**Project Description:**

This project is focused on mastering the intricacies of deep learning through the development and optimization of a neural network tailored for the task of digit recognition. Utilizing the renowned MNIST dataset, known for its complexity yet accessibility, the project goes through the comprehensive process of building, training, evaluating, and fine-tuning a deep learning model. This hands-on experience aims to demystify the operational mechanics of deep networks, showcasing their capability to interpret and classify high-dimensional data derived from handwritten digits. This project seeks to bridge theoretical knowledge and practical application, providing a robust foundation for future projects in artificial intelligence, data analysis, and beyond, where understanding the nuances of neural network behavior is paramount.

**Task 1: Build and train a network to recognize digits**

Tasks 1A-1D can be found in the task1p1 python notebook. We decided to use notebooks because it is easier to show our outputs in them. Tasks 1E and 1F were done in task1p2 notebook.

1. **Get the MNIST digit data set**

We downloaded the dataset through PyTorch’s Datasets Module and looped through the list of images to display the first 6 digits

**A number in different sizes

Description automatically generated with medium confidence**

1. **Build a network model**

Below is a diagram of the network built. It can be found in networks.py

A diagram of a block diagram

Description automatically generated

1. **Train the model**

­We trained the network for 5 epochs, testing after each epoch and calculated the average loss to better understand the trend. It can be seen that training for more epochs results in lower loss and makes the model more accurate. However, training for more epochs can also increase losses, therefore, the perfect amount of training epochs needs to be determined by the programmer.

**A graph with numbers and a line

Description automatically generated**

1. **Save the network to a file**

We saved the network weights using the torch.save command and giving it the network weights after training.

****

1. **Read the network and run it on the test set**

We used load\_state\_dict to load the weights we had previously saved. We then got the mnist test dataset from pytorch datasets and visualized the inferences from the model. The output was a list of 10 values indicating the model’s confidence in each value. It could be seen that the inference was 98 – 100 sure of the correct value.

**A collage of numbers

Description automatically generated**

1. **Test the network on new inputs**

We tested the network on our handwritten digits. We used paint to write our digit, then we resized the image, inverted it, normalized the values, converted the picture into a tensor and did MNIST normalization on it.

We got 8/10 digits correct on our handwritten dataset and when we did not invert the digits, we got 1/10 digits correct which means that our digits have to be white on black for the MNIST model’s interpretation. This makes sense because the model was trained on only white on black images and had we incorporated a more spectrally diverse dataset, we would have gotten a better output. One more thing to note is that we wrote the 7 in a different way (Like we write in southeast Asian countries sometimes) and the model still recognized it pretty well. That kind of 7 is only repeated 1.2% of the times in the training dataset.

**A number in black squares

Description automatically generated** **A number with numbers and symbols

Description automatically generated with medium confidence**

**Task 2: Examine your network**

The entirety of task 2 and its extensions can be found in task2.ipynb

1. **Analyze the first layer**

We printed out the 10 filters as well as visualized them. It seems that they are detecting edges of different kinds. One of them (Filter 8) had a very dark mask which means it should isolate the background more. The opposite can be said and expected from filter 9.

A screenshot of a computer program

Description automatically generated A screenshot of a computer program

Description automatically generated

A screenshot of a filter

Description automatically generated

1. **Show the effect of the filters**

We applied the filters using opencv filter2D function to the first test image and the filters definitely make sense. Most of them are isolating edges at some angle and the predictions on filter 7 and 9 were also correct as they seem to be isolating different foreground/background elements rather than edges.

**A screenshot of a grid of numbers

Description automatically generated**

**Task 3: Transfer Learning on Greek Letters**

The third task can be found in the task3 notebook. We did transfer learning by freezing the weights on all but the last layer. It took 18 epochs to get a fairly consistent model that got 1/12 test images incorrect. In the image, 0 means it predicted alpha, 1 means beta and 2 means gamma

**A screenshot of a computer program

Description automatically generatedA graph with a line going up

Description automatically generatedA group of black symbols

Description automatically generated**

**Task 4: Design your own experiment**

**Extensions**

The extensions we proceeded with are

* **Replace the first layer of the MNIST network with a filter bank of your choosing (e.g. Gabor filters) and retrain the rest of the network, holding the first layer constant. How does it do?**

Replacing the first layer with 10 Gabor filters allows us to better visualize what the filters are extracting. We also trained the model in the last cell of task1p1 and it can be seen that the network performs slightly worse than the linear filters. This makes sense because the Gabor filters only extract edges and do nothing to the background or foreground.

A graph on a screen

Description automatically generated

A group of squares with numbers

Description automatically generated

* **There are many pre-trained networks available in the PyTorch package. Try loading one and evaluate its first couple of convolutional layers as in task 2.**

We evaluated Resnet18 in the task2 notebook. Below is the structure of the neural network. It has 4 layers with multiple 7x7 filter layers with 64 filters. We visualized the first 10 and it seems that resnet has a wider range of filters, some extract edges, some colours and some look for features that aren’t even in the image I provided. This already makes sense how resnet is very powerful because it is just more thorough with the features it extracts from the image.

ResNet(

(conv1): Conv2d(3, 64, kernel\_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)

(bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(relu): ReLU(inplace=True)

(maxpool): MaxPool2d(kernel\_size=3, stride=2, padding=1, dilation=1, ceil\_mode=False)

(layer1): Sequential(

(0): BasicBlock(

(conv1): Conv2d(64, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(relu): ReLU(inplace=True)

(conv2): Conv2d(64, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

)

(1): BasicBlock(

(conv1): Conv2d(64, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(relu): ReLU(inplace=True)

(conv2): Conv2d(64, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

)

)

(layer2): Sequential(

(0): BasicBlock(

(conv1): Conv2d(64, 128, kernel\_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)

(bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(relu): ReLU(inplace=True)

(conv2): Conv2d(128, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(downsample): Sequential(

(0): Conv2d(64, 128, kernel\_size=(1, 1), stride=(2, 2), bias=False)

(1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

)

)

(1): BasicBlock(

(conv1): Conv2d(128, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(relu): ReLU(inplace=True)

(conv2): Conv2d(128, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

)

)

(layer3): Sequential(

(0): BasicBlock(

(conv1): Conv2d(128, 256, kernel\_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)

(bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(relu): ReLU(inplace=True)

(conv2): Conv2d(256, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(downsample): Sequential(

(0): Conv2d(128, 256, kernel\_size=(1, 1), stride=(2, 2), bias=False)

(1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

)

)

(1): BasicBlock(

(conv1): Conv2d(256, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(relu): ReLU(inplace=True)

(conv2): Conv2d(256, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

)

)

(layer4): Sequential(

(0): BasicBlock(

(conv1): Conv2d(256, 512, kernel\_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)

(bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(relu): ReLU(inplace=True)

(conv2): Conv2d(512, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(downsample): Sequential(

(0): Conv2d(256, 512, kernel\_size=(1, 1), stride=(2, 2), bias=False)

(1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

)

)

(1): BasicBlock(

(conv1): Conv2d(512, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(relu): ReLU(inplace=True)

(conv2): Conv2d(512, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

)

)

(avgpool): AdaptiveAvgPool2d(output\_size=(1, 1))

(fc): Linear(in\_features=512, out\_features=1000, bias=True)

)

A screenshot of a computer generated image

Description automatically generated

**Reflection**

This project taught us a lot about deep learning and networks. It was the first time one of us got to work on pytorch and we were just amazed with how easy it is to just whip up a neural network. Of course, making the network a good one is a difficult task. We had a lot of fun with building our models and although we did not experiment much with hyperparameters like learning rate, we have developed a good foundational understanding of deep networks.

**Acknowledgement**

<https://stackoverflow.com/questions/63746182/correct-way-of-normalizing-and-scaling-the-mnist-dataset>

<https://pytorch.org/tutorials/beginner/basics/tensorqs_tutorial.html>

<https://pytorch.org/tutorials/beginner/basics/data_tutorial.html>

<https://pytorch.org/tutorials/beginner/basics/transforms_tutorial.html>

<https://pytorch.org/tutorials/beginner/basics/buildmodel_tutorial.html>

<https://pytorch.org/tutorials/beginner/basics/autogradqs_tutorial.html>

<https://pytorch.org/tutorials/beginner/basics/optimization_tutorial.html>

<https://pytorch.org/tutorials/beginner/basics/saveloadrun_tutorial.html>