Trisa Nguyen

ADTA 5550

July 25, 2022

Final Project

**PART I: A Dataset of Image or Audio Files**

Dataset: Oxford-IIIT Pet dataset

Official website:

Dataset features:

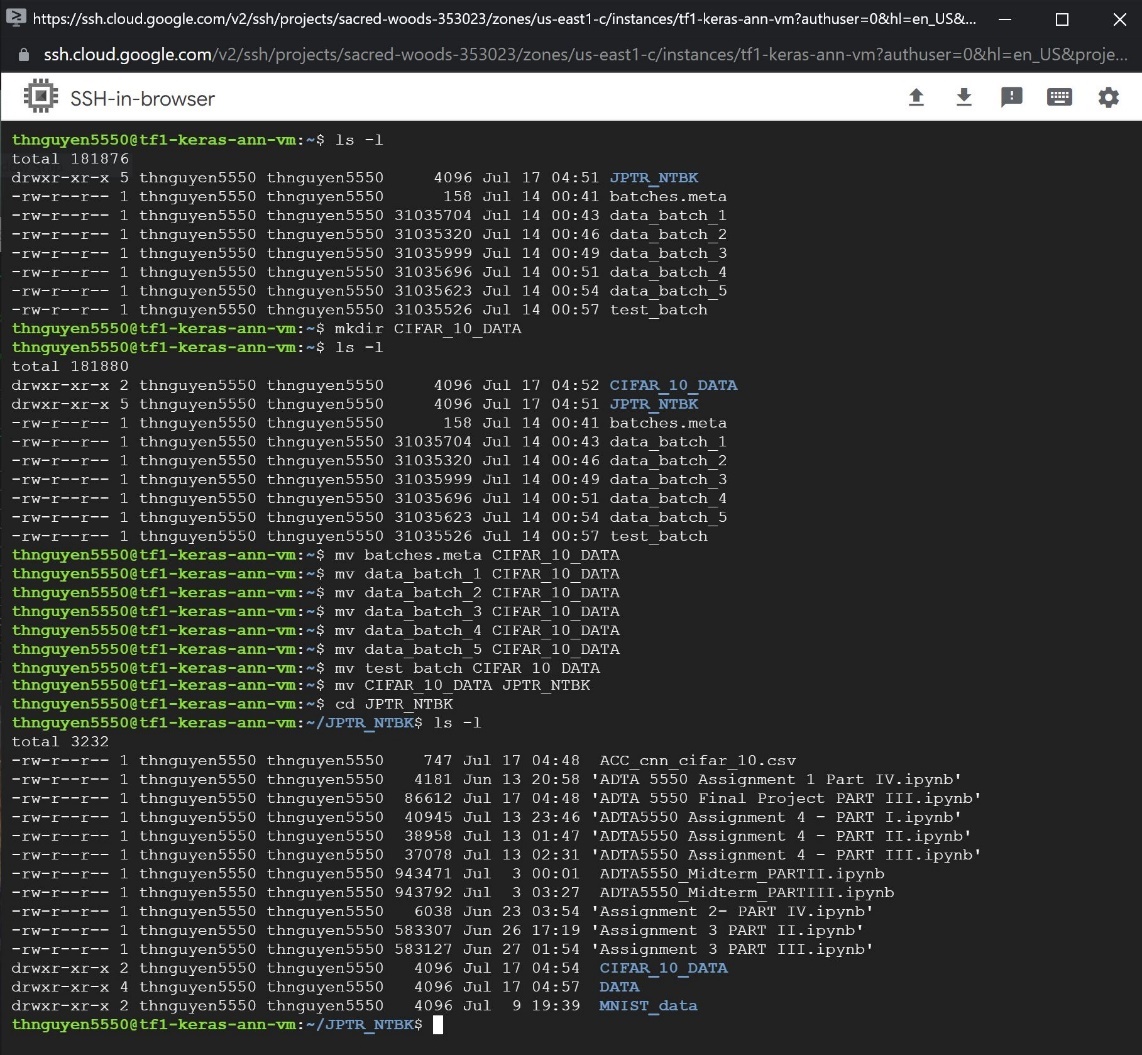
The Oxford-IIIT pet dataset is a 37-category pet image dataset with about 200 images for each class. The images have large variations in scale, pose, and lighting. There is a total of 7,349 labels (pet breeds) with 2 species (cat and dog). The dataset was split into 3,669 for the testing dataset and 3,680 for training dataset. There are 7,345 JPEG and 4 PNG images. 7,346 images are RBG and 3 images are RGBA.

**PART II: Obtain CIFAR-10 Dataset**

Graphical user interface, text

Description automatically generated

In the above image, I opened an SSH instance via GCP and uploaded the 7 files that belong to the CIFAR dataset provided in Canvas.



Then I executed the ls -l command to check that the files uploaded. Next, I created a new directory called “CIFAR\_10\_DATA” and moved all 7 files into the new directory. I then moved CIFAR\_10\_DATA into JPTR\_NTBK and executed the ls -l command again to check that the directory was successfully moved.

**PART III: Build, Train, and Test CNN on CIFAR-10 Dataset**

**CNN Model Architecture**

**A picture containing graphical user interface

Description automatically generated**

**CNN Model Design**

* Convolution Layer 1
  + Convolution shape: [batch, H, W, depth] = [1, 32, 32, 32]
  + 2D data size = 32 x 32
  + Input shape: 32 x 32 x 3
  + Output shape: 32 x 32 x 32
  + Filter/Kernel/Window size = 5 x 5
  + Filter shape = weight shape: [5, 5, 3, 32]
  + Stride = 1
  + Stride: [1, 1, 1, 1]
  + Padding = SAME
  + Activation Function Layer: ReLU
* ReLU Layer 1
  + No filter (only processes data, no extract/learn features)
  + Input shape: 32 x 32 x 32
  + Output shape: 32 x 32 x 32
* Pooling Layer 1
  + Pooling Method: Max pooling
  + Filter/Kernel/Window size = 2 x 2
  + Filter/Kernel/Window shape: [1, 2, 2, 1]
  + Stride = 2
  + Stride shape = [1, 2, 2, 1]
  + Padding = SAME
  + Input channels: 32 inputs
  + Input shape: 32 x 32 x 32
  + Output channels: 32 outputs
  + Output shape: 16 x 16 x 32
* Convolution Layer 2
  + Convolution shape: [batch, H, W, depth] = [1, 16, 16, 64]
  + 2D data size = 16 x 16
  + Input shape: 16 x 16 x 32
  + Output shape: 8 x 8 x 64
  + Filter/Kernel/Window size = 5 x 5
  + Filter shape = weight shape: [5, 5, 32, 64]
  + Stride = 1
  + Stride: [1, 1, 1, 1]
  + Padding = SAME
  + Activation Function Layer: ReLU
* ReLU Layer 2
  + No filter (only processes data, no extract/learn features)
  + Input shape: 16 x 16 x 32
  + Output shape: 16 x 16 x 32
* Pooling Layer 2
  + Pooling Method: Max pooling
  + Filter/Kernel/Window size = 2 x 2
  + Filter/Kernel/Window shape: [1, 2, 2, 1]
  + Stride = 2
  + Stride shape = [1, 2, 2, 1]
  + Padding = SAME
  + Input channels: 64 inputs
  + Input shape: 16 x 16 x 32
  + Output channels: 64 outputs
  + Output shape: 8 x 8 x 64
* Fully Connected (FC) Layer 1
  + FC\_1 shape: [inputs(in\_channels), outputs(out\_channels)]
    - = [8 x 8 x 64, 1024]
* FC Dropout Layer
  + FC\_2 shape: [inputs(in\_channels), outputs(out\_channels)]
    - = [ 1024, 1024]
* FC Final Output Layer
  + FC\_3 shape: [inputs(in\_channels), outputs(out\_channels)]
    - = [1024, 10]

**Results of testing model:**

Table

Description automatically generated with medium confidence

**Report on results of test:**

The result of this 7 layer CNN model with 5000 steps resulted in an average accuracy of 62.92%. The first accuracy check at step 0 was 10% and by the 1000th step, the accuracy jumped to about 61%. For the rest of the steps the accuracy fluctuated between 62%-68%. If we take a look at the average accuracy towards the later end of the phase, at 3000 – 4900 steps, the average accuracy was 68%.

Chart, line chart

Description automatically generated

**PART IV: Compare CNN Performance**

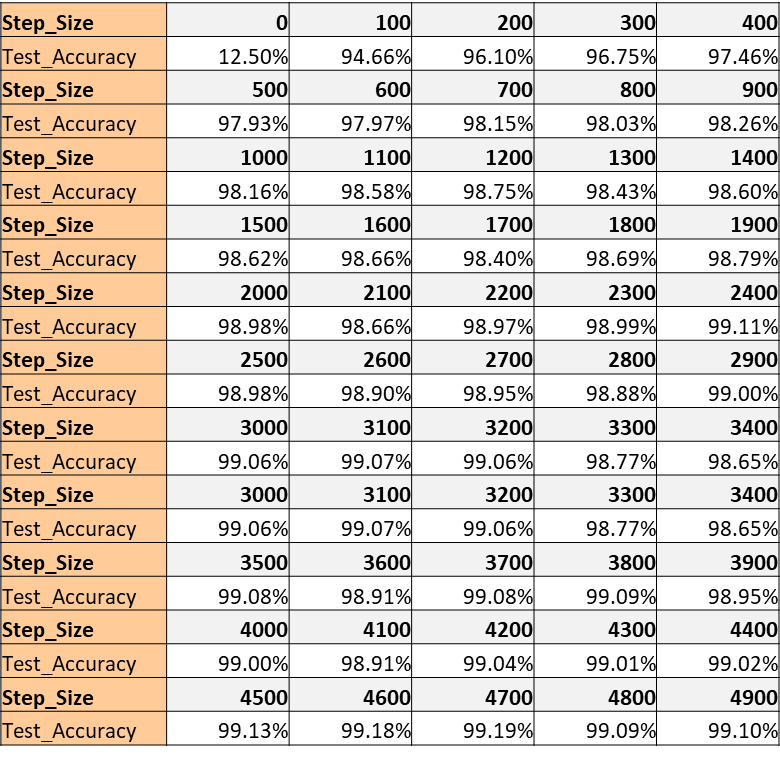
**CIFAR-10**

Table

Description automatically generated with medium confidence**Chart, line chart

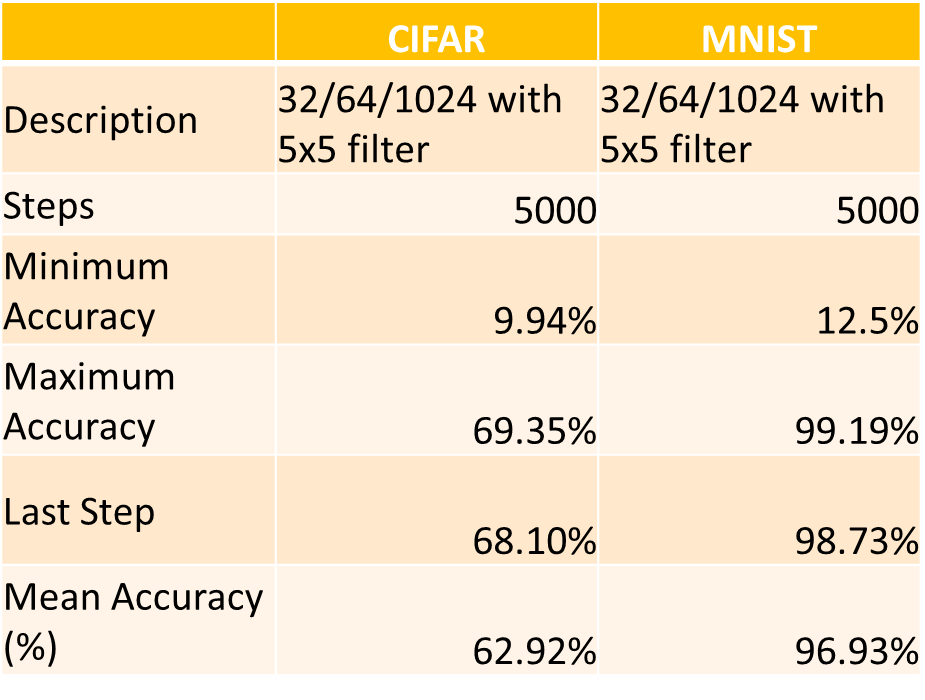
Description automatically generated**

**MNIST**

Chart

Description automatically generated

**Comparison:**



Both CNN models were designed similarly, with 7 layers using 5x5 filters, and trained on 5,000 steps. However, upon looking at the accuracy of both models, there is a noticeable gap in performance between them. By looking at the table in the above, it is evident that the MNIST model outperforms the CIFAR-10 model.

If we compare the datasets used on the CNN models, it provides some plausible reason as to why there is a large performance gap. The MNIST data set images are 28x28 pixels while the CIFAR 10 dataset are 32x32 pixels making the CIFAR 10 images more complex to process due to their size. The MNIST dataset images are also grayscale meaning that it’ll have fewer input channels of either black or white. Meanwhile, the CIFAR-10 data set images are colored and requires 3 input channels - red, green, and blue. Once again this means that the CIFAR 10 images are more complex to process. If we also look at the amount of training images each model uses, it can be seen that the MNIST (55k) has about 10% more training images than CIFAR-10 (50k). This means that the MNIST model had more unique data to train on than CIFAR-10. Lastly, if we look at batch sizes, the MNIST model used batch sizes of 50 while the CIFAR-10 model used batch sizes of 100. Using larger batch sizes may cause some overlapping, thus lower accuracy.

**PART V: Improve CNN Performance**

**Proposed improvements:**

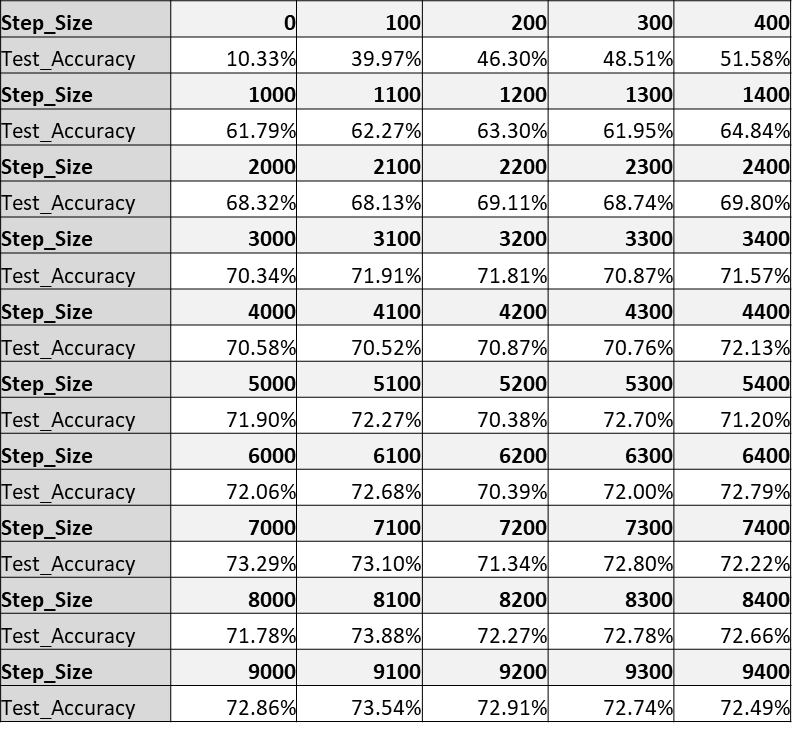
1. Increase number of convolution layers
   * 2 🡪 3
2. Increase number of pooling layers
   * 2 🡪 3
3. Increase number of hidden fully connected layers
   * 2🡪 3
4. Increase number of steps
   * 5,000 🡪 10,000
5. Decrease filter size
   * 5x5 🡪 3x3

To improve the CIFAR-10 CNN model’s performance I propose 5 changes. The first two is to increase the number of convolution and pooling layers from 2 to 3 to further reduce the output size that is fed into the classification phase of the model. Next is to increase the number of hidden fully connected layers in the classification network from 2 to 3 to allow further processing of the features during the classification phase. Next is increasing the number of steps from 5,000 to 10,000 to allow for more training time. Last is decreasing the filter size from 5x5 to 3x3 to preserve more detail that may have been lost considering the complexity of the colored images.

**Results of testing the model:**

Graphical user interface, text

Description automatically generated

 **Chart

Description automatically generated**

**Report on the results:**

The first accuracy check is little over 10%. From then until the 3000th step, the accuracy increases to 70% and fluctuates between 70% to 72% until the last step. The overall average test accuracy was 69%, which is about a 6% increase from the original model. As we know as the model trains you can see how it becomes more accurate but what I was interested in was the tail end of the data points that were collected to get a sense for how accurate this model truly is. By taking the averages of the accuracies between 6000 and 9900 steps, we can see that this model accurately classifies images by about 73%, which is about 4% increase from the original model.

**PART VI: Project Report**

**Introduction:**

This project covers a variety of topics covered in this course but mostly focuses on designing, building, and testing convolutional neural networks, or CNN. The first part of the project focuses on finding a public dataset that contains images or audio files that are appropriately labels and ready to use for deep learning. With that dataset we gathered critical information such as data structure, data size, format, and so on. Next, we will obtain the CIFAR-10 dataset which we will then use to build, train, and test a CNN model. Once we have our model we will compare its performance against a CNN model built from a previous assignment. Then taking the information we’ve learned throughout this course we will attempt to improve that model. Finally, we will review the steps we will review the steps we took throughout this project and summarize what we learned in the process.

**Summary:**

In PART III, I first designed the model drafting the architecture and listing the critical information for each layer. This design acted as a blue print for building out the model’s code. After designing the model, I built the model using Tensorflow in Jupyter notebook. Finally, at every 100 steps of the training process I collected an accuracy data point to measure the overall accuracy of the model against the test data set.

In PART IV, I compared accuracies between convolutional neural networks. Using the model accuracy data points from the CIFAR-10 CNN model I compared them against the accuracy data points from the MNIST model. Upon comparing them it was evident that the MNIST model outperformed the CIFAR-10 model. To determine the reason for this performance gap, I evaluated some of the similarities and differences between the datasets.

In PART V, I used the information that I had gathered in the comparison phase of PART IV and drafted a variety of proposed changes that could be implemented to the CIFAR-10 model to improve its accuracy. After applying those changes and testing the re-designed model, I collected new accuracy data points and compared those with the original model built in PART III. Ultimately, the changes did improve the model’s accuracy by about 4% looking only at the tail end of the training process. If we look at the overall average accuracy, it was a 6% increase.

**Conclusions:**

In PART III I learned how to design a convolution neural network’s architecture which proved to be slightly trickier than the other models we’ve drafted. Designing the model visually as well as laying out the critical information layer by layer helped me understand the pieces of this model fit together and truly acted as a blueprint when writing the TensorFlow code in Jupyter notebook.

In PART IV of this project, I learned how to gather the accuracy data points from testing every 100 steps during neural network training using python and comparing them among my models. Seeing the different levels of accuracy between the CIFAR-10 model and the MNIST model helped clarify the gap performance. It was interesting to see that even though the models had mostly the same architecture, there was a large gap when comparing model performance. Ultimately, I learned that there may be a variety of factors that impacted the accuracy of the model, and it is always prudent to run a few tests to see how accurate you can train a model to be.

In PART V of this project, I learned how to take some of the changes I initially proposed and apply them to an updated CNN model. From this I learned that drafting my proposals based on the comparisons of my model against higher performing models can be more insightful than randomly attempting to change the model parameters. After implementing my proposed changes, I also learned to recognize the point of diminishing returns. A model may become more accurate at 20,000 steps for example but the processing power to complete that training may not be worth the time or as practical for a potentially miniscule bump in accuracy. Lastly, I learned that improving the accuracy of CNNs has a learning curve and although there are recommended best practices they may not always apply to your particular model or dataset.

**PART VII: Final Presentation Videos: YouTube Link**

[**https://youtu.be/enfIgV1K1XU**](https://youtu.be/enfIgV1K1XU)