

**APPLY WATSON CAPABILITIES TO DETECT GEOLOGY
ROCK MATERIAM FROM AN IMAGE**

1. INRODUCTION

1.1 Overview

The “Applying Watson Capabilities To Detect Geogology Rock Material From An Image” is a webapp built using deep learning and IBM Watson Capabilities to detect the type of rock material (Blue Calcite, Limestone, Marble, Olivine, Red Crystal) with image as input.

Rocks are a fundamental component of Earth. The automatic identification of rock type in the field would aid geological surveying, education, and automatic mapping. It is a basic part of geological surveying and research, and mineral resources exploration. They contain the raw materials for virtually all modern construction and manufacturing and are thus indispensable to almost all the endeavours of an advanced society. In addition to the direct use of rocks, mining, drilling, and excavating provide the material sources for metals, plastics, and fuels. Natural rock types have a variety of origins and uses. The three major groups of rocks (igneous, sedimentary, and metamorphic) are further divided into sub-types according to various characteristics. It is an important technical skill that must be mastered by students of geoscience.

In this project rock classification, we are going to find the type of rock such as 'Blue Calcite', 'Limestone', 'Marble', 'Olivine' and 'Red Crystal' were we are using CNN model to analyses the type of rock. The objective of the project is to build a web application to detect the type of the rock. The model takes input from the user and compares it with the pre trained model and the rock type is classified and showcased on the UI along with its chemical composition.

1.2 Purpose

Mining companies can use this app to know the type of rock material found in the mines. This comes handy in places where human reach is difficult. At such times this can be integrated to an automated machine so that it can take snaps of the rock then the image is analyzed. Rock type identification is a basic part of geological surveying and research, and mineral resources exploration. Rocks can be identified in a variety of ways, such as visually (by the naked eye or with a magnifying glass), under a microscope, or by chemical analysis. Visual inspection assesses properties such as color, composition, grain size, and structure.

2. LITERATURE SURVEY

2.1 Existing problem

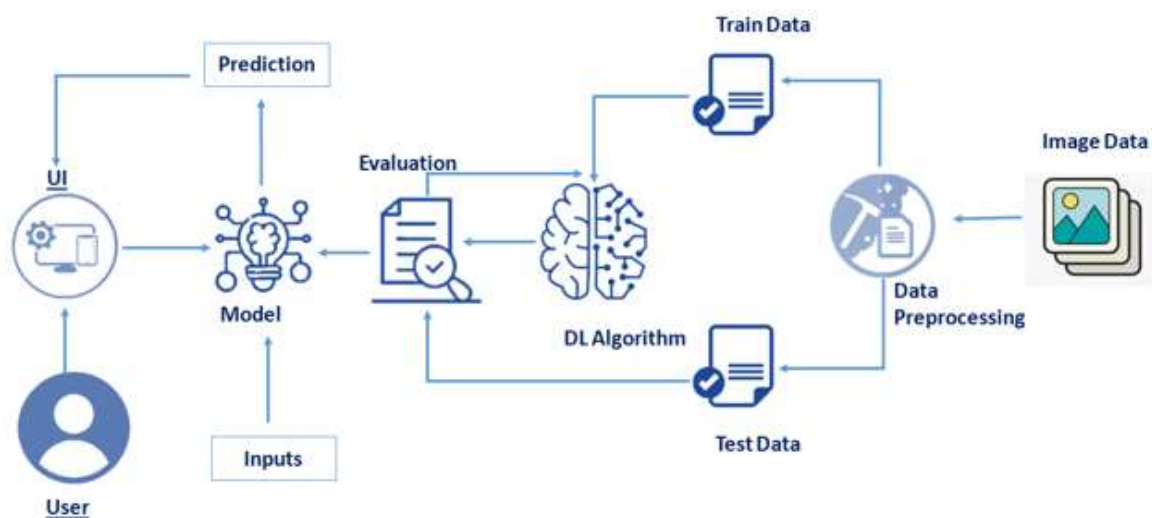
The current system of detecting rock is gathering the sample of the rock from the mine and it is sent to the lab for testing, there it undergoes for testing for few days. Based on the reports from the tests the type of rock and its properties are decided and further decision is taken.

2.2 Proposed problem

An idea made to solve this sort of problem is the "Applying Watson Capabilities To Detect Geology Rock Material From An Image." The webapp takes image as input and based on the features of the input image it decides the type of the rock material it is and its properties. This is possible by training the model with train data that is images with labels showing the features of materials like colour, texture, surface etc.,

3. THEORITICAL ANALYSIS

3.1 BLOCK DIAGRAM



3.2 HARDWARE AND SOFTWARE REQUIREMENTS

3.2.1 SOFTWARE REQUIREMENTS

- **A Device with a constant internet connection.**
- **IBM Cloud**

IBM Cloud provides solutions that enable higher levels of compliance, security, and management, with proven architecture patterns and methods for rapid delivery for running mission-critical workloads

- **IBM Watson**

IBM Watson is AI for business. Watson helps organizations predict future outcomes, automate complex processes, and optimize employees' time.

- **Python**

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. It was created by Guido van Rossum, and first released on February 20, 1991. Its high-level built-in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy-to-learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

- **Anaconda Navigator**

Anaconda Navigator is a free and open-source distribution of the Python and R programming languages for data science and machine learning related applications. It can be installed on Windows, Linux, and macOS. Conda is an open-source, cross-platform, package management system. Anaconda comes with so many nice tools like JupyterLab, Jupyter Notebook, QtConsole, Spyder, Glueviz, Orange, Rstudio, Visual Studio Code. For this project, we will be using Jupyter notebook and Spyder.

- **Jupyter Notebook**

The Jupyter Notebook is an open source web application that you can use to create and share documents that contain live code, equations, visualizations, and text. Jupyter Notebook is maintained by the people at Project Jupyter. Jupyter Notebooks are a spin-off project from the IPython project, which used to have an IPython Notebook project itself. The name, Jupyter, comes from the core supported programming languages that it supports: Julia, Python, and R. Jupyter ships with the IPython kernel, which allows you to write your programs in Python, but there are currently over 100 other kernels that you can also use.

- **Spyder**

Spyder, the Scientific Python Development Environment, is a free integrated development environment (IDE) that is included with Anaconda. It includes editing, interactive testing, debugging, and introspection features.

- **Tensor flow**

TensorFlow is an end-to-end open-source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries, and community resources that lets researchers push the state-of-the-art in ML and developers can easily build and deploy ML-powered applications.

- **Keras**

Keras leverages various optimization techniques to make high-level neural network API easier and more performant. It supports the following features:

Consistent, simple, and extensible API.

- **Flask**

Web framework used for building. It is a web application framework written in python which will be running in local browser with a user interface. In this application, whenever the user interacts with UI and selects emoji, it will suggest the best and top movies of that genre to the user.

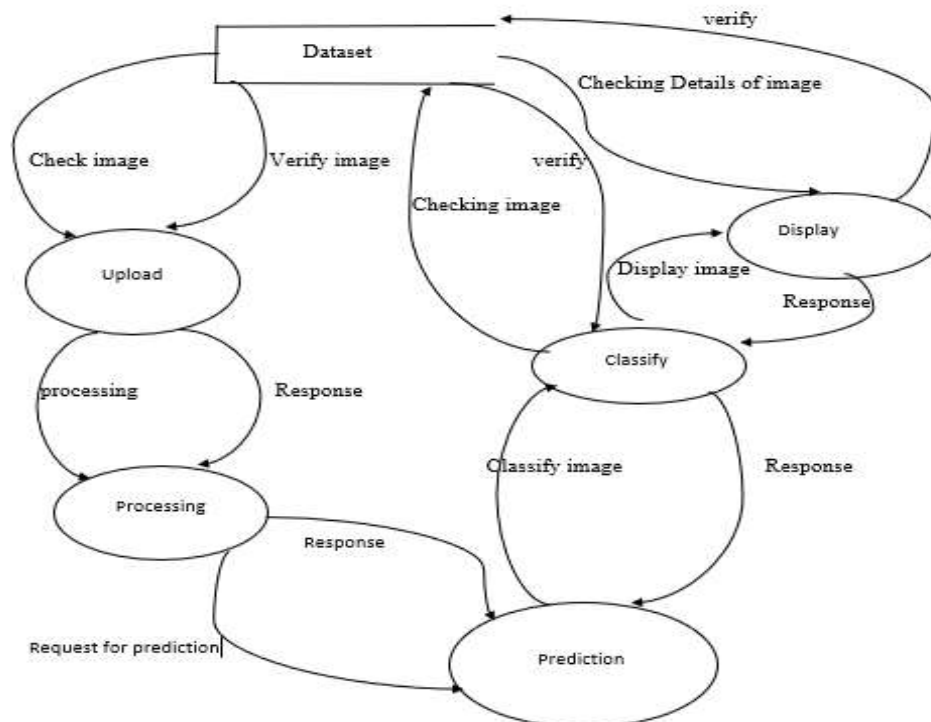
3.2.2 HARDWARE REQUIREMENTS

- o Operating system: window 7 and above with 64bit
- o Processor Type -Intel Core i3-3220
- o RAM: 4Gb and above
- o Hard disk: min 100GB

4. EXPERIMENTAL INVESTIGATIONS

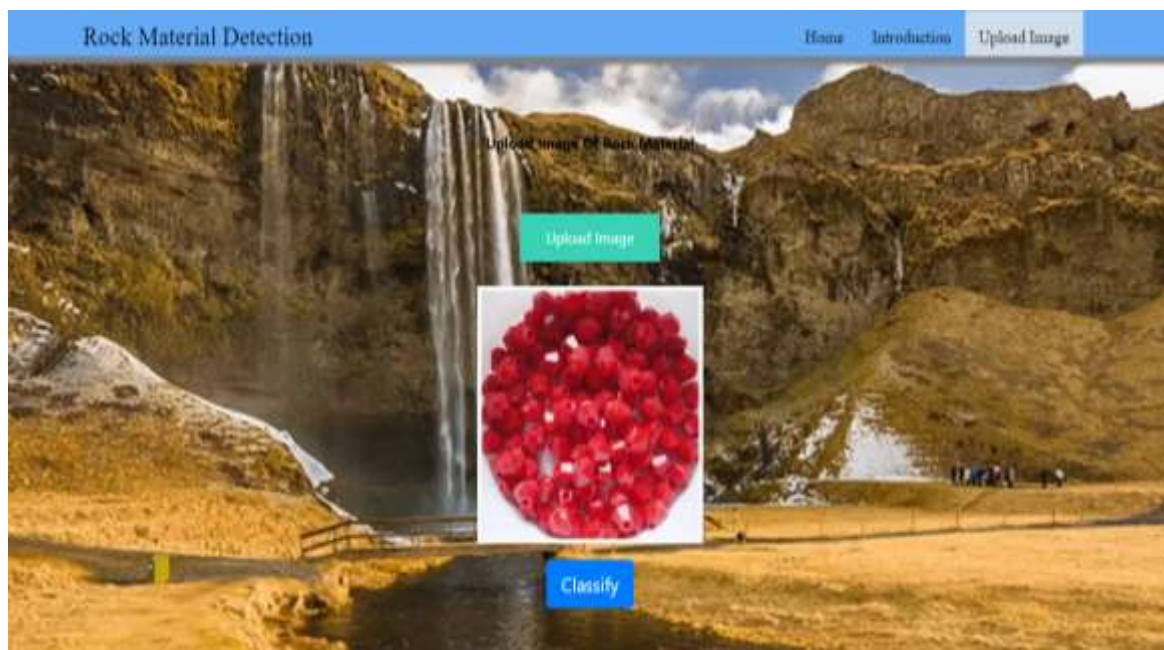
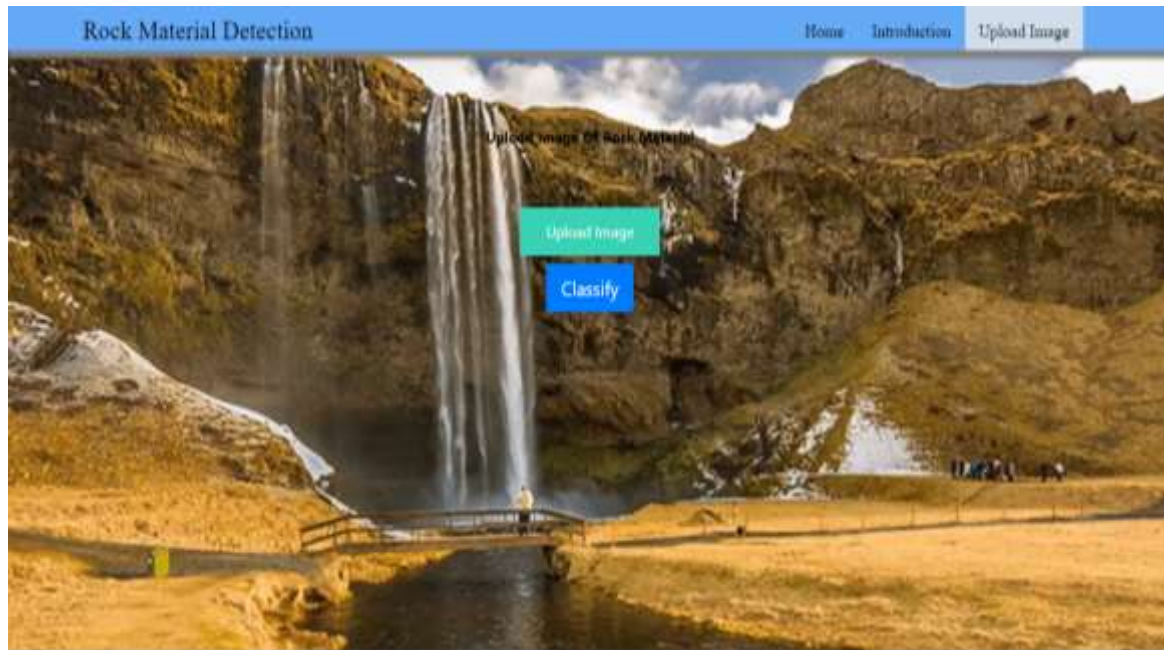
Up on several investigations it is found that miners are happy to use the rock detection using the image rather than traditional lab method as this is time consuming and cost efficient and moreover there is no actions that harm any of the nature, using this is very easy. Also in future the scope of the model can be extended beyond imagination. Based on the above investigation this idea is useful to put into use.

5. FLOWCHART



6. RESULT





Rock Material Detection
[Home](#)
[Introduction](#)
[Upload Image](#)


Rock Classified is
Red Crystal



CRYSTAL NAME:	RED CRYSTAL
COLOR:	Red
LOCATION:	Switzerland
TEXTURE :	Crystal
HARDNESS:	6.5-7,5 Hardness
CHEMICAL COMPOSITION:	Alumina


Rock Material Detection
[Home](#)
[Introduction](#)
[Upload Image](#)

Rock Classified is
Green Crystal

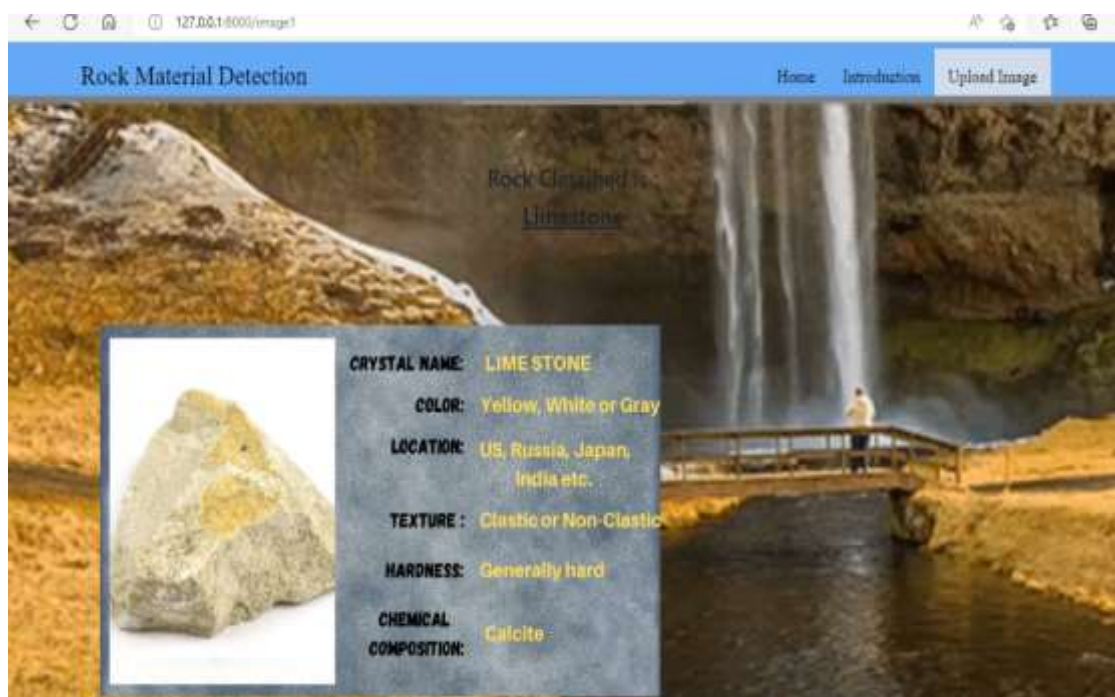
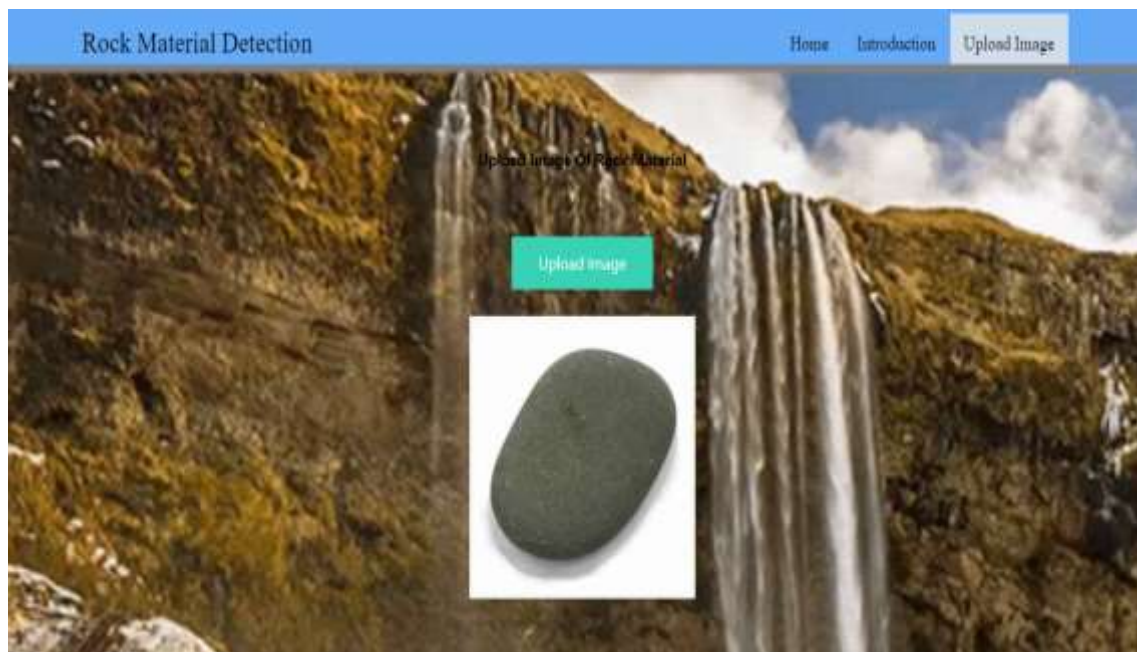


Rock Material Detection
[Home](#)
[Introduction](#)
[Upload Image](#)

Rock Classified is
Olivine



CRYSTAL NAME:	OLIVINE
COLOR:	Green, yellow-green
LOCATION:	Canada
TEXTURE :	Granular
HARDNESS:	6.5 and 7.0 Hardness
CHEMICAL COMPOSITION:	Forsterite, Fayalite



7. ADVANTAGES &DISADVANTAGES

8.1 Advantages

- Easy to use.
- Time efficient.
- Cost efficient.
- Safe to use
- Improve the quality of site investigations by calling for the minimum input data as classification parameters.
- Enable better engineering judgement and more effective communication on a project.

8.2 Disadvantages

- Results may not be perfect.
- New rock cannot be identified.
- Furthur improvement is needed.
- Images should be detailed.
- Some knowledge on technology is needed to use.

8. APPLICATIONS

- Mining
- Rock Collection
- Research
- Used for used for various engineering design and stability analysis of underground structures.
- Rock classification application is a digital educational tool that contributes to the improvement and quality of teaching and learning of rocks and geological contents within natural science subjects in the formal school system.

9. CONCLUSION

The “Applying Watson Capabilities To Detect Geogology Rock Material From An Image” was developed to make detection of rock material from an image without any lab tests. The model takes input from the user and compares it with the pre trained model and the rock type is classified and showcased on the UI along with its chemical composition.

The conclusions can be deduced from the development of the project are Onspot detection ,Easy to use and Does not require any process. The objective of the project is to build a web application to detect the type of the rock. The model takes input from the user and compares it with the pre trained model and the rock type is classified and showcased on the UI along with its chemical composition.

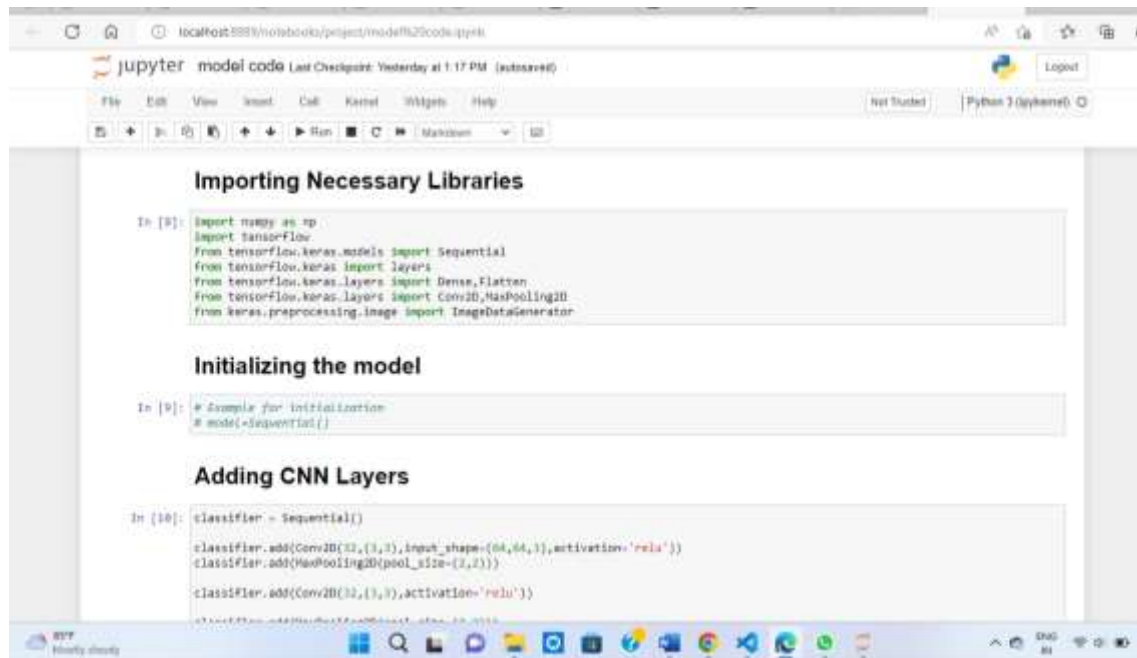
10.FUTURE SCOPE

Many more features can be integrated to give a better user experience to the customer, such as automating the image capture by attaching it to a machine that can go to quarry easily and reach places which are difficult for a man to reach. Also the accuracy of the model can be increased by taking extra inputs.

11. BIBILOGRAPHY

- [https://en.wikipedia.org/wiki/Rock_\(geology\)](https://en.wikipedia.org/wiki/Rock_(geology))
- <https://www.mdpi.com/2227-7390/7/8/755/pdf>
- <https://github.com/SmartPracticeschool/IIIPS-INT-3008-Rockidentification-using-deep-convolution-neural-network>
- https://github.com/amritanjali123/Rock_classification
- https://www.researchgate.net/publication/319590805_Rock_images_classification_by_using_deep_convolution_neural_network

12.APPENDIX



The screenshot shows a Jupyter Notebook interface with the following sections and code:

Importing Necessary Libraries

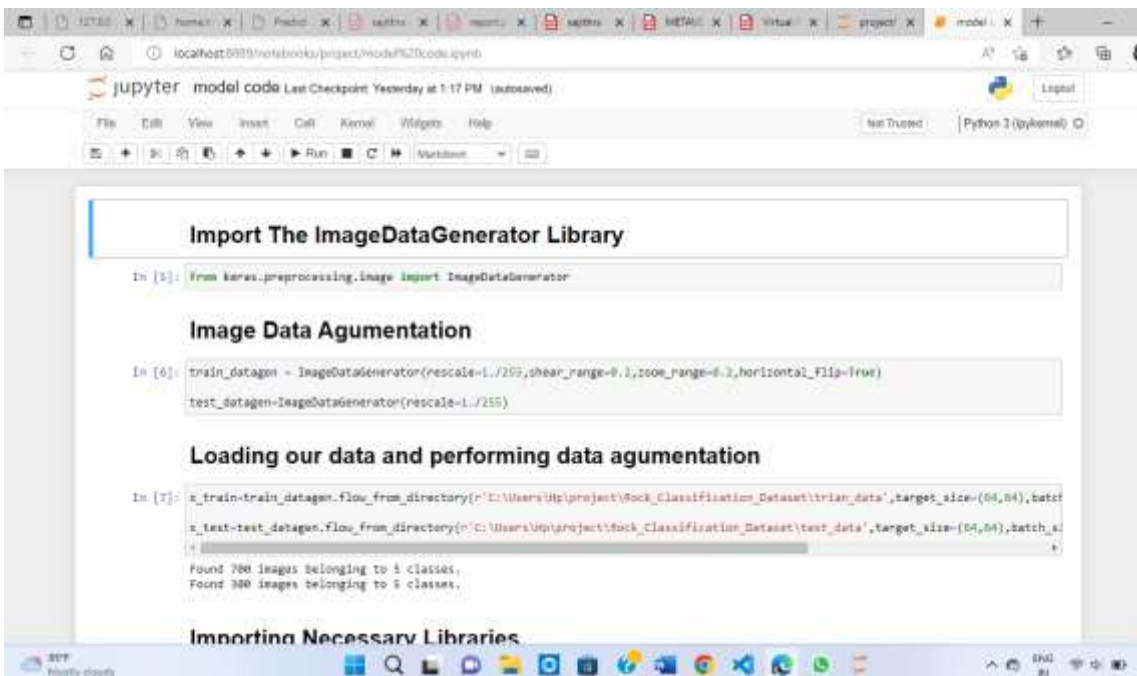
```
In [8]: import numpy as np
import tensorflow
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from keras.preprocessing.image import ImageDataGenerator
```

Initializing the model

```
In [9]: # Example for initialization
# model=Sequential()
```

Adding CNN Layers

```
In [10]: classifier = Sequential()
classifier.add(Conv2D(32,(3,3),input_shape=(64,64,3),activation='relu'))
classifier.add(MaxPooling2D(pool_size=(2,2)))
classifier.add(Conv2D(32,(3,3),activation='relu'))
```



The screenshot shows a Jupyter Notebook interface with the following sections and code:

Import The ImageDataGenerator Library

```
In [3]: from keras.preprocessing.image import ImageDataGenerator
```

Image Data Augmentation

```
In [6]: train_datagen = ImageDataGenerator(rescale=1./255,shear_range=0.2,zoom_range=0.2,horizontal_flip=True)
test_datagen=ImageDataGenerator(rescale=1./255)
```

Loading our data and performing data augmentation

```
In [7]: s_train=train_datagen.flow_from_directory(r'C:\Users\Up\project\Rock_Classification_Dataset\train_data',target_size=(64,64),batch_size=32)
s_test=test_datagen.flow_from_directory(r'C:\Users\Up\project\Rock_Classification_Dataset\test_data',target_size=(64,64),batch_size=32)

Found 760 images belonging to 5 classes.
Found 360 images belonging to 5 classes.
```

Importing Necessary Libraries

classifier.add(Flatten())

Adding Dense Layers

```
In [11]: classifier.add(Dense(units=128,activation='relu'))
classifier.add(Dense(units=5,activation='softmax'))
classifier.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 62, 62, 32)	800
max_pooling2d (MaxPooling2D)	(None, 31, 31, 32)	0
conv2d_1 (Conv2D)	(None, 29, 29, 32)	9248
max_pooling2d_1 (MaxPooling2D)	(None, 14, 14, 32)	0
Flatten (Flatten)	(None, 6272)	0
dense (Dense)	(None, 128)	882044
dense_1 (Dense)	(None, 5)	645

dense_1 (Dense) (None, 5) 645

Total params: 813,733
Trainable params: 813,733
Non-trainable params: 0

Compiling the model

```
In [26]: classifier.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
```

Fitting the model

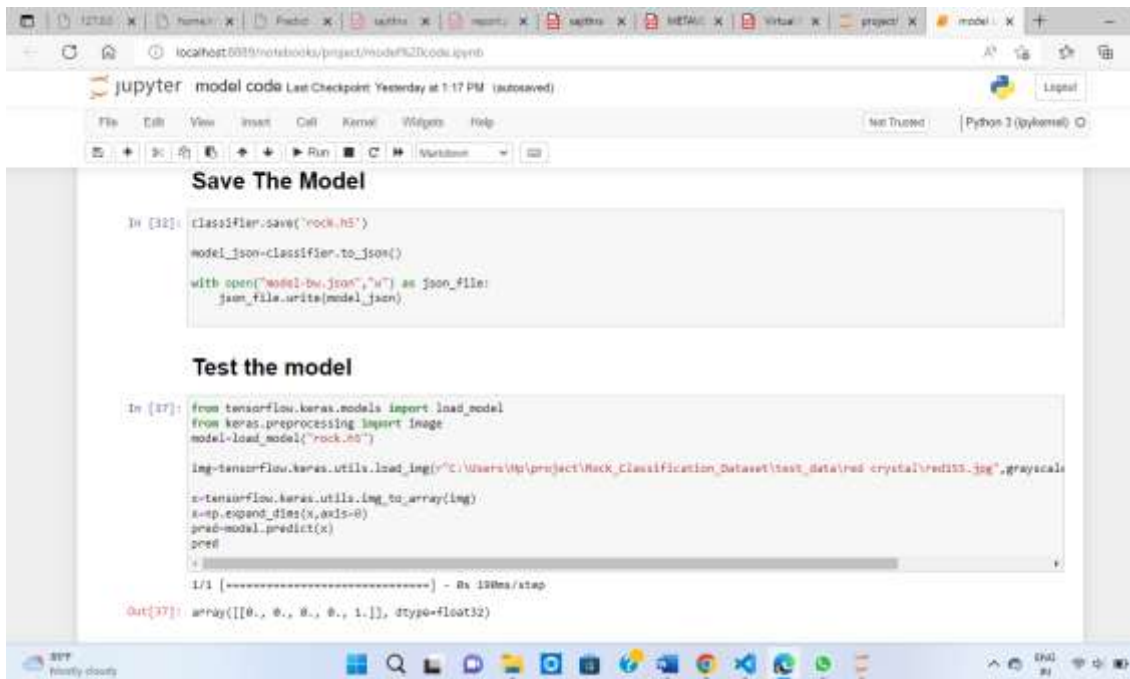
```
In [29]: classifier.fit(
x_train,steps_per_epoch = len(x_train),
epochs=20,validation_data=x_test,validation_steps=len(x_test)
)
```

```
Epoch 1/20
148/148 [=====] - 31s 213ms/step - loss: 0.8972 - accuracy: 0.5586 - val_loss: 0.6827 - val_accuracy: 0.7480
Epoch 2/20
148/148 [=====] - 18s 125ms/step - loss: 0.5922 - accuracy: 0.7457 - val_loss: 0.5639 - val_accuracy: 0.8267
Epoch 3/20
```

```
Epoch 4/20
140/140 [=====] - 19s 135ms/step - loss: 0.4134 - accuracy: 0.8220 - val_loss: 0.5458 - val_accuracy: 0.8367
Epoch 5/20
140/140 [=====] - 15s 105ms/step - loss: 0.4010 - accuracy: 0.8345 - val_loss: 0.7074 - val_accuracy: 0.6733
Epoch 6/20
140/140 [=====] - 14s 108ms/step - loss: 0.4428 - accuracy: 0.8288 - val_loss: 0.8746 - val_accuracy: 0.7233
Epoch 7/20
140/140 [=====] - 23s 161ms/step - loss: 0.3837 - accuracy: 0.8886 - val_loss: 0.8485 - val_accuracy: 0.6667
Epoch 8/20
140/140 [=====] - 18s 127ms/step - loss: 0.3185 - accuracy: 0.8771 - val_loss: 0.4987 - val_accuracy: 0.8167
Epoch 9/20
140/140 [=====] - 14s 102ms/step - loss: 0.3873 - accuracy: 0.8700 - val_loss: 0.4843 - val_accuracy: 0.8633
Epoch 10/20
140/140 [=====] - 12s 83ms/step - loss: 0.3248 - accuracy: 0.8657 - val_loss: 0.5708 - val_accuracy: 0.7833
Epoch 11/20
140/140 [=====] - 12s 87ms/step - loss: 0.2942 - accuracy: 0.8814 - val_loss: 0.4599 - val_accuracy: 0.8667
Epoch 12/20
140/140 [=====] - 14s 102ms/step - loss: 0.2748 - accuracy: 0.8929 - val_loss: 0.4983 - val_accuracy: 0.8867
Epoch 13/20
140/140 [=====] - 13s 91ms/step - loss: 0.2364 - accuracy: 0.9100 - val_loss: 0.4009 - val_accuracy: 0.8767
Epoch 14/20
140/140 [=====] - 12s 89ms/step - loss: 0.2173 - accuracy: 0.9114 - val_loss: 0.4747 - val_accuracy: 0.9033
Epoch 15/20
140/140 [=====] - 12s 87ms/step - loss: 0.2878 - accuracy: 0.8888 - val_loss: 0.3898 - val_accuracy: 0.8909
Epoch 16/20
140/140 [=====] - 13s 89ms/step - loss: 0.2631 - accuracy: 0.8943 - val_loss: 0.3615 - val_accuracy: 0.8933
Epoch 17/20
140/140 [=====] - 14s 99ms/step - loss: 0.2256 - accuracy: 0.9129 - val_loss: 0.6732 - val_accuracy: 0.8000
Epoch 18/20
140/140 [=====] - 13s 93ms/step - loss: 0.2321 - accuracy: 0.9120 - val_loss: 0.4578 - val_accuracy: 0.8933
Epoch 19/20
140/140 [=====] - 12s 89ms/step - loss: 0.2151 - accuracy: 0.9143 - val_loss: 0.7961 - val_accuracy: 0.7433
Epoch 20/20
140/140 [=====] - 13s 93ms/step - loss: 0.3069 - accuracy: 0.8771 - val_loss: 0.6276 - val_accuracy: 0.8733
```

```
140/140 [=====] - 12s 87ms/step - loss: 0.2942 - accuracy: 0.8814 - val_loss: 0.4599 - val_accuracy: 0.8667
Epoch 12/20
140/140 [=====] - 14s 102ms/step - loss: 0.2748 - accuracy: 0.8929 - val_loss: 0.4983 - val_accuracy: 0.8867
Epoch 13/20
140/140 [=====] - 13s 91ms/step - loss: 0.2364 - accuracy: 0.9100 - val_loss: 0.4009 - val_accuracy: 0.8767
Epoch 14/20
140/140 [=====] - 12s 89ms/step - loss: 0.2173 - accuracy: 0.9114 - val_loss: 0.4747 - val_accuracy: 0.9033
Epoch 15/20
140/140 [=====] - 12s 87ms/step - loss: 0.2878 - accuracy: 0.8888 - val_loss: 0.3898 - val_accuracy: 0.8909
Epoch 16/20
140/140 [=====] - 13s 89ms/step - loss: 0.2631 - accuracy: 0.8943 - val_loss: 0.3615 - val_accuracy: 0.8933
Epoch 17/20
140/140 [=====] - 14s 99ms/step - loss: 0.2256 - accuracy: 0.9129 - val_loss: 0.6732 - val_accuracy: 0.8000
Epoch 18/20
140/140 [=====] - 13s 93ms/step - loss: 0.2321 - accuracy: 0.9120 - val_loss: 0.4578 - val_accuracy: 0.8933
Epoch 19/20
140/140 [=====] - 12s 89ms/step - loss: 0.2151 - accuracy: 0.9143 - val_loss: 0.7961 - val_accuracy: 0.7433
Epoch 20/20
140/140 [=====] - 13s 93ms/step - loss: 0.3069 - accuracy: 0.8771 - val_loss: 0.6276 - val_accuracy: 0.8733

Out[20]: <keras.callbacks.History at 0x263b997debd>
```

The screenshot shows a Jupyter Notebook interface with two code cells. The first cell, titled "Save The Model", contains code to save a Keras classifier model as a JSON file. The second cell, titled "Test the model", contains code to load the saved model and test it on a specific image. The output of the second cell shows the model's prediction for the image.

```
In [12]: classifier.save('rock.h5')
model_json=classifier.to_json()
with open("model-bw.json","w") as json_file:
    json_file.write(model_json)

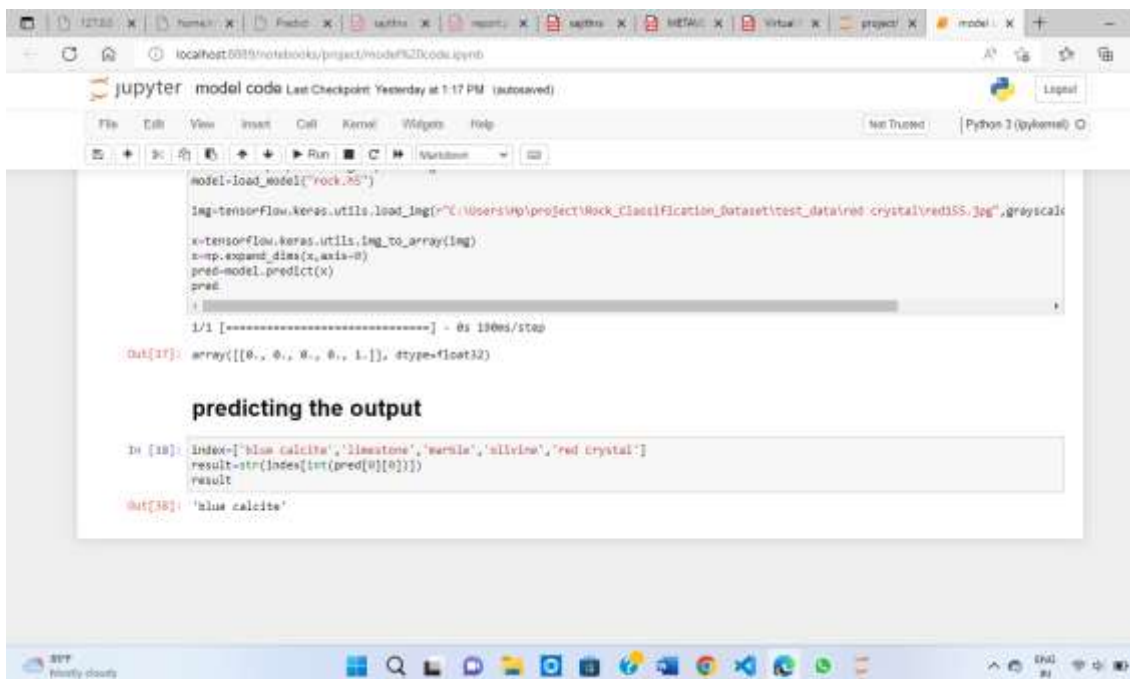
Test the model

In [17]: from tensorflow.keras.models import load_model
from keras.preprocessing import image
model=load_model("rock.h5")

img=tensorflow.keras.utils.load_img(r"C:\Users\Up\project\Rock_Classification_Dataset\test_data\red_crystal\red35.jpg",grayscale=True)
x=tensorflow.keras.utils.img_to_array(img)
x=np.expand_dims(x,axis=0)
pred=model.predict(x)
pred

1/1 [=====] - 8s 130ms/step

Out[17]: array([[0., 0., 0., 0., 1.]], dtype=float32)
```



The screenshot shows a Jupyter Notebook interface with two code cells. The first cell contains code to load the saved model and test it on a specific image. The second cell, titled "predicting the output", contains code to print the predicted class names for the image. The output of the second cell shows the predicted class name for the image.

```
model=load_model("rock.h5")

img=tensorflow.keras.utils.load_img(r"C:\Users\Up\project\Rock_Classification_Dataset\test_data\red_crystal\red35.jpg",grayscale=True)
x=tensorflow.keras.utils.img_to_array(img)
x=np.expand_dims(x,axis=0)
pred=model.predict(x)
pred

1/1 [=====] - 8s 130ms/step

Out[17]: array([[0., 0., 0., 0., 1.]], dtype=float32)

predicting the output

In [18]: index=['blue calcite','limestone','marble','slivine','red crystal']
result=str(index[int(pred[0][0])])
result

Out[18]: 'blue calcite'
```