NNGA Assignment 2 Report
The Caltech257 Dataset Classification

Submitted to

Dr. Mohamed Moustafa

Submitted by

Kirollos N. Sorour 900153483

Semester: Spring 2021

Table of Contents

Introduction

In this Assignment, the object was to develop a neural network of our own design to classify the 257 classes in the Caltech256 dataset. In my approach, I tried different neural network designs to find the best network that best classifies this dataset with the highest top 5 accuracy.

Neural networks are typically known for object classification and feature detection. This is one of the reasons why they became extremely popular in all real life applications nowadays. In that light, top leading companies and tech teams from all over the world have been competing nonstop searching for a best neural network structure to provide the best classification to all kinds of objects. This is similar to what happens in the global ImageNet competition.

In this assignment, we were asked to use either a fully connected neural network, a convolutional neural network, or a deep belief network. However, I was always fond of mixing two types of networks together just like how we can use an unsupervised model and then fine tune the results with the supervised training. That mere thought kept a lot of time designing and thinking about the best approach to use to conquer this problem

Implementation

One important note to begin with is that reading in and manipulating the data was somewhat harder than creating the model itself. That is simply due to the fact that I was not very familiar with scripting languages before this assignment, so I had to learn python scripting as a prerequisite to this assignment which was brand new to myself. I ended up creating a separate python script that I used to separate the files form the training data from the testing data.

I used many designs including fully connect neural network (FCNN) and a convolutional neural network (CNN), afterwards, I have some further studies about the tools I used and I found that it was pretty popular to do what is known as Transfer Learning. Transfer Learning is when the designer uses one of the pre-trained high-performing networks such as VGG16, VGG19, or Xception but while removing their fully connect layers; therefore, excluding their top layer to remove their classifying trained functionality. This way we get to include our fully connected layer at the end of this network instead of the old one which was fitted to the ImageNet dataset.

It is safe to say that the run time of a lot of these parts of code including a certain dataset or reading in the data would take a large amount of time to a computer using the CPU version of the TensorFlow library like mine. This is why the code in my approach is divided into several scripts. I have done so mainly for two reasons; the first of which is to save up time and effort when using different models, and the second of which is not to have to re-run all the CPU consuming code in case I need to change a single line of code in the output for example.

Hierarchy

The hierarchy I would like to discuss in this part of the report is the file system hierarchy, and how the implementation scripts fit well together.

First of all, for the project folder, it had a hierarchy as below:

```
Assignment2:
|----- training_data:
|---------- (257 classes) + data_move_script.py
|----- test_data:
|---------- (257 classes folders)
|----- preprocess_test.py
|----- preprocess_train.py
|----- train.py
```

Later on in this project, I developed a preprocess pair of scripts for each of the FCNN, VGG16, and the Xception, each having the proper notation.

For the scripts used, they are mainly 2 files. First, is the script that preprocesses the data to be ready for our model to train, and the second script is the one used for the actual training no matter what design we are using since we typically finish each design with a typical fully connected layer to classify the images.

The Preprocessing Scripts

Mainly in this assignment, I am heavily focusing on the concept of transfer learning using really well established feature extractors from the web and using their output as an input to the decision layer which happens to be a FCNN. In this implementation, I am using Keras, TensorFlow and all their accompanying libraries to design the models used in this assignment. Later on in this section, I will describe what each script is designed to perform.

VGG16 Preprocessing:

This script simply takes each image in the folder specified and pipelines it into a predefined model in Keras which is the VGG16 model in this case. However, the model is used without its top module, so it acts only as a feature extractor instead of a classifier. Moreover, the output from this stage is appended to an array and the corresponding label is appended to a similar array. This is when we save all the array content into two separate files to be used in the training process. These files are usually stored as .dat files. A default input array had a size of nx64x64x3 where n is the number of images used and the output of the pre-trained model was nx2x2x512. This output array is saved to the .dat file to be used in the next script.

Xception Preprocessing:

This scripts runs very similarly to the previous script except for the fact that the pre-trained network we are using here is Xception instead of VGG16. Xception has a huge advantage over VGG16 which is its smaller size. It has a smaller memory footprint and it’s less tiresome to download. The size of the input vector used in this script is a bit larger than before due to the minimum image size restriction to this model. The input size is nx128x128x3 and the output size is nx4x4x2048. This output array is saved to the .dat file to be used, alongside the labels array, in the next script.

FCNN Preprocessing:

The fact that the training only happens on a final FCNN will make it simple to perform the last part of the assignment, which requires the use of only a FCNN. It goes without saying that the only preprocessing we need in that case is to set the images in the required structure for the model to work so the data array would have an input size of nx64x64x3. This array is saved to the .dat file to be used in the next script.

The Training Script

In the training script, I start by loading the training and testing data from memory, and display their shapes in the output to tell how each preprocessing method affected the dataset. Right afterwards, the model is built which consists of 3 dense layers each having fully connected nodes such that the first has 1024 nodes, the second has 512 nodes also both with a relu activation function, then the prediction layer has 257 nodes which represent the number of classes in the problem ending with a softmax activation function. Then, the model summary is displayed, and it's compiled and fitted to the training data.

Right afterwards, the model structure is saved with its weights in a JSON and h5 format for the possibility of using this model in future predictions. The last part of this script is when the test data evaluation happens. The top1 and top5 accuracy are calculated manually for predicting each of the individual images. This is mainly the script used in the training. The script is pretty simple due to the usage of the keras interface which abstracts a lot of the C++ TensorFlow backend while maintaining relatively high performance.

For the training parameters used, I chose to use the number of epochs to be 40, the batch size to be 512, while activating the shuffle flag and print a verbose level 2 of details. I also decided to include 2 dropout layers between the fully connect layers model to make over-fitting harder for the model. The loss function used is categorical-cross-entropy loss function which goes very well with the Softmax last layer activation function. I also used my optimizer to be the Adam optimizer which is well known for its ability to optimize learning to decrease the loss value.

Later on in this project, I decided to also try a typical convolutional neural network so that I can compare between different types of networks. Therefore, I am using the same preprocessing script as the fully connected neural network. However, I am adding a new script similar to the training script but adds initial couple of convolutional layers at the beginning of the training model used. These convolutional layers follow the same formula used in the VGG models which is a repeating block of Conv2D layers followed by a Max-pooling layer. In this script, I am repeating this block only twice since it takes forever to run on this PC. However, I also didn’t manage to finish this run due to memory and CPU performance issues. That’s why the CNN run was never finished; however, I recorded its initial loss and accuracy values

Results and Conclusions

In this section of the report, the results and the conclusions of this assignment will be discussed, I am going to look at the results of each model independently. I am also attaching the whole program output in a .txt format for each model.

		Typical		Transfer Learning	
		FCNN	CNN	VGG16	Xception
Trainable parameters		13M	13.5M	2.75M	34M
Training Start	Loss	5.5725	11.714	9.1441	4.8742
	Top1 Accuracy	0.0304	0.0183	0.0409	0.1647
	Top5 Accuracy	0.1147	0.0613	0.0843	0.2498
Training End	Loss	5.3861	Not Finished	1.5446	0.9276
	Top1 Accuracy	0.0301		0.6226	0.7610
	Top5 Accuracy	0.1175		0.8343	0.9179
Testing	Loss	5.6689		2.8643	1.8234
	Top1 Accuracy	0.0039		0.42257	0.636
	Top5 Accuracy	0.01945		0.6368	0.8078

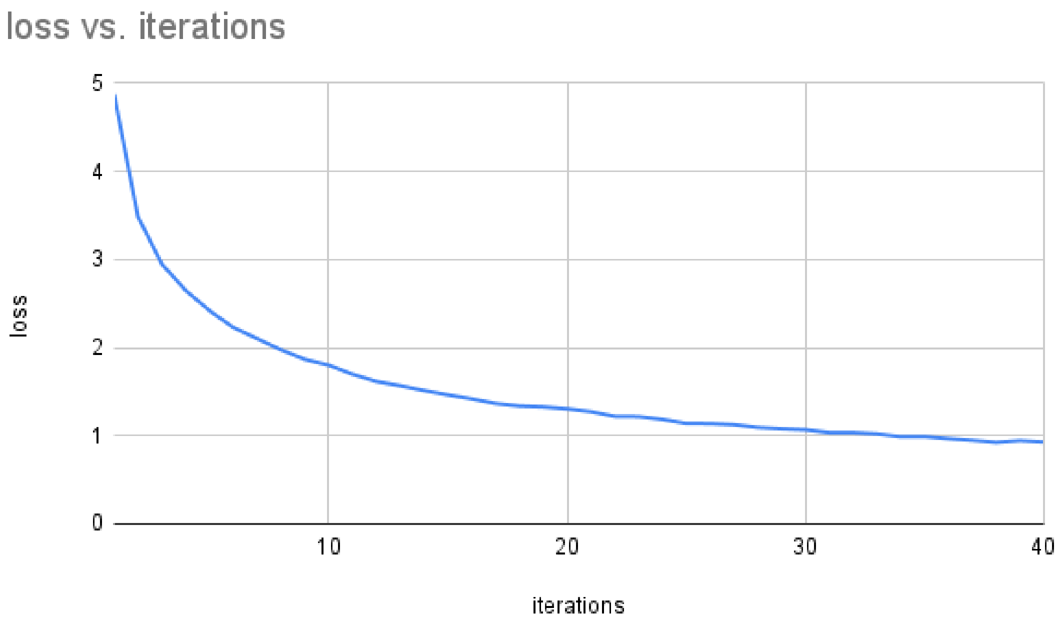
These results make me want to emphasize on a simple fact that we do not have to reinvent the wheel for everything to work fine. We can just use what others made for the following generations especially in a problem like this. We just need to customize the model to our current problem and benefit from the learning it had gained over the past datasets it has seen on the web.

Moreover, I would like to stress on the fact that the training parameters and the network size determine a lot about a networks ability to learn efficiently from the dataset.

First of all, the data batch size, which is the number of image after which the weights get updates, is very important in the learning process since it allows for faster learning and more efficient classification, since it limits the number of false data or outliers therefore, making it harder for the model to over-fit, and at the same time, it makes the model use a bigger gradient and changing the weights respectively. Secondly, the dropout layers used in this model are great to avoid over-fitting since they purposely let go of some of the learning the network made to make it harder to reach over-fitting.

Looking at the table above, one can see that the Xception model scored highest on the testing data top5 accuracy with a score of about 80.8%, while the VGG16 model achieved second with a score of about 64%. On the typical side of things, the fully connected neural network achieved last with a top5 accuracy score of about 2% which indicates that the model needs way more training epochs.

Next to discuss is the decision when to stop training, and here this was only answered based on the Xception model and had the same result used for the other models to produce meaningful comparisons between each and every one of them. Following this paragraph is a graph showing the loss values over training epochs.



This the graph for the output produced by the Xception model, and it shows that after about 40 epochs, the training loss reaches a value less than 1 which was good enough for me to stop the training seeing the other models errors and the current accuracy of this model as well.

Challenges

This assignment was a bit challenging for me due to many reasons, most of them are quite personal. However, the most pressing challenges I had in this assignment was my need to acquire knowledge about the python programming language as a prerequisite for this assignment in order for me to develop these simple codes using a ready-made packages like Keras and TensorFlow.

Moreover, I was already behind in the class due to my military service, so I had to learn python, then learn how to use the packages and afterwards study the whole course material properly and take notes then finish the assignments. This was a bit challenging for me since I needed time to organize my thoughts together and still follow up with all my military service requirements and work.

One huge challenge for me was how to organize the files in two different folders, instead of all them being gathered in one folder. This is when I had to work on a separate task of scripting a python script to fulfill this task instead of tediously coping them folder by folder.

Another challenge for me was that I am working on my military PC with limited internet connection and very slow CPUs with 3GB of RAM. That made me uncomfortable working around the environment setting and the packages download and scripts running. Other than that, the assignment was very beneficial.