10 Monkey Species Image Dataset Analysis

Module: CS985 Machine Learning For Data Analytics
Student: Rokas Labeikis
Student ID: 201349799

Objective and problem

The core problem of the assignment is to create a deep learning model, which classifies images of monkeys into 1 of 10 classes.

The objective is to create an sufficient deep learning classifier which performs well. However, more important objective, for the author, is to learn and explore machine learning framework- Tensorflow. Due to the later reason, more than 1 approach of image classification will be researched and the performance of the models will not be considered a priority.

Explored techniques include:

- Linear image classifier;
- Transfer learning.

Development environment

TensorFlow version: 1.7.0
Python version: 2.7.13

Introduction to the dataset

Analyzed dataset was taken from *www.kaggle.com* and it is called "10 Monkey Species". The original dataset consists of 2 directories ("training" and "validation") and a text file containing all the labels and titles. Each directory contains 10 subdirectories, labeled from "n0" to "n9" (these labels will be considered as classes). Finally, each of the classes subdirectories contains a series of images (images are 400x300 pixels or larger and they are in JPEG format) of a certain monkey type (class).

Below the contents of the "monkey_labels.txt" file are presented. The table encompasses the overall structure of the dataset.

	Label	Latin Name	Common Name	Train Images	Validation Images
0	n0	alouatta_palliata\t	mantled_howler	131	26
1	n1	erythrocebus_patas\t	patas_monkey	139	28
2	n2	cacajao_calvus\t	bald_uakari	137	27
3	n3	macaca_fuscata\t	japanese_macaque	152	30
4	n4	cebuella_pygmea\t	pygmy_marmoset	131	26
5	n5	cebus_capucinus\t	white_headed_capuchin	141	28
6	n6	mico_argentatus\t	silvery_marmoset	132	26
7	n7	saimiri_sciureus\t	common_squirrel_monkey	142	28
8	n8	aotus_nigriceps\t	black_headed_night_monkey	133	27
9	n9	trachypithecus_johnii	nilgiri_langur	132	26

Obtaining and preparing the dataset

Many different approaches for importing the image data were considered, one of them was retrieving all the image paths and directly reading the image from the storage, preprocessing it (resizing, reshaping etc.) and using the newly retrieved image each time it was needed (example, batch of images are needed for each training step). However, this method was discarded as it was extremely inefficient when dealing with large image sets, because constant storage accessing requires lots of CPU clock cycles.

Because of this, the function “read_data” for reading and preparing the data was developed. This function takes some directory as an input, then it goes through all of the subdirectories while building the data frame in the process. At the end, it returns a newly constructed data frame and halts. The produced data frame consists of numerical image representation, its label, path to the real image, integer encoded label and one hot encoded label. This approach proved to be much more efficient, since images are stored as arrays in memory.

Lastly, in order to make sure that everything went smoothly with the input reading and the data frame preparations a series of checks were in place:

- Data frames were checked for null or missing values;
- Data frame format was tested to be as expected.

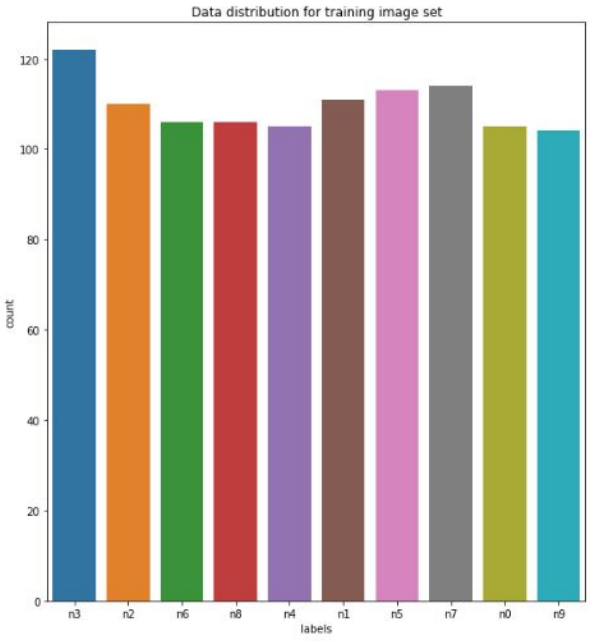
Exploring the dataset

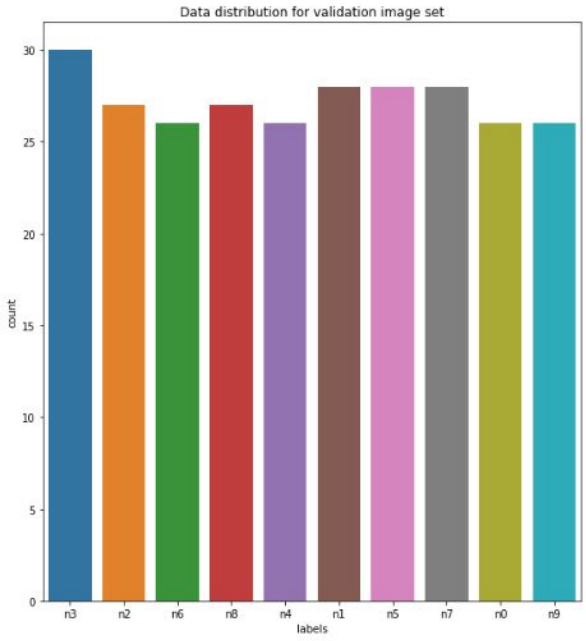
There are 1096 images dedicated for model training and 272 for testing. In total there are 1368 number of images.

Random sample of 9 images along with their labels are printed out below. This is done in order to see the type of images we are dealing with and if the labels are matching correctly.

Data distribution

As we can see, from the diagrams below, the data is almost evenly distributed, while only the "n3" class has a higher number of entries in both sets, however the difference is not significant so it should have no impact on the results.





Models

Transfer Learning

Transfer learning is a technique that takes a piece of a model that has already been trained on a related task and reuses it in a creation of a new model. This technique was the first choice for 10-way image classification problem because, it is known that retrained models perform generally well and training a completely new model to classify images requires lots of time, processing power and a huge sample size.

In order to train the model, author used a script located in './retrain.py' with following parameters:

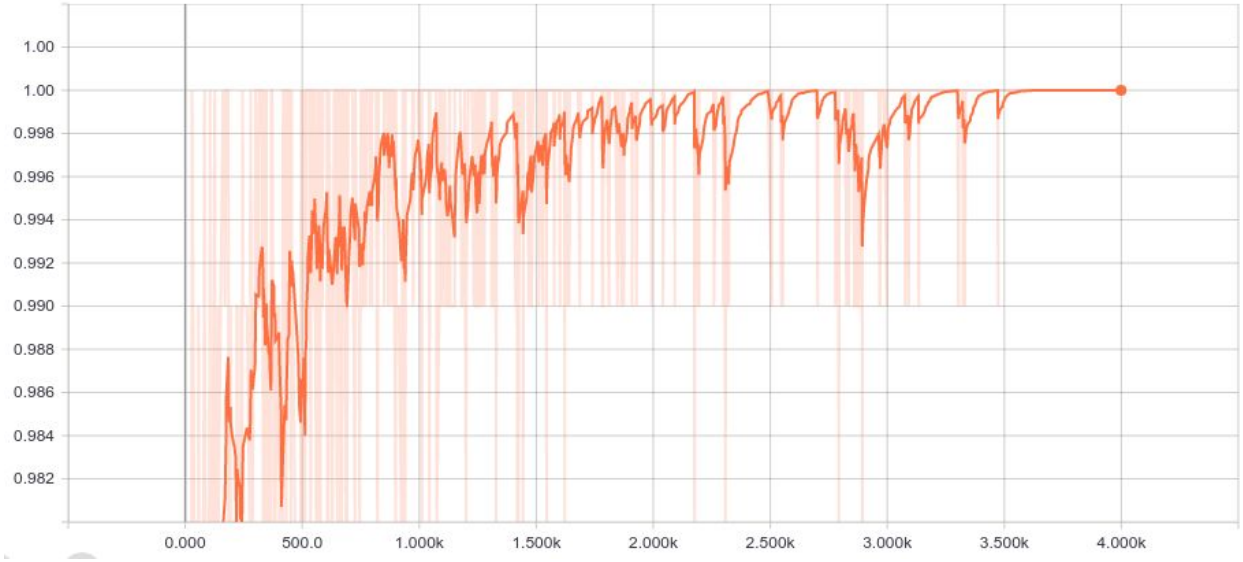
- --bottleneck_dir './retrained_model/bottlenecks'
- --output_graph './retrained_model/output_graph.pb'
- --image_dir './input/training'
- --output_labels './retrained_model/output_labels.txt'

Full command (takes around 20-30 minutes to complete):

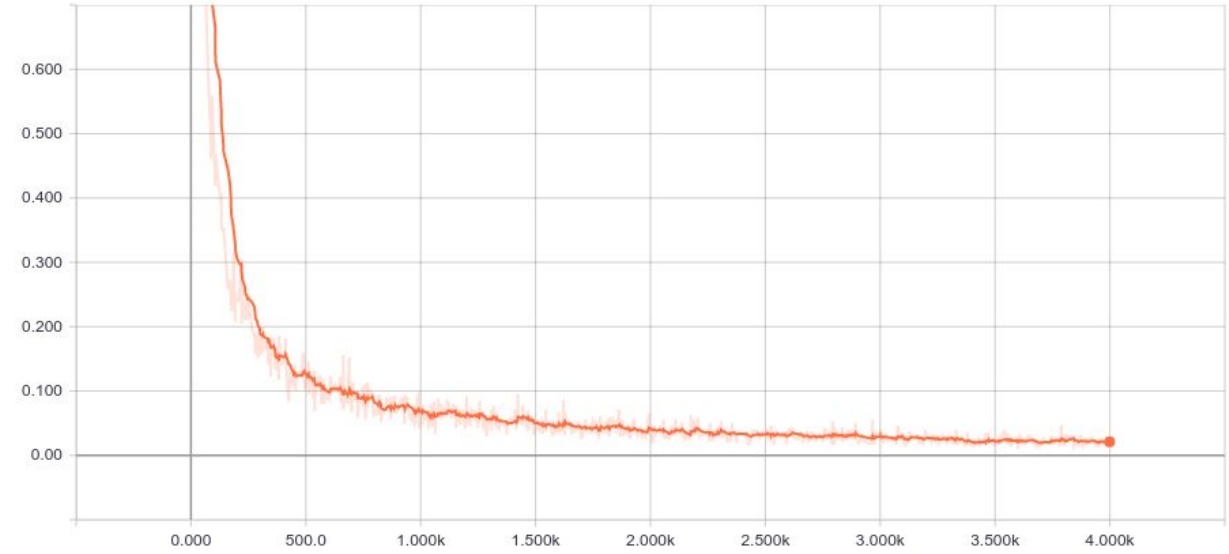
python retrain.py --bottleneck_dir './retrained_model/bottlenecks' --output_graph './retrained_model/output_graph.pb' --image_dir './input/training' --output_labels

'./retrained_model/output_labels.txt'

Below a graph is shown, which represents the change of accuracy during the training.



The second graph represents constant shifting of the cross entropy while training the model. Both graphs were directly taken from TensorFlow Hub.



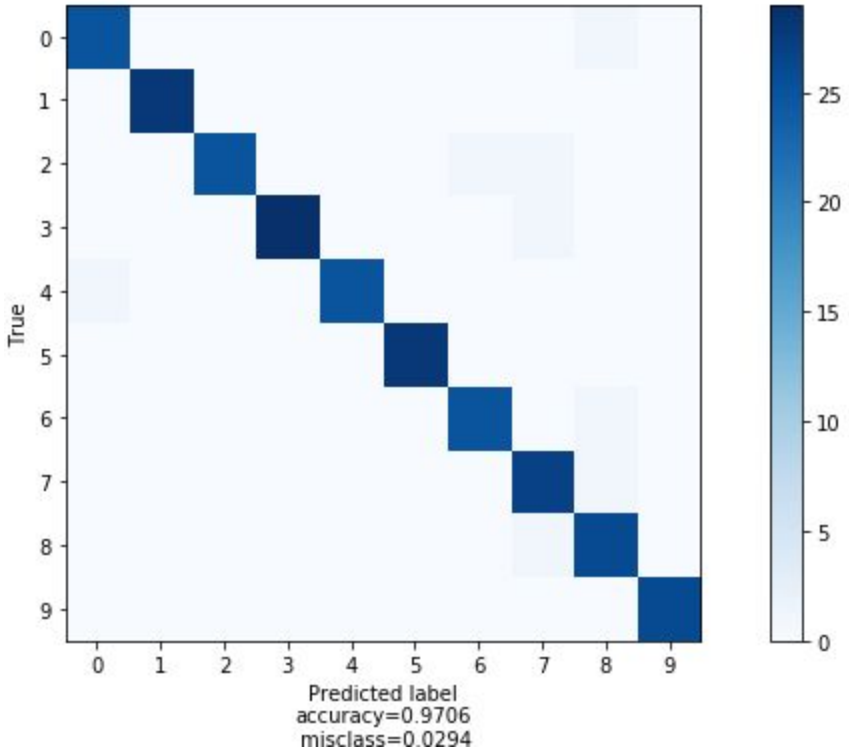
Transfer learning Validation

In order to properly validate the new classifier, the author had to write a python script. The script loads all the labels, loads the newly generated graph, shuffles the testing dataset and runs the classifier on every image in the set. The loop outputs the

confidence level and the classification result after each image. A sample validation output is shown below:

```
('n3', 0.9962218, 'Correct!')
('n7', 0.92149746, 'Correct!')
('n3', 0.9941415, 'Correct!')
('n1', 0.9924239, 'Correct!')
('n9', 0.9987066, 'Correct!')
('n8', 0.9707032, 'Correct!')
('n2', 0.83481497, 'Correct!')
('n4', 0.96305114, 'Correct!')
('n3', 0.9744708, 'Correct!')
('n0', 0.994269, 'Correct!')
('n3', 0.9987514, 'Correct!')
('n5', 0.9837325, 'Correct!')
('n7', 0.62718266, 'False! -Correct was: ', 'n8', ' Path: ', 'input/validation/n8/n813.jpg')
```

The final accuracy of the retrained model was around 0.9706. It is clearly visible that the model is performing extremely well- misclassifying only few instances from the validation image set. Furthermore, the accuracy can be improved to be almost 100% if the images were left in higher resolution, however this was not done because of performance reasons. Below the evaluation confusion matrix is presented:



Linear model

A linear classifier uses object's features to identify which class it belongs to by making a classification decision based on the value of a linear combination of the characteristics. Such classifiers work well generally for problems with many features, reaching accuracy levels comparable to non-linear classifiers while taking less time to train and use. Although image classification is typically too complex for a simple linear classifier, the author chose to explore its intricacies as it provided great learning experience because of the variety techniques used.

In order to use the classifier it first needed to be defined as an TensorFlow custom estimator with expected features ([150,150,3] shape), number of classes (10) and directory of the model.

Then the function "generate_input_fn()" was defined. It generates an expected "input_fn()" function for training and validating the classifier. This function takes in the "train" or "test" data frames and the batch size. It reads the image data from the passed data frame, shuffles the images and converts the images and expected labels to tensors. Finally, it returns batches of images and corresponding labels.

Finally, the procedure to train the classifier was initiated with 2000 training steps passed as a parameter. A snapshot of the training output is presented below:

```
INFO:tensorflow:loss = 92.10341, step = 1
INFO:tensorflow:global_step/sec: 44.6067
INFO:tensorflow:loss = 3334953.0, step = 101 (2.187 sec)
INFO:tensorflow:global_step/sec: 66.6263
INFO:tensorflow:loss = 1548382.0, step = 201 (1.472 sec)
INFO:tensorflow:global_step/sec: 68.2619
INFO:tensorflow:loss = 1413623.0, step = 301 (1.465 sec)
INFO:tensorflow:global_step/sec: 68.3958
INFO:tensorflow:loss = 297406.0, step = 401 (1.462 sec)
INFO:tensorflow:global_step/sec: 69.1665
INFO:tensorflow:loss = 111411.25, step = 501 (1.446 sec)
```

Linear classifier validation

As expected linear classifier performed quite poorly, averaging only around 55% accuracy, however it still performs better than random guessing. Also, the performance

of the model may further be improved by adjusting the options or by storing images in higher resolution. The output of the validation is presented below:

```
INFO:tensorflow:Evaluation [10/100]
INFO:tensorflow:Evaluation [20/100]
INFO:tensorflow:Evaluation [30/100]
INFO:tensorflow:Evaluation [40/100]
INFO:tensorflow:Evaluation [50/100]
INFO:tensorflow:Evaluation [60/100]
INFO:tensorflow:Evaluation [70/100]
INFO:tensorflow:Evaluation [80/100]
INFO:tensorflow:Evaluation [90/100]
INFO:tensorflow:Evaluation [100/100]
INFO:tensorflow:Finished evaluation at 2018-04-20-16:52:54
INFO:tensorflow:Saving dict for global step 12000: accuracy = 0.5665, average_loss = 14274.221, global_step = 1200
0, loss = 570968.8
```

Linear classifier predictions

In order to make the classifier work for singular images the author defined a predictor. It takes a random image from the test set (could be easily changed to a specific image) and starts the classification process on that image. It outputs the expected label and actual label after it is done. Sample output is shown below:

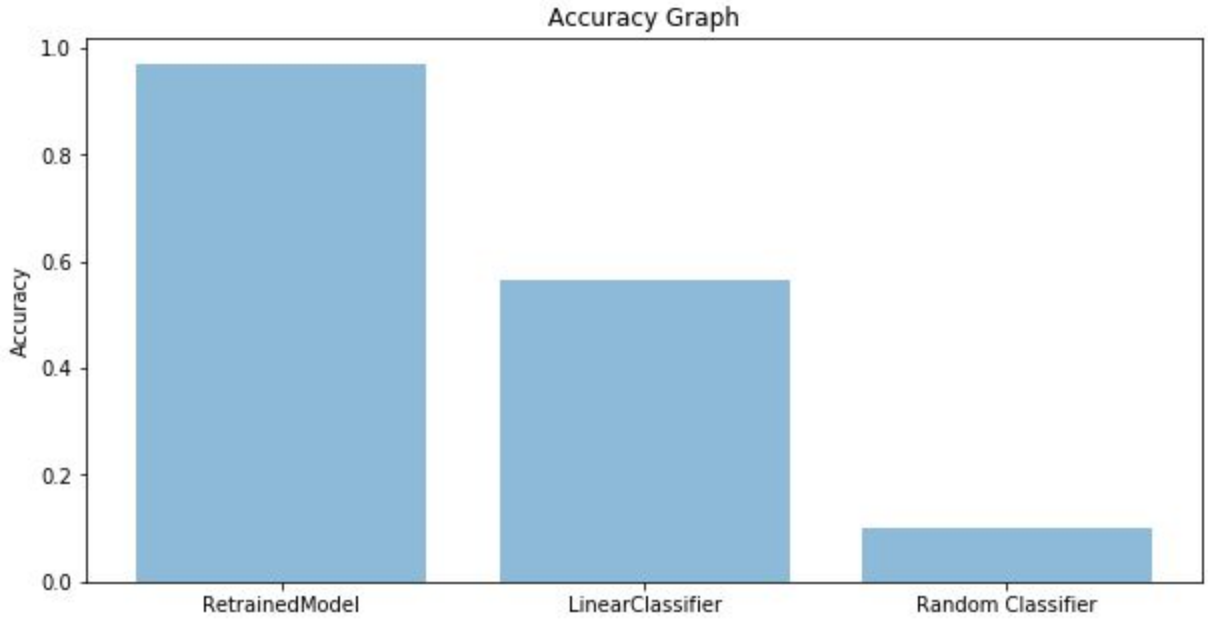
```
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from ./linear_model/model.ckpt-2000
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
Actual image label: n0
Predicted output: n0
```

Evaluation

Each model was evaluated with respect to the original problem in their corresponding validation sections, this section briefly compares accuracy between the examined models and a random classification model. Given that there are 10 classes and the test image set has almost even class distribution, assumption was made that the random classifier has 10% accuracy.

Average accuracy scores:

- Random Classifier: 0.1
- LinearClassifier: 0.5665
- RetrainedModel: 0.970588235294



Conclusion

To summarize, the 10 Monkey Species image dataset was examined in order to solve the 10 way classification problem - deducing to which class does a given image belong to. While analyzing the dataset the author had to deal with various new challenges, such as importing images, preprocessing (reshaping, rescaling etc) images, transforming data to tensors, as well as, preparing, training and validating the TensorFlow classifiers.

First step was importing and preparing the data, thus, new data frame was created for each: training and testing image sets. These data frames were used to assist further image dataset analysis. The data frame features were:

- Image paths;
- Image representations (array);
- Image labels;
- Image integer encoded labels;
- Image one hot encoded labels.

Moreover, few different image classifiers were implemented and their accuracy scores were compared.The retrained model classifier, as expected, had an edge over linear classifier with an average accuracy score of more than 97%. While linear classifier was accurate only around 55% of the time.

Finally, this project was a valuable learning experience for the author, who learned how to cope with TensorFlow data manipulation, visualization, and analysis techniques. The issue of discovering the most appropriate methods for image dataset analysis, for example, proved to be both engaging and challenging. Along with the latter insights, the author also had an opportunity to improve Python coding principles.