RUTGERS UNIVERSITY
School of Engineering
Department of Electrical & Computer Engineering

ECE 472 – Robotics & Computer Vision– Fall 2022

Project 1 - Deep Learning & Image Classification

Name (last, first) : Mehmet Ali Soner

netID : mas996

RUID: 196000499

Date: November 4, 2022

1 Problem 1

ResNet is a Convolutional Neural Network that incorporated residual learning. If we take the activation function of one layer to be $\mathscr{F}(x)$ and the input to be x, then we would have what a skip connection is called:

$$H(x) = \mathscr{F}(x) + x$$

We feed the input of one layer into the next one without modifying it all; so it skips one layer and gets added to the next one. This idea was inspired by how the brain functions. The human brain demonstrates a similar phenomenon. The main problem with previous CNN architectures was the inability to train networks that had more depth. It was expected that with more depth, we would achieve higher accuracy but this wasn't the case with traditional CNNs. The figure above shows the higher error rate of the 56-layered network in comparison to the 20-layered network. There are multiple explanations for this but the one that occurs the most is the vanishing/exploding gradients problem. The derivative of the sigmoid function has a maximum value of 0.25. If we have more layers and if we keep on multiplying the partial derivatives of each layer, the gradients will decrease to a value close to zero. This has performance issues since the weights remain unchanged in the network. ResNets overcame that problem through the skip connections and the ReLU function, which doesn't cause a small derivative. With ResNets, we can train deeper networks, which in the end can achieve higher accuracy.

2 Problem 2

The first modification that is need is in the first convolutional layer. The MNIST dataset doesn't contain any RGB values and only has one dimension in the third dimension of the image matrix. To clarify, an image you read from the MNIST dataset has shape:

```
train_data = datasets.MNIST(
    root = 'data',
    train = True,
    transform = transforms.ToTensor(),
    download = True,
)
print(train_data.data[0].size())
```

So, the first convolutional layer is modified accordingly to

```
model.conv1 = nn.Conv2d(1, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias
                                                =False)
```

To freeze the layers, we use

```
# Freeze layers

for param in model.parameters():
    param.requires_grad = False
```

and to unfreeze the last layer, which we want to modify, we use

```
# Unfreeze and modify last layer

for param in model.fc.parameters():
    param.requires_grad = True
```

Now, the last layer can be changed so it outputs the amount of classes that is in MNIST, which is 10.

```
# Modify number of classes/features, we only got 10 (0,1,2,...9)

num_features = model.fc.in_features
model.fc = nn.Linear(num_features,10)
```

Finally, I defined some hyperparameters, loss functions and optimizers:

```
batch_size = 32
loss_func = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr = 0.001)
num_epochs = 10
```

For the training and evaluation, I have used the same snippet of code for problem 2 and problem 3. For training:

```
# Train model
def train_model(model,num_epochs,loader,loss_func,optimizer):
    model.train()
    for epoch in range(num_epochs):
        losses = []

        for batch_idx, (data, labels) in enumerate(loader):
            # Get data to cuda if possible
            data = data.to(device=device)
            labels = labels.to(device=device)

            # zero gradients
            optimizer.zero_grad()

            # forward
            scores = model(data)

            # calcualte loss + backpropagation
            loss = loss_func(scores, labels)
            losses.append(loss.item())
            loss.backward()

            # adjust learning weights
            optimizer.step()

        print(f"Cost at epoch {epoch} is {sum(losses)/len(losses)}")
    print("Finished training")
```

This code was pretty standard and was found in almost every resource that I used, including the tutorial on PyTorch. In this function, the program iterates through the network for num_ epochs times. Before any operation, the data and the labels are moved to the device. In this case I used a RTX 3090. The three main steps we do in every epoch are going forward, backwards and updating in the network.

With epochs = 3, the model performed at 89.55% accuracy on the training set and 89.73% accuracy on the test set. With epochs = 10, the model performed at 93.33% accuracy on the training set and 92.21% accuracy on the test set.

3 Problem 3

For this problem, I found a dataset which had divided the training set into two folders ”cats” and ”dogs”. This made importing and loading the data a lot easier.

```
# Prepare data
train_data = ImageFolder(root='dog_cat/training_set',transform=preprocess_res)
test_data = ImageFolder(root='dog_cat/test_set',transform=preprocess_res)

# DataLoaders
train_loader = DataLoader(train_data,batch_size=32,shuffle=True)
test_loader = DataLoader(test_data, batch_size=32, shuffle=True)
```

For the networks, I chose ResNet50 and AlexNet. The modification for ResNet50 is almost identical to the one in Problem 2. The only differences are the size of the output classes, which in this case is 2 in comparison to 10 from before and the fact that inputs have RGB values.

```
# Init ResNet50
model = models.resnet50(pretrained=True)
```

```
# Freeze layers
for param in model.parameters():
    param.requires_grad = False

# Unfreeze and modify last layer
for param in model.fc.parameters():
    param.requires_grad = True
num_features = model.fc.in_features
model.fc = nn.Linear(num_features,2)
model = model.to(device)
```

The general idea for the modification of AlexNet is the same. Freeze the layers except the last one and modify the output feature size. However, since the architecture of AlexNet is different from ResNet, there are a few extra steps:

```
# Init AlexNet
model2=models.alexnet(pretrained=True)
# Freeze layers
for param in model2.parameters():
    param.requires_grad = False
# Unfreeze last layer and modify
model2.classifier[4] = nn.Linear(4096,1024)
model2.classifier[6] = nn.Linear(1024,2)
model2 = model2.to(device)
```

The hyperparameters for this problem were the following:

```
# Define loss, optimizer, num_epochs

loss_func = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr = 0.001)
optimizer2 = optim.Adam(model2.parameters(), lr = 0.001)
num_epochs = 3
```

With ResNet50, the model performed at 98.71% accuracy on the training set and 98.02% accuracy on the test set. With AlexNet, the model performed at 98.35% accuracy on the training set and 98.35% accuracy on the test set.