**Task 4**

**Github :** [**https://github.com/dardan-gashi01/DardanGashi\_IN3063coursework**](https://github.com/dardan-gashi01/DardanGashi_IN3063coursework)

**Introduction:**

For this task I will be creating a convolutional neural network using pytorch and analysing how different parameters such as the batch size and the learning rate and number or epochs 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 to investigate this. I will also be creating a dropout to help me understand how dropouts can impact our results and prevent our model from overfitting. I will mostly be exploring the convolutional neural network to understand the layers and how they work to make the classification model more accurate, and I will be describing how it works in the report also with the layers explained and how I laid it out.

**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 must 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:**

Epochs is how many passes we do whilst training and the outcome I am expecting is the more epochs we have the better accuracy we should get because there is more training to be done in a model with 50 epochs compared to a model with 25.

From running my program, I had them both using the same parameters except epochs so for one of them I have the epochs of 50 and for the other one I have 25 epochs and I ran them both one at a time and got their results.

**A picture containing table

Description automatically generated**Graphical user interface, application

Description automatically generated

50 epochs

25 epochs

These 2 confusion matrices have contradicted my prediction and have gone the other way and 25 epochs has given the better accuracy overall compared to the 50 epochs one. Some scores for the 50 epoch one give me better results than the 25 for example the classes truck, frog and dog these 3 have a better result than the 25 epoch one but the other classes plane, car, bird, cat, deer, ship and horse. The 25 epoch one is because it hasn’t trained for the optimal time but for the 50 epochs it’s because it has overfit the model and we can see that in this graph:

Chart, line chart

Description automatically generated

As we can see from this graph, we have the training loss going down totally fine but then the validation loss decreases initially for 4-5 epochs and then increases the rest of the way for the rest of the epochs until 50. Now this isn’t good as due to this it decreased out accuracy of the model and it means that I will need a stopping criterion which will be spoken about later in the report and how I implemented one and how it impacted my results.

**Learning rate:**

Smaller the lr the higher the accuracy

Learning rate is how fast a model trains and the outcome I am expecting is that the lower learning rate which I used 0.001 as my lowest is going to perform better for accuracy than the learning rate with 0.01.

After testing with the same parameters for both model but just changing the learning rate for both so one has the learning rate of 0.01 and the other with 0.001, I got the results of:

**Graphical user interface, application

Description automatically generated**Graphical user interface, application

Description automatically generated

0.01 LR

0.001 LR

From the matrices above we can see that the model with a lower learning rate has performed better than the one with the higher for all classes except the one which is frog.

This is the outcome that was expected because if you have a learning rate that is high it will finish faster but won’t learn as optimal as the lower learning rate model which will take longer but will learn more optimally and get better results on average. So, when creating a model you have to take the costs and weigh them up so the higher learning rate still got good results for a shorter cost of running whereas the other model got better results but for the cost of running for longer so there are downsides in both ways and the person making the model can decide the cost they want to take either running time or accuracy.

**Optimizer:**

I also tested changing the optimizer to see how it would impact the accuracy of the model. For the models above I used the optimizer Adam for all of them and in this section, I will compare the optimizers Adam and Stochastic gradient descent (SGD). I will compare them without using a stopping criterion and will compare the epoch by loss graphs and the confusion matrices to see which one performs better.

Graphical user interface, application

Description automatically generated**Graphical user interface, application

Description automatically generated**

Adam

SGD

From the confusion matrices we can see that the better performing optimizer is the Adam optimizer for the same model as it gets a higher accuracy and performs faster than SGD which gives it the edge in this because it takes a shorter time to run with the addition of getting better overall results. However due to the graphs below we can see that the SGD model is not overfitting like figure 2.1 (Adam model) which does overfit and therefore SGD can be better in that sense.

**Stopping criteria:**

I decided to add a stopping criteria because in most of my model I would get a graph like Adam one above and so it means we would be overfitting and we don’t really want to overfit so I made a piece of code that checks the last 3 entries into the validation loss list and gets the difference between them and if it spots 2 increasing changes then the model will stop training and save results as normal. I done this manually making this function myself.

def difference(arr):

return [item-arr[i-1] for i, item in enumerate(arr) if i != 0]

if len(diff) >= 2:

#print(diff)

if((diff[-1] > 0) and (diff[-2] > 0)):

break

The first function is so it checks the difference of the list and creates a new list with those items in it. Then the second piece of code is putting it into use in the training loop so if the conditions are met then it stops the training with the break keyword. To compare how this impacted my results I tested one model with all same parameters and no stopping criteria and the other with the same parameters and a stopping criterion and compare the results.

In figure 1.1 we have the accuracy after implementing a stopping criterion and in figure 2.1 its before we implemented on and in the results, we can see that the accuracy of the stopping criteria one is better on average and when I run my accuracy calculation, I get an accuracy of 61% and when I run the accuracy calculation in the code without stopping criteria, I get an accuracy of 58%. So, from this we see that implementing the dropout gives us a bit better accuracy due to not overfitting the model like we see in figure 2.2 compared to figure 1.2 which we have the stopping.

**Evaluation of the task:**

Overall, task 4 went well and I managed to create a convolutional neural network using pytorch that uses both optimizers, Adam and SGD, and experimented how parameters and optimizers have an impact on the final result with things such as confusion matrices, to show how accurate or model was, and the epoch loss graph. This helped me visualise how it all works and helped get a better understanding of how the layers in the convolutional neural network work together with different activation functions to get the best results with the model we use. This task also helped me understand how dropout can help our model’s accuracy because it doesn’t overfit the model and how overfitting can have a negative impact on the model at the end. I discussed these all in the report with figures to back up my arguments and used different parameters and discussed how some parameters can affect the others and which ones give the best results overall with figures and graphs I used in this whole report. The programming for this task was interesting to learn and allowed me to further explore neural networks by using a library that has built in functions to make the programming slightly easier as we didn’t have to do any manual work like task 3 and allowed me to experiment more without the worry of breaking the program.

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>

Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generatedChart, line chart

Description automatically generatedGraphical user interface, application

Description automatically generatedA picture containing chart

Description automatically generated

Fig 3.1

Fig 2.2

Fig 1.2

Fig 1.1

Fig 2.1

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)