Name: Mohit Shailesh Kulkarni

UTA ID: 1002031021

Assignment - 2

LeNet 5:

Convolution Neural Network is the foundation of deep learning algorithms. Lenet 5 is the first CNN and it is used to classify the 2d images in a grayscale method. Lenet was trained on grayscale images and shape of images are 32*32*1. There are 7 layers the first layer will convert the image to 28*28*6 then the layer max pooling will do 14*14*6 then the next conv layer will do 10*10*16 and the pooling will do 5*5*16= 400 and last 3 layers tells about the neurons. This will give us the image as 1d.

Output:

```
LeNet5(
  (conv1): Conv2d(1, 6, kernel_size=(5, 5), stride=(1, 1))
  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
  (conv3): Conv2d(16, 120, kernel_size=(5, 5), stride=(1, 1))
  (linear1): Linear(in_features=120, out_features=84, bias=True)
  (linear2): Linear(in_features=84, out_features=10, bias=True)
  (tanh): Tanh()
  (avgpool): AvgPool2d(kernel_size=2, stride=2, padding=0)
)
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1            [-1, 6, 28, 28]             156
              Tanh-2            [-1, 6, 28, 28]               0
         AvgPool2d-3            [-1, 6, 14, 14]               0
            Conv2d-4           [-1, 16, 10, 10]           2,416
              Tanh-5           [-1, 16, 10, 10]               0
         AvgPool2d-6             [-1, 16, 5, 5]               0
            Conv2d-7            [-1, 120, 1, 1]          48,120
              Tanh-8            [-1, 120, 1, 1]               0
            Linear-9                   [-1, 84]          10,164
             Tanh-10                   [-1, 84]               0
           Linear-11                   [-1, 10]             850
================================================================
Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.11
Params size (MB): 0.24
Estimated Total Size (MB): 0.35
----------------------------------------------------------------
output.shape :  torch.Size([64, 10])
```

Figure A

In Figure A it shows the images conversion in each layer of the Lenet 5 CNN model

```
00:08:09 ---
Epoch: 0/14, Train Loss: 0.2966, Val Loss: 0.1241, Train Acc: 91.2483, Val Acc: 96.2200
00:08:34 ---
Epoch: 1/14, Train Loss: 0.0958, Val Loss: 0.0719, Train Acc: 97.0500, Val Acc: 97.6300
00:08:59 ---
Epoch: 2/14, Train Loss: 0.0645, Val Loss: 0.0694, Train Acc: 98.0400, Val Acc: 97.7000
00:09:24 ---
Epoch: 3/14, Train Loss: 0.0493, Val Loss: 0.0551, Train Acc: 98.4583, Val Acc: 98.2500
00:09:48 ---
Epoch: 4/14, Train Loss: 0.0397, Val Loss: 0.0516, Train Acc: 98.7550, Val Acc: 98.2700
00:10:13 ---
Epoch: 5/14, Train Loss: 0.0317, Val Loss: 0.0451, Train Acc: 98.9500, Val Acc: 98.5200
00:10:38 ---
Epoch: 6/14, Train Loss: 0.0286, Val Loss: 0.0473, Train Acc: 99.1067, Val Acc: 98.5000
00:11:03 ---
Epoch: 7/14, Train Loss: 0.0238, Val Loss: 0.0418, Train Acc: 99.2717, Val Acc: 98.6400
00:11:28 ---
Epoch: 8/14, Train Loss: 0.0214, Val Loss: 0.0395, Train Acc: 99.3050, Val Acc: 98.6300
00:11:54 ---
Epoch: 9/14, Train Loss: 0.0186, Val Loss: 0.0416, Train Acc: 99.4067, Val Acc: 98.6700
00:12:19 ---
Epoch: 10/14, Train Loss: 0.0157, Val Loss: 0.0447, Train Acc: 99.4567, Val Acc: 98.6300
00:12:44 ---
Epoch: 11/14, Train Loss: 0.0138, Val Loss: 0.0451, Train Acc: 99.5633, Val Acc: 98.6300
00:13:09 ---
Epoch: 12/14, Train Loss: 0.0137, Val Loss: 0.0459, Train Acc: 99.5833, Val Acc: 98.7300
00:13:34 ---
Epoch: 13/14, Train Loss: 0.0127, Val Loss: 0.0447, Train Acc: 99.5883, Val Acc: 98.6300
```

Figure B

In Figure B Epoch is set to 14 it shows the 14 epoch iterations the training data loss and validating data loss of every iteration and the accuracy of training and validating dataset of every iteration.

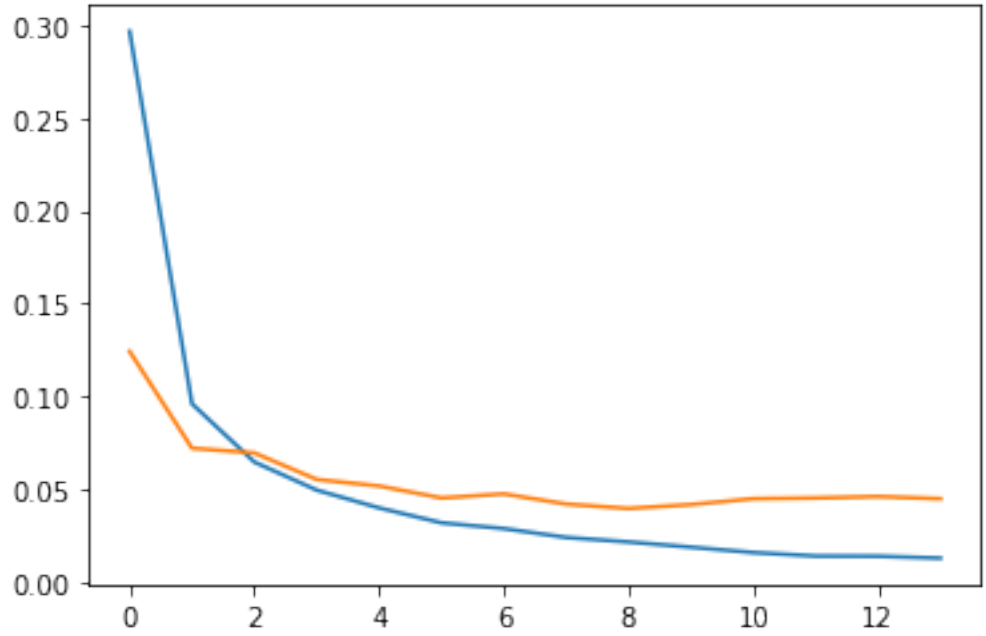


Figure C

In Figure C there is a Plot of Training loss and Validating loss with training loss as blue line and validating loss as red line. As we can see there are some bumps in validating loss as compared to training loss.

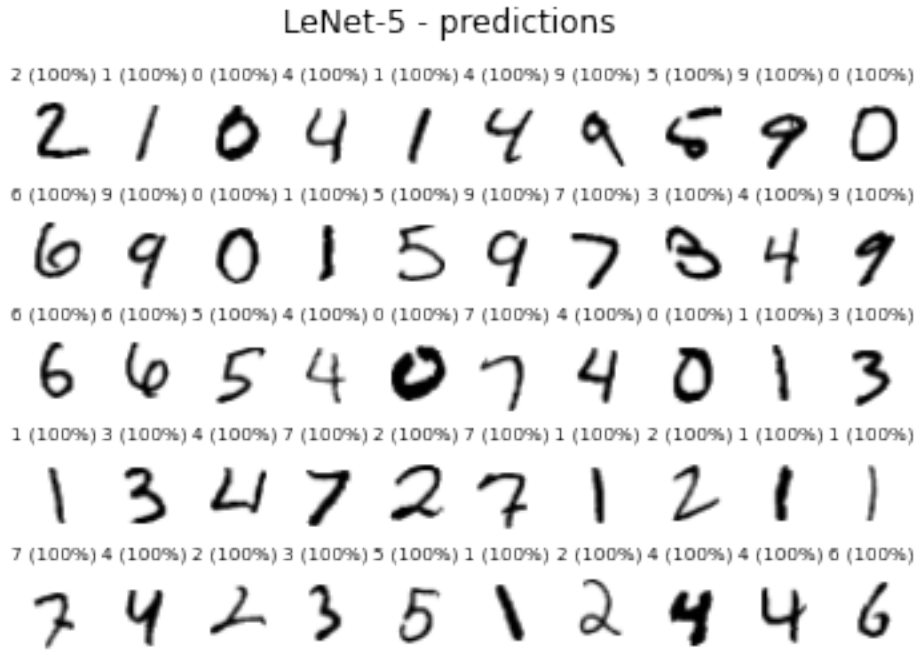


Figure D

In figure D it shows the grid of 5 X 10 of images and shows the output as the number we are getting and the accuracy of the number found. For example in row 1 column 1 the number in the image is 2 and we get the accuracy 100 %.

Analysis:

As per my analysis we have successfully implemented the Lenet 5 CNN in Pytorch on MNIST dataset. At first we have transform the image sizes and which type of device is used. Then there is a Lenet5 function which has the operations for all the 7 layers. Then it shows the epoch iterations and the values which are required as shown in the Figure B. Then It shows the plot of the training loss and validating loss which is shown in the figure C. And at the last it shows the which numbers are extracted form the dataset the accuracy of it and classifying it in various classes.

References:

https://towardsdatascience.com/implementing-yann-lecuns-lenet-5-in-pytorch-5e05a0911320

https://www.analyticsvidhya.com/blog/2021/03/the-architecture-of-lenet-5/

https://datahacker.rs/lenet-5-implementation-tensorflow-2-0/

https://blog.paperspace.com/writing-lenet5-from-scratch-in-python/

Bellow is the screenshot of whole code and output in Google Collab

```
'''
Assignment-2
Name: Mohit Shailesh Kulkarni
UTA ID: 1002031021
References:
https://towardsdatascience.com/implementing-yann-lecuns-lenet-5-in-pytorch-5e05a0911320
https://www.analyticsvidhya.com/blog/2021/03/the-architecture-of-lenet-5/
https://datahacker.rs/lenet-5-implementation-tensorflow-2-0/
'''
import matplotlib.pyplot as plt
import numpy as np
from datetime import datetime

import torch
from torch import nn, optim
from torch.utils.data import DataLoader
from torchvision import datasets, transforms
import torch.nn.functional as F
from torchsummaryX import summary as summaryX
from torchsummary import summary


device = ("cuda" if torch.cuda.is_available() else "cpu") # This is for the trainning from


device
```

```
'cpu'
```

```
transform = transforms.Compose([
          transforms.Resize((32, 32)),
          transforms.ToTensor()
          ])

train_set = datasets.MNIST(root='DATA_MNIST/', download=True, train=True, transform=transf
trainloader = torch.utils.data.DataLoader(train_set, batch_size=64, shuffle=True)

test_set = datasets.MNIST(root='DATA_MNIST/', train=False, transform=transform)
testloader = torch.utils.data.DataLoader(test_set, batch_size=64, shuffle=True)

train_data_size = len(train_set)
test_data_size = len(test_set)


training_data = enumerate(trainloader)
batch_idx, (images, labels) = next(training_data)
print(images.shape)
print(labels.shape)

testing_data = enumerate(testloader)
batch_idx, (images, labels) = next(testing_data)
print(images.shape)
```

```
print(labels.shape)
```

```
torch.Size([64, 1, 32, 32])
torch.Size([64])
torch.Size([64, 1, 32, 32])
torch.Size([64])
```

```
class LeNet5(nn.Module):
  def   init  (self):
    super(LeNet5, self).  init  ()

    self.conv1 = nn.Conv2d(in_channels = 1, out_channels = 6,
                           kernel_size = 5, stride = 1, padding = 0)
    self.conv2 = nn.Conv2d(in_channels = 6, out_channels = 16,
                           kernel_size = 5, stride = 1, padding = 0)
    self.conv3 = nn.Conv2d(in_channels = 16, out_channels = 120,
                           kernel_size = 5, stride = 1, padding = 0)
    self.linear1 = nn.Linear(120, 84)
    self.linear2 = nn.Linear(84, 10)
    self.tanh = nn.Tanh()
    self.avgpool = nn.AvgPool2d(kernel_size = 2, stride = 2)

  def forward(self, out):
    out = self.conv1(out)
    out = self.tanh(out)
    out = self.avgpool(out)
    out = self.conv2(out)
    out = self.tanh(out)
    out = self.avgpool(out)
    out = self.conv3(out)
    out = self.tanh(out)

    out = out.reshape(out.shape[0], -1)
    out = self.linear1(out)
    out = self.tanh(out)
    out = self.linear2(out)
    return out

model = LeNet5()
out = torch.randn(64,1,32,32)
output = model(out)

print(model)
summary(model, (1,32,32))
print("output.shape : ",output.shape)
```

```
LeNet5(
  (conv1): Conv2d(1, 6, kernel_size=(5, 5), stride=(1, 1))
  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
  (conv3): Conv2d(16, 120, kernel_size=(5, 5), stride=(1, 1))
  (linear1): Linear(in_features=120, out_features=84, bias=True)
  (linear2): Linear(in_features=84, out_features=10, bias=True)
  (tanh): Tanh()
  (avgpool): AvgPool2d(kernel_size=2, stride=2, padding=0)
)
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1            [-1, 6, 28, 28]             156
              Tanh-2            [-1, 6, 28, 28]               0
         AvgPool2d-3            [-1, 6, 14, 14]               0
            Conv2d-4           [-1, 16, 10, 10]           2,416
              Tanh-5           [-1, 16, 10, 10]               0
         AvgPool2d-6             [-1, 16, 5, 5]               0
            Conv2d-7            [-1, 120, 1, 1]          48,120
              Tanh-8            [-1, 120, 1, 1]               0
            Linear-9                   [-1, 84]          10,164
             Tanh-10                   [-1, 84]               0
           Linear-11                   [-1, 10]             850
================================================================
Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.11
Params size (MB): 0.24
Estimated Total Size (MB): 0.35
----------------------------------------------------------------
output.shape :  torch.Size([64, 10])
```

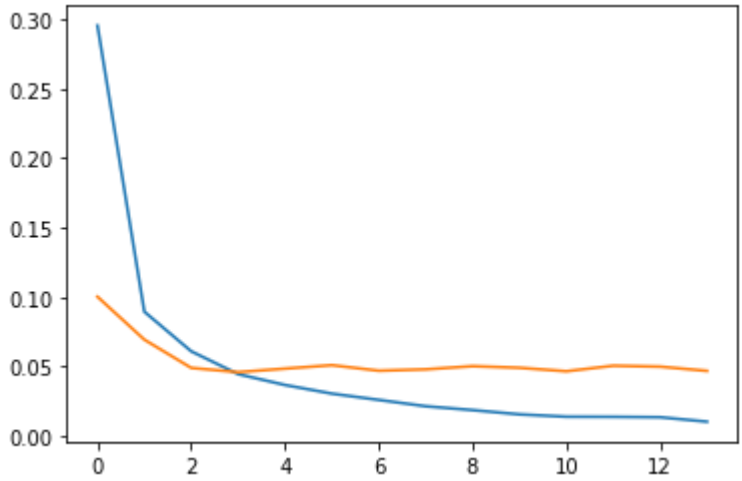
```
optimizer = optim.Adam(model.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss()


epochs = 14
train_loss, val_loss = [], []

for epoch in range(epochs):

    total_train_loss = 0
    total_val_loss = 0

    model.train()
    total = 0

    # training our model
    for idx, (image, label) in enumerate(trainloader):

        image, label = image.to(device), label.to(device)
        optimizer.zero_grad()
        pred = model(image)

        loss = criterion(pred, label)
        total_train_loss += loss.item()

        pred = torch.nn.functional.softmax(pred, dim=1)
        for i, p in enumerate(pred):
            if label[i] == torch.max(p.data, 0)[1]:
                total = total + 1
```

```
            loss.backward()
            optimizer.step()

        total_train_loss = total_train_loss / (idx + 1)
        train_loss.append(total_train_loss)

        # validating our model
        model.eval()
        total1 = 0
        for idx, (image, label) in enumerate(testloader):
            image, label = image.to(device), label.to(device)
            pred = model(image)
            loss = criterion(pred, label)
            total_val_loss += loss.item()

            pred = torch.nn.functional.softmax(pred, dim=1)
            for i, p in enumerate(pred):
                if label[i] == torch.max(p.data, 0)[1]:
                    total1 = total1 + 1

        train_acc = 100 * total / train_data_size
        valid_acc = 100 * total1 / test_data_size

        total_val_loss = total_val_loss / (idx + 1)
        val_loss.append(total_val_loss)

        if epoch % 1 == 0:
            print(f'{datetime.now().time().replace(microsecond=0)} --- ''\nEpoch: {}/{}, Train
```

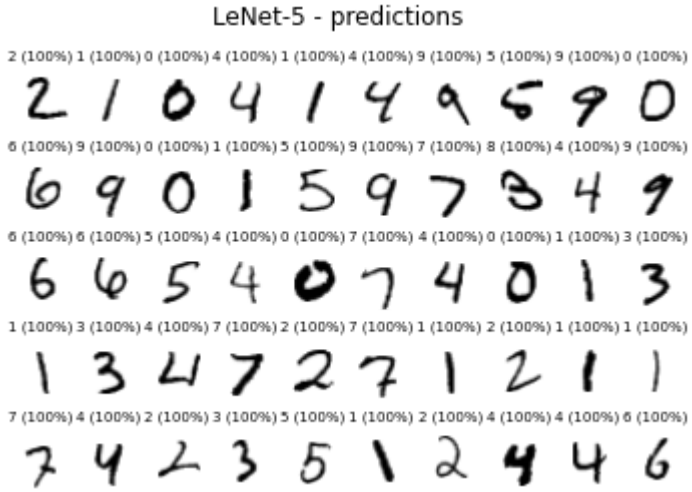
```
 01:42:21 ---
 Epoch: 0/14, Train Loss: 0.2957, Val Loss: 0.1002, Train Acc: 91.1283, Val Acc: 96.9
 01:42:47 ---
 Epoch: 1/14, Train Loss: 0.0894, Val Loss: 0.0693, Train Acc: 97.2483, Val Acc: 97.8
 01:43:12 ---
 Epoch: 2/14, Train Loss: 0.0609, Val Loss: 0.0489, Train Acc: 98.1233, Val Acc: 98.5
 01:43:37 ---
 Epoch: 3/14, Train Loss: 0.0444, Val Loss: 0.0459, Train Acc: 98.5933, Val Acc: 98.5
 01:44:03 ---
 Epoch: 4/14, Train Loss: 0.0366, Val Loss: 0.0484, Train Acc: 98.8917, Val Acc: 98.4
 01:44:28 ---
 Epoch: 5/14, Train Loss: 0.0304, Val Loss: 0.0509, Train Acc: 98.9850, Val Acc: 98.4
 01:44:53 ---
 Epoch: 6/14, Train Loss: 0.0258, Val Loss: 0.0469, Train Acc: 99.1783, Val Acc: 98.5
 01:45:18 ---
 Epoch: 7/14, Train Loss: 0.0213, Val Loss: 0.0479, Train Acc: 99.3000, Val Acc: 98.6
 01:45:42 ---
 Epoch: 8/14, Train Loss: 0.0185, Val Loss: 0.0502, Train Acc: 99.4300, Val Acc: 98.6
 01:46:08 ---
 Epoch: 9/14, Train Loss: 0.0154, Val Loss: 0.0490, Train Acc: 99.5017, Val Acc: 98.5
 01:46:32 ---
 Epoch: 10/14, Train Loss: 0.0137, Val Loss: 0.0465, Train Acc: 99.5783, Val Acc: 98.
 01:46:57 ---
 Epoch: 11/14, Train Loss: 0.0137, Val Loss: 0.0505, Train Acc: 99.5417, Val Acc: 98.
 01:47:22 ---
 Epoch: 12/14, Train Loss: 0.0134, Val Loss: 0.0498, Train Acc: 99.5450, Val Acc: 98.
 01:47:48 ---
 Epoch: 13/14, Train Loss: 0.0102, Val Loss: 0.0468, Train Acc: 99.6550, Val Acc: 98.
```

```
plt.plot(train_loss)
plt.plot(val_loss)
```

```
[<matplotlib.lines.Line2D at 0x7f15109420d0>]
```




```
ROW_IMG = 10
N_ROWS = 5

fig = plt.figure()
for index in range(1, ROW_IMG * N_ROWS + 1):
    plt.subplot(N_ROWS, ROW_IMG, index)
    plt.axis('off')
    plt.imshow(test_set.data[index], cmap='gray_r')

    with torch.no_grad():
        model.eval()
        probs = model(test_set[index][0].unsqueeze(0))

    title = f'{torch.argmax(probs)} (100%)'

    plt.title(title, fontsize=7)
fig.suptitle('LeNet-5 - predictions');
```



Colab paid products - Cancel contracts here

✓ 0s completed at 7:47 PM