Tensorflow Project: Gemstone Classification

Hong Son, Yiyun Zhang, Brian Kong

INFO 607 - Applied Database Technologies

May 26, 2021

Introduction

Our area of study is classifying gemstones. A gemstone is mineral, stone, or organic matter that can be cut and polished or used as jewelry or other ornament (Maula & et. al, 2017). We are fascinated by the number of different types of gemstones and the perceived value of gemstones. Some gemstones such as diamond, ruby, sapphire, and emerald carry more value than other gemstones. Thus, they are categorized into various types. The four characteristics used to categorize the type of gemstone based on physical aspects consist of the hardness, density, refraction of light, and the color of the mineral. Our research expands from a previous research done by Syarif Hidayatullah State Islamic University where they worked on image processing problems around three types of gemstone (Ruby, Sapphire, and Emerald). In this study, the researchers used the Artificial Neural Network (ANN) machine learning model and obtained relatively high accuracy for the classification problem. Our study takes it up a notch by looking at more gemstone classes through Convolutional Neural Networks (CNN), a model that is a kind of ANN. A CNN, specifically, has one or more layers of convolution units. This will be further discussed in the CNN section of this report. After using Tensorflow, we discovered that Tensorflow is a useful framework for deep learning problems and that it is not too hard to pick up.

Goal and Purpose of the Project

The goal of this project is to explore the functionality of Tensorflow by attempting image classification. This project could also help with the creation of a gemstone identification application. Researchers, professionals, or individuals interested in the collection and identification of gemstones would be interested in our project. For example, if an individual wants to repair his or her gemstones, they could use the gemstone application to get the right gemstones. Another is in the development of a gemstone collection and managing the collection. The simple use case is coming across a gemstone and trying to discover the type of gemstone.

In-Scope and Out-Scope Goals

In-Scope

1. Explore Tensorflow functionality through gemstone classification with image recognition.
2. Attempt to correctly categorize gemstones using tensorflow with Convolutional Neural Networks (CNN)
3. Visualize and document the results.

Out-Scope

1. Create a full application to automatically import images and classify them.
2. Distinguishing between fake and real gemstones.
3. Exploring other models such as RNN and ANN.

Dataset

We will be using a dataset that consists of 3,200+ images of different gemstones and 87 classes. The data is publicly available and is on kaggle (https://www.kaggle.com/lsind18/gemstones-images). The pictures were scraped from minerals.net and rasavgems.com so we assume that the majority of the images are real gemstones. The train data consists of ~2,800 files while the test data consist of ~40 files. The file format for the files are jpg (.jpg) file format. Below are examples of ten random gemstone instances from the dataset. In further sections, we did not experience any problems with the images. In some of the modeling work, we cropped the images so that the model can pick up on the gemstone details.

Random Gemstone Instances (10 Shown)

The following are the classes from the dataset:

```
array(['Aquamarine', 'Almandine', 'Andradite', 'Ametrine', 'Alexandrite',
       'Amethyst', 'Andalusite', 'Amazonite', 'Amber', 'Aventurine Green',
       'Blue Lace Agate', 'Chalcedony', 'Beryl Golden', 'Cats Eye',
       'Benitoite', 'Bixbite', 'Carnelian', 'Bloodstone',
       'Aventurine Yellow', 'Chalcedony Blue', 'Chrome Diopside',
       'Chrysoberyl', 'Chrysocolla', 'Danburite', 'Diaspore', 'Fluorite',
       'Dumortierite', 'Chrysoprase', 'Citrine', 'Emerald', 'Diamond',
       'Coral', 'Kyanite', 'Iolite', 'Grossular', 'Hiddenite', 'Kunzite',
       'Jade', 'Hessonite', 'Goshenite', 'Jasper', 'Garnet Red',
       'Labradorite', 'Lapis Lazuli', 'Onyx Black', 'Onyx Red',
       'Moonstone', 'Opal', 'Morganite', 'Malachite', 'Onyx Green',
       'Larimar', 'Pearl', 'Peridot', 'Prehnite', 'Rhodochrosite',
       'Quartz Rose', 'Quartz Rutilated', 'Quartz Lemon', 'Pyrite',
       'Rhodolite', 'Pyrope', 'Quartz Beer', 'Quartz Smoky', 'Rhodonite',
       'Ruby', 'Serpentine', 'Sapphire Purple', 'Scapolite',
       'Sapphire Blue', 'Spessartite', 'Sapphire Pink', 'Sodalite',
       'Sphene', 'Sapphire Yellow', 'Spinel', 'Spodumene', 'Sunstone',
       'Tanzanite', 'Variscite', 'Tourmaline', 'Topaz', 'Tigers Eye',
       'Zoisite', 'Tsavorite', 'Turquoise', 'Zircon'], dtype='<U17')
```

Gemstone Classes (87 Total)

Software/Hardware Needed

Google Colaboratory (Colab) was used for our work. Google Colab is free and requires no setup. Tensorflow is already pre-installed and optimized so we don't have to worry about installation or pre-requisites (Tensorflow Blog, 2018). For any libraries that we needed to install, we could just !pip install (We did not install any additional libraries). It is also easy to share and work on collaboratively. The CNN requires hardware acceleration. This can be done by going to Runtime, change runtime, and then selecting GPU. Our work was written in Python.

Tensorflow & Keras

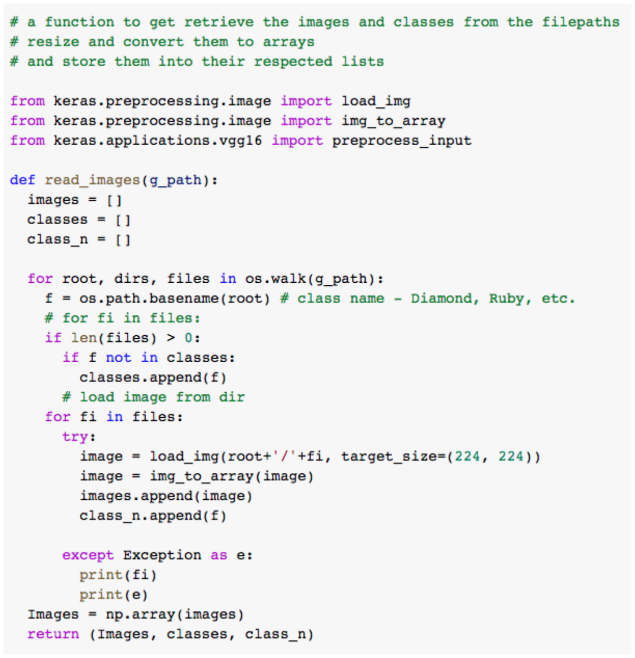
Deep Learning is gaining a lot of popularity due to its supremacy in accuracy when trained with huge amounts of data (Mahapatra, 2018). Tensorflow is a framework created by Google for creating Deep Learning models (Kofler, 2017). It was written in C++ but can be accessed through Python API's (what we used). It can be used to create complex applications with great accuracy. Tensorflow can be used to solve problems related to images, videos, text, or even audio. Tensorflow was also created with processing limitations in mind. They can be run on almost all computers and even some smartphones. A computer with Intel Core I3 and 8 GB of RAM would not have any performance issues running Tensorflow (Mahapatra, 2018). But an important thing to note is that Deep Learning requires high-end computers in the back end to execute the algorithms (Mahapatra, 2018). The main advantage of Tensorflow is that it simplifies things for the developer. It takes care of a lot of the detailed work when it comes to tuning algorithms so that the developer can focus on a more macro view of the application. Keras is a neural networking library and has deep learning models publicly available. In our case, we used the VGG model from Keras to test out for gemstone classification.

Convolutional Neural Networks (CNN)

CNN is a kind of Artificial Neural Networks (ANN), which is a part of Neural Networks (NN). Neural Networks, is a class of Machine Learning algorithms that can be used for Deep Learning and Big Data problems. ANN is a collection of neurons or nodes that pass a signal from one to another. The number of units and the way they are connected is the network architecture. The classes of ANN include Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), and more. To simplify the explanation of CNN, a CNN, has one or more layers of convolution units. The layers work together to create an output. The convolution units reduce the number of units so that it reduces overfitting. With image classification, the images go through a series of convolution layers, pooling, and then classified through softmax (Heo, 2020). Polling layers are used to reduce the dimensionality (parameters) and summarize the region in the convolution layer. The softmax return value(s) (up to 1) about what the image may be (eg. Dog = .05, Cat = .95).

Data Transformation

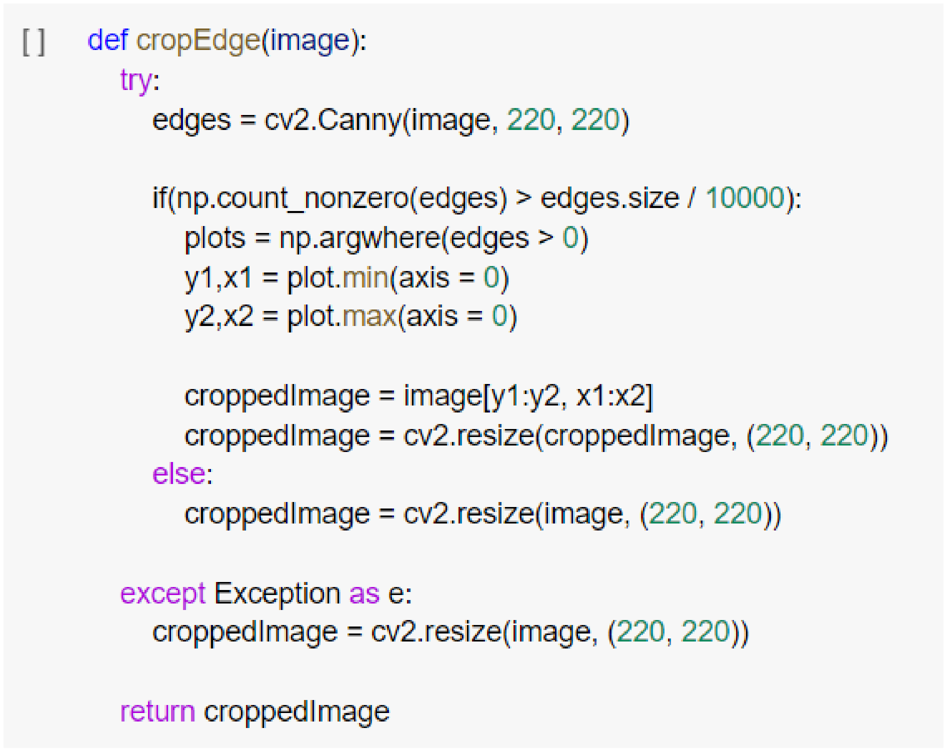
We first downloaded and uploaded the work to a shared folder on our Google Drive. The data can be found here - https://drive.google.com/drive/folders/1Zlhlr-agWRXO2VLOVMQn_Hx9LOs3SCrU?usp=sharing. The structure is Dataset → Archive → Test/Train. The test and train folders contain the folders of the gemstone classes (e.g Alexandrite). Inside the gemstone classes folder are the gemstone images (e.g alexandrite_3.jpg). So we first had to use this arrangement to read the images and get the classes. We create a function, read_images, that deals with creating a list of the images and classes. For each file in the directories, we used Keras' functions (load_img, img_to_array) to load, set the size, and append the images to an array. We also took the classes from the respective folders.



```python
# a function to get retrieve the images and classes from the filepaths
# resize and convert them to arrays
# and store them into their respected lists

from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.applications.vgg16 import preprocess_input

def read_images(g_path):
  images = []
  classes = []
  class_n = []

  for root, dirs, files in os.walk(g_path):
    f = os.path.basename(root) # class name - Diamond, Ruby, etc.
    # for fi in files:
    if len(files) > 0:
      if f not in classes:
        classes.append(f)
      # load image from dir
    for fi in files:
      try:
        image = load_img(root+'/'+fi, target_size=(224, 224))
        image = img_to_array(image)
        images.append(image)
        class_n.append(f)

      except Exception as e:
        print(fi)
        print(e)
  Images = np.array(images)
  return (Images, classes, class_n)
```

Getting the images and classes from the folders

In our own CNN model, we also crop the images so that the model would be able to concentrate more on the gemstone than the background. OpenCV lets us crop the images through looking for the edges and removing the edges. Then we resize the image and returned them for the modeling.



```python
[ ]   def cropEdge(image):
        try:
          edges = cv2.Canny(image, 220, 220)

          if(np.count_nonzero(edges) > edges.size / 10000):
            plots = np.argwhere(edges > 0)
            y1,x1 = plot.min(axis = 0)
            y2,x2 = plot.max(axis = 0)

            croppedImage = image[y1:y2, x1:x2]
            croppedImage = cv2.resize(croppedImage, (220, 220))
          else:
            croppedImage = cv2.resize(image, (220, 220))

        except Exception as e:
          croppedImage = cv2.resize(image, (220, 220))

        return croppedImage
```

Cropping Images

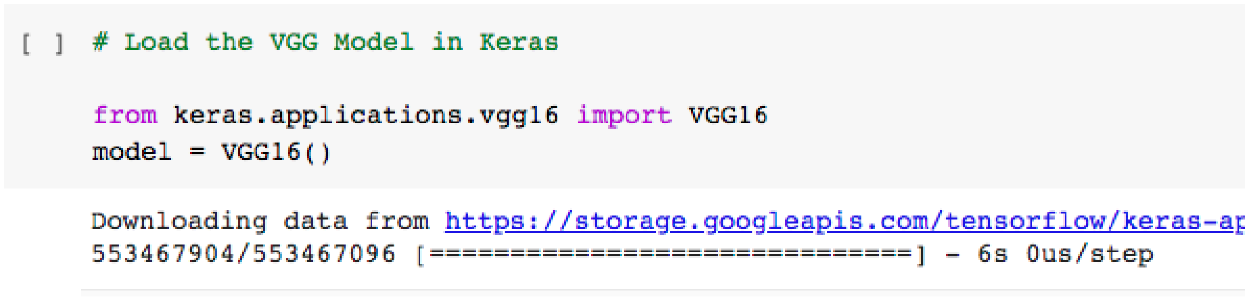
Methods Employed

In our study, we created two Jupyter notebooks to implement a pre-trained VGG16 model as well as a self-built CNN model. VGG16 is a CNN model which uses 13 convolutional layers and 3 fully connected layers, hence the name VGG16. This is a pre-trained model by Keras and was able to produce .901 top 5 accuracy in ImageNet (a dataset of around 140,000,000 images and 1,000 classes). Our own CNN model is a sequential model consisting of 5 convolution blocks with a max pool layer in each one. Activation function relu and softmax.

System Implementation

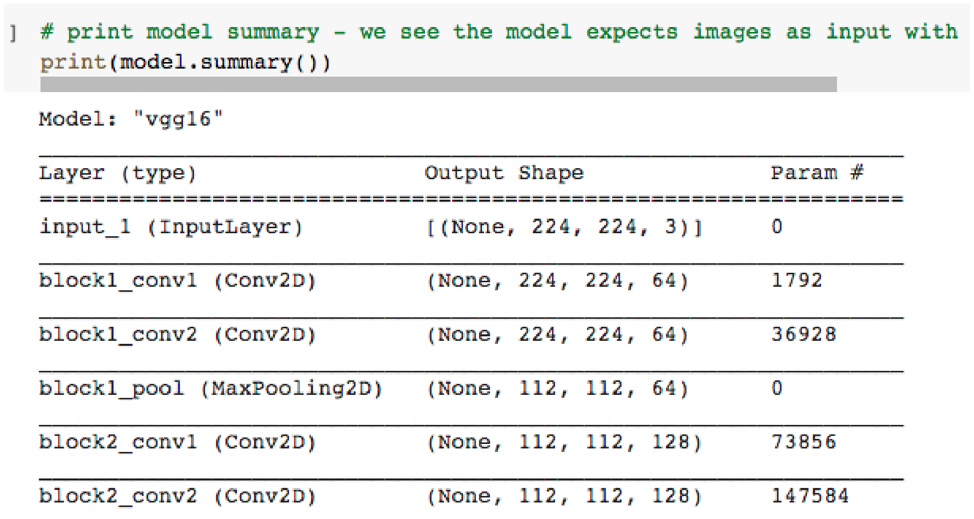
VGG16 Pre-Trained Work

The implementation was more straightforward due to it being pre-trained. Loading the model just involved importing the VGG16 model from keras.applications.vgg16 and calling VGG16().



```
[ ] # Load the VGG Model in Keras

    from keras.applications.vgg16 import VGG16
    model = VGG16()

    Downloading data from https://storage.googleapis.com/tensorflow/keras-ap
    553467904/553467096 [==============================] - 6s 0us/step
```

Next, we can look at the model summary using the summary function. We can see from the summary that the model is expecting the images to be inputted with 224 X 224 pixel images and 3 channels (color). The weights are already pre-determined by the model. Something we also found after the modeling is that there are only 1000 classes in the prediction part of the summary. This will be discussed in the results.



```
] # print model summary - we see the model expects images as input with
  print(model.summary())

  Model: "vgg16"
  _________________________________________________________________
  Layer (type)                Output Shape              Param #
  =================================================================
  input_1 (InputLayer)        [(None, 224, 224, 3)]     0
  _________________________________________________________________
  block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792
  _________________________________________________________________
  block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928
  _________________________________________________________________
  block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0
  _________________________________________________________________
  block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856
  _________________________________________________________________
  block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584
```

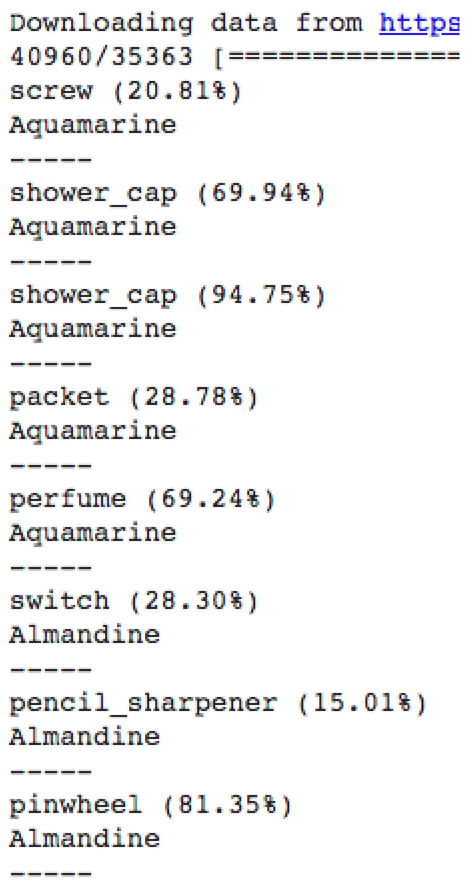
After looking through the summary, we can prepare the input (our gemstone images in the arrays) through the pre_process_input() function. Then we can predict the images through the predict() function. In 1000 classes, it will attempt to predict the images. The % between the predictions indicates how much percentage it belonged to the class (out of 1000 classes).

```
from keras.applications.vgg16 import preprocess_input
# prepare the image for the VGG model
image = preprocess_input(Test_Imgs)
```

```
# predict the probability across all output classes
yhat = model.predict(image)
```

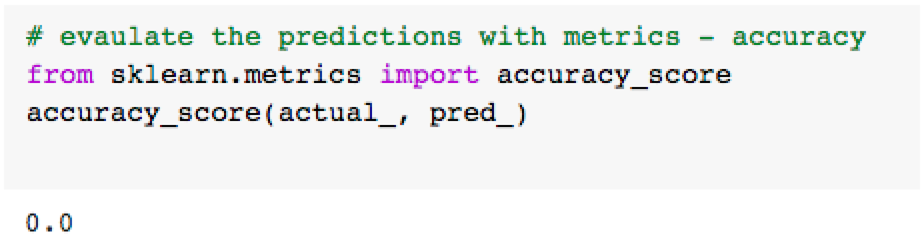
After we had the predictions, we wanted to look at the initial 8 predictions and see what the results were. After looking at the top 8, we weren't too surprised by the results. The first row is the prediction with % matched while the second row is the actual gemstone class.

```
from keras.applications.vgg16 import decode_predictions
from matplotlib.pyplot import figure

# convert the probabilities to class labels
label = decode_predictions(yhat)
# # retrieve the most likely result, e.g. highest probability
new_label = []
for i, l in enumerate(label):
  new_label.append(label[i][0])
# print the classification
figure(figsize=(2, 2), dpi=80)

# print first 8 predictions along with the actual classes
i=0
for l, x in zip(new_label[0:8], Test_act[0:8]):
  print('%s (%.2f%%)' % (l[1], l[2]*100))
  print(x)
  print('-----')
```



```
Downloading data from https
40960/35363 [==============
screw (20.81%)
Aquamarine
-----
shower_cap (69.94%)
Aquamarine
-----
shower_cap (94.75%)
Aquamarine
-----
packet (28.78%)
Aquamarine
-----
perfume (69.24%)
Aquamarine
-----
switch (28.30%)
Almandine
-----
pencil_sharpener (15.01%)
Almandine
-----
pinwheel (81.35%)
Almandine
-----
```

After we took a look at the first eight predictions, we calculated the accuracy score for the prediction. Before we calculated the score, we lowercase and took out special characters in the predictions and gemstone labels. Our accuracy score, using sklearn.metrics, is 0%. This is discussed in the results section of the report.



```
# evaulate the predictions with metrics - accuracy
from sklearn.metrics import accuracy_score
accuracy_score(actual_, pred_)
```

```
0.0
```

<u>CNN Sequential Work</u>
We built our model by defining a series layer parameters, model parameters and model architect based on the VGG16 model. In other words, our model is a similar and mini version of the VGG16 model: 1 Conv2D 32 Pool, 1 Conv2D 64 Pool, 3 Conv2D 128 Pool, 1 FLAT, 1 Drop, 1x Dense 512, 1 Dense Len/Classes, while VGG16 model has all of these layers but with more numbers.

Import Keras:

```
from keras import optimizers
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, AveragePooling2D
from keras.layers import Activation, Dropout, Flatten, Dense
from keras.preprocessing.image import ImageDataGenerator
```

Built Model:

Conv2D: 2D convolution layer that is spatial convolution of images.

MaxPooling2D: Moving window across 2D input spaces.

32: batch size that represents the number of training samples used during gradient descent.

3: kernel size that represents a pixel window 3x3 that will scan across the whole image.

Activation: rectified linear activation function(ReLU), output the input directly if positive. Softmax, outputs a vector containing a list of potential outcomes of probability distributions.

Flatten: function that transforms a multidimensional vector to a single dimension.

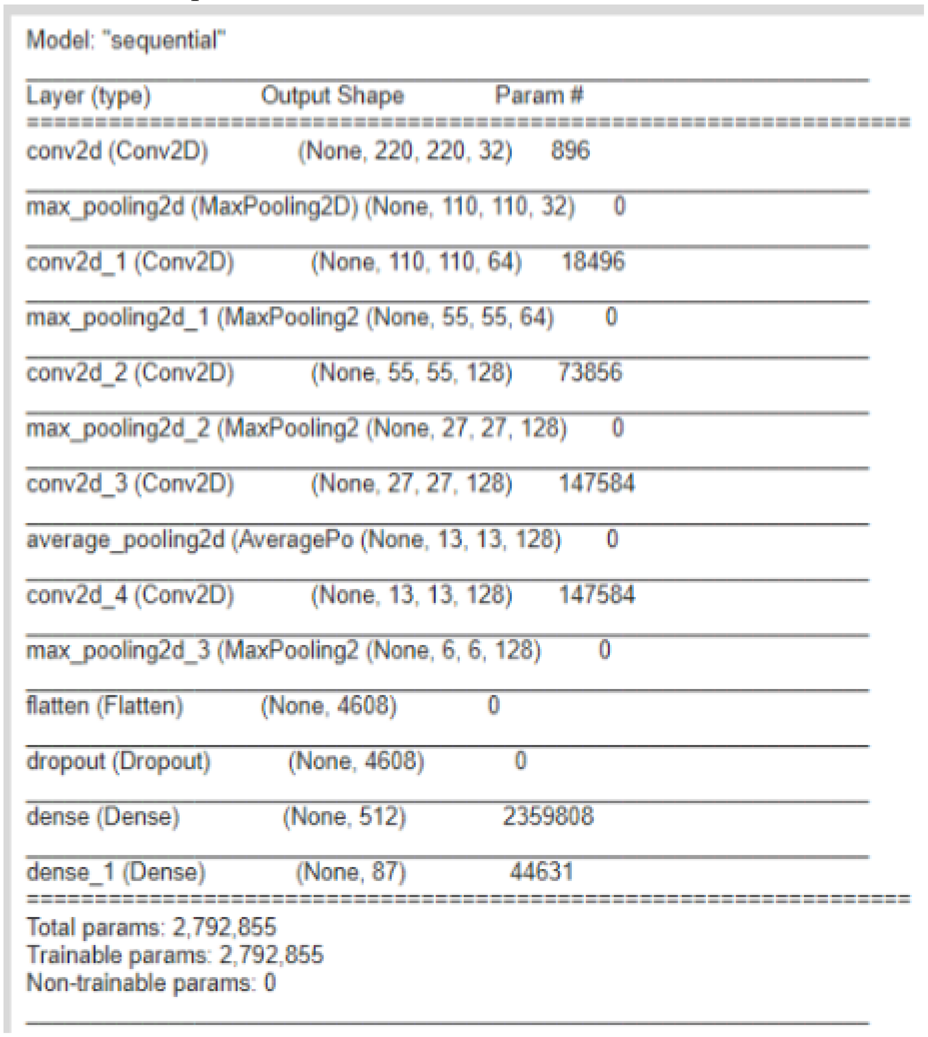
Dropout: a layer that helps reduce overfitting by randomly setting input's fraction to 0.

Dense: basic layer that connects all nodes and feeds all output from the previous layer to neurons.

```
myModel = Sequential()

myModel.add(Conv2D(32, (3, 3), activation = 'relu', padding = 'same', input_shape = (220, 220, 3)))
myModel.add(MaxPooling2D((2, 2)))

myModel.add(Conv2D(2 * 32, (3, 3), activation = 'relu', padding = 'same'))
myModel.add(MaxPooling2D((2, 2)))

myModel.add(Conv2D(4 * 32, (3, 3), activation = 'relu', padding = 'same'))
myModel.add(MaxPooling2D((2, 2)))

myModel.add(Conv2D(4 * 32, (3, 3), activation = 'relu', padding = 'same'))
myModel.add(AveragePooling2D(pool_size = (2, 2), strides = (2, 2)))

myModel.add(Conv2D(4 * 32, (3, 3), activation = 'relu', padding = 'same'))
myModel.add(MaxPooling2D((2, 2)))

myModel.add(Flatten())
myModel.add(Dropout(0.5))
myModel.add(Dense(16 * 32, activation = 'relu'))
myModel.add(Dense(87, activation = 'softmax'))
myModel.summary()
```

Model Summary:

Our model has 2,792,855 parameters, all of them are trainable. The VGG16 model has 138,357,544 trainable parameters.



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 220, 220, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 110, 110, 32)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 110, 110, 64)      18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 55, 55, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 55, 55, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 27, 27, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 27, 27, 128)       147584
_________________________________________________________________
average_pooling2d (AveragePo (None, 13, 13, 128)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 13, 13, 128)       147584
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 6, 6, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 4608)              0
_________________________________________________________________
dropout (Dropout)            (None, 4608)              0
_________________________________________________________________
dense (Dense)                (None, 512)               2359808
_________________________________________________________________
dense_1 (Dense)              (None, 87)                44631
=================================================================
Total params: 2,792,855
Trainable params: 2,792,855
Non-trainable params: 0
_________________________________________________________________
```

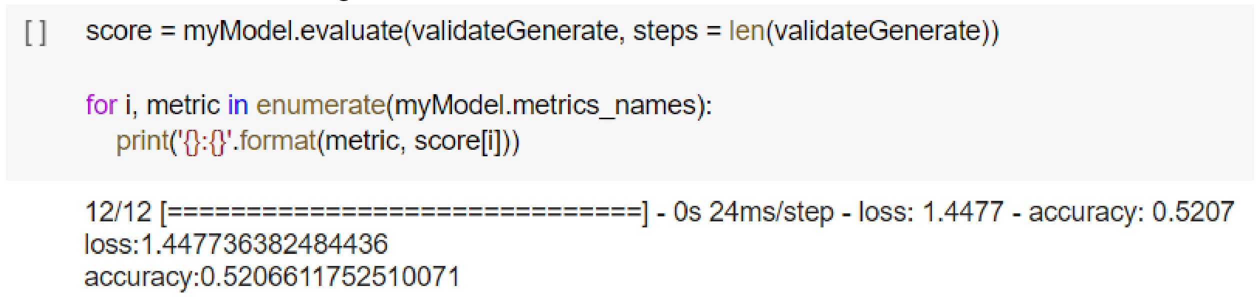
Fit Model:

Epochs: the number of times the algorithm goes through the entire dataset.
Verbose: mode of model output, 0 means silent, 1 is progress bar.

```
myModel.fit(
    trainGenerate,
    steps_per_epoch = len(trainX) // 32,
    epochs = 15,
    validation_data = validateGenerate,
    validation_steps = len(validateX) // 32,
    verbose = 1
)
```

Check result:

The sequential model has its own evaluation function to evaluate the model. The accuracy is the number of right predictions divided by the total number of predictions. All of these calculations are using the test dataset.

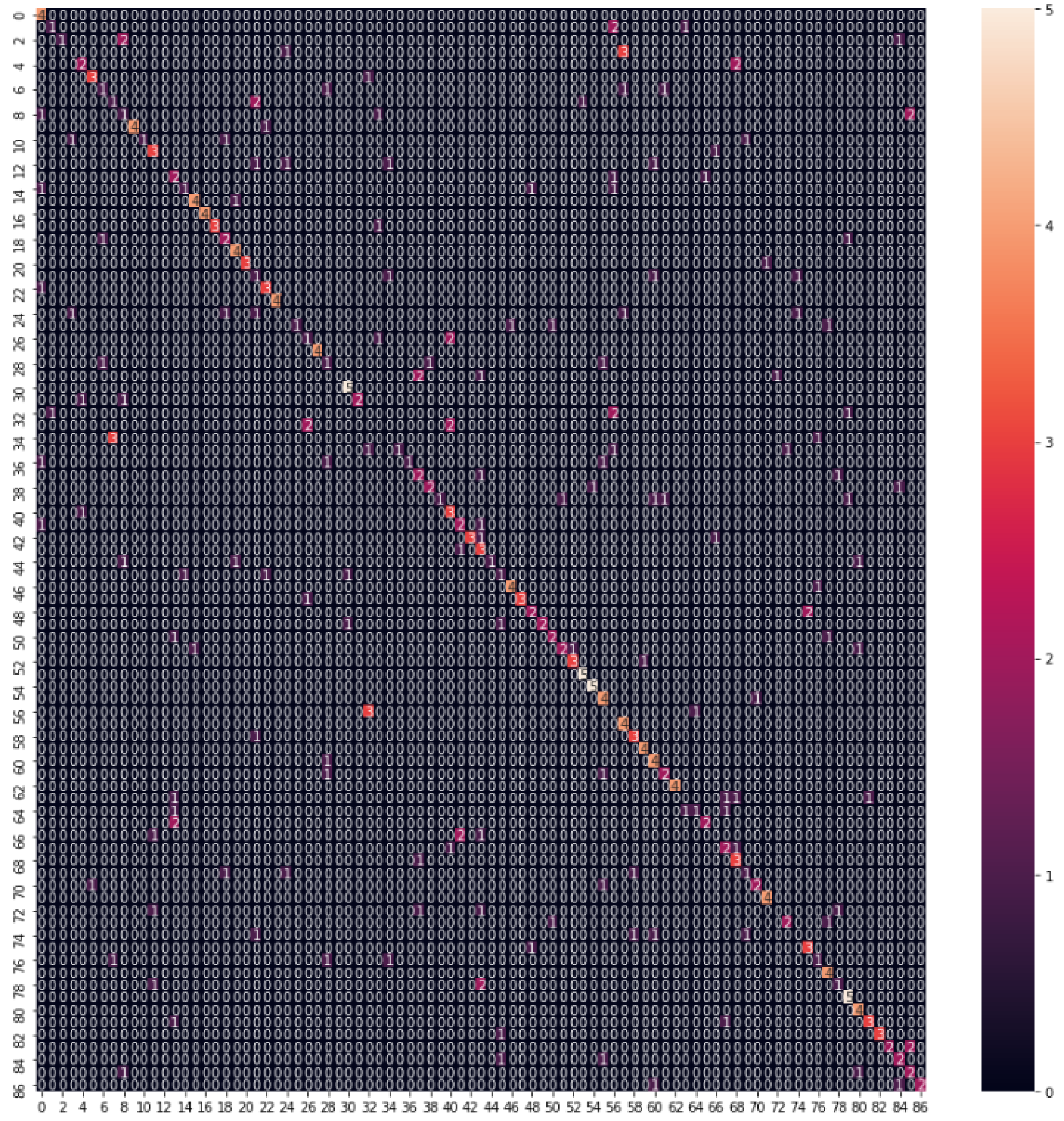


```
[ ]  score = myModel.evaluate(validateGenerate, steps = len(validateGenerate))

     for i, metric in enumerate(myModel.metrics_names):
       print('{}:{}'.format(metric, score[i]))
```

```
12/12 [==============================] - 0s 24ms/step - loss: 1.4477 - accuracy: 0.5207
loss:1.447736382484436
accuracy:0.5206611752510071
```

The F1 score, which is F measure, shows the accuracy of the test. Unlike accuracy_score, it calculates both precision and recall value. Therefore, the F1 score is a good metric to determine the accuracy in a balanced way. The 'macro' method calculates metrics for each label, and finds their unweighted mean. This does not take label imbalance into account.

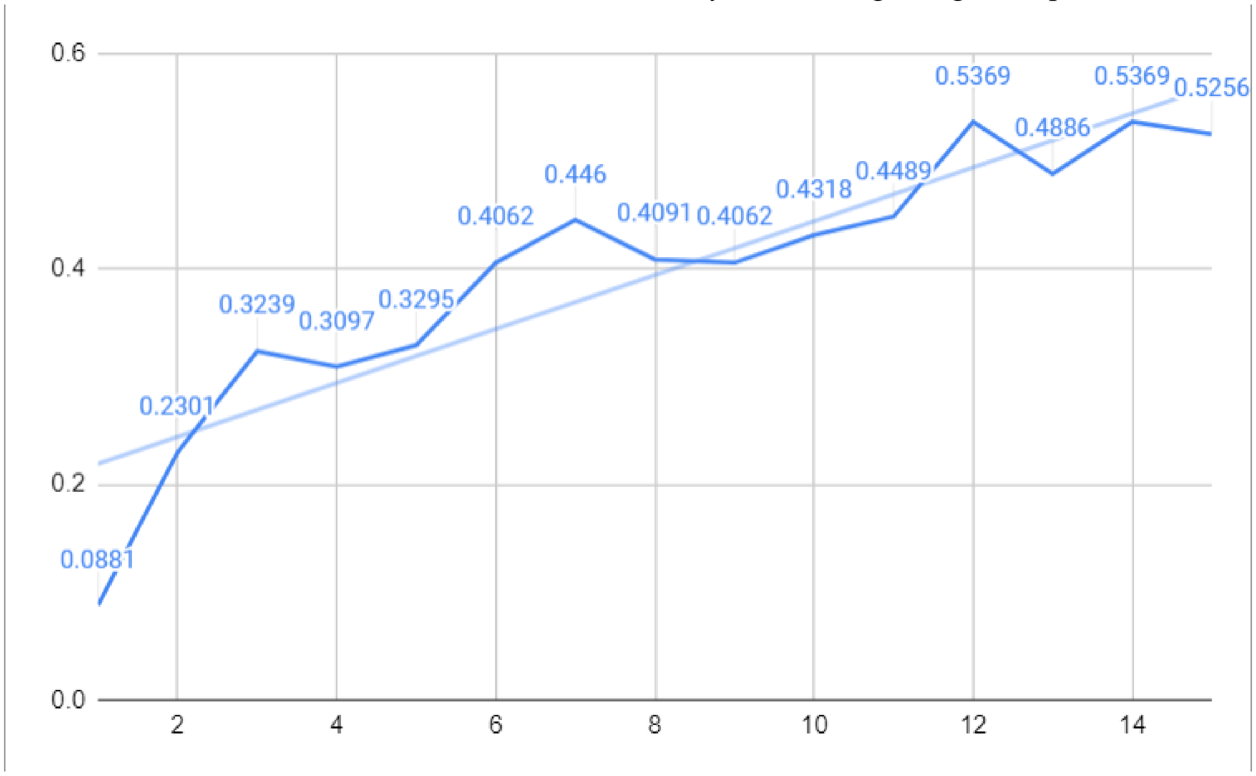
F1 measure: 0.4969



Results and Discussion

As expected, the pre-trained VGG16 model did not perform well. This is likely due to its unfamiliarity with gemstones. Many of the gemstones were classified as common objects such as shower caps, screws and tennis balls. This would likely see much more success with more general objects similar to the data it was trained with. After doing the modeling, we also found that the model only has been trained on 1000 classes. Based on further research, we found that

Our own trained CNN model has better results than the pre-trained VGG16 model. It has accuracy 0.5256 after 15 epochs, based on its trendline, the accuracy will reach above 90% percent after 30 epochs. The explanation of the better performance may be due to the conformity and integrity of the dataset: the model is trained from scratch using only gemstone images, there are no other data classes that interfere with the similarity and training/recognition process.



Limitations and Future Considerations

One thing we didn't take in mind is the classes that VGG16 can predict. We went ahead and wanted to test it out. With more time, we would have looked more into the classes during our research section, in which we spent a lot of time learning tensorflow and CNN. After doing a search around the internet for VGG16 ImageNet classes, we found this resource which tells us about the classes - https://gist.github.com/yrevar/942d3a0ac09ec9e5eb3a. We can see that there are no gemstones here. This list shows common animals, equipment, professions, places, and more. This shows that the VGG16 was trained on common things found everyday. There is the aspect of fine-tuning the model with our gemstone data but we decided to create our own CNN model. This could be something we look at in the future.

Testing (How I can replicate your experiments)

To replicate our project, you can download the dataset from Kaggle or use our shared link (https://drive.google.com/drive/folders/1Zlhlr-agWRXO2VLOVMQn_Hx9LOs3SCrU?usp=sharing). Next, you can create a project folder on Google Drive with the project name. After that you can put the dataset and the iPython notebooks onto the project folder. To run the code in the python files, please change file paths on the files to your file paths. The code should then execute and finish (execution time around ~30-1 hr).

Conclusion

After working on this project, we learned alot about the functionality of the CNN Model for Image classification. To summarize, the images go through a series of convolution layers, pooling, and then classified through softmax. Tensorflow helped reduce a lot of the workload from our end. Google already integrated the installation to Google Colab and for this problem, we didn't have to write a lot of logic that would be needed if we didn't use Tensorflow. Finally, Tensorflow can be used for difficult problems like the one we performed. We obtained 50% from around 18 epochs so we expect it to be better with more epochs.

References

Heo, M. (2020). [TensorFlow 2 Deep Learning] CNN.
https://www.youtube.com/watch?v=A60X_NDte7o

Kofler, M. (2017). Deep Learning with Tensorflow: Part 1 — theory and setup. Towards Data Science. https://towardsdatascience.com/deep-learning-with-tensorflow-part-1-b19ce7803428

Mahapatra, S. (2018). Why Deep Learning over Traditional Machine Learning? Towards Data Science.
https://towardsdatascience.com/why-deep-learning-is-needed-over-traditional-machine-learning-1b6a99177063

Maula, Ismatul, et al. "Development of a Gemstone Type Identification System Based on HSV Space Colour Using an Artificial Neural Network Back Propagation Algorithm." International Conference on Science and Technology (ICOSAT 2017)-Promoting Sustainable Agriculture, Food Security, Energy, and Environment Through Science and Technology for Development. Atlantis Press, 2017.
https://www.researchgate.net/publication/325465942_Development_of_a_Gemstone_Type_Identification_System_Based_on_HSV_Space_Colour_Using_an_Artificial_Neural_Network_Back_Propagation_Algorithm

Pai, A. (2020). CNN vs. RNN vs. ANN – Analyzing 3 Types of Neural Networks in Deep Learning. Analytics Vidhya.
https://www.analyticsvidhya.com/blog/2020/02/cnn-vs-rnn-vs-mlp-analyzing-3-types-of-neural-networks-in-deep-learning/

Tensorflow Blog. (2018). Colab: An easy way to learn and use TensorFlow. https://blog.tensorflow.org/2018/05/colab-easy-way-to-learn-and-use-tensorflow.html