Project 2 Performance Report and Final Analysis

CS 5600 – 12/15/2023

By Tom Prouty

Introduction

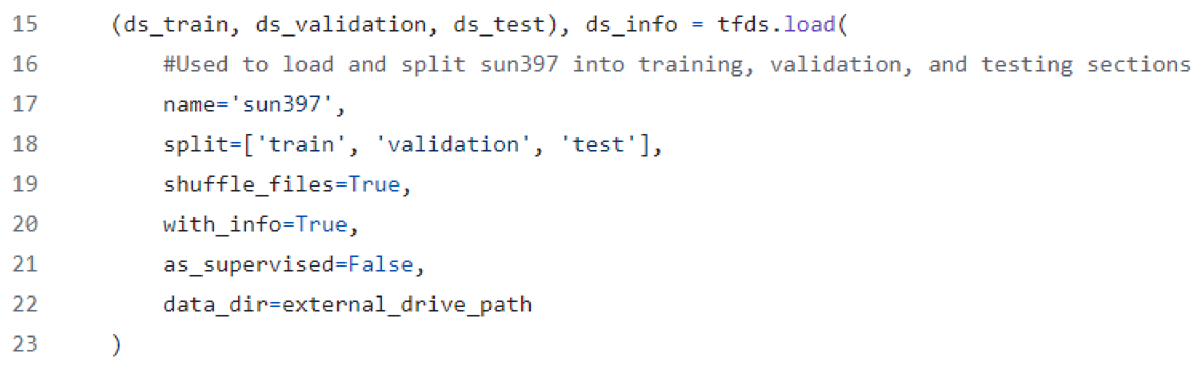
The goal of this project was to implement a multi-category image classifier using a convolutional neural network. My reason for pursuing this project/goal is because I wanted to develop a greater understanding of the limits and requirements needed for large scale image classification. I have seen a lot of discussion about image recognition, whether that be faces or places, in the last few years. As a part of attending CS5600 I hope to better understand why we are trying to implement these types of systems. And, as a computer engineer, I want to understand the data and hardware requirements needed for the creation of these systems. Therefore, creating an image classifier that contains multiple categories is a good idea to both affirm my understanding of CNNs and increase my awareness of the restrictions surrounding them. Of course, to make the CNN for the project, I needed a dataset to train it on.

The dataset used for this project was the SUN397 dataset; a scene recognition dataset created by Princton University. The dataset contains 397 categories for scenes and approximately 108,000 images with a minimum of 100 images per category. I thought this dataset would be good for a multi-category image classifier as the dataset was from a reputable source. It also contains enough images for training a neural network to recognize each category. I did have several issues when implementing the dataset. Those are explained in the "Discussion of Problems" section below.

In addition to dataset selection, having a base model to build a CNN would also be appropriate. This was because in Class Project 1, there was significant difficulty training some of the neural networks given some of my hardware limitations. Therefore, I imported a VGG19 model from TensorFlow and added on to the end of that model. I also froze the imported layers in an attempt to reduce the amount of time needed to train the CNN. This process was only marginally successful.

Project Implementation

The first step taken as a part of this project was to download the SUN397 dataset to the external hard drive. This was so that the dataset, which is 36 GB, could be loaded directly without having to be redownloaded every time an alteration to the python model creation file was made. The following code is was downloaded for the dataset and then split it into train, validation, and testing subsets.



```
(ds_train, ds_validation, ds_test), ds_info = tfds.load(
    #Used to load and split sun397 into training, validation, and testing sections
    name='sun397',
    split=['train', 'validation', 'test'],
    shuffle_files=True,
    with_info=True,
    as_supervised=False,
    data_dir=external_drive_path
)
```

Figure 1: Image of Code to Load and Split SUN397 Dataset

It was quickly determined to be necessary to preprocess the images from the dataset as they may be of different sizes. Since the images in SUN397 are generally of high detail it appeared that while some fine details may be lost in the preprocessing, larger macro-structures of the images would be retained. Many of the images were 300 by 244 pixels and altering them to 244 by 244 caused minimal distortion and allowed for a single input shape to the CNN. This conversion generally resulted in somewhat blurred photos when converted to jpegs, but shapes were still discernable.

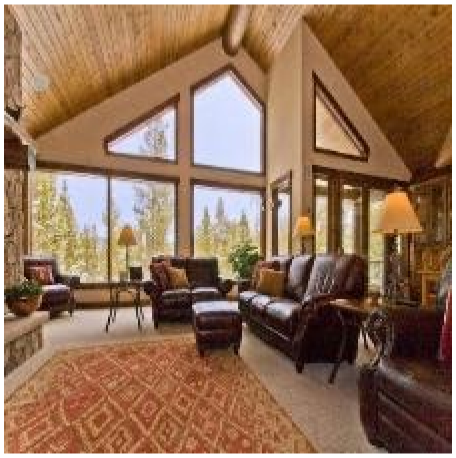


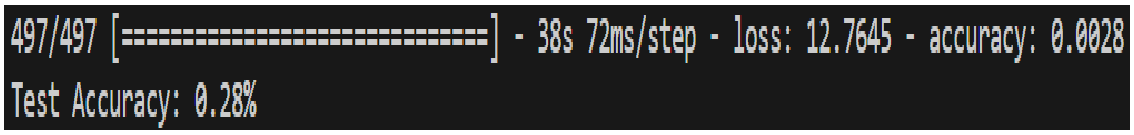
Figure 2: Example of Image Used in Training after Preprocessing

I initially implemented a CNN that would be trained from "scratch" on the SUN397 dataset. However, a few issues quickly became apparent. In initial runs, the time to successfully train the CNN for 10 epochs was around 1 to 2 hours. To solve this training time issue, I worked to see if I could use a bit of my laptop's inbuilt GPU for processing capabilities. This was successful but not to the desired degree of improvement. The second problem encountered was potentially assessed to be an overfitting issue. After the first few epochs of training, the CNN's general accuracy on the training subset would rapidly exceed its general accuracy on the validation subset. To try and combat the overfitting problem, several different attempts were made to reduce the accuracy difference. The final revision was one that entailed adding drop-out layers to decrease dependency on large weights. This seemed to be the most effective for limiting the CNN's difference between training accuracy and validation accuracy. The choice was made to import the VGG19 model to use as a starting point for the CNN. The VGG19 model had already been trained for multi-category classification and using it as the first portion of the CNN would reduce some of the problems with overfitting. And, by freezing the imported layers, the training time for the model could be limited to reasonable levels.

Results

The final evaluation results for this CNN are as follows:

- It took approximately 50 minutes to train the model for 6 epochs.
- The general accuracy was evaluated at approximately 28%, (this is from evaluating the model on the full test set as well as a sample test set designed to be downloaded from a GitHub repository.
- The following image shows a picture of the results after running the evaluation python code



```
497/497 [==============================] - 38s 72ms/step - loss: 12.7645 - accuracy: 0.0028
Test Accuracy: 0.28%
```

Figure 3: Image of Final Test Results

Figure 3. showcases the general accuracy of the CNN and not the accuracy of each category. The reason for calculating the general accuracy of the CNN is due to the sample test set (used for evaluation) containing only 10 images per category. Therefore, the individual category accuracy could potentially vary significantly from the true large-scale accuracy due to the low sample size. An issue I ran into regarding testing for general accuracy was that the validation and test accuracy of the CNN would tend to stall at around 26 – 29%. Training past these "stall" points resulted in overfitting and a reduction in the overall test accuracy, even with the drop-out layers and use of VGG19.

Upon further review of the test, this low general accuracy score made more sense. The scenes contained in SUN397 can be very complex in the content details they contain. The CNN model designed for this project was likely not complex enough to correctly track the important details in each scene category well enough to correctly classify images from each category. This resulted in the CNN trending towards image recognition, or overfitting, rather than the intended purpose of image classification.

Discussion of Problems

There were several problems that I came across as I was working on this project. One of the largest is a topic that we have discussed in class (at large). This being the issue of resources. Trying to train the CNN usually took so long that it was nearly impossible to make small scale changes to improve the CNN's accuracy. Additionally, my computer often had temperature problems when trying to perform training. Both of these issues directly relate back to our class discussion on how resource intensive cloud computing is, as well as the issue of waste heat on both the hardware and surroundings. It was a constant concern of mine that my CPU or GPU would start to overheat and be damaged if I made the model too complex. Naturally, this most likely impacted the CNN's final accuracy on the test dataset. From the class discussion, I had a general understanding as to how hardware can be a limiting factor for neural net training. After having done this project, I have a far more real experience with why resources can be such a large issue for Deep Learning and other neural net training algorithms/systems.

Another challenge I came across was how difficult it is to create a neural net that can perform image classification on more than one category. I had initially thought that implementing a multi-category image classifier would only be slightly more complicated than creating an ANN or CNN capable of recognizing one type of image. Now, I understand more deeply why attempting to classify something as complex as a person's room or type of building is significantly harder than originally thought. I believe that the goal of the project, while achievable, was significantly more complicated than I initially assumed.

Upon review there are a few things that I could have done to mitigate the problems I experienced in this project. The first one would have been to more formally examine the dataset that I wanted to train my CNN on. From what I understand now, scene classification can be significantly difficult, far harder than I had initially intended. Also, a more formal review would have allowed me to have a better understanding of the dataset's structure before I began trying to implement my CNN. There were multiple times I got errors because I had not correctly accessed the splits of the dataset. A possible solution to the training time problem would have been for me to have implemented the computation capabilities of my GPU before attempting to train my CNN, as setting up the GPU took a significant amount of time. I believe if I had spent more time investigating the overfitting issues, I could have devised a better model for the CNN, as the drop-out layers I used on the initial CNN model only slowed the issue and did not prevent it from occurring.

Conclusion

I know that I learned a lot from doing this project even if I didn't get to fully make an accurate multi-category image classifier. I developed a greater understanding of how a CNN can be designed and trained. I also got experience with both downloading and using datasets for the purpose of neural net training. Finally, I got to implement some of the ideas I had in class and some of the lessons that I learned from the other projects and class assignments. Overall, while not as successful as initially planned, I am happy with what I have managed to accomplish.

Link To GitHub:

https://github.com/BlueJCash/CS5600_Project2_TomP/tree/main