Machine Learning Engineer Nanodegree

Capstone Project

Definition

Project Overview

Deep learning has allowed for the design of object recognition and is becoming more powerful by the day. A good real-life example of this is when you go to take a picture of yourself (a "selfie") and notice that there is a square around your face and any other faces in the picture[1]. Similarly, deep learning with deep neural networks can be used in areas to identify disease in plants[2]. This is accomplished under the hood using layers upon layers of convolutional neural networks which are constantly outperforming the previously listed machine learning techniques on image recognition. There was no particular academic research that was considered on this topic however the dog breed classification project was quite similar to this problem.

Problem Statement

A great obstacle to landmark recognition research is the lack of large annotated datasets. Thankfully Google has presented machine learning enthusiasts the largest worldwide dataset on Kaggle.com[3] boasting 15,000 classes.

The main objective of this challenge is to build a model which recognizes the correct landmark (if any) in a dataset of challenging test images.

Metrics

Accuracy of the model can be given by using the number of landmark id's in the test dataset and the number of images in the test dataset. Accuracy was chosen as a measurement as it is easy to understand at a basic level. Accuracy for a balanced dataset can prove to be a very powerful measurement and hence as pre-processing will take place it will be used as one of the two metrics. This is calculated as such:

---

1 Omkar M. Parkhi. 2015. Deep Face Recognition. [ONLINE] Available at: https://www.robots.ox.ac.uk/~vgg/publications/2015/Parkhi15/parkhi15.pdf.

2 Srdjan Sladojevic. 2016. Deep Neural Networks Based Recognition of Plant Diseases by Leaf Image Classification,". [ONLINE] Available at: https://www.hindawi.com/journals/cin/2016/3289801/.

3 https://www.kaggle.com/c/landmark-recognition-challenge

$$\text{Accuracy} = \left(\frac{\sum_{x=0}^{x=N}\left(x \cdot \frac{n_{landmark_{test}}}{n_{images_{test}}}\right)}{n_{images_{test}}}\right) = \frac{Number\ of\ correctly\ predicted\ classes}{Total\ number\ of\ predictions}$$

A benchmark can be created by guessing the landmarks at random, it is expected that the model will give a higher accuracy than this.

The official GAP (Global Average Precision) at k, where k=1 metric is used as the evaluation technique officially by Google and is as such:

$$GAP = \frac{1}{M}\sum_{i=1}^{N} P(i)rel(i)$$

Where:

- N is the total number of predictions returned by the system, across all queries
- M is the total number of queries with at least one landmark from the training set visible in it (note that some queries may not depict landmarks)
- P(i) is the precision at rank ii
- rel(i) denotes the relevance of prediction ii: it's 1 if the ii-th prediction is correct, and 0 otherwise

Global average precision is a powerful measurement for data which is unbalanced. As the original dataset is quite unbalanced, the GAP will be calculated and evaluated. Ordinary precision would not fare too well in the case of an unbalanced dataset and so the global average precision is an optimal choice for this project. This evaluation metric will also be evaluated to see if results are consistent with accuracy as it is possible that accuracy measurements end up affected due to the skewness (poor balance) of the data.

Analysis

Data Exploration and visualisation

The dataset given by the Google-Landmark Recognition challenge consists of 1,048,575 images in the dataset to be split between training, testing and validation sets. There exists three columns, firstly an ID column, a URL column and a landmark_id column. The ID column is unique for every image, the URL column provides a URL to the image and the landmark_id contains a number representing a particular landmark so there are many URLs which share the same landmark_id but different id's.

It is important to have an understanding of how balanced the data is. For visualisation purposes I have taken the 150 most frequently occurring landmarks and 3% of each landmarks data and have plotted the frequency of a URL appearing as each landmark as such:

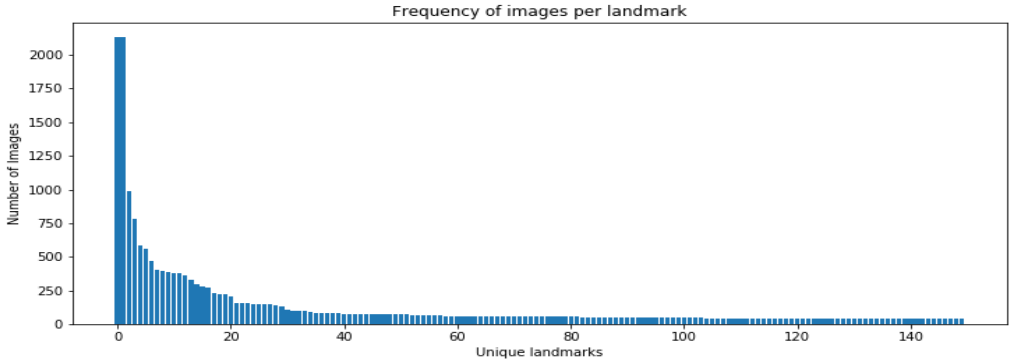


Figure 1. Frequency of images in descending order of raw

From this image, it is evident that the data is skewed with a right tail (positively skewed). If the whole dataset with approximately 15,000 unique landmarks was shown the right tail would be significantly longer and hence it is evident that this dataset is extremely unbalanced.

To gain a deeper understanding of how the data works, consider landmark_id 9633, which is the most frequent landmark_id. By opening two random URLs with the landmark_id 9633 we see the following:

Figure 2. Random example of landmark_id 9633

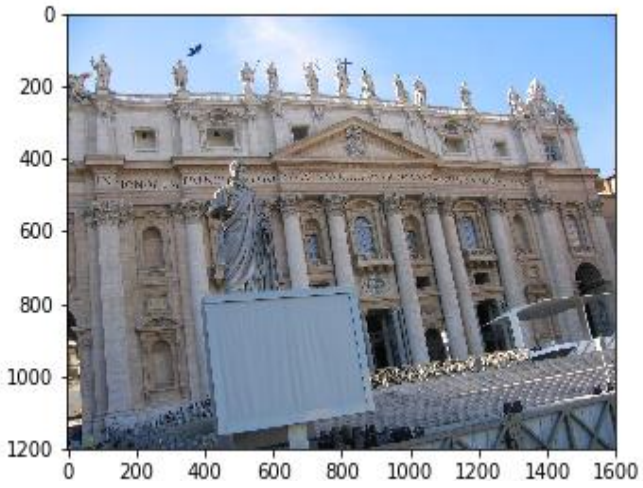


Figure 3. Random example of landmark_id 9633

Both images are taken of St. Peter's Basilica at the Vatican City, a popular landmark within Italy. The first thing we notice is that the pictures have a lot of noise in them such as people and other objects. It is also important to notice that the images were of different shape and size which can be seen by the axes which is further discussed later in the report.

Figure 4 shows some statistics regarding the raw dataset. Here we can see that the standard deviation is 607.39 with a mean of 70.29. The largest dataset contains 42619 images whereas the smallest contains only 1.

```
#Show details of the data before sampling
train_data['landmark_id'].value_counts().describe()
```

```
count    14918.000000
mean        70.289248
std        607.389633
min          1.000000
25%          5.000000
50%         12.000000
75%         39.000000
max      42619.000000
Name: landmark_id, dtype: float64
```

Figure 4. Statistics of the raw dataset

Algorithms and Techniques

Convolutional neural network (CNN) will be used to create the model to predict what landmark id an image belongs to. I will utilise transfer learning through this project by importing the Xception[4] pre-trained model through the keras application[5] library. This model consists of 36 convolutional layers forming the feature extraction base of the network. Francois Chollet provides in his Xception publication an excellent architectural diagram of the layers within the Xception Model (page 1255)[6].

The Xception model will take in an image of dimensions (299,299,3) and run it through a series of separable convolutions which are essentially spatial convolutions which are iterated independently over each channel of input which is then continued on by a pointwise convolution. Max pooling layers are also applied in the model which is used to down sample the image as it partitions the image of (299,299,3) dimensions into a set of rectangles and outputs the maximum. The model then will extract the bottleneck features to be used as an input into a fully connected neural network that I will define and ultimately being fit to predict the accuracy of an image being correctly classified.

Benchmark

Originally I had proposed to create a benchmark by creating an algorithm which randomly guesses which image belongs to which class. For the purposes of discussion and graph creation, I decided to use a CNN which I had previously defined in the dog breed classifier

---

[4] *François Chollet. 2017. Xception: Deep Learning with Depthwise Separable Convolutions. [ONLINE] Available at: https://arxiv.org/abs/1610.02357*

[5] *https://keras.io/applications/#xception*

[6] *François Chollet. 2017. Xception: Deep Learning with Depthwise Separable Convolutions. [ONLINE] Available at: https://arxiv.org/abs/1610.02357*

project. This consisted of four 2D convolutional layers all complemented with max pooling layers, two dropout layers, one flattening layer and two dense layers as outlined in figure 5.

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 224, 224, 8)	104
max_pooling2d_1 (MaxPooling2	(None, 112, 112, 8)	0
conv2d_2 (Conv2D)	(None, 112, 112, 16)	528
max_pooling2d_2 (MaxPooling2	(None, 56, 56, 16)	0
conv2d_3 (Conv2D)	(None, 56, 56, 32)	2080
max_pooling2d_3 (MaxPooling2	(None, 28, 28, 32)	0
conv2d_4 (Conv2D)	(None, 28, 28, 64)	8256
max_pooling2d_4 (MaxPooling2	(None, 14, 14, 64)	0
dropout_1 (Dropout)	(None, 14, 14, 64)	0
flatten_1 (Flatten)	(None, 12544)	0
dense_1 (Dense)	(None, 140)	1756300
dropout_2 (Dropout)	(None, 140)	0
dense_2 (Dense)	(None, 140)	19740

Total params: 1,787,008
Trainable params: 1,787,008
Non-trainable params: 0

Figure 5. CNN of benchmark model

Figure 6 shows the training accuracy reach approximately 30 epochs before plateauing at its maximum accuracy of 93%. However it is interesting to see that after approximately 20 epochs, the validation accuracy begins to plateau to approximately 24%. This is a clear indication of over fitting the model. On the test accuracy, the benchmark model scores **28.57% accuracy** and this will be used for comparison sake.

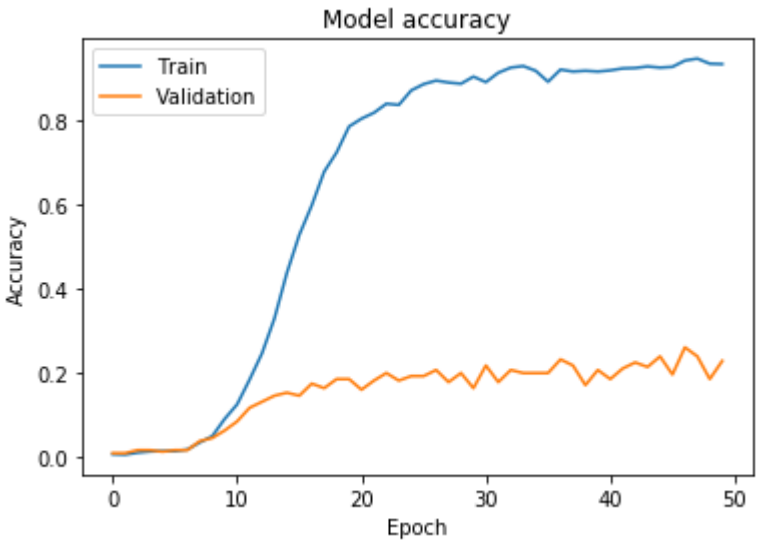


Figure 6. CNN of benchmark model

Analysis

Data Preprocessing

As the dataset is quite large, I decided to use a subset of the training images. I had quite a lot of experimentation with reducing the dataset. Initially I took 150 most frequently occurring landmark ID's and then a 3% of ID's from each landmark ID in order to preserve the ratio of images using a stratified sampling process (see figure 1). This reduced the number of training images to 19,887. I then however came to realise that by looking at only the first 150 classes the data became extremely unbalanced and hence the proposed accuracy prediction may be a negligible parameter to use.

Secondly, the method I ended up using in this project was to first create a table in descending order (highest to lowest) of URL's per landmark ID. At this point I then indexed the sample so that 70 classes were taken below the median and 70 classes were taken above the median, to result in a total of 140 classes (landmarks) being used.

Figure 7 shows the frequency graph for the 140 selected classes (landmarks). It can now be seen that the number of url's per landmark ID does not vary anywhere near as much as it used to.

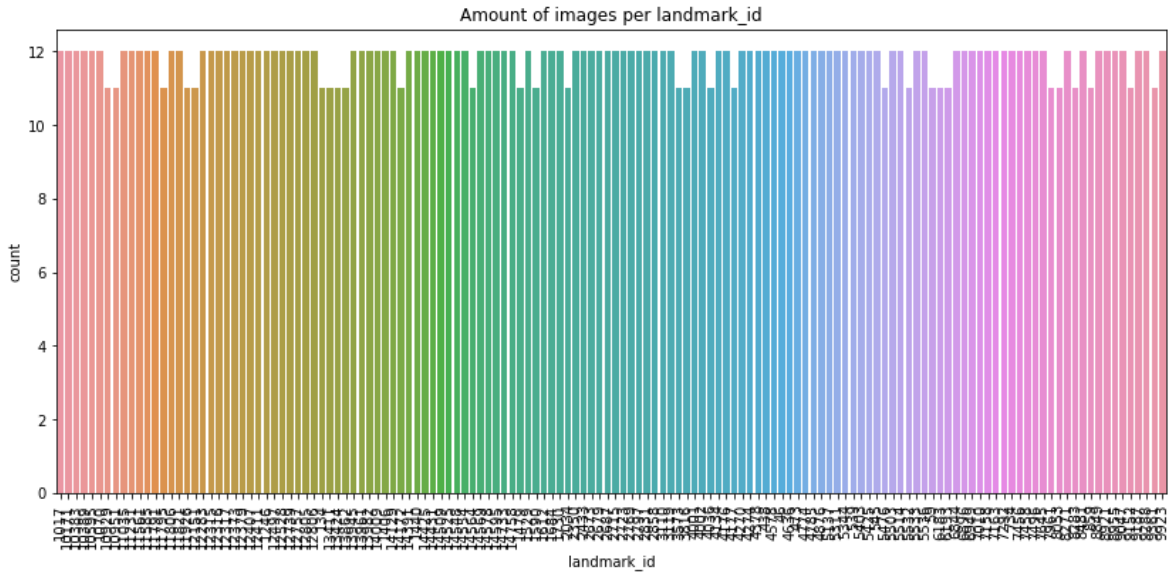


Figure 7. Frequency of sampled dataset

Figure 8 shows some statistics of the sampled dataset. It can now be seen that the mean is 11.80 with a standard deviation of 0.41. The minimum amount of URL's per landmark is 11 and the maximum 12. In this case, using accuracy as a measurement will be very relevant to measuring the model.

```
#Show details of the data (note the std)
training_samp['landmark_id'].value_counts().describe()
```

```
count    140.000000
mean      11.792857
std        0.406714
min       11.000000
25%       12.000000
50%       12.000000
75%       12.000000
max       12.000000
Name: landmark_id, dtype: float64
```

Figure 8. Statistics of sampled dataset

A 70/15/15 split was used to create train/test/validation sets. There were 1091 samples of training data, 280 samples of testing data and 280 sample of validation data. All data was downloaded using the data downloading script provided by Kaggle and is saved as download_script.py[7].

As some images have expired, there were 24, 3 and 3 images which failed to download for training, testing and validation sets respectively. The reason for limiting the size of the images was that I was performing the project on the same workspace that was used for the dog breed classifier deep learning project as this was the most effective way that I could use a GPU. The disk space is limited to 4gb and hence I had to work with this.

Finally, the images are to be rescaled from an interval from [0,255] to [0,1] as the model would not be able to process such a high range[8].

Implementation

- The Xception model must be defined with the input shape of (299,299,3), from here a train, validation and test model will be created which creates the files on disk to be loaded back in.
- The files are loaded back into the workspace where train, validation and test targets are defined.
- The architecture is defined, figure 9 shows the architecture which I used (this is inspired by the same architecture that I used in the dog breeding project).
- The model is compiled using a range of optimizers and parameters such as momentum.
- Both training and validation features are fit to the model over 50 epochs. To keep track of the values, the best weight is saved to disk.
- This weight is then loaded into the workspace and accuracy is calculated.
- Finally the global average precision is calculated.

[7] *https://www.kaggle.com/tobwey/landmark-recognition-challenge-image-downloader/comments*
[8] *https://www.linkedin.com/pulse/keras-image-preprocessing-scaling-pixels-training-adwin-jahn*

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
global_average_pooling2d_24  (None, 2048)              0
_________________________________________________________________
dense_47 (Dense)             (None, 500)               1024500
_________________________________________________________________
dropout_24 (Dropout)         (None, 500)               0
_________________________________________________________________
dense_48 (Dense)             (None, 140)               70140
=================================================================
Total params: 1,094,640
Trainable params: 1,094,640
Non-trainable params: 0
_________________________________________________________________
```

Figure 9. CNN of final model

From a coding point of view, the pre-processing of data was the toughest. The data was very unbalanced as previously mentioned and trying to make sense of almost 15,000 classes was quite difficult without being able to visualise due to the size. The way I overcame this was by looking at the statistical analysis of the data (see figure 4) and changing the data until the standard deviation was small and the max/min spread was small. The steps of code came quite naturally as the difficulty of image recognition comes in defining the CNN and compiling with correct optimizers. A lot of this heavy lifting was done through transfer learning.

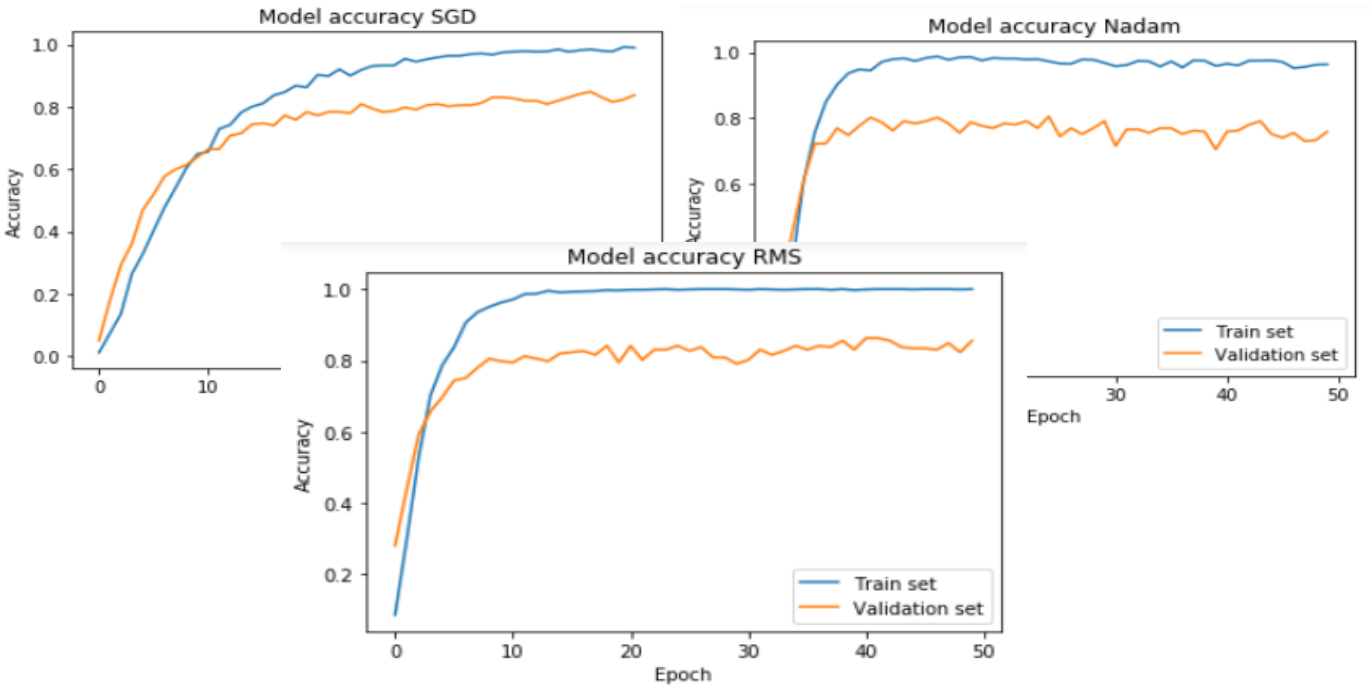
Refinement

The most obvious parameter that I tweaked the most often was the optimizer[9] whilst compiling the model. I initially used a stochastic gradient descent optimizer and tweaked the momentum parameter and learning rate parameter. I found that a momentum of 0.9 and a momentum of 0.5 to be quite consistent yielding in an accuracy of 78.34%.

Secondly, I decided to use the Nadam (Nesterov Adam) optimizer at its default values which resulted with an accuracy of 76.90%.

Lastly, I decided to use the RMSprop optimizer with its default parameters, resulting in an all-time high accuracy of 81.95%.

It can be seen from figure 10, 11 and 12 that all graphs plateau off reasonably quickly with SGD taking the longest. All train graphs eventually reach 1.0 or close to it with the validation accuracies hovering around 0.9-0.95. It is seen that overfitting occurs to some extent for all cases and that the smallest occurs with SGD followed by RMS. Considering the 3.7% difference between the SGD and RMS optimizers and bearing in mind that GPU power is needed to run these models over larger amounts of epochs, I decided to use the RMSprop optimizer for the remainder of the project.



[9] *https://keras.io/optimizers/*

Figure 10. Accuracy growth for SGD,Nadam,RMS optimizers

Results

Model Evaluation and Validation

The main limitation I faced in creating an effective model was that I did not have a GPU or a lot of ram in my personal computer. I decided to use the Udacity workspace with the GPU time left over from the dog predictor project. The main limitation here was not the time I had to use the GPU but the HDD space available to me, 4gb.

For this reason, I decided to completely make use of transfer learning and Keras application library so that the intense computations would not have to be made and I could tweak the model as I wished[10].

An interesting study which looked at identifying images given a number of categories with a reasonably well-balanced dataset was undertaken[11]. In this study, the VGG-16, ResNet-152, Inception V3 and Xception models were compared with the parameter of measurement being accuracy. It can be seen that the Xception model performs extremely well and as the dataset which I used was similar to the dataset of the study, I decided to use the Xception model based on this.

	Top-1 accuracy	Top-5 accuracy
VGGNet – 1st Runner Up in ILSVRC 2014 **VGG-16**	0.715	0.901
ResNet – Winner in ILSVRC 2015 **ResNet-152**	0.770	0.933
Inception-v3 – 1st Runner Up in ILSVRC 2015 **Inception V3**	0.782	0.941
Xception	**0.790**	**0.945**

Figure 11. Study shown how Xception performs against other common models

[10] *DISHASHREE GUPTA. 2017. Transfer learning & The art of using Pre-trained Models in Deep Learning. [ONLINE] Available at: https://www.analyticsvidhya.com/blog/2017/06/transfer-learning-the-art-of-fine-tuning-a-pre-trained-model/.*

[11] *SH Tsang. 2018. Review: Xception — With Depthwise Separable Convolution, Better Than Inception-v3 (Image Classification). [ONLINE] Available at: https://towardsdatascience.com/review-xception-with-depthwise-separable-convolution-better-than-inception-v3-image-dc967dd42568*

It is important to validate the robustness of the model to see how the accuracy may vary when parameters are changed. In the original workflow, all data was preprocessed such that the train images were flipped both vertically and horizontally (data augmentation). To test out the robustness, I decided to test the model without horizontally or vertically flipping the images in the dataset and by using the RMSProp optimizer.

Figure 12 shows both graphs detailing how train and validation accuracies increase with epochs. Both graphs look quite similar early, however slightly differ in the later stages. Quantitatively, we see that the accuracy in the sensitivity model is 80.15% compared to 81.95%. Both traits prove that the model is robust.

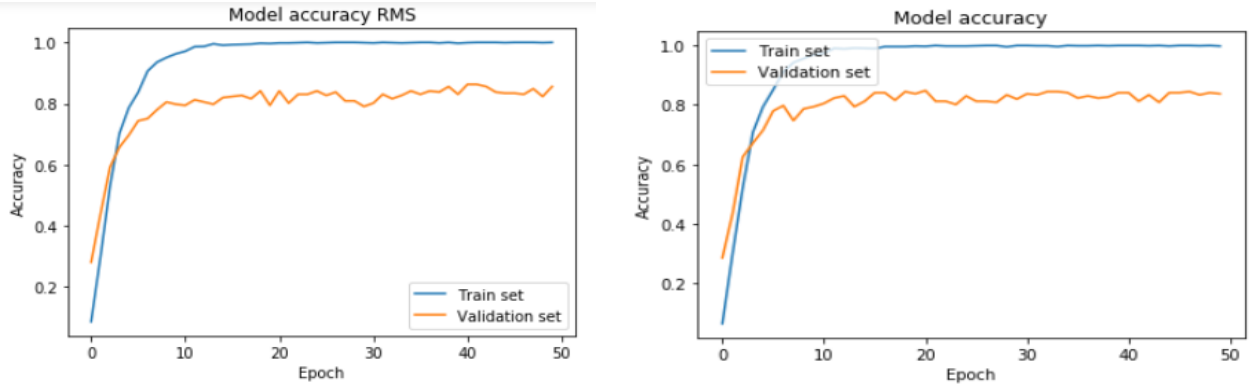
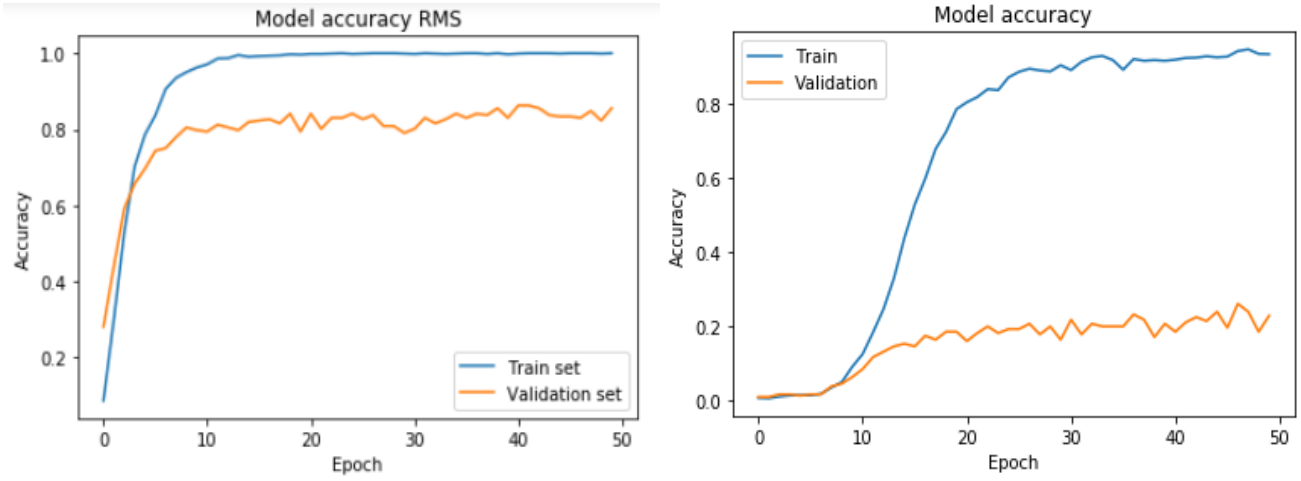


Figure 12. Model accuracy with and without horizontal flipping

Justification

A benchmark model was created as described above whereby a basic CNN was created and tested on the same dataset. In the benchmark model, an accuracy of 28.57% was obtained. The model which I have defined in this project gives an accuracy of 81.95%, this gives a percentage difference of 53.38%.



The Google landmark challenge required participants to calculate the global average precision, I believe that this was due to the unbalanced dataset. The dataset which I used was a balanced sample of this. For completeness I calculated the GAP and the model had a GAP of 96.71%. Some of the highest performing models in the Google Landmark challenge had GAP's of around 40%. It is important to note that the dataset that I used is skewed differently and is over 600 times smaller than the original dataset. Based on this, the GAP scored here should not be used for comparative sake in the challenge however can be considered solely for this project.

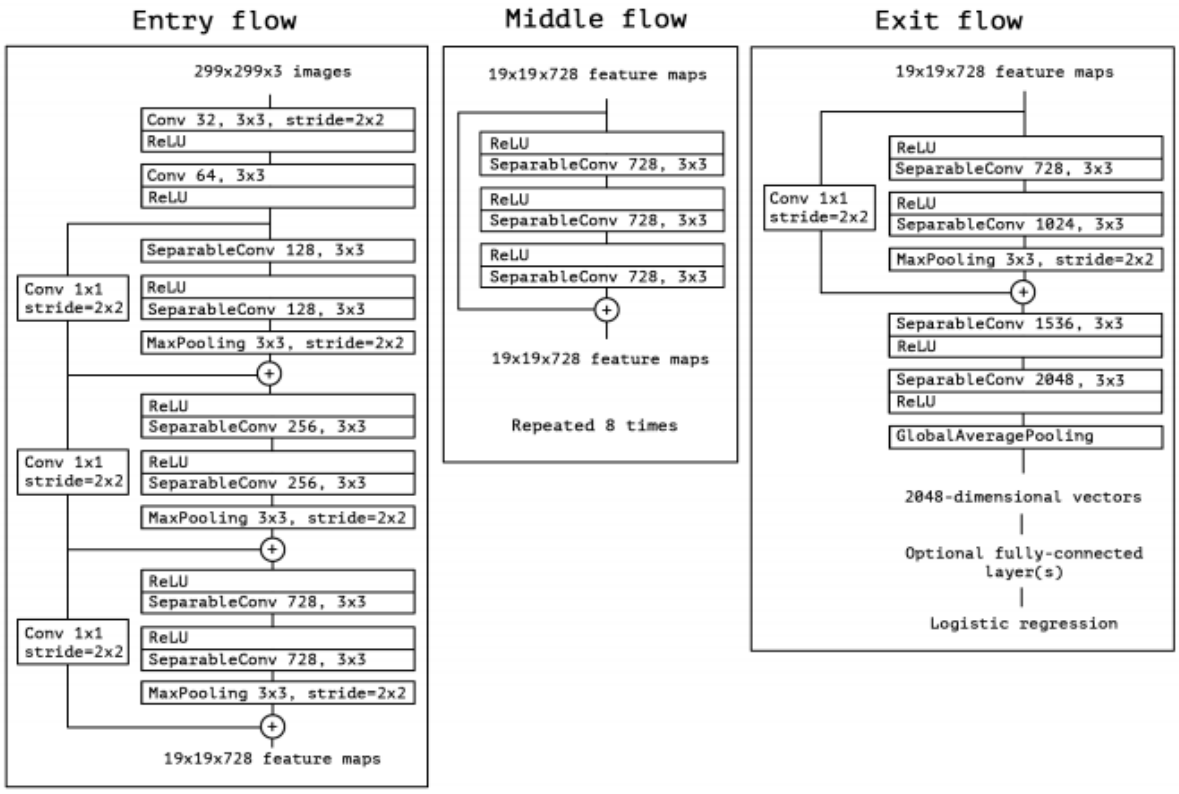
Conclusion

Figure 13. Model accuracy Xception Model vs Bencmark

The model architecture used is outlined in figure 14 below. Figure 14 shows the architecture of the Xception model. Each image is converted to a size of 299x299x3 when creating the train, validation and test generators. This is passed in to the "Entry flow" section of the architecture. The Early Flow stage consists of convolutional layers using RelU activation and max pooling layers. In the "Middle flow section" the model repeats itself 8 times through a series of separable convolutional layers. Finally the "Exit flow" passes through more separable convolutional layers, a max pooling layer and a global average pooling layer. After this, under the "Optional fully-connected layer(s)" section, the architecture as shown in figure 5 is applied and then the model undergoes logistic regression for its fit.[12]

Figure 14. Xception model architecture

[12] *François Chollet. 2017. Xception: Deep Learning with Depthwise Separable Convolutions. [ONLINE] Available at: https://arxiv.org/abs/1610.02357*



I put the model to the test and decided to use two testing images from class 2687 as shown in figure 15. It shows the sign outside the Talvisodan-monumentti which is a war monument in Finland. The reason I chose this class was that one image is in nice clear weather whereas one is in snowy weather and the sign is covered in snow. The fact that the model could identify that these are the same place adds to the conclusion that the model is an accurate and powerful one.

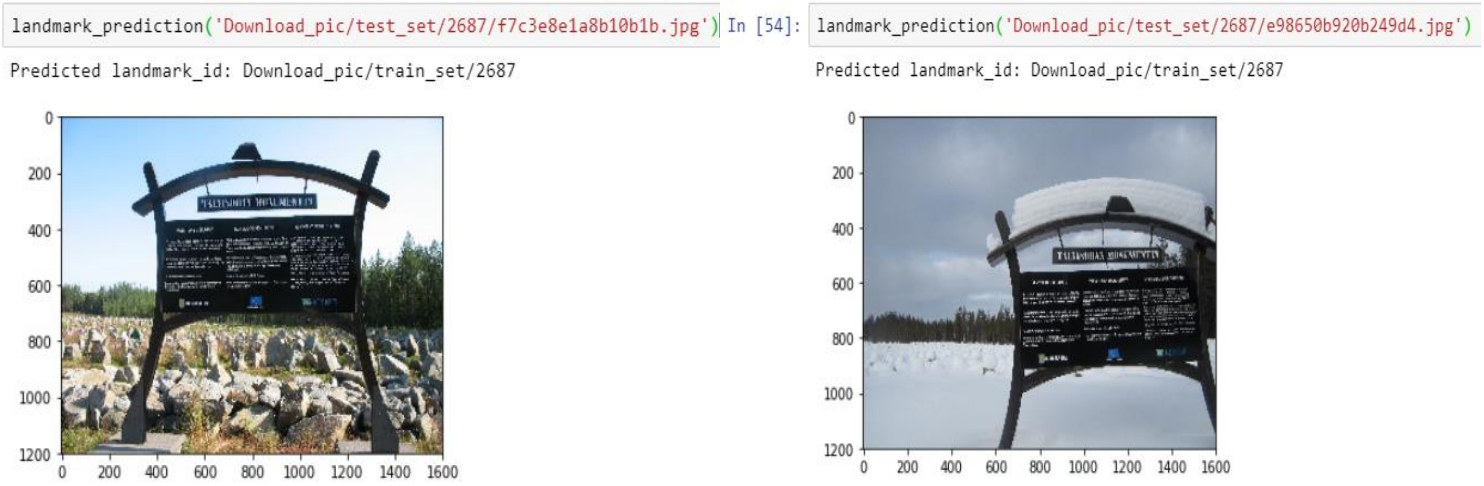


Figure 15. Class 2687 test images correct predictions

Reflection

Upon the acceptance of my proposal for this project, I had loaded the data into python and began analysing the dataset in more detail. Initially I took 2% of the entire dataset and then used a stratified sampling technique to maintain the ratio of images per class. After further research I concluded that a highly unbalanced dataset would deem the accuracy measurement to be almost negligible and only the global average precision measurement to have meaning. I then decided to use a sample of the dataset whereby 140 classes (unique landmarks) and 11-12 images per landmark existed. I looked at the standard deviation difference and hence concluded that the dataset was balanced.

At this point I needed to create a train, test and validation dataset and so I used Numpy's split technique to create a 60/20/20 split. At this point all of these "images" are merely url's on a spreadsheet. I struggled at downloading bulk URL's at first, I searched the web trying to find ways, I began manually downloading these from google until I realised that everyone running my project would have to do the same. I later found a script written on the official Kaggle challenge[13] which was an auto downloader. I loaded that python code into my workspace and utilized it.

Next was the pre-processing of data. An interesting pre processing input to the ImageDataGenerator function was "preprocessing_function=prerocess_inbuilt"[14] which was from keras.applications.xception. I tried to experiment with this input however it didn't seem to have any positive impact on my results. There was not much documentation around this available and so I decided to leave it out. I did however decide to use the horizontal and vertical flip techniques as a form of data augmentation.

I created the train/test/validation generators noting that the Xception model requires an image input size of (299,299,3) as shown in figure 14. Predict_generator was then used to generate and save the bottleneck feature where I then loaded the files back in to define the target parameters for the train/validation/test datasets. A basic architecture which was inspired by the one I used in the dog breeding project was defined three times for three different optimizers. The main objective of this was to see which would yield the best results to go forward with. RMSProp, SGD and Nadam optimizers were all researched and available in the Keras library and each were compiled. Each were fit and ran for 50 epochs and a checkpoint was saved to memory holding the best weight.

RMSProp had the best attributes as an optimizer (accuracy of 81.95%) and was hence used. I was quite surprised by this as a lot of research that I had looked at used stochastic gradient descent optimizers. I then calculated the global average precision to be consistent with the Kaggle challenge and received an exceptionally high 96.71% which I then concluded was not to be used as a comparative measurement due to the difference of my dataset to the original.

---

13 https://www.kaggle.com/tobwey/landmark-recognition-challenge-image-downloader/comments

14 https://github.com/keras-team/keras/issues/9624

A sensitivity model was then carried out whereby I removed any pre-processing from the dataset and the whole procedure was repeated over 50 epochs again. A test accuracy of 80.15% was calculated, proving the rigidness of my model.

For visualisation purposes, I borrowed and tweaked some code from the dog breeding project called called extract_bottleneck_features[15] and used it to predict what landmark_id a few of the test images belonged to.

The most difficult part of this entire project was the pre-processing of data. The reason for this is that my computer would take hours and then give memory errors when trying to save down the bottleneck features (.npy files). I eventually made the dataset so small that I could do this but then when it came to running 50 epochs, the run-time was hours. That is when I approached two very helpful mentors on the Udacity student hub, Graham R and James L who guided me to use the remaining GPU time on dog breeding workspace and confirmed that using a much smaller dataset for the purpose of this project was absolutely acceptable, many thanks to them.

Improvement

The main limiting factor of this project was the amount of data that I could use due to computational restraints. If I had a larger HDD with a powerful GPU I would expand the dataset to its original size. I believe this would help to counteract the overfitting aswell.

I would consider using models such as VGG19, AlexNet or one of the ResNet models[16]. I believe that with both a larger dataset and experimenting with different pre trained models that the accuracy and global average precision are likely to decrease however the model will be much more powerful at identifying a larger group of images.

15 https://github.com/udacity/dog-project
16 https://github.com/jcjohnson/cnn-benchmarks