**Week 4 Report Summer Internship**

Name: Mohamed Elfaki

University: Eastern Mediterranean University (Famagusta, Cyprus)

Department: Software Engineering

Training Organization: Cloud Solutions for IT & Communications Co, AI department

**Outline:**

1. Introduction
2. Convolutional Neural Networks Fundamentals
3. CNN From scratch implementation
4. CNN using PyTorch
5. References
6. **Introduction**

This week’s focus will be solely on Convolutional Neural networks. I will start by exploring and explaining the inner workings of a CNN, then go onto trying to implement it by scratch using absolutely no libraries other than NumPy and pandas. Then I will go onto implementing a CNN using PyTorch. The purpose of this is to give me a comprehensive understanding of how CNNs work so when I use PyTorch to implement them, the inner workings aren’t a complete black box.

1. **Convolutional Neural Networks Fundamentals**

Last week we went through regular neural networks. We established many details such as how they consist of neurons having parameters such as weights and biases. As well as the fact that they reside in layers. Now convolutional neural networks aren’t dramatically different from neural networks; they consist of the same neural network but also add a special part called a convolution.

A convolution happens when a filter slides along an image scanning for features, each filter (also called a kernel) focuses on outputting different features such as edges or shapes. Then these features are pooled in order to reduce dimensionality while keeping the important features.

A CNN runs in this order, first a convolution is applied, this is then usually put through a ReLU function and then pooled, and this can be done a number of times usually 3-5 , after this the final product is flattened and sent to the FC (Fully Connected) layer which is essentially our original neural network. The Cross Entropy loss calculates the error. Back propagation happens all the way back to the original image which find all things that causes error from convolution to pooling to the basic neural network and updates the weights accordingly.

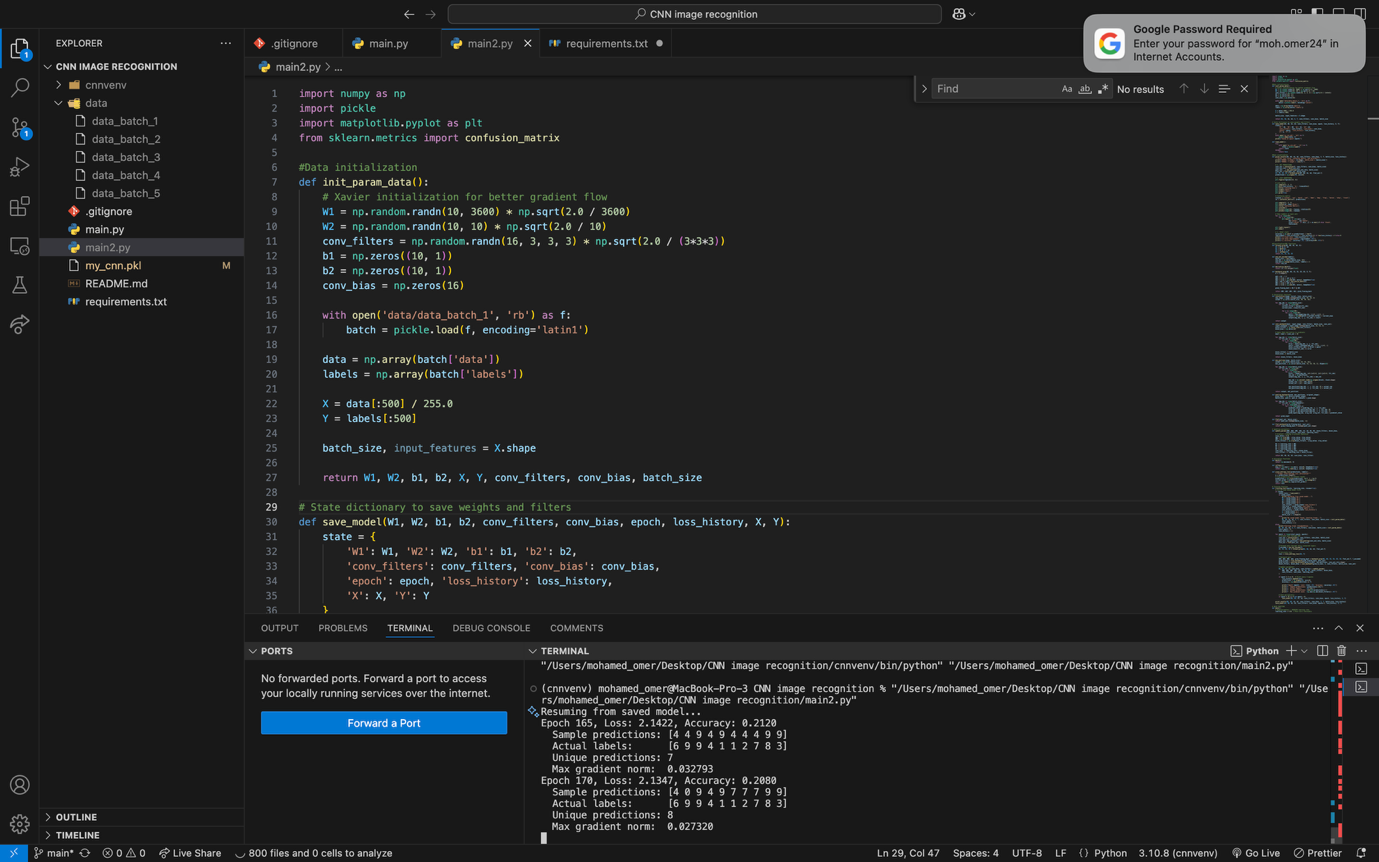
This process runs many times in order to adjust the weights and reduce the loss every epoch.

1. **CNN from scratch Implementation**

To fully understand the basics of CNNs I decided to implement a CNN from scratch, no PyTorch, no TensorFlow, no nothing. Before starting I made a rough sketch of the architecture I planned to use. 1 Convolution layer and 1 NN layer, though I had not realized it yet, this architecture was too simple to give any meaningful results with a dataset as complex as CIFAR-10. But I continued anyways, I made many functions. I used an abstraction-based methodology and made everything I needed into functions, those included forward and backward propagation for my NN as well as convolution layer.

I had a function for initializing parameters, when I completed the code and ran it, I realized something. This code runs too slow; I had used FOUR nested loops and nonetheless python ones with no other libraries to assist me. So I came up with an Idea, I will make function that save the epochs I’ve done so far into a .pkl file, so that I can leave this running overnight and if anything, unexpected happens I always have my saved weights.

After doing this for a couple of hours, I only managed to get to 21% accuracy. The following is an image of me achieving the 21% accuracy in the terminal below along with some of the code:

****

As you can see the accuracy is written as 0.212 which translates to 21.2%, I wasn’t satisfied with this, but I had also understood the idea, I knew to achieve higher accuracy I had to add more convolution layers. But I felt like instead of adding more conv layers from scratch, using PyTorch would be a more practical step now.

1. **CNN using PyTorch**

When I first researched PyTorch and its functions, I realized how good and easy to use it is. The functions that I made which took me a while were made in PyTorch more efficiently and easier to use. That is when I realized how few lines of code were required compared to from scratch method.

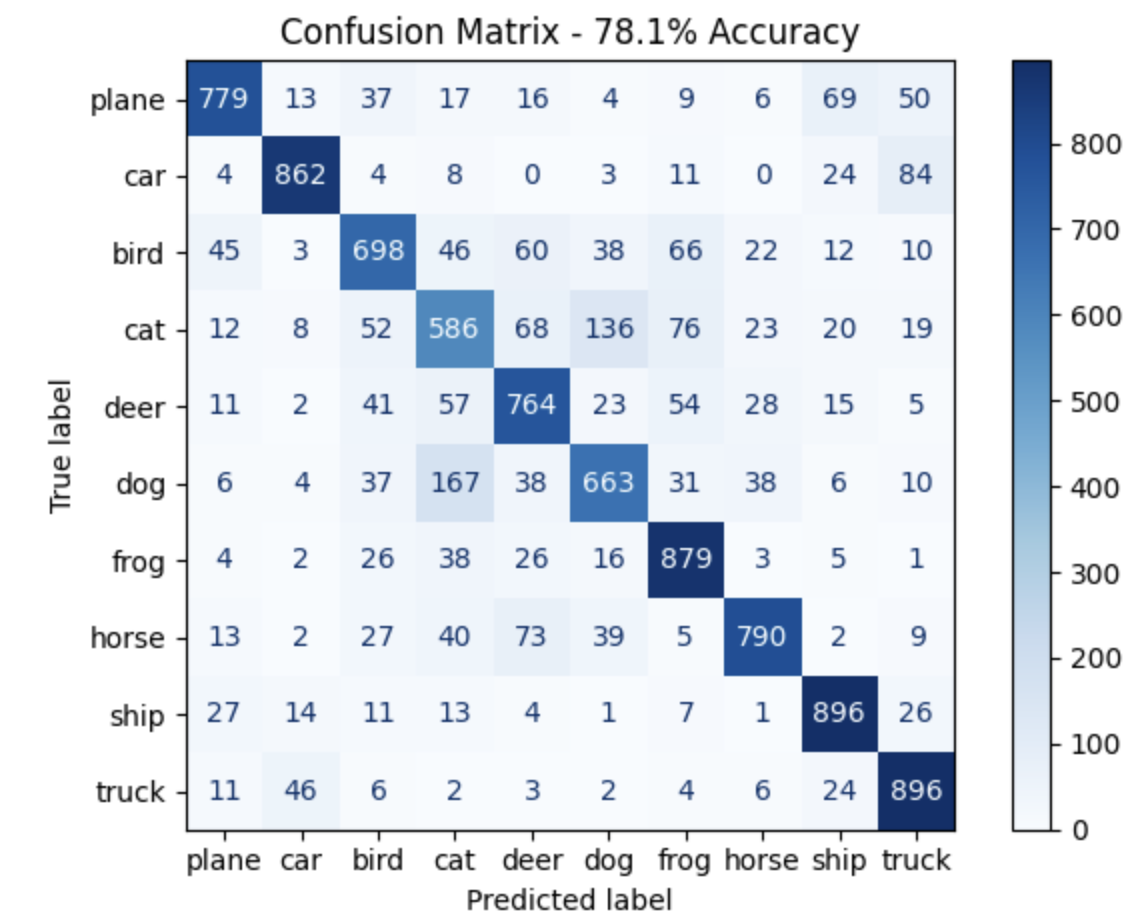
PyTorch is also GPU accelerated thus I was able to use my computers GPU and train my model much faster, this was absolutely amazing as last time I had to wait hours to train my model that achieved 21% accuracy whereas now I only must wait minutes for a model that performs significantly better.

I managed a 78.1% accuracy using a simple CNN in PyTorch. Though I used a 4-layer convolution and 2-layer FC layer it was very simple to implement thanks to PyTorch. I also used a dropout to stop the network to become dependent on certain neurons and find deeper patterns.

The following are the overall statistics of my PyTorch-based CNN:

**A graph of a line

AI-generated content may be incorrect.**

****

1. **References**

* [**https://en.wikipedia.org/wiki/Convolutional\_neural\_network**](https://en.wikipedia.org/wiki/Convolutional_neural_network)
* [**https://taylorandfrancis.com/knowledge/Engineering\_and\_technology/Artificial\_intelligence/CNN/**](https://taylorandfrancis.com/knowledge/Engineering_and_technology/Artificial_intelligence/CNN/)
* [**https://docs.pytorch.org/tutorials/beginner/basics/buildmodel\_tutorial.html**](https://docs.pytorch.org/tutorials/beginner/basics/buildmodel_tutorial.html)
* [**https://pyimagesearch.com/2021/07/19/pytorch-training-your-first-convolutional-neural-network-cnn/**](https://pyimagesearch.com/2021/07/19/pytorch-training-your-first-convolutional-neural-network-cnn/)