Department of Mining Engineering
Indian Institute of Technology (BHU)

TITLE:

Raw Materials or Minerals Image Processing

An Interim Report of the UG Project in FULFILLMENT

OF THE REQUIREMENTS FOR THE DEGREE OF

Bachelor Of Technology In

Mining Engineering.

By -

Kumar Shivam (19155044)

Department of Mining Engineering

Indian Institute of Technology BHU, Varanasi (221005)

SEMESTER: 6TH AND SESSION (2021-2022)

CONTENTS

Department of Mining Engineering
Indian Institute of Technology (BHU)

Certificate

This is to certify that the thesis entitled "Raw materials or Minerals Image Processing" submitted by Kumar Shivam, Roll No. 19155044, in partial fulfillment of the requirements for the award of a Bachelor of Technology degree in Mining Engineering at the Indian Institute of Technology (BHU) Varanasi is an authentic work carried out by him under my supervision and guidance. To the best of my knowledge, the matter embodied in the thesis has not been submitted to any other University/Institute for the award of any Degree or Diploma.

DATE: 06/05/2022

Signature of Sir

Indian Institute of Institute BHU Varanasi

ACKNOWLEDGEMENT

It will be simple to name all those people who helped me to get this thesis done. However, it will be tough to thank them enough.

First and foremost, I express my sincere gratitude and indebtedness to Mr. R.P.Singh for allowing me to carry on the present topic, "Raw materials or Minerals Image Processing," and later on for his inspiring guidance, constructive criticism, and valuable suggestions throughout this project work. I am very much thankful to him for their able guidance and pain taking effort in improving my understanding of this project.

I would also like to give special thanks to our Institute- IIT BHU, Varanasi, and the Department of Mining Engineering for providing all the necessary facilities during the project.

I would be failing in my duties if I don't mention the reference and inspiration from the works of others whose details are mentioned in the reference section. I acknowledge my indebtedness to all of them.

At last, my sincere thanks to all my friends who have patiently extended all sorts of help in accomplishing this assignment.

Kumar Shivam

Date: 06/05/2022

ABSTRACT

Recently, Image processing (IP) and Machine learning (ML) algorithms have been successfully used in a wide variety of industry sectors. The raw materials processing methods play an important role in the excavation industry. The area is nowadays a target for novel, innovative technology applications. One of them is image processing as an instrument for the determination of different material properties. The paper proposes a tool – database system – dedicated to supporting image processing and computer vision methods application in the mining industry. The idea and implementation of the system is presented. The database was supplied with a set of coal and mineral images. The images were taken with a variety of sites and google images. The usefulness of the system was tested by the application of a pipeline of image processing algorithms. The goal was to estimate the grain size composition of coal and rock matter presented in the images. With the help of Machine Learning, we convert the images and search for edge detection, then after using stacked LSTM models we predict the classification and type of minerals.

Furthermore, we propose a real-time application for image analysis in the context of mineral processing. Finally, we propose some future directions, including suggestions on mining data collection and developing of an exploration tool based on IP and ML algorithms by taking advantage of cloud computing technology to analyze geoscience data in order to identify target areas with high prospectively potential. The experiments proved the usefulness of a developed database system dedicated to image processing.

Introduction of the Topic

Image processing and analysis is a discipline in which application in the mining industry becomes more and more important. The application of image processing in Run of Mine (ROM) and aggregate analysis has been growing since the 1990s. The image processing was used for grain size distribution determination, ore classification, coal gangue identification, etc. Despite the machine vision method used, the effective and efficient way of information storage and retrieval is necessary. There is also a need for the preparation of a rock grains image database, which can be used for the development and testing of dedicated computer vision methods. The twenty-first century is bringing radical changes in the mineral industry.

High speed and low-cost computational and sensing devices make it possible to reach remote locations and instantaneously control the processes. Image processing which was very useful for some selected applications, is gaining popularity in the mineral industry in a changed scenario. The AI techniques (both image processing and machine learning algorithms) valorize the different types of mining data previously considered unusable to improve the efficiency and performance and the redefinition of mining professions at the global scale. Finally, the use of these advanced technological elements found in AI to optimize the main and critical mining operations would immensely impact the industry and help it grow exponentially. In this project, The system was implemented, and a feed with minerals samples images and some coal images was given to the machine, which then distinguishes between minerals.

Mineral Processing

The majority of mineral exploration efforts created by the mining industry are failures. Only 1% of greenfield projects and 5% of brownfield projects yielded any mineral discoveries. Published figures of the proportion of exploration targets that ultimately become beneficial mines range from 1 in 1,000 (0.01%) to 1 in 3,333 (0.03%). Mineral exploration companies accept high levels of risk. A. General Formulation Mineral Prospectivity Mapping uses modern image processing and machine learning algorithms to uncover common and specific patterns in the database of the desired mineralization style. Once this is accomplished, the trained model is then applied to estimate the likelihood of mineralization in other comparable exploration environments.

Mathematically, the Mineral Prospectivity Mapping may be described as follows:

Train Xi / i N = 1,...,N suite of geoscientific data maps.

y^train: known mineral occurrence locations.

Train an ML model (theta, X^train) by tuningθ to fit the data X ^train to the labels Trainy according to the cost function L.

Then, the mineral potential can be estimated in new areas using available exploration data.

Material and methods

General assumptions and ideas The computer vision methods application in any discipline is involved processing a certain number of images. The concept of application of database technology to support management given in suggests the following approach to the problem:

- Database systems are targeted in processing large amounts of data, therefore, they are a natural choice as a platform for bulk machine vision applications.
- The processing effort should be moved from client computers to database servers, thus improving mobility and lowering requirements for end user's equipment.
- Machine vision algorithms should be implemented in User Defined Functions, which repository should be stored at the database server.
- It should be possible to write images formed as a result of the image processing method application back to the database.
- The results of the machine vision application should be stored as a set of parameters in the database, forming the knowledge base.
- All data stored in the database system should be available for end-users and further processing.

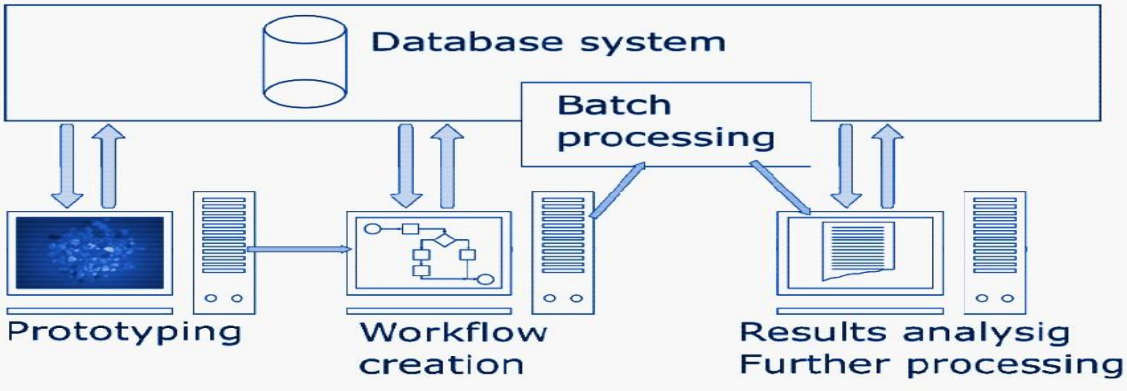
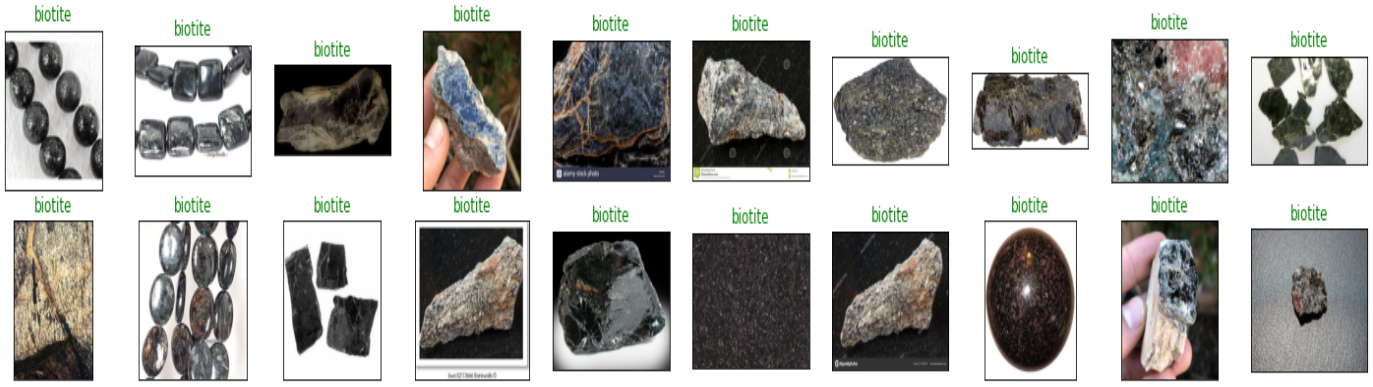


Figure 1. Schema of database usage for images processing and analysis.

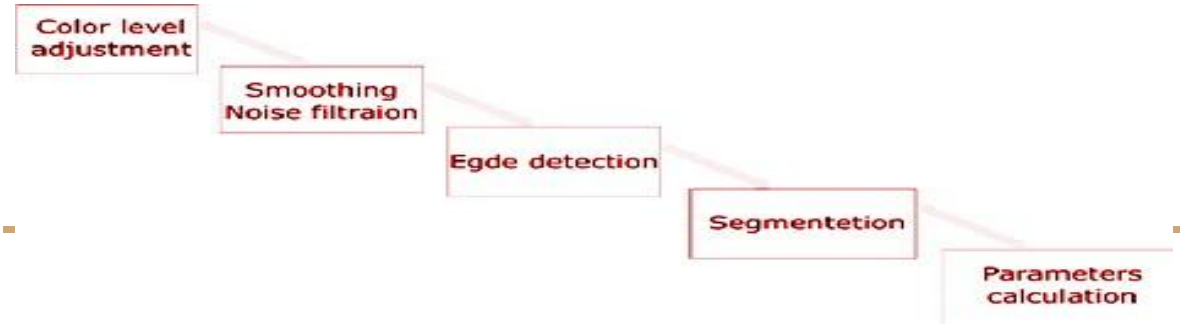
1. Dataset

During several images acquisition sessions, over 500 images were taken and stored in the database system. The photographed samples consisted of enriched and sieved minerals, coal, and barren rock with mixed granularity and coal originating wastes. There were also a few pictures of crushed Run-Of-Mine (ROM). Because of imaging scene character, the automatic exposure parameters selection could not be directly used. For each sample, basic information about its origin, as well as a description, was stored. For the majority of samples laboratory, fuel analysis results are also available. A typical photograph of different mineral samples.



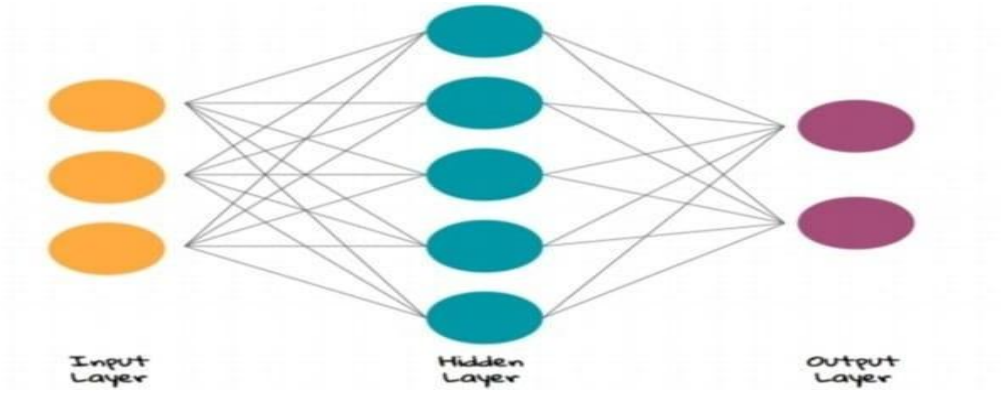
2. Image Processing Test

The overall procedure used was schematically presented in the figure below. The selected images stored in the database were cropped to ensure no other objects than coal or rocks visible. The resulting pictures had the sizes of 3872x2096. Then the images were transformed to HSV color space and only the V (Value) channel, representing image brightness was taken for further processing.

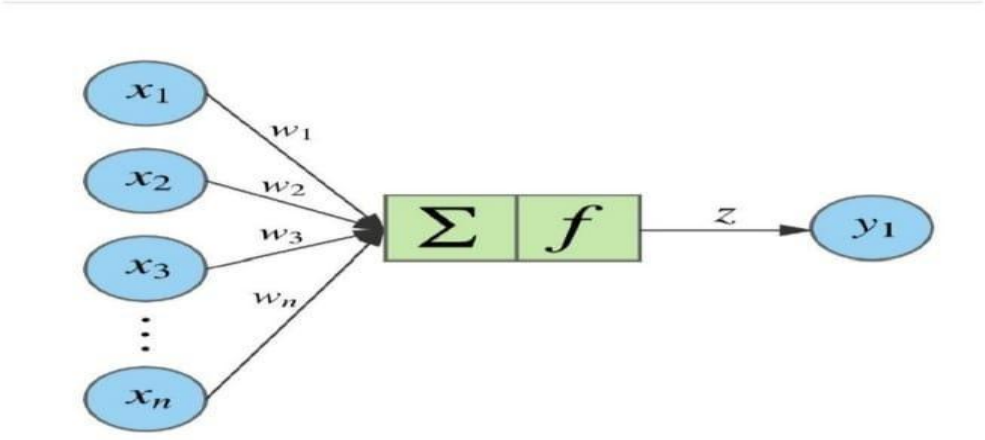


3. ARTIFICIAL NEURAL NETWORK (ANN)

Artificial Neural Networks (ANN) are multi-layer fully-connected neural nets that look like the figure below. They consist of an input layer, multiple hidden layers, and an output layer. Every node in one layer is connected to every other node in the next layer. We make the network deeper by increasing the number of hidden layers.



If we zoom in to one of the hidden or output nodes, what we will encounter is the figure below.



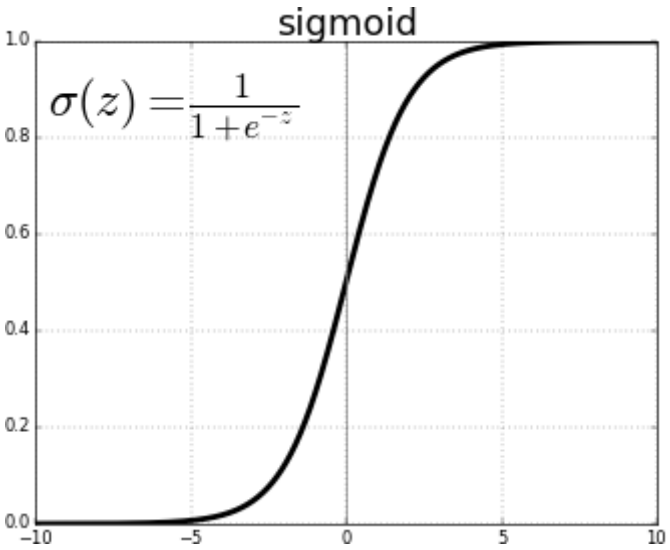
A given node takes the weighted sum of its inputs and passes it through a non-linear activation function. This is the node's output, which then becomes the input of another node in the next layer. The signal flows from left to right, and the final output is calculated by performing this procedure for all the nodes. Training this deep neural network means learning the weights associated with all the edges (W1, W2, W3…, Wn). The equation for a given node looks as follows. The weighted sum of its inputs passed through a non-linear activation function. It can be represented as a vector dot product, where n is the number of inputs for the node.

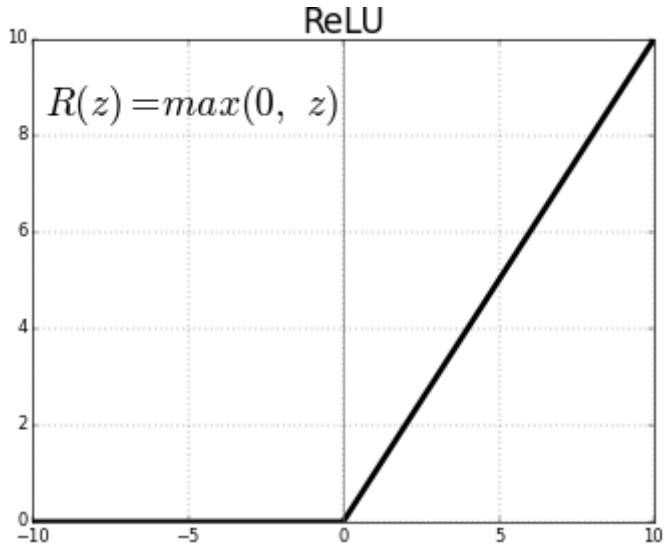
$$z = f(b + x \cdot w) = f\left(b + \sum_{i=1}^{n} x_i w_i\right)$$

$$x \in d_{1\times n},\ w \in d_{n\times 1},\ b \in d_{1\times 1},\ z \in d_{1\times 1}$$

There are many popular activation functions.

Some are mentioned below:

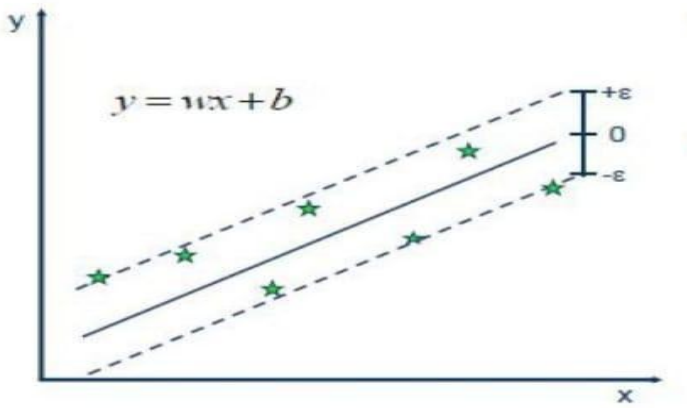




4. SUPPORT VECTOR MACHINES (SVM)

"Support Vector Machine" (SVM) is a supervised machine learning algorithm that can be used for both classification or regression challenges. However, it is mostly used in classification problems. In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is a number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiates the two classes very well.

However, the main idea is always the same: to minimize error, individualizing the hyperplane maximizes the margin, keeping in mind that part of the error is tolerated. The objective function of SVR is to minimize the coefficients — more specifically, the 2-norm of the coefficient vector. The error term is instead handled in the constraints, where we set the absolute error less than or equal to a specified margin, called the maximum error, ϵ (epsilon). We can tune epsilon to gain the desired accuracy of our model.



- Solution:

$$\min \frac{1}{2}\|w\|^2$$

- Constraints:

$$y_i - wx_i - b \le \varepsilon$$

$$wx_i + b - y_i \le \varepsilon$$

There are many hyperparameters of SVM/SVR; some important hyperparameters are explained below:

C (Regularization term): The C parameter tells the SVM optimization how much you want to avoid misclassifying each training example. For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly. Conversely, a very small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassified more points.

Kernel: A kernel is a way of computing the dot product of two vectors xx and yy in some (possibly very high dimensional) feature space. A very simple and intuitive way of thinking about kernels (at least for SVMs) is a similarity function. Given two objects, the kernel outputs some similarity score. The objects can be anything starting from two integers, two real-valued vectors, trees, whatever provided that the kernel function knows how to compare them. There are many types of kernels e.g. linear kernel, Polynomial kernel, RBF kernel, Gaussian Kernel, etc... They all use different methods to find similarity between 2 objects.

Epsilon: Epsilon (ϵ) is the maximum absolute error permitted when calculating the constraints. We can tune epsilon to gain the desired accuracy of our model. (A smaller epsilon generally leads to a tighter constraint and thus regularizes the model more).

Snap Of the Python CODE

For Acutal Code:
*I am adding my **GitHub** link below*
https://github.com/shivam5600/Raw-Materials-or-Minerals-Image-Processing

Snaps:

Load Library

In [6]:

```
import torch
from torchvision.datasets import ImageFolder
import torch.nn.functional as F
from torchvision import transforms
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import random_split
from torchvision.utils import make_grid
from torch.utils.data import DataLoader, SubsetRandomSampler

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import time
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
from collections import Counter
```

Load Data and Get Insight

```
root_folder = '../input/minerals-identification-dataset/minet'
target_label = ['biotite', 'bornite', 'chrysocolla', 'malachite',
                'muscovite', 'pyrite', 'quartz']
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```
#load data and tranform it into tensor
dataset = ImageFolder(root_folder, transform=transforms.ToTensor())
print('Data size: ',len(dataset))
dataset.classes
```

```
Data size:  956
```

```
['biotite',
 'bornite',
 'chrysocolla',
 'malachite',
 'muscovite',
 'pyrite',
 'quartz']
```

```
#insert count into dataframe
df = pd.DataFrame(count, index=np.arange(1))
df = df.transpose().reset_index()
df.columns = ['Mineral', 'count']
df
```

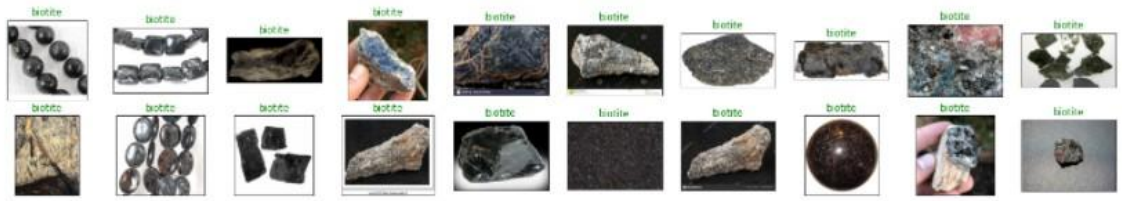
Out[10]:

	Mineral	count
0	biotite	68
1	bornite	170
2	chrysocolla	163
3	malachite	235
4	muscovite	77
5	pyrite	98
6	quartz	145

In [11]:

```
#plot barplot for the sake of easy to read
sns.barplot(df['Mineral'], df['count'])
plt.title('Dataset for each label');
plt.xticks(rotation=30)
plt.grid(axis='y')
```

```
for i in range(20):
    image, label = dataset[i]
    ax = fig.add_subplot(2, 10, i+1, xticks=[], yticks = [])
    ax.imshow(image.permute(1,2,0))
    ax.set_title(target_label[label], color='green')
```



In [10]:

```
#count number for each label
count = {}

for i in range(len(dataset)):
    _, labels = dataset[i]
    label = target_label[labels]
    if label not in count:
        count[label] = 1
    elif label in count:
        count[label] += 1

#insert count into dataframe
df = pd.DataFrame(count, index=np.arange(1))
df = df.transpose().reset_index()
df.columns = ['Mineral', 'count']
```

```
def plot_dist(indexes, dataset=dataset):
    #dist = {}
    count = Counter()
    for i in indexes:
        _, label = dataset[i]
        count[target_label[label]] += 1

    # for i in indexes:
    #     img, labels = dataset[i]
    #     label = target_label[labels]
    #     if label not in dist:
    #         dist[label] = 1
    #     elif label in dist:
    #         dist[label] +=1

    dist_2 = dict(sorted(count.items(), key=lambda kv: kv[1], reverse=True))
    plt.bar(dist_2.keys(), dist_2.values())
    plt.xticks(rotation=30)
    plt.title('Data distribution'); plt.ylabel('count')
    plt.show()
```

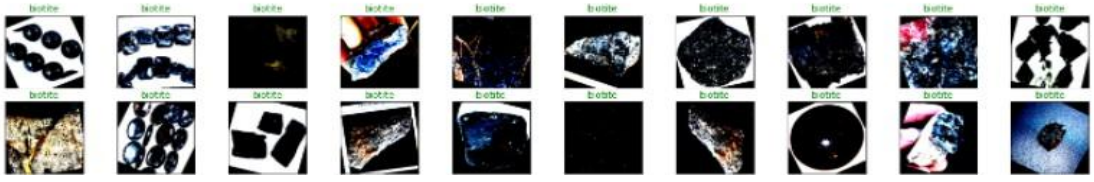
In [16]:

```
# split data to get the same distributiom


#get index and its label
idx_label = {}
for i in range(len(dataset)):
    _, label = dataset[i]
    idx_label[i] = label
```

Load All Data

In [13]:

```
#transform format to augmetation dataset beacuse our dataset only has 956 images
#I reload the data and do multiple transformation and resize it
data_transform = transforms.Compose([transforms.Resize((224, 224)),
                                     transforms.RandomRotation(30),
                                     transforms.RandomVerticalFlip(p=0.5),
                                     transforms.RandomHorizontalFlip(p=0.5),
                                     transforms.ToTensor(),
                                     transforms.Normalize(mean = [0.485, 0.456, 0.406],std = [0.
229, 0.224, 0.225])]) #imagenet mean and std

load_data = ImageFolder(root_folder, transform=data_transform)

#check the images result from transformation
fig = plt.figure(figsize=(25, 4))
for i in range(20):
    image, label = load_data[i]
    ax = fig.add_subplot(2, 10, i+1, xticks=[], yticks = [])
    ax.imshow(image.permute(1,2,0))
    ax.set_title(target_label[label], color='green')
```



Mineral Classification

Notebook Data Logs Comments (0)

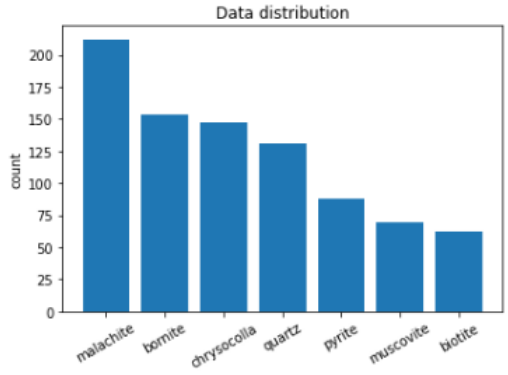
In [16]:

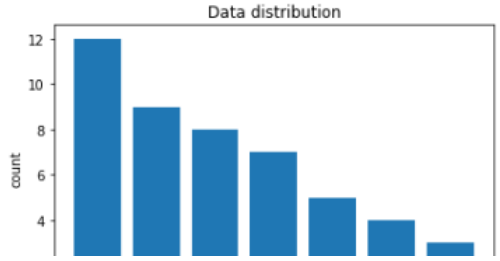
```
# split data to get the same distributiom


#get index and its label
idx_label = {}
for i in range(len(dataset)):
    _, label = dataset[i]
    idx_label[i] = label

#split for data validation
x_train, x_val, y_train, y_val = train_test_split(list(idx_label.keys()), list(idx_label.values
()),
                                                  stratify = list(idx_label.values()), test_size
=0.05)
```

In [17]:

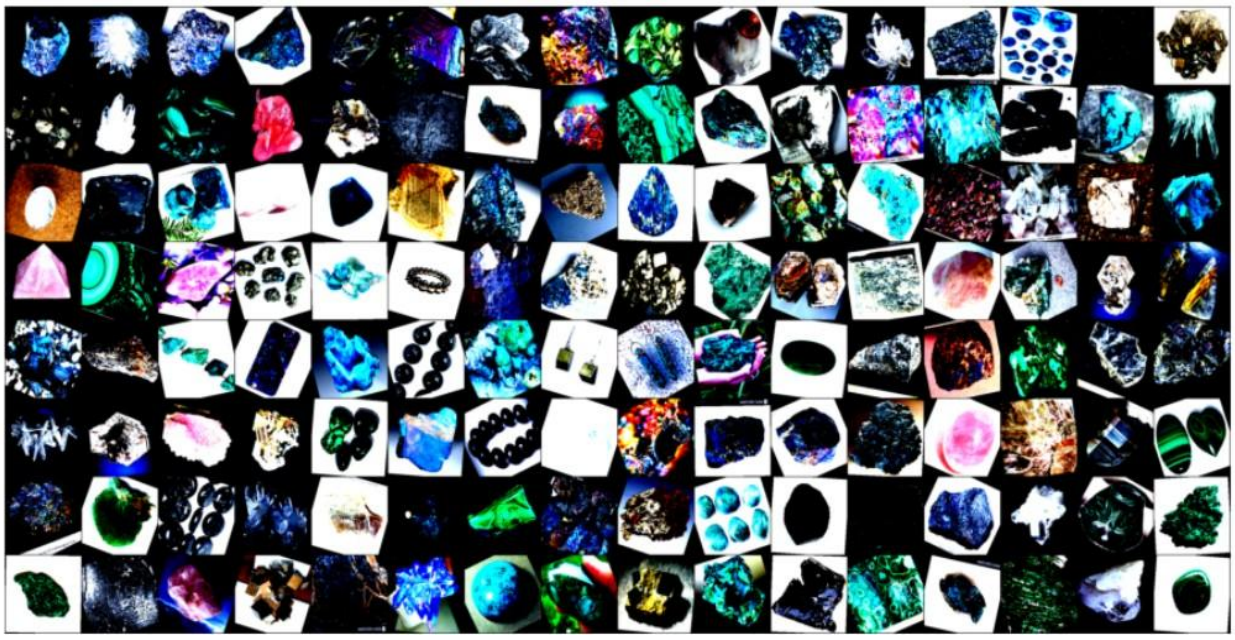
```
#exclude validation index from dataset
x_val
idx_label_2 = {}
for idx, label in idx_label.items():
    if idx not in x_val:
        idx_label_2[idx] = label

#split data train and test after exlude x_val
x_train, x_test, y_train, y_test = train_test_split(list(idx_label_2.keys()), list(idx_label_2.v
alues()),
                                                    stratify = list(idx_label_2.values()), test_
size=0.05)
print(len(x_train))
print(len(x_val))
print(len(x_test))
```

In [18]:
```
plot_dist(x_train)
plot_dist(x_val)
plot_dist(x_test)
```






```
images.shape: torch.Size([128, 3, 224, 224])
```



In [25]:

```
def fit(epochs, model, train_loader, val_loader, criterion, optimizer):
    train_losses = []
    test_losses = []
    train_accu = []
    val_accu = []
    fit_time = time.time()
    for e in range(epochs):
        since = time.time()
        running_loss = 0
        train_acc = 0
        for image, label in train_loader:
            optimizer.zero_grad()
            image = image.to(device); label = label.to(device);

            output = model(image)
            ps = torch.exp(output)
            _, top_class = ps.topk(1, dim=1)
            correct = top_class == label.view(*top_class.shape)
            train_acc += torch.mean(correct.type(torch.FloatTensor))

            loss = criterion(output, label)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()

        else:
            model.eval()
            test_loss = 0
            accuracy = 0
            with torch.no_grad():
                for image, label in val_loader:
                    image = image.to(device); label = label.to(device);
```

```
            model.eval()
            test_loss = 0
            accuracy = 0
            with torch.no_grad():
                for image, label in val_loader:
                    image = image.to(device); label = label.to(device);

                    output = model(image)
                    loss = criterion(output, label)

                    ps = torch.exp(output)
                    _, top_class = ps.topk(1, dim=1)
                    correct = top_class == label.view(*top_class.shape)
                    accuracy += torch.mean(correct.type(torch.FloatTensor))

                    test_loss += loss.item()

            train_losses.append(running_loss/len(train_loader))
            test_losses.append(test_loss/len(val_loader))
            train_accu.append(train_acc/len(train_loader))
            val_accu.append(accuracy/len(val_loader))
            model.train()
            print("Epoch: {}/{}.. ".format(e+1, epochs),
                  "Train Loss: {:.3f}.. ".format(running_loss/len(train_loader)),
                  "Test Loss: {:.3f}.. ".format(test_loss/len(val_loader)),
                  "Train Accuracy: {:.3f}.. ".format(train_acc/len(train_loader)),
                  "Test Accuracy: {:.3f}.. ".format(accuracy/len(val_loader)),
                  "Time: {:.2f}s" .format((time.time()-since)))

    history = {'train_loss' : train_losses, 'val_loss': test_losses,
               'train_accuracy': train_accu, 'val_accuracy':val_accu}
    print('Total time: {:.2f} m' .format((time.time()- fit_time)/60))
    return history
```

```
acy: 0.854..  Time: 39.29s
Epoch: 27/40..  Train Loss: 0.584..  Test Loss: 0.480..  Train Accuracy: 0.784..  Test Accur
acy: 0.812..  Time: 39.00s
Epoch: 28/40..  Train Loss: 0.588..  Test Loss: 0.453..  Train Accuracy: 0.792..  Test Accur
acy: 0.812..  Time: 37.95s
Epoch: 29/40..  Train Loss: 0.571..  Test Loss: 0.461..  Train Accuracy: 0.804..  Test Accur
acy: 0.771..  Time: 38.13s
Epoch: 30/40..  Train Loss: 0.559..  Test Loss: 0.440..  Train Accuracy: 0.793..  Test Accur
acy: 0.812..  Time: 38.42s
Epoch: 31/40..  Train Loss: 0.560..  Test Loss: 0.418..  Train Accuracy: 0.806..  Test Accur
acy: 0.875..  Time: 36.44s
Epoch: 32/40..  Train Loss: 0.544..  Test Loss: 0.445..  Train Accuracy: 0.807..  Test Accur
acy: 0.833..  Time: 37.38s
Epoch: 33/40..  Train Loss: 0.511..  Test Loss: 0.463..  Train Accuracy: 0.816..  Test Accur
acy: 0.833..  Time: 37.75s
Epoch: 34/40..  Train Loss: 0.501..  Test Loss: 0.448..  Train Accuracy: 0.826..  Test Accur
acy: 0.812..  Time: 38.07s
Epoch: 35/40..  Train Loss: 0.500..  Test Loss: 0.491..  Train Accuracy: 0.806..  Test Accur
acy: 0.812..  Time: 37.96s
Epoch: 36/40..  Train Loss: 0.510..  Test Loss: 0.439..  Train Accuracy: 0.816..  Test Accur
acy: 0.854..  Time: 37.03s
Epoch: 37/40..  Train Loss: 0.495..  Test Loss: 0.462..  Train Accuracy: 0.822..  Test Accur
acy: 0.812..  Time: 37.57s
Epoch: 38/40..  Train Loss: 0.489..  Test Loss: 0.446..  Train Accuracy: 0.813..  Test Accur
acy: 0.833..  Time: 38.74s
Epoch: 39/40..  Train Loss: 0.487..  Test Loss: 0.489..  Train Accuracy: 0.830..  Test Accur
acy: 0.833..  Time: 37.08s
Epoch: 40/40..  Train Loss: 0.469..  Test Loss: 0.466..  Train Accuracy: 0.836..  Test Accur
acy: 0.792..  Time: 38.01s
Total time: 25.55 m
```

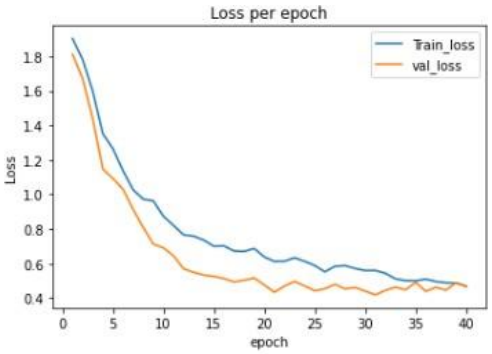
In [28]:

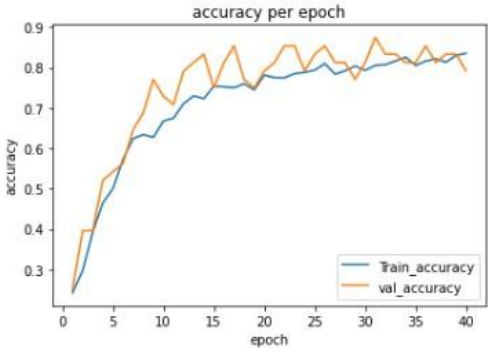
```
#save mode
save_model(model_1, optimizer, 'mineral_seq_own.pt')
```

```
/opt/conda/lib/python3.7/site-packages/torch/serialization.py:402: UserWarning: Couldn't ret
rieve source code for container of type Mineral_1. It won't be checked for correctness upon
loading.
  "type " + obj.__name__ + ". It won't be checked "
```

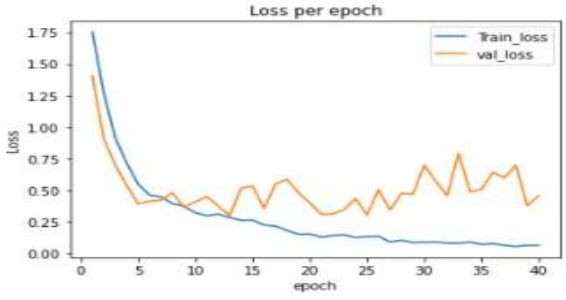
In [29]:

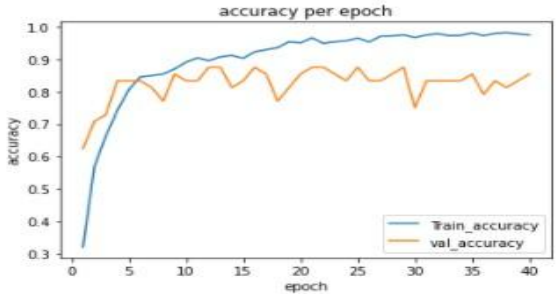
```
def plot_loss(history, n_epoch):
    epoch = [x for x in range(1, n_epoch+1)]
    plt.plot(epoch, history['train_loss'], label='Train_loss')
    plt.plot(epoch, history['val_loss'], label='val_loss')
    plt.title('Loss per epoch')
    plt.ylabel('Loss')
    plt.xlabel('epoch')
    plt.legend();
    plt.show()

def plot_accuracy(history, n_epoch):
    epoch = [x for x in range(1, n_epoch+1)]
    plt.plot(epoch, history['train_accuracy'], label='Train_accuracy')
    plt.plot(epoch, history['val_accuracy'], label='val_accuracy')
    plt.title('accuracy per epoch')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend();
    plt.show()
```





In [33]:
```
plot_loss(history_VGG, epoch)
plot_accuracy(history_VGG, epoch)
```





Prediction and Evaluation

In [40]:

```
def predict_label(model, dataloader):
    prediction_list = []
    labels = []
    model.to(device)
    model.eval()
    for i, batch in enumerate(dataloader):
        image, label = batch
        image = image.to(device); label = label.to(device)

        out = model(image)
        ps = torch.exp(out)
        _, top_class = torch.max(ps , 1)
        preds = np.squeeze(top_class.cpu().numpy())
        prediction_list.append(preds)
        labels.append(label.cpu().numpy())
    return np.squeeze(prediction_list), np.squeeze(labels)
```

In [35]:

```
def predict_plot(test_loader, model, target_label=target_label, n=20):

    # obtain one batch of test images
    dataiter = iter(test_loader)
    images, labels = dataiter.next()
    images.numpy()

    # move model inputs to cuda, if GPU available
    train_on_gpu = torch.cuda.is_available()
    if train_on_gpu:
        images = images.cuda()

    model.eval()
    # get sample outputs
    model.to(device)
    output = model(images)
    # convert output probabilities to predicted class
    _, preds_tensor = torch.max(output, 1)
    preds = np.squeeze(preds_tensor.cpu().numpy()) #np.squeeze(preds_tensor.numpy()) if not train
_on_gpu else np.squeeze(preds_tensor.cpu().numpy())
    images = images.cpu()

    # plot the images in the batch, along with predicted and true labels
    fig = plt.figure(figsize=(25, 4))
    for idx in np.arange(n):
        ax = fig.add_subplot(2, n/2, idx+1, xticks=[], yticks=[])
        plt.imshow(images[idx].permute(1 ,2, 0))
        ax.set_title("{} ({})".format(target_label[preds[idx]], target_label[labels[idx]]),
                     color=("green" if preds[idx]==labels[idx].item() else "red"))
    plt.show()
```

My Model

In [37]:

```
model_mineral = load_model('mineral_seq_own.pt', inferece=True)
model_mineral
```

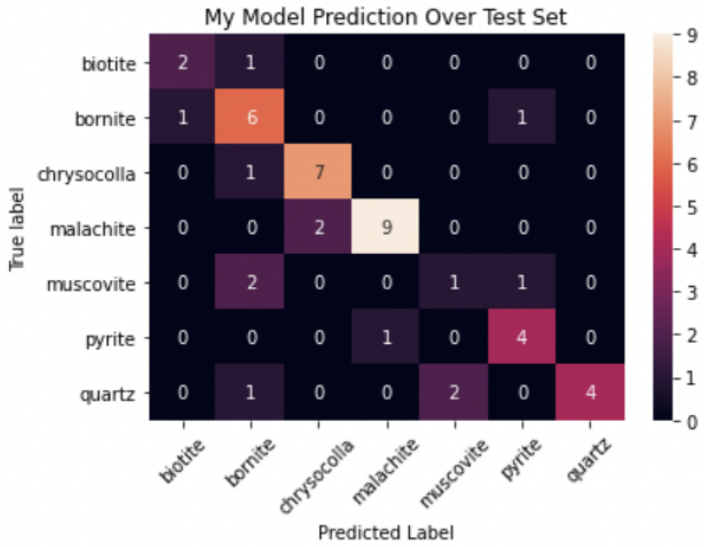
Out[37]:

```
Mineral_1(
  (net): Sequential(
    (0): Conv2d(3, 48, kernel_size=(11, 11), stride=(3, 3))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=3, stride=1, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(48, 128, kernel_size=(5, 5), stride=(1, 1))
    (4): ReLU()
    (5): MaxPool2d(kernel_size=3, stride=1, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(128, 128, kernel_size=(4, 4), stride=(1, 1))
    (7): ReLU()
    (8): MaxPool2d(kernel_size=4, stride=3, padding=0, dilation=1, ceil_mode=False)
    (9): Conv2d(128, 64, kernel_size=(3, 3), stride=(1, 1))
    (10): ReLU()
    (11): MaxPool2d(kernel_size=3, stride=3, padding=0, dilation=1, ceil_mode=False)
    (12): Flatten()
    (13): Linear(in_features=2304, out_features=512, bias=True)
    (14): ReLU()
    (15): Dropout(p=0.3, inplace=False)
    (16): Linear(in_features=512, out_features=7, bias=True)
    (17): LogSoftmax()
  )
)
```

In [64]:
```
#how model perfome in test_data
y_predict, y_true = predict_label(model_mineral, test_loader)
```

In [65]:
```
#plot confusion matric
print(classification_report(y_true, y_predict))
sns.heatmap(confusion_matrix(y_true, y_predict), annot=True)
plt.ylabel('True label'); plt.xlabel('Predicted Label')
plt.yticks(np.arange(0.5, len(target_label)), labels=target_label, rotation=0);
plt.xticks(np.arange(0.5, len(target_label)), labels=target_label, rotation=45)
plt.title('My Model Prediction Over Test Set')
plt.show()
```

```
              precision    recall  f1-score   support

           0       0.67      0.67      0.67         3
           1       0.55      0.75      0.63         8
           2       0.78      0.88      0.82         8
           3       0.90      0.82      0.86        11
           4       0.33      0.25      0.29         4
           5       0.67      0.80      0.73         5
           6       1.00      0.57      0.73         7

    accuracy                           0.72        46
   macro avg       0.70      0.68      0.67        46
weighted avg       0.74      0.72      0.72        46
```



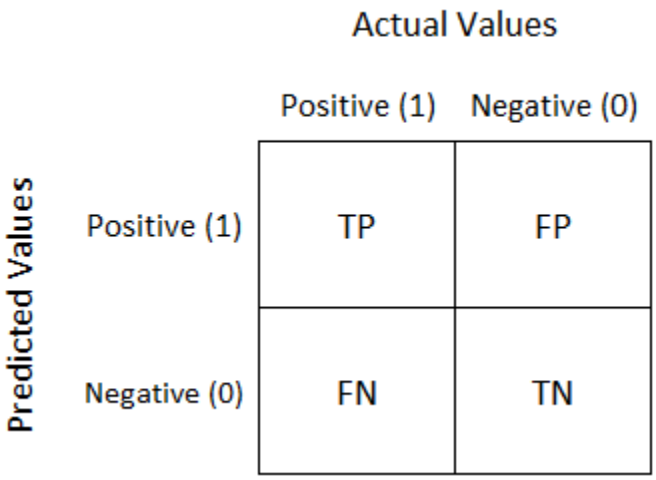
CLASSIFICATION MATRIX:

True Positives (TP) - These are the correctly predicted positive values which mean that the value of the actual class is yes and the value of the predicted class is also yes.

True Negatives (TN) - These are the correctly predicted negative values which mean that the value of the actual class is no and value of the predicted class is also no.

False Positives (FP) – When the actual class is no and the predicted class is yes.

False Negatives (FN) – When the actual class is yes but the predicted class is no.



About the accuracy Parameters:

- **Precision** - Precision is the ratio of correctly predicted positive observations to the total predicted positive observations.

Precision = TP/TP+FP

- **Recall** (Sensitivity) - Recall is the ratio of correctly predicted positive observations to the all observations in actual class - yes.

Recall = TP/TP+FN

- **F1 score** - F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. Intuitively it is not as easy to understand as accuracy, but F1 is usually more useful than accuracy, especially if you have an uneven class distribution. Accuracy works best if false positives and false negatives have similar cost. If the cost of false positives and false negatives are very different, it's better to look at both Precision and Recall.

  F1 Score = 2*(Recall * Precision) / (Recall + Precision)

This was an approach used by beginners to get required results on image processing however better results may be available by using various other algorithms.

Total WorkPlan

In this Project, we went through the required literature reading part which includes gathering data of different minerals from different websites and data collection platforms and transforming the images into a similar size by sklearn tools. We further got the knowledge of different modeling processes required to build a platform recognizing different minerals by their images. We further looked into Artificial Neural Network and SVM.

Now, after doing all those works, we used tensorflow and PyTorch as an application for this project. We further build the model of image processing and predict the accuracy of the model and refine the model using hyperparameter tuning to get maximum model efficiency. Along with the code, we will look into the final results of our created model.

Conclusion:

The main goal of this paper was to explore the different ways to apply the advanced **image processing and machine learning algorithms for mineral** prospectivity mapping and mineral processing. A set of image processing operations are performed showing the usefulness of the designed database. And also, the development of machine vision for image analysis in the context of mineral processing. With the help of data analysis the configuration and size of the images are configured in order to remove discrepancies.

The further development of the system is done and precision of about 65-70% is observed.

References:

[1] Wikipedia “Image Processing techniques”

[2] Kaggle\Mining Images Dataset

[3] Persson A L 1998 Engineering Geology 50 177 – 186 ISSN 0013-7952 URL http://www.sciencedirect.com/science/article/pii/S001379529800009X

[4] Tessier J, Duchesne C and Bartolacci G 2007 Minerals Engineering 20 1129 – 1144 ISSN 0892-6875 URL http://www.sciencedirect.com/science/article/pii/S0892687507001161

[5] Wang R and Liang Z 2011 Automatic separation system of coal gangue based on DSP and digital image processing 2011Symposium on Photonics and Optoelectronics (SOPO) (IEEE) pp 1–3

[6] Perez C A, Estévez P A, Vera P A, Castillo L E, Aravena C M, Schulz D A and Medina L E 2011 International Journal of Mineral Processing 101 28 – 36 ISSN 0301-751

[7] URL http://www.sciencedirect.com/science/article/pii/S0301751611001037 [6] 2019 URL http://www.image-net.org/

[8] Mlynarczyk M, Ladniak M and Piorkowski A 2014 Physicochemical Problems of Mineral Processing 50(2) 563–573

[9] Liskov B and Wing J M 1993 Family values: A behavioral notion of subtyping Tech. rep. Carnegie-Mellon Univ Pittsburgh PA Dept of Computer Science

[10] Zhang Z 1999 Flexible camera calibration by viewing a plane from unknown orientations Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on 1 (IEEE) pp 666–673

[11] Rodriguez-Galiano, V., et al. "Machine learning predictive models for mineral prospectivity: An evaluation of neural networks, random forest, regression trees and support vector machines." Ore Geology Reviews 71 (2015): 804-818.

[12] Partington, G. A., and M. J. Sale. "Prospectivity mapping using GIS with publicly available earth science data–a new targeting tool being successfully used for exploration in New Zealand." Pacrim 2004 Congress Volume, Adelaide. 2004.

[13] https://cartesgeoscientifiques.mem.gov.ma

[14] Ouanan, H., Ouanan, M., & Aksasse, B. (2018, July). Deep Learning Technology for Identifying a Person of Interest in the Real World. In International Conference on Advanced Intelligent Systems for Sustainable Development (pp. 220-227). Springer, Cham.

[15] Ouanan, H., Ouanan, M., & Aksasse, B. (2018). Non-linear dictionary representation of deep features for face recognition from a single sample per person. Procedia Computer Science, 127, 114-122.

Thanks and Regards

Kumar Shivam

19155044

Mining Enggineering (Btech)