss = 0.1497 Test Accuracy = 95%
Epoch 8, 10%     train_loss: 0.0365 train_accuracy:94% took: 0.75s
Epoch 8, 20%     train_loss: 0.0354 train_accuracy:94% took: 0.71s
Epoch 8, 30%     train_loss: 0.0341 train_accuracy:94% took: 0.64s
Epoch 8, 40%     train_loss: 0.0339 train_accuracy:94% took: 0.80s
Epoch 8, 50%     train_loss: 0.0331 train_accuracy:95% took: 0.59s
Epoch 8, 60%     train_loss: 0.0319 train_accuracy:95% took: 0.89s
Epoch 8, 70%     train_loss: 0.0331 train_accuracy:95% took: 0.61s
Epoch 8, 80%     train_loss: 0.0317 train_accuracy:95% took: 0.90s
Epoch 8, 90%     train_loss: 0.0336 train_accuracy:94% took: 0.76s
Epoch 8, 100%    train_loss: 0.0346 train_accuracy:94% took: 0.63s
Test loss = 0.1478 Test Accuracy = 94%
Epoch 9, 10%     train_loss: 0.0328 train_accuracy:94% took: 1.06s
Epoch 9, 20%     train_loss: 0.0346 train_accuracy:94% took: 0.60s
Epoch 9, 30%     train_loss: 0.0307 train_accuracy:95% took: 0.61s
```

```
Epoch 9, 40%     train_loss: 0.0311 train_accuracy:96% took: 0.52s
Epoch 9, 50%     train_loss: 0.0384 train_accuracy:94% took: 0.53s
Epoch 9, 60%     train_loss: 0.0336 train_accuracy:94% took: 0.52s
Epoch 9, 70%     train_loss: 0.0314 train_accuracy:94% took: 0.52s
Epoch 9, 80%     train_loss: 0.0317 train_accuracy:94% took: 0.56s
Epoch 9, 90%     train_loss: 0.0281 train_accuracy:96% took: 0.52s
Epoch 9, 100%    train_loss: 0.0331 train_accuracy:95% took: 0.58s
Test loss = 0.1504 Test Accuracy = 95%
Epoch 10, 10%    train_loss: 0.0321 train_accuracy:95% took: 0.76s
Epoch 10, 20%    train_loss: 0.0314 train_accuracy:95% took: 0.52s
Epoch 10, 30%    train_loss: 0.0333 train_accuracy:95% took: 0.54s
Epoch 10, 40%    train_loss: 0.0329 train_accuracy:94% took: 0.51s
Epoch 10, 50%    train_loss: 0.0309 train_accuracy:95% took: 0.53s
Epoch 10, 60%    train_loss: 0.0312 train_accuracy:94% took: 0.52s
Epoch 10, 70%    train_loss: 0.0288 train_accuracy:95% took: 0.51s
Epoch 10, 80%    train_loss: 0.0262 train_accuracy:96% took: 0.55s
Epoch 10, 90%    train_loss: 0.0304 train_accuracy:94% took: 0.59s
Epoch 10, 100%   train_loss: 0.0360 train_accuracy:94% took: 0.52s
Test loss = 0.1536 Test Accuracy = 95%
```

Sanity check

In [11]:
```
#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 = "SGD")

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 = "SGD")

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 = "SGD")

del LN
```

```
weight decay:0.0000 learning rate:0.00100000
Epoch 1, 10%     train_loss: 0.1384 train_accuracy:48% took: 0.84s
Epoch 1, 20%     train_loss: 0.1356 train_accuracy:54% took: 0.82s
Epoch 1, 30%     train_loss: 0.1336 train_accuracy:65% took: 0.69s
Epoch 1, 40%     train_loss: 0.1305 train_accuracy:70% took: 1.03s
Epoch 1, 50%     train_loss: 0.1269 train_accuracy:71% took: 0.74s
Epoch 1, 60%     train_loss: 0.1217 train_accuracy:73% took: 0.67s
Epoch 1, 70%     train_loss: 0.1100 train_accuracy:77% took: 0.69s
Epoch 1, 80%     train_loss: 0.1001 train_accuracy:80% took: 0.74s
Epoch 1, 90%     train_loss: 0.0911 train_accuracy:84% took: 0.70s
Epoch 1, 100%    train_loss: 0.0859 train_accuracy:83% took: 0.82s
weight decay:0.0010 learning rate:0.00100000
Epoch 1, 10%     train_loss: 0.1368 train_accuracy:55% took: 0.76s
Epoch 1, 20%     train_loss: 0.1345 train_accuracy:67% took: 0.70s
Epoch 1, 30%     train_loss: 0.1292 train_accuracy:70% took: 0.85s
Epoch 1, 40%     train_loss: 0.1275 train_accuracy:69% took: 0.83s
Epoch 1, 50%     train_loss: 0.1219 train_accuracy:72% took: 0.79s
Epoch 1, 60%     train_loss: 0.1154 train_accuracy:73% took: 0.87s
Epoch 1, 70%     train_loss: 0.1102 train_accuracy:73% took: 0.73s
Epoch 1, 80%     train_loss: 0.0984 train_accuracy:82% took: 0.68s
Epoch 1, 90%     train_loss: 0.0884 train_accuracy:84% took: 0.75s
Epoch 1, 100%    train_loss: 0.0760 train_accuracy:86% took: 0.72s
weight decay:10000.0000 learning rate:0.00100000
```

```
---------------------------------------------------------------------------
-------
ValueError                                Traceback (most recent cal
l last)
<ipython-input-11-c846da48a923> in <module>
     20 LN = LeNet()
     21 train_net(LN, which_model = "LeNet", whether_norm = True, ba
tch_size=5, n_epochs=1, learning_rate = learning_rate,
---> 22               weight_decay = weight_decay, print_train_proce
ss = True, print_test_process = False, validation = False,loss_metho
d = "SGD")
     23
     24 del LN

<ipython-input-2-373f73c14a16> in train_net(net, loss_method, which_
model, whether_norm, batch_size, n_epochs, learning_rate, weight_dec
ay, print_train_process, print_test_process, validation)
    207             loss_size = loss(outputs, labels)
    208             if np.isnan(loss_size.data):
--> 209                 raise ValueError("loss explode due to large
regularization or learning rate")
    210
    211             loss_size.backward()

ValueError: loss explode due to large regularization or learning rat
e
```

As regularization increases, loss also increases. When regularization is too large, loss explodes.

In [12]:
```
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 = "SGD")

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 = "SGD")

del LN

weight_decay = 0.001; learning_rate = 0.1
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 = "SGD")

del LN
```

```
weight decay:0.0010 learning rate:0.00001000
Epoch 1, 10%     train_loss: 0.1412 train_accuracy:31% took: 0.81s
Epoch 1, 20%     train_loss: 0.1408 train_accuracy:33% took: 0.69s
Epoch 1, 30%     train_loss: 0.1411 train_accuracy:32% took: 0.69s
Epoch 1, 40%     train_loss: 0.1409 train_accuracy:33% took: 0.70s
Epoch 1, 50%     train_loss: 0.1410 train_accuracy:33% took: 0.70s
Epoch 1, 60%     train_loss: 0.1405 train_accuracy:36% took: 0.70s
Epoch 1, 70%     train_loss: 0.1408 train_accuracy:34% took: 0.68s
Epoch 1, 80%     train_loss: 0.1406 train_accuracy:35% took: 0.67s
Epoch 1, 90%     train_loss: 0.1406 train_accuracy:36% took: 0.68s
Epoch 1, 100%    train_loss: 0.1406 train_accuracy:34% took: 0.81s
weight decay:0.0010 learning rate:0.00100000
Epoch 1, 10%     train_loss: 0.1383 train_accuracy:46% took: 0.74s
Epoch 1, 20%     train_loss: 0.1358 train_accuracy:60% took: 0.69s
Epoch 1, 30%     train_loss: 0.1333 train_accuracy:71% took: 0.68s
Epoch 1, 40%     train_loss: 0.1297 train_accuracy:72% took: 0.69s
Epoch 1, 50%     train_loss: 0.1251 train_accuracy:72% took: 0.68s
Epoch 1, 60%     train_loss: 0.1178 train_accuracy:75% took: 0.67s
Epoch 1, 70%     train_loss: 0.1138 train_accuracy:71% took: 0.72s
Epoch 1, 80%     train_loss: 0.1044 train_accuracy:75% took: 0.89s
Epoch 1, 90%     train_loss: 0.0950 train_accuracy:80% took: 1.00s
Epoch 1, 100%    train_loss: 0.0896 train_accuracy:82% took: 0.75s
weight decay:0.0010 learning rate:0.10000000
```

```
---------------------------------------------------------------------------
-------
ValueError                                Traceback (most recent cal
l last)
<ipython-input-12-e4af65fb3672> in <module>
     20 LN = LeNet()
     21 train_net(LN, which_model = "LeNet", whether_norm = True, ba
tch_size=5, n_epochs=1, learning_rate = learning_rate,
---> 22               weight_decay = weight_decay, print_train_proce
ss = True, print_test_process = False, validation = False,loss_metho
d = "SGD")
     23
     24 del LN

<ipython-input-2-373f73c14a16> in train_net(net, loss_method, which_
model, whether_norm, batch_size, n_epochs, learning_rate, weight_dec
ay, print_train_process, print_test_process, validation)
    207             loss_size = loss(outputs, labels)
    208             if np.isnan(loss_size.data):
--> 209                 raise ValueError("loss explode due to large
regularization or learning rate")
    210
    211             loss_size.backward()

ValueError: loss explode due to large regularization or learning rat
e
```

When learning rate is too small, loss barely changes. When learning rate is too large, loss explodes.

Hyperparameter Optimization(random search)

In [15]:
```
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 = "SGD")
    except:
        print("loss explodes. learning rate:{:.8f} regularization:{:.8f}".format(learning_rate, weight_decay))

    del LN
```

```
Test loss = 0.0937 Test Accuracy = 97%
train_loss: 0.02109754 train_accuracy:96% learning rate:0.00582210 regularization:0.00007245 running time:43.6651
Test loss = 0.2049 Test Accuracy = 92%
train_loss: 0.04114626 train_accuracy:92% learning rate:0.00043608 regularization:0.00243426 running time:39.1342
Test loss = 0.5368 Test Accuracy = 75%
train_loss: 0.10642646 train_accuracy:75% learning rate:0.00012184 regularization:0.00000132 running time:41.3061
Test loss = 0.1220 Test Accuracy = 95%
train_loss: 0.02056650 train_accuracy:96% learning rate:0.00531958 regularization:0.00012685 running time:43.9437
Test loss = 0.4507 Test Accuracy = 83%
train_loss: 0.09391274 train_accuracy:81% learning rate:0.00019317 regularization:0.00000188 running time:44.8019
Test loss = 0.1494 Test Accuracy = 95%
train_loss: 0.02318160 train_accuracy:95% learning rate:0.00367561 regularization:0.00048379 running time:44.1863
Test loss = 0.6241 Test Accuracy = 71%
train_loss: 0.12537603 train_accuracy:73% learning rate:0.00006051 regularization:0.00003437 running time:41.8498
Test loss = 0.6833 Test Accuracy = 54%
train_loss: 0.13626477 train_accuracy:54% learning rate:0.00001902 regularization:0.00005666 running time:45.4452
Test loss = 0.1799 Test Accuracy = 94%
train_loss: 0.03628243 train_accuracy:92% learning rate:0.00064598 regularization:0.00000743 running time:43.1933
Test loss = 0.1173 Test Accuracy = 95%
train_loss: 0.02536073 train_accuracy:95% learning rate:0.00314349 regularization:0.00357941 running time:41.4396
```

Choose the pair with highest accuracy rate and smallest loss.

In [17]:
```
learning_rate = 0.00582210; weight_decay = 0.00007245
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 = "SGD")
del LN
```

```
Epoch 1, 10%     train_loss: 0.1324 train_accuracy:60% took: 0.79s
Epoch 1, 20%     train_loss: 0.0987 train_accuracy:78% took: 0.73s
Epoch 1, 30%     train_loss: 0.0554 train_accuracy:89% took: 0.69s
Epoch 1, 40%     train_loss: 0.0419 train_accuracy:92% took: 0.72s
Epoch 1, 50%     train_loss: 0.0417 train_accuracy:92% took: 0.71s
Epoch 1, 60%     train_loss: 0.0389 train_accuracy:91% took: 0.69s
Epoch 1, 70%     train_loss: 0.0342 train_accuracy:92% took: 0.70s
Epoch 1, 80%     train_loss: 0.0334 train_accuracy:93% took: 0.70s
Epoch 1, 90%     train_loss: 0.0357 train_accuracy:93% took: 0.71s
Epoch 1, 100%    train_loss: 0.0323 train_accuracy:93% took: 0.71s
Test loss = 0.1423 Test Accuracy = 94%
Epoch 2, 10%     train_loss: 0.0256 train_accuracy:94% took: 1.04s
Epoch 2, 20%     train_loss: 0.0308 train_accuracy:94% took: 0.84s
Epoch 2, 30%     train_loss: 0.0320 train_accuracy:93% took: 0.81s
Epoch 2, 40%     train_loss: 0.0309 train_accuracy:94% took: 0.71s
Epoch 2, 50%     train_loss: 0.0291 train_accuracy:93% took: 0.72s
Epoch 2, 60%     train_loss: 0.0248 train_accuracy:94% took: 0.69s
Epoch 2, 70%     train_loss: 0.0253 train_accuracy:95% took: 0.71s
Epoch 2, 80%     train_loss: 0.0272 train_accuracy:94% took: 0.74s
Epoch 2, 90%     train_loss: 0.0293 train_accuracy:95% took: 0.87s
Epoch 2, 100%    train_loss: 0.0234 train_accuracy:96% took: 0.82s
Test loss = 0.1142 Test Accuracy = 96%
Epoch 3, 10%     train_loss: 0.0270 train_accuracy:94% took: 1.11s
Epoch 3, 20%     train_loss: 0.0286 train_accuracy:94% took: 0.79s
Epoch 3, 30%     train_loss: 0.0217 train_accuracy:95% took: 0.74s
Epoch 3, 40%     train_loss: 0.0199 train_accuracy:96% took: 0.74s
Epoch 3, 50%     train_loss: 0.0248 train_accuracy:95% took: 0.74s
Epoch 3, 60%     train_loss: 0.0284 train_accuracy:95% took: 0.88s
Epoch 3, 70%     train_loss: 0.0196 train_accuracy:96% took: 0.80s
Epoch 3, 80%     train_loss: 0.0252 train_accuracy:94% took: 0.74s
Epoch 3, 90%     train_loss: 0.0226 train_accuracy:95% took: 0.71s
Epoch 3, 100%    train_loss: 0.0289 train_accuracy:94% took: 0.79s
Test loss = 0.1144 Test Accuracy = 95%
Epoch 4, 10%     train_loss: 0.0258 train_accuracy:95% took: 1.09s
Epoch 4, 20%     train_loss: 0.0239 train_accuracy:95% took: 0.76s
Epoch 4, 30%     train_loss: 0.0202 train_accuracy:96% took: 0.74s
Epoch 4, 40%     train_loss: 0.0252 train_accuracy:94% took: 0.74s
Epoch 4, 50%     train_loss: 0.0231 train_accuracy:95% took: 0.76s
```

```
Epoch 4, 60%     train_loss: 0.0181 train_accuracy:96% took: 0.86s
Epoch 4, 70%     train_loss: 0.0235 train_accuracy:96% took: 0.86s
Epoch 4, 80%     train_loss: 0.0240 train_accuracy:96% took: 0.77s
Epoch 4, 90%     train_loss: 0.0187 train_accuracy:96% took: 0.75s
Epoch 4, 100%    train_loss: 0.0205 train_accuracy:96% took: 0.73s
Test loss = 0.1025 Test Accuracy = 96%
Epoch 5, 10%     train_loss: 0.0230 train_accuracy:95% took: 1.01s
Epoch 5, 20%     train_loss: 0.0230 train_accuracy:95% took: 0.87s
Epoch 5, 30%     train_loss: 0.0176 train_accuracy:96% took: 0.81s
Epoch 5, 40%     train_loss: 0.0242 train_accuracy:95% took: 0.73s
Epoch 5, 50%     train_loss: 0.0222 train_accuracy:95% took: 0.75s
Epoch 5, 60%     train_loss: 0.0156 train_accuracy:96% took: 0.74s
Epoch 5, 70%     train_loss: 0.0182 train_accuracy:96% took: 0.76s
Epoch 5, 80%     train_loss: 0.0265 train_accuracy:96% took: 0.74s
Epoch 5, 90%     train_loss: 0.0227 train_accuracy:95% took: 0.76s
Epoch 5, 100%    train_loss: 0.0212 train_accuracy:96% took: 0.75s
Test loss = 0.0909 Test Accuracy = 95%
Epoch 6, 10%     train_loss: 0.0170 train_accuracy:96% took: 1.02s
Epoch 6, 20%     train_loss: 0.0189 train_accuracy:96% took: 0.73s
Epoch 6, 30%     train_loss: 0.0176 train_accuracy:97% took: 0.89s
Epoch 6, 40%     train_loss: 0.0205 train_accuracy:96% took: 0.80s
Epoch 6, 50%     train_loss: 0.0182 train_accuracy:96% took: 0.74s
Epoch 6, 60%     train_loss: 0.0203 train_accuracy:96% took: 0.76s
Epoch 6, 70%     train_loss: 0.0272 train_accuracy:94% took: 0.76s
Epoch 6, 80%     train_loss: 0.0207 train_accuracy:96% took: 0.75s
Epoch 6, 90%     train_loss: 0.0175 train_accuracy:96% took: 0.78s
Epoch 6, 100%    train_loss: 0.0271 train_accuracy:94% took: 0.76s
Test loss = 0.0812 Test Accuracy = 97%
Epoch 7, 10%     train_loss: 0.0163 train_accuracy:96% took: 0.99s
Epoch 7, 20%     train_loss: 0.0188 train_accuracy:96% took: 0.76s
Epoch 7, 30%     train_loss: 0.0204 train_accuracy:96% took: 0.87s
Epoch 7, 40%     train_loss: 0.0140 train_accuracy:97% took: 0.74s
Epoch 7, 50%     train_loss: 0.0136 train_accuracy:97% took: 0.81s
Epoch 7, 60%     train_loss: 0.0240 train_accuracy:95% took: 0.89s
Epoch 7, 70%     train_loss: 0.0213 train_accuracy:96% took: 0.97s
Epoch 7, 80%     train_loss: 0.0187 train_accuracy:96% took: 0.74s
Epoch 7, 90%     train_loss: 0.0224 train_accuracy:95% took: 0.74s
Epoch 7, 100%    train_loss: 0.0243 train_accuracy:94% took: 0.72s
Test loss = 0.0873 Test Accuracy = 95%
Epoch 8, 10%     train_loss: 0.0201 train_accuracy:96% took: 1.03s
Epoch 8, 20%     train_loss: 0.0181 train_accuracy:97% took: 0.84s
Epoch 8, 30%     train_loss: 0.0181 train_accuracy:96% took: 0.74s
Epoch 8, 40%     train_loss: 0.0215 train_accuracy:95% took: 0.75s
Epoch 8, 50%     train_loss: 0.0128 train_accuracy:97% took: 0.96s
Epoch 8, 60%     train_loss: 0.0259 train_accuracy:95% took: 0.76s
Epoch 8, 70%     train_loss: 0.0188 train_accuracy:96% took: 0.75s
Epoch 8, 80%     train_loss: 0.0168 train_accuracy:96% took: 0.77s
Epoch 8, 90%     train_loss: 0.0140 train_accuracy:97% took: 0.73s
Epoch 8, 100%    train_loss: 0.0199 train_accuracy:96% took: 0.71s
Test loss = 0.0960 Test Accuracy = 95%
```

```
Epoch 9, 10%     train_loss: 0.0190 train_accuracy:96% took: 0.97s
Epoch 9, 20%     train_loss: 0.0164 train_accuracy:96% took: 0.74s
Epoch 9, 30%     train_loss: 0.0205 train_accuracy:96% took: 0.75s
Epoch 9, 40%     train_loss: 0.0193 train_accuracy:96% took: 1.01s
Epoch 9, 50%     train_loss: 0.0166 train_accuracy:96% took: 0.99s
Epoch 9, 60%     train_loss: 0.0183 train_accuracy:96% took: 0.92s
Epoch 9, 70%     train_loss: 0.0156 train_accuracy:96% took: 1.15s
Epoch 9, 80%     train_loss: 0.0221 train_accuracy:96% took: 0.78s
Epoch 9, 90%     train_loss: 0.0179 train_accuracy:96% took: 0.75s
Epoch 9, 100%    train_loss: 0.0137 train_accuracy:97% took: 0.72s
Test loss = 0.0890 Test Accuracy = 97%
Epoch 10, 10%    train_loss: 0.0191 train_accuracy:96% took: 1.03s
Epoch 10, 20%    train_loss: 0.0117 train_accuracy:98% took: 0.73s
Epoch 10, 30%    train_loss: 0.0161 train_accuracy:96% took: 0.74s
Epoch 10, 40%    train_loss: 0.0175 train_accuracy:96% took: 0.74s
Epoch 10, 50%    train_loss: 0.0157 train_accuracy:97% took: 0.95s
Epoch 10, 60%    train_loss: 0.0171 train_accuracy:97% took: 0.88s
Epoch 10, 70%    train_loss: 0.0195 train_accuracy:96% took: 0.82s
Epoch 10, 80%    train_loss: 0.0159 train_accuracy:96% took: 0.92s
Epoch 10, 90%    train_loss: 0.0216 train_accuracy:96% took: 0.87s
Epoch 10, 100%   train_loss: 0.0143 train_accuracy:97% took: 0.87s
Test loss = 0.0890 Test Accuracy = 96%
```

In [ ]: