**Task 4**

**Introduction:**

For this task I will be creating a neural network using pytorch and analysing how different parameters such as the batch size and the learning rate have an impact on the loss and the accuracy of the model. I will also be analysing how different optimizers affect the outcome of accuracy and the loss of the model, I will be using the optimizers Adam and stochastic gradient descent.

**The Data I am working with:**

The CIFAR-10 dataset has 60000 32x32 images with 10 different classes. So, 6000 images per class. The classes are plane, car, bird, cat, deer, dog, frog, horse, ship and truck and the task are to create a neural network for this dataset. So, my program will see how high I can get my accuracy whilst teaching my model and then testing on a dataset.

The dataset given to us has 2 files. One is the train file which contains 50000 images to train with and the other to test with has 10000 images.

So pytorch tends to work in tensors so I once I import the data I have to transform the data into tensors to be able to run the torch functions on it. To do this I used the transforms.Compose() function from the torchvision.transforms library. The reason we transform to tensors is because we want to be able to run the program on the GPU to speed up testing and have more computing power, but tensors are very similar to numpy arrays, so it is quite familiar to work with.

After creating the function to transform the data I then downloaded the CIFAR10 files to begin working on the code and once I imported it, using torchvision.datasets.CIFAR10, I set up my training and testing datasets so I can validate results at the end after we train the model.

To check I had the correct dataset I decided to experiment with it by displaying to images to see what I am working with, and I made a function that used matplotlib to display random images and got an output like this:

A picture containing text

Description automatically generated

This shows the type of pictures we are working with are very pixilated and helped me to understand the model, what we are working with and gave me ideas of how to maybe lay out the neural network.

Building the Model:

To build the model I did a bit of research and came across this article[[1]](#footnote-1) and the coding they used was in tensor flow, which we are not using in this module, so I just ignored the coding and looked at the diagrams in the article which explained the convolutional neural network and used this diagram to guide me through it

A picture containing chart

Description automatically generated

With the diagram above and lecture 9 I managed to understand the concept of convolutional neural networks and how they work. So, I decided to go with the filter size 5x5 and I used the input size which is 32x32 and the filter size to work out the output size.

The formula (found in lecture 9 slide 46): (wi − F + 2P)/S + 1 = output size where wi is the size of the input so 32x32, F is the filter size which is 5x5, P is the padding (which I do not use so mine is 0) and S which is the stride and mine is 1.

If I substitute into the formula, I end up getting (32 – 5 + 0)/1 + 1 = 28 so my output is 28x28.

After playing around with the conv layers and finding out what the best order is I got these results

Text

Description automatically generated

As you can see from this the original size of the image is 32x32 and after applying the first convoluting neural network we get a size of 28x28 which was calculated above.

Then when you use pooling it halves the image size and we get 14x14 after pooling

Then again, I do the formula above for CNN and I get the answer of 10x10 because

(14 – 5 + 0)/1 + 1 = 10 and we get that answer from doing conv2d once again as shown on the image.

And then we do one final pool, and we end up getting a 5x5 image which is what we were after so it’s the same size as the filter.

After figuring all the mathematics behind it and understanding the model with the layers I developed the model and got this:

Text

Description automatically generated

This forward pass is identical to the diagram I had above where it runs the convolutional layer first followed by pooling with an activation function ReLU followed by the second convolutional layer and then pooled by the activation ReLU once more.

Then I flattened the image using x.view function and run the fully connected layers after this along with the ReLU activation functions to finish the neural network

I also didn’t use the softmax layer because I set it up a different way using cross entropy Loss and I made an optimizer where I train using SGD and Adam and I display results in the next section of how they differ with accuracy and loss.

**Analysis of the model with different parameters:**

I ran tests on my model with different parameters such as how epochs effect the model, how learning rate effects the model, how the batch size effects the model and what optimizers work the best with my convolutional neural network the Adam optimizer or the stochastic gradient descent.

**Batch size:**

Batch size is how many images the program can train on and in theory if you give something more images to train with to learn they can analyse more images and see more similarities between them and be able to distinguish between them more. So, the output I am expecting is the higher the batch size the higher the accuracy of the model when testing so they are directly proportional.

From running my program for one of them was number of epochs was 25 and learning rate of 0.001 for both. The difference is that one had a batch size of 100 and the other a batch size of 20. From investigating the program that ran with the 100 batches performed better and we can see the comparison using a confusion matrix to compare accuracy.

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

100 batch-size

20 batch-size

From the matrices above we can see that the one with the 100-batch size has a higher accuracy than the one with the batch size of 20 and it is what I expected with this outcome.

However, we can still see one pattern that persists even though the batch size has been increased and that is the training for the group cats and dogs and the group truck and car. Both groups have a similar accuracy with both batches, 100-batch is still slightly higher, and this is because of how similar they are because cats and dogs all pixilated are near the same and same as cars and trucks and it still struggles to get high accuracy for those items even though it has 80 more images to learn from so that was an interesting outcome with those 2 classes.

**Number of epochs:**

Higher the epoch higher the accuracy

**Learning rate:**

Smaller the lr the higher the accuracy

**Optimizer:**

**Evaluation of the task:**

documentation

<https://pytorch.org/docs/stable/index.html>

reused this code from the tutorial 9 cactus\_classification to allow me to plot a loss by epoch graph to visualise the learning

train\_loss+=loss.item() \* data.size(0)

train\_loss = train\_loss/len(train\_loader.sampler)

train\_losses.append(train\_loss)

link I used to learn the dataset and to learn how to manipulate my data to make it useable

<https://medium.com/@sergioalves94/deep-learning-in-pytorch-with-cifar-10-dataset-858b504a6b54>

I used this to get an idea of how to lay out my CNN

https://shonit2096.medium.com/cnn-on-cifar10-data-set-using-pytorch-34be87e09844

1. Youtube.com. 2021. [online] Available at: <https://www.youtube.com/watch?v=EFg3u\_E6eHU&ab\_channel=SpanningTree> [Accessed 1 December 2021]. [↑](#footnote-ref-1)