COMP6065 – Artificial Intelligence (L5AC)

Michael Fernanlie	-	1901530133
Steven Muliamin	-	1901499280

Odd Semester 2017/2018

Contents

Introduction

Mineral Classifier is an image classifier application that utilizes a pre-trained convolutional neural network (CNN), called Inception. Inception model provides layers of neural network with the early layers containing shape detection and edge detection functionality. With the Inception model, custom dataset can be used to train the classification accuracy and will be located at the final layers of the model. The size of dataset that can be allocated currently is unlimited and naturally for a bigger training set, the amount of time required by the application to learn at startup is longer. However, big training set often benefits the prediction accuracy of the application. The inspiration for this image classifier is based on a youtuber with interest in machine learning, Siraj Raval, on one of his video on youtube "Build a Tensorflow Image Classifier in 5 Min" (https://www.youtube.com/watch?v=QfNvhPx5Px8). The git repository for this project itself is https://github.com/mfernanlie/AI-FinalProject-ImageClassifier.

As its name suggests, Mineral Classifier is able to identify a mineral given their photos. Mineral has been one of the most important natural resources for human. Its usage can be found on many occupation field. However, the name of the minerals is often hard to remember and it can't be easily be recognized by its physical appearance as well. Consequently, Mineral Classifier can help people who are not familiar with minerals to identify them. For this project, we didn't collect all variants of mineral that are known to human. But we selected 4 of them, which is Malachite, Bauxite, Amethyst, and Gypsum, for Mineral Classifier so the user can identify their proposed image of mineral that they wish to identify. It is important to note that Mineral Classifier doesn't return results of classification as in 'how alike this image is to a certain label' bur rather 'how close this particular image is to the category that we made available'.

Solution & Features

Mineral Classifier application uses a preprocessing artificial neural networks method called Convolutional Neural Network (CNN). A CNN generally consists of several layers of neural network where information forwarding occurred. Using TensorFlow, an open source platform for data processing and graphing by Google, a machine learning model called Inception model, which is a pre-trained Convolutional Neural Network (CNN). Inception is an image classification method supported by TensorFlow, which was made by Google and tested with over a hundred thousand images and tenth thousands of categories. This model consists of multiple layers and among the front layers contain shape detection and edge detection functionality. It is important to note that the only layer that we can modify is the final layer of the model, where representation of our customized datasets is concerned. Aside from the final layer, the rest of the layers was pre-trained and its training data is transferred to our implementation every time it ran. This method is called Transfer Learning. After the training dataset is given, it will be processed on final layer of the Inception model. Once the training is completed, the user can propose image to be classified.

Solution Design Architecture

Since we are using TensorFlow as our platform to develop our machine learning method, there are two choices of computational resources that we can use. We can use either GPU or CPU computational resources for this project. We decided to use the CPU to run the machine learning method with specifications of:

CPU: Intel i7-7700HQ

RAM: 8GB DDR4

For using the inception model with TensorFlow, we must first set up a virtual environment for Linux OS. In this project, Docker, a distributed package platform for Linux and Windows virtual machine, is used. After installing Docker package, run Docker server through terminal command. We used Windows PowerShell since it recognizes both Windows and Linux terminal command. Once the Docker server is running, a local folder must be created in our $HOME directory. In Windows, $HOME directory is commonly referred as C:\user\{user's_name}. The local folder that we created in $HOME directory must then be linked and accessible from Docker VM.

After the local folder we have created in $HOME, get the Inception model (CNN) from GIT repository that was provided by Siraj Raval inside Docker environment, https://github.com/llSourcell/tensorflow_image_classifier. The files we pull from the GIT repository contains both Inception which contains pre-trained data and the code for training. With that being said, since the GIT repository is pulled inside Docker environment, we can't modify or view the files in that repository. Pulling all the files from that git could take some time.

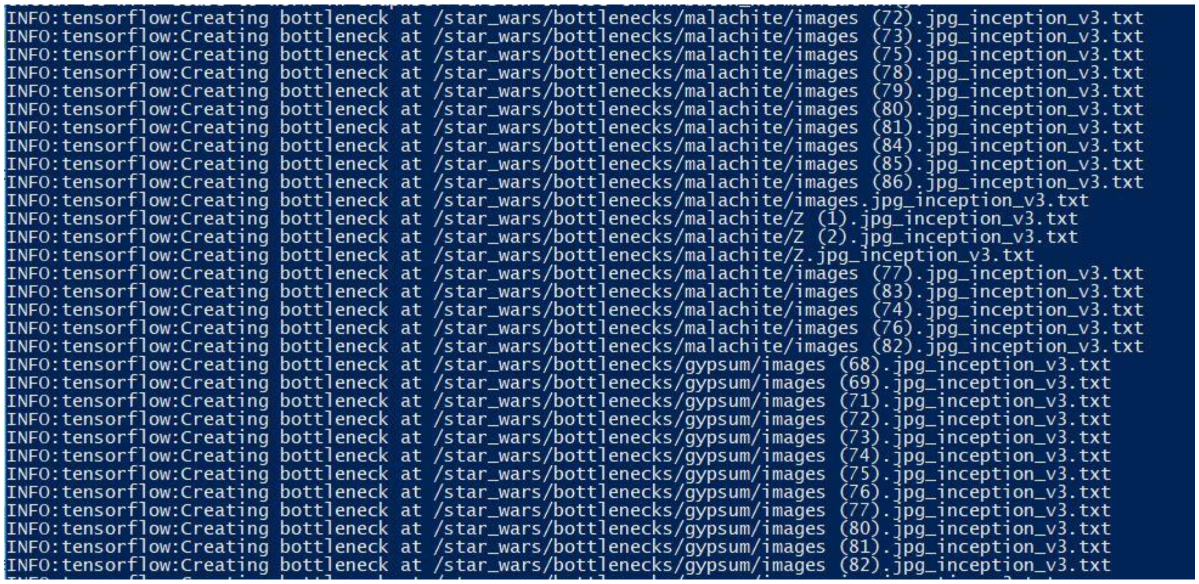
Once the Inception and the code have been pulled from the repository, the training can be done immediately. Before training is executed, it is imperative that we provide our own dataset of images (format .jpg) inside the local folder that we created in $HOME directory. The rules of storing your own customized datasets, is that for all images that is stored on a folder, the images represent that folder as a label or category in the application. The application should also accept at least two available label. With proper references to the location and command, the application will then conduct the training of the provided dataset that we store before. For this project, our dataset was taken from simple google image search. With a help from Chrome extension called Fatkun, it allows us to save all images that appeared on the first page of google image search. However, we don't take all the images that appeared on the google search as our training data. We still have to do some manual verification and filter to the data before taking it to make sure there are no bad data taken. Depending on the size of our dataset, training could take some time. In our case, we provided around 400 images of Bauxite, Malachite, Gypsum, and Amethyst. For 400 images it took around 5 minutes for the application to finish the training. The reason why the training takes shorter time for big dataset is that we used a pre-trained CNN model from Inception. Be reminded that the amount of time it took to train the images are dependent on your computer computational power. Refer to the beginning of this section for our computer specifications.

When the training is completed, a notification will appear on the Docker environment prompt. As of now, the application is ready to accept an image so it can be classified as to what category it is closest to. Be reminded again that the results this application returns will be that of a percentage. It does not imply the percentage of how similar an image is to a label or category, but rather the percentage of how close an image is to a label or category. Hence the rules of having to have at least two label or category in the application since having only one label will return 100% to that only label regardless of how similar the test images is to the trained images.

It is advised that when you're creating your own dataset for training, avoid creating two labels containing too similar images. Images with high similarity under two different labels might compromise the accuracy of the application.

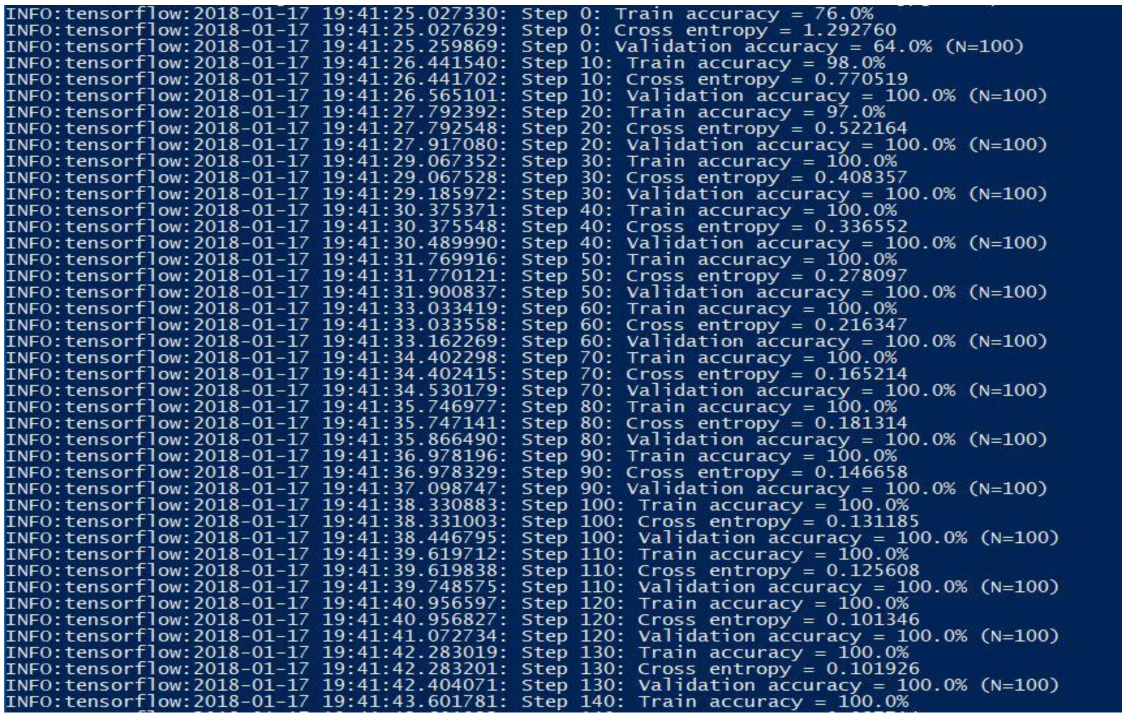
Code Snippet

Training the dataset, phase 1. Loading Inception model and bottlenecks for lower/first layers (shape detection and edge detection). Opened in Windows PowerShell.



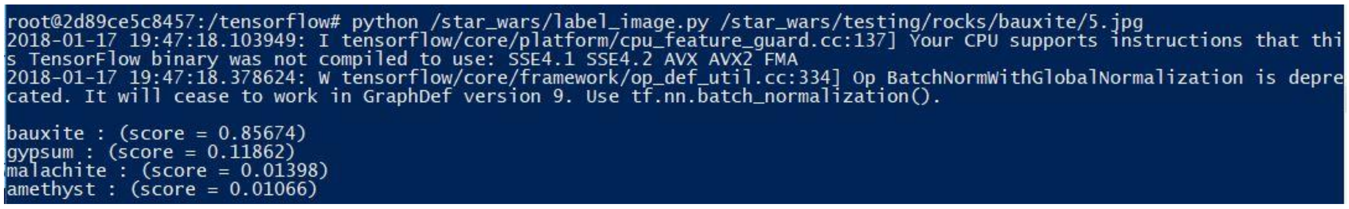
```
INFO:tensorflow:Creating bottleneck at /star_wars/bottlenecks/malachite/images (72).jpg_inception_v3.txt
INFO:tensorflow:Creating bottleneck at /star_wars/bottlenecks/malachite/images (73).jpg_inception_v3.txt
INFO:tensorflow:Creating bottleneck at /star_wars/bottlenecks/malachite/images (75).jpg_inception_v3.txt
INFO:tensorflow:Creating bottleneck at /star_wars/bottlenecks/malachite/images (78).jpg_inception_v3.txt
INFO:tensorflow:Creating bottleneck at /star_wars/bottlenecks/malachite/images (79).jpg_inception_v3.txt
INFO:tensorflow:Creating bottleneck at /star_wars/bottlenecks/malachite/images (80).jpg_inception_v3.txt
INFO:tensorflow:Creating bottleneck at /star_wars/bottlenecks/malachite/images (81).jpg_inception_v3.txt
INFO:tensorflow:Creating bottleneck at /star_wars/bottlenecks/malachite/images (84).jpg_inception_v3.txt
INFO:tensorflow:Creating bottleneck at /star_wars/bottlenecks/malachite/images (85).jpg_inception_v3.txt
INFO:tensorflow:Creating bottleneck at /star_wars/bottlenecks/malachite/images (86).jpg_inception_v3.txt
INFO:tensorflow:Creating bottleneck at /star_wars/bottlenecks/malachite/images.jpg_inception_v3.txt
INFO:tensorflow:Creating bottleneck at /star_wars/bottlenecks/malachite/Z (1).jpg_inception_v3.txt
INFO:tensorflow:Creating bottleneck at /star_wars/bottlenecks/malachite/Z (2).jpg_inception_v3.txt
INFO:tensorflow:Creating bottleneck at /star_wars/bottlenecks/malachite/Z.jpg_inception_v3.txt
INFO:tensorflow:Creating bottleneck at /star_wars/bottlenecks/malachite/images (77).jpg_inception_v3.txt
INFO:tensorflow:Creating bottleneck at /star_wars/bottlenecks/malachite/images (83).jpg_inception_v3.txt
INFO:tensorflow:Creating bottleneck at /star_wars/bottlenecks/malachite/images (74).jpg_inception_v3.txt
INFO:tensorflow:Creating bottleneck at /star_wars/bottlenecks/malachite/images (76).jpg_inception_v3.txt
INFO:tensorflow:Creating bottleneck at /star_wars/bottlenecks/malachite/images (82).jpg_inception_v3.txt
INFO:tensorflow:Creating bottleneck at /star_wars/bottlenecks/gypsum/images (68).jpg_inception_v3.txt
INFO:tensorflow:Creating bottleneck at /star_wars/bottlenecks/gypsum/images (69).jpg_inception_v3.txt
INFO:tensorflow:Creating bottleneck at /star_wars/bottlenecks/gypsum/images (71).jpg_inception_v3.txt
INFO:tensorflow:Creating bottleneck at /star_wars/bottlenecks/gypsum/images (72).jpg_inception_v3.txt
INFO:tensorflow:Creating bottleneck at /star_wars/bottlenecks/gypsum/images (73).jpg_inception_v3.txt
INFO:tensorflow:Creating bottleneck at /star_wars/bottlenecks/gypsum/images (74).jpg_inception_v3.txt
INFO:tensorflow:Creating bottleneck at /star_wars/bottlenecks/gypsum/images (75).jpg_inception_v3.txt
INFO:tensorflow:Creating bottleneck at /star_wars/bottlenecks/gypsum/images (76).jpg_inception_v3.txt
INFO:tensorflow:Creating bottleneck at /star_wars/bottlenecks/gypsum/images (77).jpg_inception_v3.txt
INFO:tensorflow:Creating bottleneck at /star_wars/bottlenecks/gypsum/images (80).jpg_inception_v3.txt
INFO:tensorflow:Creating bottleneck at /star_wars/bottlenecks/gypsum/images (81).jpg_inception_v3.txt
INFO:tensorflow:Creating bottleneck at /star_wars/bottlenecks/gypsum/images (82).jpg_inception_v3.txt
```

Training the dataset, phase 2. Training the actual dataset on the final/upper layers that we prepared.



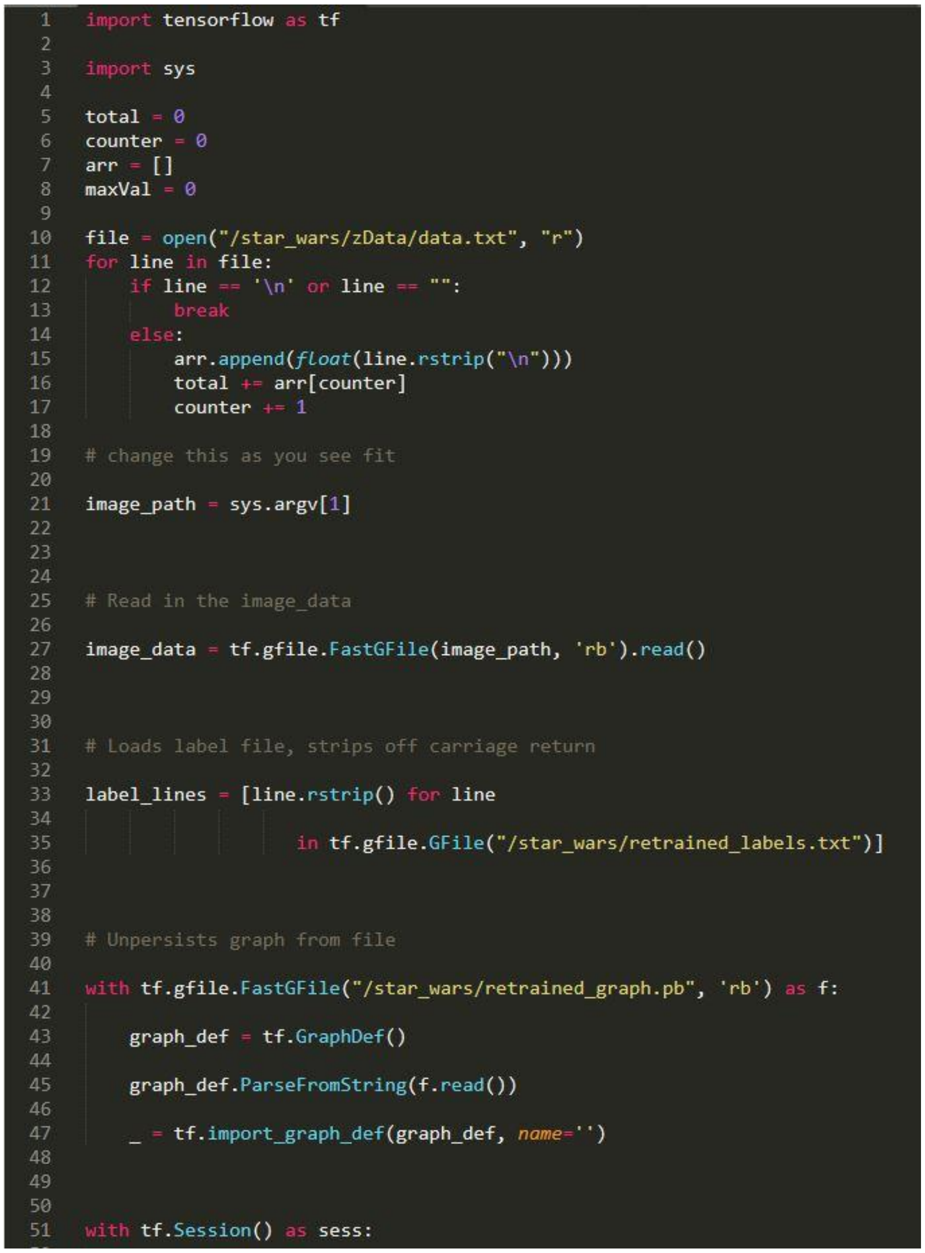
```
INFO:tensorflow:2018-01-17 19:41:25.027330: Step 0: Train accuracy = 76.0%
INFO:tensorflow:2018-01-17 19:41:25.027629: Step 0: Cross entropy = 1.292760
INFO:tensorflow:2018-01-17 19:41:25.259869: Step 0: Validation accuracy = 64.0% (N=100)
INFO:tensorflow:2018-01-17 19:41:26.441540: Step 10: Train accuracy = 98.0%
INFO:tensorflow:2018-01-17 19:41:26.441702: Step 10: Cross entropy = 0.770519
INFO:tensorflow:2018-01-17 19:41:26.565101: Step 10: Validation accuracy = 100.0% (N=100)
INFO:tensorflow:2018-01-17 19:41:27.792392: Step 20: Train accuracy = 97.0%
INFO:tensorflow:2018-01-17 19:41:27.792548: Step 20: Cross entropy = 0.522164
INFO:tensorflow:2018-01-17 19:41:27.917080: Step 20: Validation accuracy = 100.0% (N=100)
INFO:tensorflow:2018-01-17 19:41:29.067352: Step 30: Train accuracy = 100.0%
INFO:tensorflow:2018-01-17 19:41:29.067528: Step 30: Cross entropy = 0.408357
INFO:tensorflow:2018-01-17 19:41:29.185972: Step 30: Validation accuracy = 100.0% (N=100)
INFO:tensorflow:2018-01-17 19:41:30.375371: Step 40: Train accuracy = 100.0%
INFO:tensorflow:2018-01-17 19:41:30.375548: Step 40: Cross entropy = 0.336552
INFO:tensorflow:2018-01-17 19:41:30.489990: Step 40: Validation accuracy = 100.0% (N=100)
INFO:tensorflow:2018-01-17 19:41:31.769916: Step 50: Train accuracy = 100.0%
INFO:tensorflow:2018-01-17 19:41:31.770121: Step 50: Cross entropy = 0.278097
INFO:tensorflow:2018-01-17 19:41:31.900837: Step 50: Validation accuracy = 100.0% (N=100)
INFO:tensorflow:2018-01-17 19:41:33.033419: Step 60: Train accuracy = 100.0%
INFO:tensorflow:2018-01-17 19:41:33.033558: Step 60: Cross entropy = 0.216347
INFO:tensorflow:2018-01-17 19:41:33.162269: Step 60: Validation accuracy = 100.0% (N=100)
INFO:tensorflow:2018-01-17 19:41:34.402298: Step 70: Train accuracy = 100.0%
INFO:tensorflow:2018-01-17 19:41:34.402415: Step 70: Cross entropy = 0.165214
INFO:tensorflow:2018-01-17 19:41:34.530179: Step 70: Validation accuracy = 100.0% (N=100)
INFO:tensorflow:2018-01-17 19:41:35.746977: Step 80: Train accuracy = 100.0%
INFO:tensorflow:2018-01-17 19:41:35.747141: Step 80: Cross entropy = 0.181314
INFO:tensorflow:2018-01-17 19:41:35.866490: Step 80: Validation accuracy = 100.0% (N=100)
INFO:tensorflow:2018-01-17 19:41:36.978196: Step 90: Train accuracy = 100.0%
INFO:tensorflow:2018-01-17 19:41:36.978329: Step 90: Cross entropy = 0.146658
INFO:tensorflow:2018-01-17 19:41:37.098747: Step 90: Validation accuracy = 100.0% (N=100)
INFO:tensorflow:2018-01-17 19:41:38.330883: Step 100: Train accuracy = 100.0%
INFO:tensorflow:2018-01-17 19:41:38.331003: Step 100: Cross entropy = 0.131185
INFO:tensorflow:2018-01-17 19:41:38.446795: Step 100: Validation accuracy = 100.0% (N=100)
INFO:tensorflow:2018-01-17 19:41:39.619712: Step 110: Train accuracy = 100.0%
INFO:tensorflow:2018-01-17 19:41:39.619838: Step 110: Cross entropy = 0.125608
INFO:tensorflow:2018-01-17 19:41:39.748575: Step 110: Validation accuracy = 100.0% (N=100)
INFO:tensorflow:2018-01-17 19:41:40.956597: Step 120: Train accuracy = 100.0%
INFO:tensorflow:2018-01-17 19:41:40.956827: Step 120: Cross entropy = 0.101346
INFO:tensorflow:2018-01-17 19:41:41.072734: Step 120: Validation accuracy = 100.0% (N=100)
INFO:tensorflow:2018-01-17 19:41:42.283019: Step 130: Train accuracy = 100.0%
INFO:tensorflow:2018-01-17 19:41:42.283201: Step 130: Cross entropy = 0.101926
INFO:tensorflow:2018-01-17 19:41:42.404071: Step 130: Validation accuracy = 100.0% (N=100)
INFO:tensorflow:2018-01-17 19:41:43.601781: Step 140: Train accuracy = 100.0%
```

Test results, given the image to be tested it will return all the labels with percentage of how close the given image looks to the label. Ex: the image is 85% close to being labelled as ‘Bauxite’

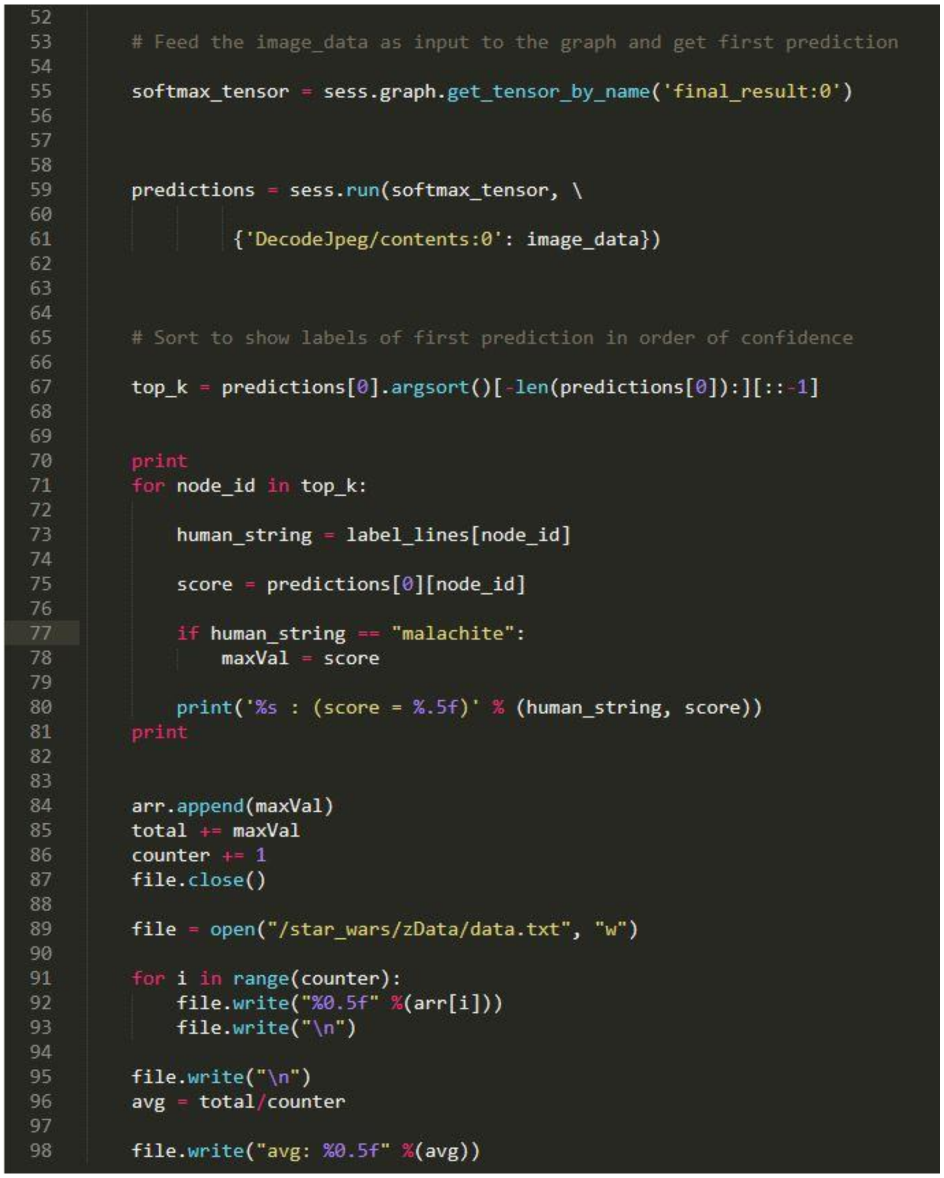


```
root@2d89ce5c8457:/tensorflow# python /star_wars/label_image.py /star_wars/testing/rocks/bauxite/5.jpg
2018-01-17 19:47:18.103949: I tensorflow/core/platform/cpu_feature_guard.cc:137] Your CPU supports instructions that thi
s TensorFlow binary was not compiled to use: SSE4.1 SSE4.2 AVX AVX2 FMA
2018-01-17 19:47:18.378624: W tensorflow/core/framework/op_def_util.cc:334] Op BatchNormWithGlobalNormalization is depre
cated. It will cease to work in GraphDef version 9. Use tf.nn.batch_normalization().

bauxite : (score = 0.85674)
gypsum : (score = 0.11862)
malachite : (score = 0.01398)
amethyst : (score = 0.01066)
```

Logical code for the image comparison and classifying in final layer.



```python
import tensorflow as tf

import sys

total = 0
counter = 0
arr = []
maxVal = 0

file = open("/star_wars/zData/data.txt", "r")
for line in file:
    if line == '\n' or line == "":
        break
    else:
        arr.append(float(line.rstrip("\n")))
        total += arr[counter]
        counter += 1

# change this as you see fit

image_path = sys.argv[1]



# Read in the image_data

image_data = tf.gfile.FastGFile(image_path, 'rb').read()



# Loads label file, strips off carriage return

label_lines = [line.rstrip() for line

                   in tf.gfile.GFile("/star_wars/retrained_labels.txt")]



# Unpersists graph from file

with tf.gfile.FastGFile("/star_wars/retrained_graph.pb", 'rb') as f:

    graph_def = tf.GraphDef()

    graph_def.ParseFromString(f.read())

    _ = tf.import_graph_def(graph_def, name='')



with tf.Session() as sess:
```

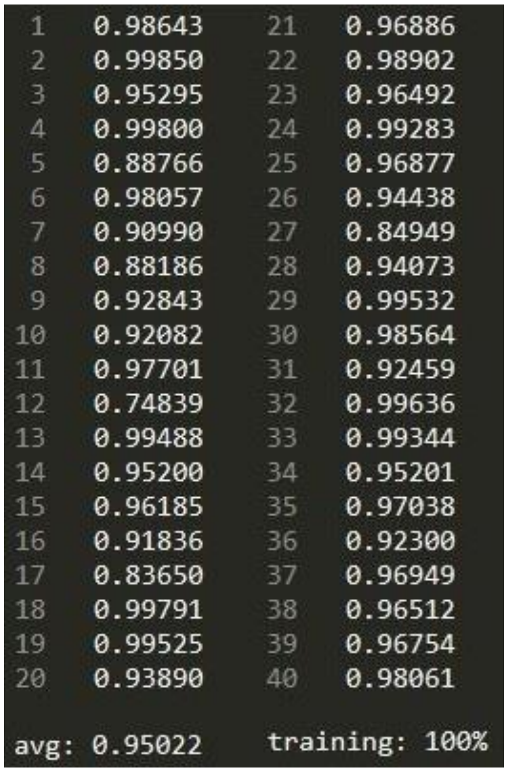


```python

    # Feed the image_data as input to the graph and get first prediction

    softmax_tensor = sess.graph.get_tensor_by_name('final_result:0')



    predictions = sess.run(softmax_tensor, \

             {'DecodeJpeg/contents:0': image_data})



    # Sort to show labels of first prediction in order of confidence

    top_k = predictions[0].argsort()[-len(predictions[0]):][::-1]


    print
    for node_id in top_k:

        human_string = label_lines[node_id]

        score = predictions[0][node_id]

        if human_string == "malachite":
            maxVal = score

        print('%s : (score = %.5f)' % (human_string, score))
    print


    arr.append(maxVal)
    total += maxVal
    counter += 1
    file.close()

    file = open("/star_wars/zData/data.txt", "w")

    for i in range(counter):
        file.write("%0.5f" %(arr[i]))
        file.write("\n")

    file.write("\n")
    avg = total/counter

    file.write("avg: %0.5f" %(avg))
```

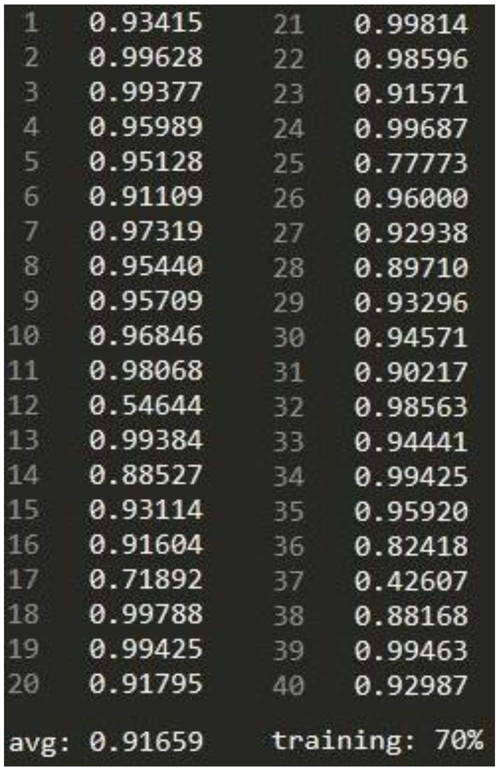
Test Results

In this section, we’re showing an analysis of Mineral Classifier program. The test case is divided into 4 scenario, where the training dataset contains 100%, 70%, 40% or 20% of images. For 100% images or standard scenario, the training set contains 400 images with a hundred for each mineral/label. So for 70% there will be 70 images for each mineral/label and total of 280 images for training dataset. For the testing case however, we will always have 40 test images with 10 images for each mineral/label regardless of the amount of the training dataset.

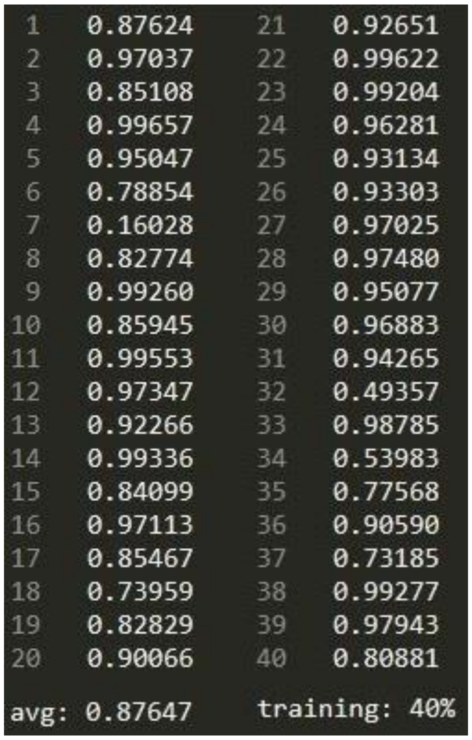
Example of testing



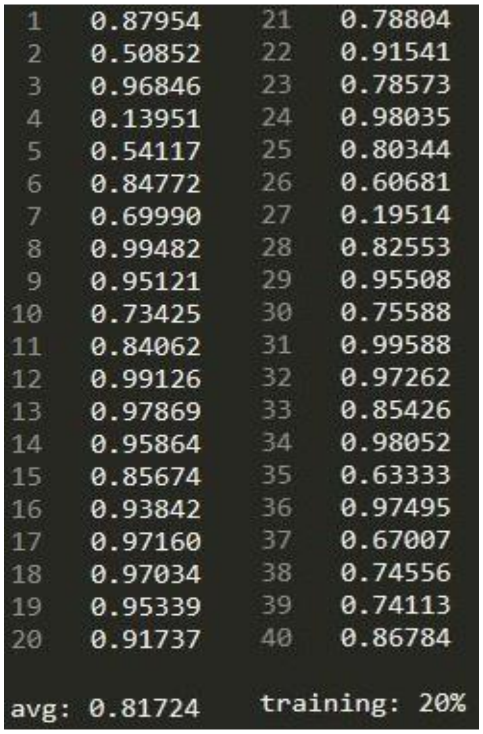
```
 1   0.98643    21   0.96886
 2   0.99850    22   0.98902
 3   0.95295    23   0.96492
 4   0.99800    24   0.99283
 5   0.88766    25   0.96877
 6   0.98057    26   0.94438
 7   0.90990    27   0.84949
 8   0.88186    28   0.94073
 9   0.92843    29   0.99532
10   0.92082    30   0.98564
11   0.97701    31   0.92459
12   0.74839    32   0.99636
13   0.99488    33   0.99344
14   0.95200    34   0.95201
15   0.96185    35   0.97038
16   0.91836    36   0.92300
17   0.83650    37   0.96949
18   0.99791    38   0.96512
19   0.99525    39   0.96754
20   0.93890    40   0.98061

avg: 0.95022    training: 100%
```



```
 1   0.93415    21   0.99814
 2   0.99628    22   0.98596
 3   0.99377    23   0.91571
 4   0.95989    24   0.99687
 5   0.95128    25   0.77773
 6   0.91109    26   0.96000
 7   0.97319    27   0.92938
 8   0.95440    28   0.89710
 9   0.95709    29   0.93296
10   0.96846    30   0.94571
11   0.98068    31   0.90217
12   0.54644    32   0.98563
13   0.99384    33   0.94441
14   0.88527    34   0.99425
15   0.93114    35   0.95920
16   0.91604    36   0.82418
17   0.71892    37   0.42607
18   0.99788    38   0.88168
19   0.99425    39   0.99463
20   0.91795    40   0.92987

avg: 0.91659    training: 70%
```

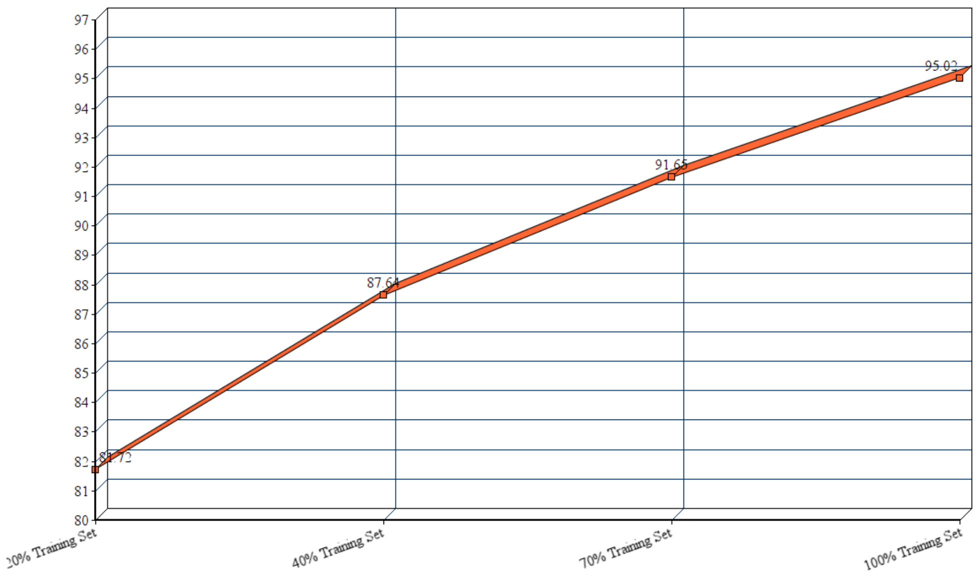


```
1   0.87624     21   0.92651
2   0.97037     22   0.99622
3   0.85108     23   0.99204
4   0.99657     24   0.96281
5   0.95047     25   0.93134
6   0.78854     26   0.93303
7   0.16028     27   0.97025
8   0.82774     28   0.97480
9   0.99260     29   0.95077
10  0.85945     30   0.96883
11  0.99553     31   0.94265
12  0.97347     32   0.49357
13  0.92266     33   0.98785
14  0.99336     34   0.53983
15  0.84099     35   0.77568
16  0.97113     36   0.90590
17  0.85467     37   0.73185
18  0.73959     38   0.99277
19  0.82829     39   0.97943
20  0.90066     40   0.80881

avg: 0.87647      training: 40%
```



```
1   0.87954     21   0.78804
2   0.50852     22   0.91541
3   0.96846     23   0.78573
4   0.13951     24   0.98035
5   0.54117     25   0.80344
6   0.84772     26   0.60681
7   0.69990     27   0.19514
8   0.99482     28   0.82553
9   0.95121     29   0.95508
10  0.73425     30   0.75588
11  0.84062     31   0.99588
12  0.99126     32   0.97262
13  0.97869     33   0.85426
14  0.95864     34   0.98052
15  0.85674     35   0.63333
16  0.93842     36   0.97495
17  0.97160     37   0.67007
18  0.97034     38   0.74556
19  0.95339     39   0.74113
20  0.91737     40   0.86784

avg: 0.81724      training: 20%
```

From the results of testing, we can see that there are 40 lines of results printing float numbers. Actually it is the percentage to the correct label in each test case. And as we have 40 test images, we have 40 lines. The total correct percentage will then be sum and divided to get the average accuracy for that scenario. Seeing from the training dataset of 100% to 20% we can see that the average accuracy drops as the total training images are reduced.



Additionally, we also retrieved some data regarding category accuracy of testing done on 100% training dataset. The data we retrieved can show which category are more likely to be correctly predicted by our application.

Minerals Type	Accuracy
Bauxite	94.45%
Gypsum	93.21%
Amethyst	95.63%
Malachite	96.42%

According to the data we can see that Malachite gets the highest accuracy while Gypsum gets the lowest. But the difference between each type is very small and tolerable. This leads us to believe that the application can correctly predict the category of mineral 95% of the time.

Conclusions

An image classification program using Inception model where the lower layers are pre-trained is certainly faster and more reliable than other standard method. It is observable from our test case that even using 20% of the total training dataset, the application still correctly classifies an image at accuracy of 81%. This goes to show that the application is quick to learn new images that it hasn't seen before because some detection functionalities are pre-trained. Of course provided with larger training dataset, the application will surely perform better in correctly classifying an image.

Bibliography

GIT Repositories

https://github.com/mfernanlie/AI-FinalProject-ImageClassifier

https://github.com/llSourcell/tensorflow_image_classifier

Development Tools

https://www.tensorflow.org

https://www.docker.com/

https://chrome.google.com/webstore/detail/fatkun-batch-download-ima/nnjjahlikiabnchcpehcpkdeckfgnohf?hl=en

Video References

https://www.youtube.com/watch?v=QfNvhPx5Px8

Application Guide (by us)

https://drive.google.com/open?id=1QQTGaBs_7iZDUGBfzYdpc2Rkxugd1oYS