Universidad de Montemorelos

Facultad de Ingeniería y Tecnología

Ingeniería en Sistemas Computacionales

Automatic classification of plutonic rocks with machine learning applied to extracted colors on iOS devices

Sarah Hernández Serrano

1170469

Asesor: Dr. Germán Harvey Alférez Salinas

Co-asesores: Dr. Benjamin Clausen and Dra. Ana María Martínez Ardila

Montemorelos, Nuevo León, México

00 de mayo de 2021

Abstract

The classification of ~~different~~ rocks can provide important information about the conditions under they were formed and their tectonic environment. Color is a key property for rock classification. However, it is difficult to describe them because perceived colors ~~in rocks also~~ depend on the observer's experience ~~with similar observations.~~ Moreover, the equipment for rock classification is expensive, which makes this equipment prohibitive to geology students and amateurs. The contribution of this research work is twofold. First, since color is a key property for rock classification, feature reduction was applied in rock images extracting just the four dominant colors before ~~training the machine learning algorithms; to this end, the k-means clustering method was used.~~ Second, we present an iOS application that uses machine learning to classify images of the following plutonic rocks: granite, granodiorite, gabbro, and diorite. Different machine learning models were created and evaluated with the Logistic Regression, K-Nearest Neighbors, Decision Trees, Support Vector Machine, and Convolutional Neural Network algorithms. The best results were achieved with K-Nearest Neighbors algorithm. The evaluation results of the algorithm are the following: 82% of accuracy, 84% of precision, 89% of recall, and 83% of F1-score.

Index Terms

Machine Learning, Color Extraction, iOS Devices, Geology, Rock Classification, Features Reduction, Dominant Colors, K-means Clustering, Logistic Regression, K-Nearest Neighbors, Support Vector Machine, Convolutional Neural Network, Decision Trees.

CONTENTS

Automatic classification of plutonic rocks with machine learning applied to extracted colors on iOS devices

Sarah Hernández, Germán H. Alférez, Benjamin Clausen and Ana M. Martínez

School of Engineering and Technology, Montemorelos University, Mexico

I. Introduction

A. Background

The classification of different rocks can provide important information about the conditions under which they were formed. They can also tell us much about their tectonic environment, given that they are closely linked to the convection of tectonic plates. Their mineral and chemical makeup can be used to learn about the composition, temperature and pressure that exists within the Earth's mantle [1].

Plutonic rocks are a type of igneous rock (one of the three main types of rocks in the world and also the most common) formed when magma in the middle of the volcano cools and solidifies below the Earth's surface. They come in many different sizes and colors, and have a wide variety of uses [2, 3].

Our contribution is twofold: extract the dominant colors of plutonic rock images and use these information to classify them with the implemention of machine learning. Color changes in rocks may indicate changes in the rock's mineral assemblage, texture, organic carbon content (shales), or other properties [4]. That is why color is a key property for rock classification. In addition, machine learning, a branch of artificial intelligence, provides easy classification of images whitin mobile applications and is becoming an appealing tool in various fields of earth sciences, especially in predictions and resources estimation [5].

In [], a previous work by Elías L. Vázquez and Germán H. Alférez has be done for the classification of plutonic rocks on android devices using the whole set of features of the same dataset instead of just the color extracted features.

B. Problem Statement

The equipment for rock classification is expensive, which makes it prohibitive to geology students and amateurs. In addtion, despite the fact that color is a key property for rock classification, it is an attribute difficult to describe because perceived colors in rocks also depend on the ~~stimulus area and~~ the observer's experience with similar observations; so a color may often be named differently by different persons [4].

Sarah Hernández is a computer science engineering student at the School of Engineering and Technology of Montemorelos University, Nuevo León, Mexico, e-mail: 1170469@alumno.um.edu.mx.

Germán H. Alferez, Ph.D., is the director of the Institute of Data Science and professor at the School of Engineering and Technology of Montemorelos University, Nuevo León, Mexico, e-mail: harveyalferez@um.edu.mx.

Benjamin Clausen, Ph.D., is a professor at the Department of Earth and Biological Sciences of Loma Linda University, California, United States, e-mail: bclausen@llu.edu.

Ana M. Martínez, Ph.D., is a professor at the Department of Earth and Biological Sciences of Loma Linda University, California, United States, e-mail: anmartinez@llu.edu.

C. Justification

There are two reasons to create an iOS mobile application that allows the classification of rocks ~~through colors~~ with machine learning. First, no such application exists. A mobile application offers the flexibility to carry out the classification on real time and could be an alternative to expensive traditional methods of rock classification. Second, the extraction of the dominant colors allows geologists to more accurately detect color changes in rocks and is useful to reduce the number of features in images used to train machine learning, which provides easy classification of rocks within the application.

D. Objectives

The main objective of this research work is to create an application that uses machine learning to extract the dominant colors from images and use these colors to classify plutonic rocks. This objective is achieved with the following sub-objectives:

- Determine the best k number of clusters to use in the k-means clustering machine learning method to extract the dominant colors from images of plutonic rocks.
- Use the extracted dominant colors and the percentage of pixels belonging to each color cluster as the input to train the following machine learning algorithms: Logistic regression (LR), K-Nearest Neighbors (KNN), Decision Trees (DT), Support Vector Machine (SVM), and a Convolutional Neural Network (CNN).
- Validate the models generated to get the best suited to deploy the best model on the iOS mobile application.
- Test the effectiveness of the application with images taken from new plutonic rocks to compare the results with our previous work in which a CNN was trained with images using the whole set of features.

E. Hypothesis

The training of a machine learning model with just the dominant colors extracted from rock images can provide an effective way for the classification of plutonic rock images.

II. Theoretical Foundation

A. Underpinnings of our Approach

Our approach is based on the following concepts (see Fig. 1).

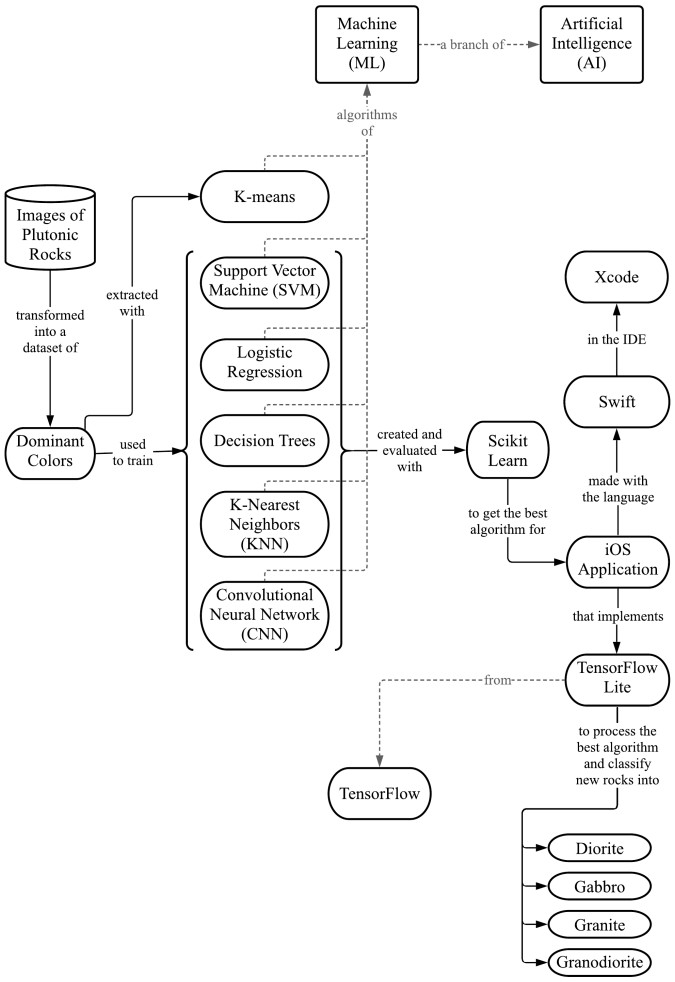

Figure 1. Underpinnings of our approach

1) Plutonic rocks

Plutonic rocks are solidified inside the earth's crust, usually from magma, although they may have been formed by a different mechanism. Plutonic rocks are generally coarse-grained, but not all coarse-grained rocks are plutonic [6].

According to [6], granite rocks are plutonic rocks with a granular texture, composed of similar amounts of quartz, potassium feldspar, and plagioclase as essential minerals, and smaller amounts of one or more minerals, such as biotite, muscovite or hornblende.

Granodiorite is a plutonic rock of the granitoid family, characterized by quartz because plagioclase constitutes more than 2/3 of the total feldspars. Generally, together with granite, it is the most abundant rock of the great batholiths. Its volcanic equivalent is dacite. Granodiorite is similar to granite, but with less potassium feldspar and more plagioclase.

Gabbro is a plutonic rock composed mainly of calcium plagioclase and clinopyroxene or orthopiroxene, with or without olivine or amphibole. It is the intrusive equivalent of basalt. It is distinguished from diorite by the nature of plagioclase, which is higher in calcium than in sodium.

Diorite has about the same structural properties as granite but darker colour and more limited supply. Commonly is composed of about two-thirds plagioclase feldspar and one-third dark-coloured minerals, such as hornblende or biotite. The presence of sodium-rich feldspar, oligoclase or andesine, in contrast to calcium-rich plagioclase, labradorite or bytownite, is the main distinction between diorite and gabbro [7].

2) Machine Learning

Machine learning is a branch of artificial intelligence that lies at the intersection of computer science, engineering, and statistics and often appears in other disciplines. Machine Learning systems can be classified according to the type of supervision they get during training [8, 9]. In this way, there are two major categories:

a) Supervised learning

In supervised learning, the training set fed to the algorithm includes the desired solutions, called labels. A typical supervised learning task is classification [9].

Some common algorithms of supervised learning are described as follows:

- Logistic Regression (LR) is commonly used to estimate the probability of an instance to belong to a particular class. This model computes a weighted sum of the input features (plus a bias term) and outputs the logistic of this result. The logistic is a sigmoid function that outputs a number between 0 and 1. This makes it a binary classifier [9].
- K-Nearest Neighbors (KNN) is a simple algorithm in which its priniciple is to find a predefined number of training samples closest in distance to the new point, and predict the label from these. Being a non-parametric method, it is often successful in classification situations where the decision boundary is very irregular [10].
- Decision Trees (DT) algorithm consists of split nodes and leaf nodes. Each split node performs a split decision and routes a data sample either to the left or the right child node. In the tree structures, leaves represent the labels, non-leaf nodes are the input features, and branches represent conjunctions of features that lead to the classifications [11, 12].
- Support Vector Machine (SVM) algorithm is a powerful and versatile machine learning model, particularly well suited for classification of complex small- or medium-sized datasets. In this algorithm, the data is plotted in a n-dimensional space (number of features) and a decision boundary (hyperplane) split it into classes [9, 8].
- Convolutional Neural Networks (CNNs) emerged from the study of the brain's visual cortex. The most important building block is the convolutional layer. The CNN´s architecture consists of several connected layers allowing the network to concentrate on small low-level features in the first hidden layer, then assemble them into larger higher-level features in the next layer, and so on [9].

b) Unsupervised learning

In unsupervised learning the training data is unlabeled. One of the most important unsupervised learning algorithms is k-means. K-means is a simple algorithm capable of clustering an unlabeled dataset very quickly and efficiently, often in just a few iterations. You have to specify the number of clusters k that the algorithm must find and it could easily label all the instances in the dataset by assigning each of them to the cluster whose centroid is closest [9].

3) Dominant colors

The dominant colors refer to the principal colors presented in an image. The dominant colors of an image are selected grouping the image pixels according to their color in the RGB (Red, Green, Blue) additive color space. The average color of each group are the dominant colors and the number of pixels of that groups are the dominant colors´ percentage.

In the RGB space, the light spectra of varying fractions of the three primary colors referred to as channels, combine to make new colors. Each channel has intensity values from 0 to 1 that are scaled by the number of bits used to represent each component. The 24-bit color cube used in this research has components in the range of 0–255. Each color in the color space can be represented as a three-value vector [13].

4) Underlying technologies

a) Scikit learn

Scikit-learn is a Python module that integrates a wide range of state-of-the-art machine learning algorithms for medium-scale supervised and unsupervised problems. This package focuses on bringing machine learning to non-specialists using a general-purpose high-level language. Emphasis is put on ease of use, performance, documentation, and API consistency. It has minimal dependencies and is distributed under the simplified BSD license, encouraging its use in both academic and commercial settings [14].

b) Tensorflow

TensorFlow[1] is an open-source library developed by Google. It has become very popular because it provides APIs that facilitates machine learning. TensorFlow also has a faster compilation time than other deep learning libraries such as Keras and Touch. TensorFlow supports both CPU and GPU computing devices [15].

c) Tensorflow Lite

TensorFlow Lite[2] is a set of tools to help developers run TensorFlow models on mobile, embedded, and IoT devices. It enables on-device machine learning inference with low latency and a small binary size [16].

TensorFlow Lite consists of two main components:

- The TensorFlow Lite interpreter, which runs specially optimized models on many different hardware types, including mobile phones, embedded Linux devices, and microcontrollers.
- The TensorFlow Lite converter, which converts TensorFlow models into an efficient form for use by the interpreter, and can introduce optimizations to improve binary size and performance.

[1]https://www.tensorflow.org

[2]https://www.tensorflow.org/lite

d) Swift

Swift[3] is an open source programming language designed for macOS, iOS, watchOS, tvOS, and more. Swift already supports all Apple platforms and Linux, with community members actively working to port to even more platforms [17]. In 2010, Apple started developing Swift, a new programming language that would rival Objective-C in type safety, security, and hardware performance. Swift is more than 2.6x faster than Objective-C and more than 8.4x faster than Python. ~~Swift 1.0 was released in September 2014~~ [18].

e) Xcode

Xcode[4] is a long-standing tool for app production and a well-known integrated development environment (IDE), enabling developers to write the code and compile apps that can be used on a variety of different devices and operating systems [19]. It also includes a unified macOS SDK and all the frameworks, compilers, debuggers, and other tools ~~you~~ need to build apps that run natively on Apple Silicon [20].

f) CocoaPods

CocoaPods[5] is a dependency manager for Swift and Objective-C Cocoa projects. The dependencies of the projects ~~just~~ are specified in a single text file called Podfile. CocoaPods resolves dependencies between libraries, fetches the resulting source code, then links it together in an Xcode workspace to build a single project. [21].

[3]https://swift.org

[4]https://developer.apple.com/xcode/

[5]https://cocoapods.org

B. Related Work

Over the last decade, there has been considerable progress in developing methodologies based in machine learning for many of Earth Science applications [22]. We present relevant articles related to the implementation of machine learning in the classifcation of images into the geoscience field.

1) Rock lithology image classification using machine learning

In [23], Ran et al. also worked on the classification of rocks from images using machine learning. A total of 2,290 images with typical rock characteristics of six rock types (granite, limestone, conglomerate, sandstone, shale, mylonite) were used in the study. The images were labeled according to the clarity of the rock and then were cropped into 14,589 sample patches of 512 × 512 pixels compressed to 128 × 128 pixels. 60% of the patches of each type of rock were selected for the training dataset, 20% for the validation dataset, and 20% for the testing dataset. Finally, they proposed a deep CNN model that was compared with four machine learning models: SVM, AlexNet, GoogleLeNet Inception v3, and VGGNet-16. The proposed model achieved the highest overall accuracy of 97.76% compared with the other models.

In [24], Cheng and Guo proposed a deep CNN to identify the rock granularity in images under three color spaces. The dataset contained 4,200 RGB images of feldspar sandstone rocks collected from an oil field in Ordos and divided into three types: coarse, medium granular, and fine feldspar sandstone. The images were normalized to 224 × 224 pixels and

converted to YCbCr and HSV color spaces. The proposed CNN was a 6-layer structure: 4 convoluted layers with ReLU as the activation function and 2 fully connected layers with Softmax as the classifier. The model was trained for each color space with 75% of the experimental data, a batch size of 100, and different kernel sizes and learning rates. The lowest error rates were obtained with a learning rate of 0.0005, the kernel sizes of 11, 5, 3, and 3 for each convolutional layer respectively, and the cross-validation for HSV color space. In RGB color space, the classification accuracy achieved 98.5%.

2) Rock classification with machine learning on Android devices

In [25], Fan et al. created a method for rock lithology recognition on Android devices based on two lightweight CNNs. A total of 3,208 images taken by a smartphone camera were selected from the China Geological Survey dataset. The images of 28 rock categories were reduced to 214 × 214 pixels and the 80% were used to train two CNNs pretrained with ImageNet dataset based on MobileNet and SqueezeNet models. The achieved occupation sizes were 19.6 M and 36.8 M for MobileNet and SqueezeNet, and 232.7 M for a heavyweight ResNet50 model built to compare the models. SqueezeNet was almost two times faster than MobileNet, and 7 times faster than ResNet50. A rock recognition software based on the trained models was developed for Android devices. The results for SqueezeNet and MobileNet on Android smartphones were as follows: the file sizes of 9.2 MB and 17.6 MB, the execution time from 736 to 366 and 1,218 to 523 milliseconds, and the recognition accuracies of 94.55% and 93.27%.

Also in [26], Fan et al. improved their work using a model based on ShuffleNet for quick and accurate rock lithology recognition with smartphones. A total of 3,795 images were taken to 30 different kinds of rocks collected from multiple locations in East China. The proposed model was trained using 80% of the dataset, 3,600 iteration steps, a learning rate of 0.008, and the parameters imported by the transfer learning method from ShuffleNet model pretrained with the ImageNet dataset. ShuffleNet occupied a space of 18.2M and was compared with MobileNet, SqueezeNet, and ResNet50 that occupied a space of 34.5M, 25M, and 219.4M respectively. An Android application was created using the models and run on three Android smartphones (Huawei, Samsung, and Oppo). The average recognition time of ShuffleNet for a single rock image was 786 milliseconds, the reached accuracy was 97.65% in the verification data set of PC and 95.30% on the test data set of the smartphones.

In [], the authors worked on the automatic classification of plutonic rocks in Android devices. The dataset contained 71 images of diorite, gabbro, granite, granodiorite, monzodiorite, and tonalite which classification was based on petrographic analysis of Cretaceous granitoids collected from Peru. To increase the number of images, a new dataset of 846 patches from around 450 × 700 to 900 × 1000 pixels was generated from the first dataset. A deep neural network was created with 4-layer structure: two convolutional, and two fully connected layers. After each convolutional layer it was a max pooling layer of 2×2 size, and a flatten and a dropout layers before the fisrt and second fully conected layers respectively. Several combinations of the rock classes were trained with 50% of the image patches, 100 epochs and a batch size of 32. The best combination was with four classes of plutonic rocks (gabbro, diorite, granite and granodiorite) achieving an accuracy of 95%, an average precision of 96%, an average recall of 95%, and an average F1-score of 95%. Finally, an Android application was developed using the proposed model trained with the best combination of rocks. The application evaluation results were: 70% for gabbro, 28.5% for diorite, 100% for granodiorite, and 28.5% for granite.

3) Mineral images recognition applying machine learning and feature extraction

In [27], Zhang et al. worked on the intelligent identification for rock-mineral images using ensemble machine learning algorithms (model stacking). A total of 481 images of four minerals (K-feldspar, perthite, plagioclase, and quartz) were obtained with the camera on the top of a microscope. The target RGB images were cropped to cover the minerals and then processed to be 299 × 299 pixels. A deep learning model based on Inception-v3 was adopted to extract high-level features (such as chromatic aberration and texture) from the images and train the algorithms of LR, SVM, KNN, Random Forest (RF), Multilayer Perceptron (MLP), and Gaussian Naive Bayes (GNB). LR, SVM, and MLP had a significant effect on extracted features, with higher accuracy (90.0%, 90.6%, and 89.8%) than the other models. The new features generated by this three models were employed for the model stacking in a new instance of LR. The stacking model showed a better performance than the single models, with an accuracy of 90.9%.

In [28], Maitre et al. worked on the automated recognition of mineral grains based on supervised machine learning. A large mosaic RGB image was constructed with several views of a sample surface taken with a stereo-zoom binocular microscope. The sample surface contained grains of 27 different mineral species (plagioclase, augite, background, hypersthene, ilmenite, magnetite, titanite, hornblende, etc.) and was scanned with an automated Scanning Electron Microscopy (SEM) that assigned to each grain a mineral specie based on its chemical signature. Both images were divided in 3,192 sub-images of 600 × 600 pixels. To label the grains of the mosaic image, the simple linear iterative clustering (SLIC) algorithm was applied for superpixel segmentation to match each superpixel of the mosaic's sub-images with the superpixels of the SEM's scan sub-images. From the computed RGB superpixel histograms, the color intensity (quantile) and peak intensity (ratio between the number of pixels in the first and second maximum bins to total number of pixels) were extracted as features for each superpixels. KNN, RF, and Classification and Regression Trees (CART) algorithms were trained with 70% of the extracted features, and tested with the other 30% using the kappa statistics, precision, recall, and f1-score indicators. The RF algorithm gave the best results with a global accuracy of 89%.

4) Comparison of the articles

Table I illustrates the comparison of the articles previously described.

As evidenced in these articles, machine learning has been an accurate way for image classification in the field of geosciences. Feature extraction was applied in [28] and [27] before the training of several supervised machine learning models to classify mineral samples. These articles showed their higher accuracies from around 90%, however, the number of extracted features per instance used for training was even too large compared with the 16 extrected features per image presented in this research.

Four articles [23, 24, 25, 26] introduce the comparison of different CNN model structures to analyze the performance of these models for rock lithology classification using all the image features rather than extracted features. Although these models showed great results in their evaluations, all of them were trained with diverse types of rocks compared to the classification of only plutonic type rocks made in this work. This type of classification becomes more difficult because the rocks of a same type have very similar characteristics.

The articles including the deployment of the created models on android mobile aplications for rock lithology classification are[25, 26]. Also in [], Elías L. Vázquez and Germán H. Alférez made a previos work using the same dataset presented in our study to deploy their proposed model on an Android application for the automatic classification of plutonic rocks. However, this model was trained with all the image features instead of selecting just the dominant colors. There are no works presenting the evaluation results of the deployment of machine learning on iOS devices.

III. Results

A. Methodology

This section describes the steps followed in this research work.

1) Data description

The ~~original~~ images were provided by the Department of Earth and Biological Sciences at Loma Linda University. The dataset contains images of four classes of plutonic rocks: granite, granodiorite, gabbro, and diorite with a total of 81 images. The images used in the following experiments are available online[6]. Table II shows the number of images per class.

Table II
Number of images per class

Class	Number of images
Granite	16
Granodiorite	21
Gabbro	19
Diorite	25
Total of images	**81**

2) Data preparation

In this step the images were processed to extract the color values from image pixels. The notebook with the source code for data preparation is available online[7]. In listing 1, the "load_images_from_folder" function in lines 6 to 18, is declared to process all the files with image extension ('.png', '.jpg', '.jpeg') from a folder. After processing an image, the color of each pixel in it is represented by the RGB color space.

```
1  import cv2, os
2  from PIL import Image, ImageOps
3  import numpy as np
4  from itertools import repeat
5
6  def load_images_from_folder(folder, size):
7    images = []
8    for filename in os.listdir(folder):
9      if(not filename.lower().endswith(('.png', '.jpg',
           '.jpeg'))):
10       print(filename, 'removed')
11       continue
12     img = Image.open(os.path.join(folder,filename))
13     fit_and_resized_image = ImageOps.fit(img, size,
           Image.ANTIALIAS)
14     img = np.array(fit_and_resized_image)
15     img = img[...,:3]
16     if img is not None:
17       images.append(img)
18   return (images, filenames)
```

Listing 1. Function to load images from a folder

In listing 2, the "load_images_from_directory" function is declared from lines 1 to 13 to iterate the subfolders contained in a main directory and process the images in them with the support of the previous declared "load_images_from_folder" function. It also assigns labels to the processed images according to the name of the folder in which they are as shown in line 9. For that reason, it is important to group the images in subfolders based on their rock class name.

Finally, in line 17, the rock images in the folder declared in line 15 are processed in RGB format.

```
1  def load_images_from_directory(path, size):
2    paths = os.listdir(path)
3    x = []; y = []; files = [];
4    for folder in paths:
5      if(os.path.isfile(folder)):
6        print(folder, 'removed')
7        continue
8      images, filenames =
           load_images_from_folder(os.path.join(path,
           folder), size)
9      lbls = list(repeat(folder, len(images)))
10     x.extend(images)
11     y.extend(lbls)
12     files.extend(filenames)
13   return (x, y, files)
14
15 IMG_PATH = '../Rock images/train'
16 IMG_SIZE = (512, 512)
17 x, y, files = load_images_from_directory(IMG_PATH,
       IMG_SIZE)
```

Listing 2. Function to load images from a directory

3) Determining the ~~best k~~ number of color clusters

The k-means algorithm was used to extract the dominant colors from the images. ~~First, it was necessary to calculate the optimal number of clusters (groups of pixels with similar color) to use in the algorithm. In listing 3,~~ the Elbow method was used for this purpose. This method consists of iterating in a range of possible cluster numbers to use in the k-means algorithm and determine the best number. The possible cluster numbers are from 2 to 20 as shown in line 4 and then are iterated in lines 6 to 9. At each iteration each cluster number is selected as the k number of the k-means algorithm and

[6]https://github.com/sarah-hs/Color-extraction/tree/main/Rock-images
[7]http://bit.ly/3pyh3JL

trained in line 8 with an instance of the images processed at the previous step. In line 9 the score obtained in k-means at each iteration is appended to a list of scores. From lines 11 to 16 the k-means scores are plotted with their respective k numbers. The number at the elbow of the plot is the best k number of clusters, that number is 4 in this experiment (see Fig. 2).

```
1  img = x[0]
2  reshape = img.reshape((img.shape[0] * img.shape[1],
       img.shape[2]))
3
4  Nclusters = range(2, 20)
5  score = []; distortions = []
6  for n in Nclusters:
7    k = KMeans(n_clusters=n)
8    k.fit(reshape)
9    score.append(k.score(reshape))
10
11 plt.plot(Nclusters,score, 'bx-')
12 plt.xlabel('Number of Clusters')
13 plt.ylabel('Score')
14 plt.title('Elbow Curve')
15 plt.axes().set_xticks(Nclusters)
16 plt.show()
```

Listing 3. Determining the best number of clusters with the Elbow method

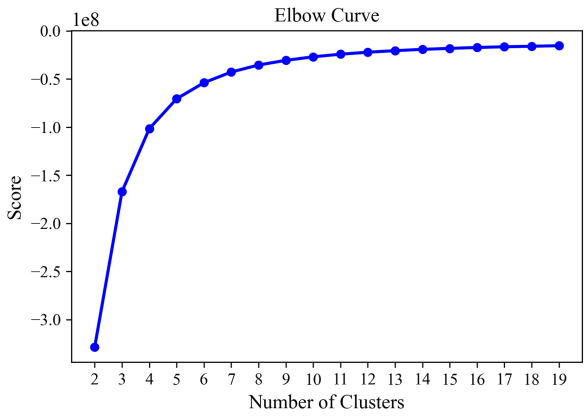

Figure 2. The Elbow method showing the optimal k number of clusters

4) Color extraction

This step describes the process for extracting the dominant colors from the rock images.

In listing 4, the "get_dominant_colors" function receives the pixel values of an image to train the k-means algorithm with 4 clusters, which is the number of clusters selected in the previous step. In line 6, k-means works by separating the pixels of that image into k groups (clusters) of similarly coloured pixels. The colors at each cluster center reflect the average of the attributes of all members of a cluster. The percentage of a cluster calculated in lines 8 to 10 is the number of pixels within that cluster. From lines 14 to 18, the center color in each cluster and its percentage is added to the features list returned in line 20. This list has sixteen elements: the four main colors in RGB (red, green, and blue) color format and their percentage (see Fig. 3).

```
1  from sklearn.cluster import KMeans
2  CLUSTERS = 4
3
4  def get_dominant_colors(img):
5    reshape = img.reshape((img.shape[0] * img.shape[1],
       img.shape[2]))
6    cluster = KMeans(n_clusters=CLUSTERS).fit(reshape)
7
8    labels = np.arange(0,
       len(np.unique(cluster.labels_)) + 1)
9    (hist, _) = np.histogram(cluster.labels_, bins =
       labels)
10   hist = hist.astype("float"); hist /= hist.sum()
11
12   features = []
13
14   colors = sorted([(percent, color) for (percent,
       color) in zip(hist, cluster.cluster_centers_)])
15
16   for (percent, color) in colors:
17     features.extend(color)
18     features.append(percent)
19
20   return features
```

Listing 4. Function to extract the dominant colors from a single image

In listing 5, the processed data of the rock images is iterated to extract the dominant colors with the previous explained "get_dominant_colors" function of listing 4. These colors are added to a new array called "extracted_colors". In line 6, the extracted colors are saved together with their respective rock label in a CSV file. This file is used in the next step to train the machine learning algorithms. The CSV is available online[8].

```
1  extracted_colors = []
2  for img in images:
3    features = get_dominant_colors(img)
4    extracted_colors.append(features)
5
6  np.savetxt(CSV_DIR, np.column_stack((extracted_colors,
       labels)), delimiter=",")
```

Listing 5. Iterate original data to extract dominant colors

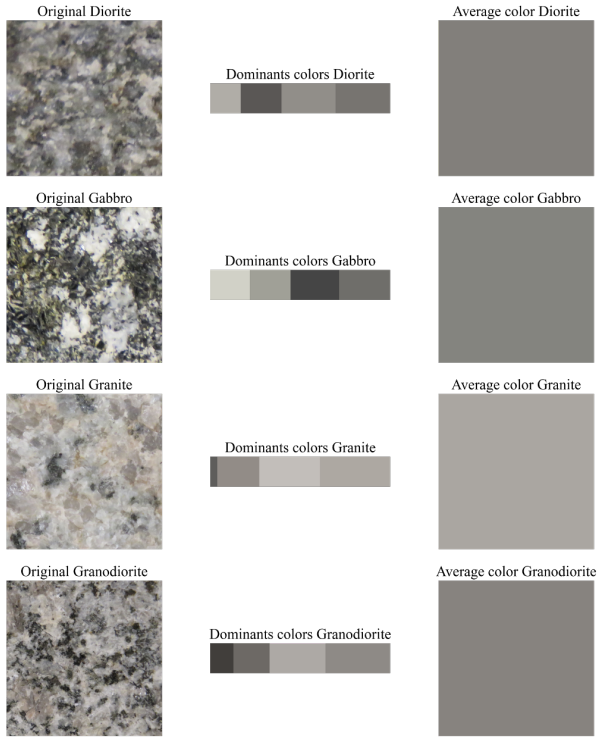

Figure 3. Dominant colors and average color of sample images

[8]https://github.com/sarah-hs/Color-extraction/blob/main/colors.csv

5) Training of the different algorithms with the extracted dominant colors

Five machine learning models were trained for this experiment: LR, KNN, DT, SVM, and a CNN. The notebook showing the training and validation process of the generated models is available online[9].

First, in listing 6, the main dominant colors data is loaded from the CSV file of the previous step into x and y Numpy arrays and splitted into train, test, and validation sets. The "train_test_split" function was used in line 9 to split the data and get a validation set with 20% of the data. In line 10, the other 80% of the split in line 9 was splitted again in 20% for the test set and 80% for the train set. Three sets were used in the experiments: 20% for validation, 16% for testing, and 64% for training from the total of the data. The train set composed for "xtrain" and "ytrain" of the line 10 was used to train all the algorithms.

```
1  import numpy as np
2  data = np.loadtxt(CSV_FILE, delimiter=",")
3
4  x = data[:, :-1]
5  y = data[:, -1]
6
7  # Val = 20%, test = 16%, train = 64%
8  from sklearn.model_selection import train_test_split
       as splitter
9  xtt, xval, ytt, yval = splitter(x, y, train_size=0.8,
       random_state=42)
10 xtrain, xtest, ytrain, ytest = splitter(xtt, ytt,
       train_size=0.8, random_state=42)
```

Listing 6. Splitting the data in train and test

The following parameters were used to set the models as ~~shown in table III.~~

~~Table III~~
~~THE PARAMETERS SET IN THE MODELS~~

~~Model~~	~~Parameters~~	~~Value~~
~~LR~~	~~Solver~~	~~L-BFGS~~
~~KNN~~	~~N_neighbors~~	~~3~~
~~DT~~	~~Class_weights~~	~~Balanced~~
~~SVM~~	~~Kernel~~ ~~Gamma~~	~~RBF~~ ~~Auto~~
~~CNN~~	~~Loss~~ ~~Optimizer~~	~~Categorical_crossentropy~~ ~~Adam~~

- The L-BFGS (Limited-memory Broyden–Fletcher–Goldfarb–Shanno) solver is one of the quasi-Newton methods that approximate the BFGS algorithm, which utilizes a limited amount of computer memory. The solver's methodology starts iterating at a random point (xt) to calculate the minimum from the second derivate of the original function at that point and assign it to a new point (xt+1). The new point will become the starting point for the next iteration. In this way, L-BFGS quickly converges on the solution [29].
- The number of neighbors used in KNN represents the neighbor samples analyzed from the dataset to classify a new sample into the class to which most of their neighbors belong.
- By utilizing class weights, we can increase the importance of a particular class during training. The balanced mode for class weight uses the values of the labels in the train set to automatically adjust the node weights inversely proportional to the class frequencies in the input data [29].
- The Radial Basis Function (RBF) kernel is useful to eliminate the computational requirement of an algorithm. When it is used, the features need to be scaled and the gamma value is setting for [29].
- The auto value in gamma paremeter of scikit learn is represented by $1/numberOfFeatures$. An small gamma will give you low bias and high variance solutions [29, 30].
- Categorical cross-entropy is the most diffused classification cost function, adopted by logistic regression and the majority of neural architectures. This convex function can be easily optimized using stochastic gradient descent techniques and is an excellent choice for classification problems [31].
- Adam (adaptive moment estimation) algorithm is an efficient optimization method that computes adaptive learning rates for each parameter to optimize the weights of the model [29].

Below, listing 7 presents the set and training process of the models. LR, KNN, DT, SVM, and CNN were set as listed in lines 3, 8, 13, 18, and 36 and trained in lines 4, 9, 14, 19, and 37 respectively.

The CNN was compiled in line 36 with the categorical crossentropy loss function and the Adam optimizer. Also several numbers of epochs and batches were tried until the most optimal values were obtained as shown in its training at line 37.

```
1  # Logistic Regression
2  from sklearn.linear_model import LogisticRegression
3  logReg = LogisticRegression()
4  logReg.fit(xtrain, ytrain)
5
6  # K-Nearest Neighbors
7  from sklearn.neighbors import KNeighborsClassifier
8  knn = KNeighborsClassifier(n_neighbors=3)
9  knn.fit(xtrain, ytrain)
10
11 # Decision Trees
12 from sklearn.tree import DecisionTreeClassifier
13 decisionTree =
       DecisionTreeClassifier(class_weight='balanced')
14 decisionTree.fit(xtrain, ytrain)
15
16 # Support Vector Machines
17 from sklearn.svm import SVC
18 svmc = SVC()
19 svmc.fit(xtrain, ytrain)
20
21 # Convolutional Neural Network
22 import tensorflow as tf
23 from matplotlib import pyplot
24 from tensorflow.keras.models import Sequential
25 from tensorflow.keras.layers import Dense, Flatten,
       Dropout, Conv1D, MaxPooling1D
26 from tensorflow.keras.utils import to_categorical
27
28 model = Sequential()
29 model.add(Conv1D(filters=32, kernel_size=2,
       activation='relu',
       input_shape=(n_timesteps,n_features)))
```

[9]https://github.com/sarah-hs/Color-extraction/blob/main/ML-training.ipynb

```
model.add(Conv1D(filters=32, kernel_size=2,
    activation='relu'))
model.add(Dropout(0.5))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(100, activation='relu'))
model.add(Dense(n_outputs, activation='softmax'))
model.compile(loss='categorical_crossentropy',
    optimizer='adam', metrics=['accuracy'])
model.fit(xtrain, ytrain, epochs=10, batch_size=32,
    verbose=0)
```

Listing 7. Models training

Table IV describes the CNN model structure used in this experiment.

Table IV
CNN MODEL STRUCTURE

Layer	Kernel size	Activation function	Output shape
Input features	-	-	16
Conv1D	2 × 2	RELU	16 × 32
Conv1D	2 × 2	RELU	16 × 32
Dropout (50%)	-	-	16 × 32
MaxPooling1D	2 × 2	-	16 × 32
Flatten	-	-	16 × 32
Dense	-	RELU	16 × 32
Dense	-	Softmax	16 × 32

6) Evaluation of the machine learning algorithms

Evaluating a model is a core part of building an effective machine learning project. For this purpose are used the following four metrics to evaluate the algorithms [32]:

- Accuracy is the ratio of number of correct predictions to the total number of input samples.

$$Accuracy = \frac{CorrectPredictions}{PredictionsMade}$$

- Precision is the number of correct positive results divided by the number of positive results predicted by the classifier.

$$Precision = \frac{TP}{TP + FP}$$

- Recall is the number of correct positive results divided by the number of all relevant samples (all samples that should have been identified as positive).

$$Recall = \frac{TP}{TP + FN}$$

- F1-score is used to measure a test's accuracy, the Harmonic Mean between precision and recall. It tells you how precise your classifier is (how many instances it classifies correctly), as well as how robust it is (it does not miss a significant number of instances).

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

The greater the F1-score, the better is the performance of our model.

Table V shows the accuracy of all the trained algorithms evaluated with the test set of the previous step.

Table V
RESULTS OF ALL THE ALGORITHMS IN TERMS OF ACCURACY

Model	Accuracy
Logistic Regression	0.29
KNN	0.82
Decision Trees	0.70
SVM	0.41
CNN	0.41

Tables VI-X show the results of each model evaluation in terms of precision, recall and F1-score. The best results were obtained with the KNN model with the highest average values of 82% for accuracy, 84% for precision, 89% for recall, and 83% for f1-score.

Table VI
RESULTS OF LOGISTIC REGRESSION

Class	Precision	Recall	F1-score
Granite	1.00	1.00	1.00
Granodiorite	0.86	1.00	0.92
Gabbro	0.50	1.00	0.67
Diorite	1.00	0.57	0.73
Average	**0.84**	**0.89**	**0.83**

Table VII
RESULTS OF K-NEAREST NEIGHBORS

Class	Precision	Recall	F1-score
Granite	1.00	1.00	1.00
Granodiorite	0.86	1.00	0.92
Gabbro	0.50	1.00	0.67
Diorite	1.00	0.57	0.73
Average	**0.84**	**0.89**	**0.83**

Table VIII
RESULTS OF DECISION TREES

Class	Precision	Recall	F1-score
Granite	1.00	0.50	0.67
Granodiorite	0.62	0.83	0.71
Gabbro	1.00	1.00	1.00
Diorite	0.67	0.57	0.62
Average	**0.82**	**0.73**	**0.75**

Table IX
RESULTS OF SUPPORT VECTOR MACHINE

Class	Precision	Recall	F1-score
Granite	0.50	0.50	0.50
Granodiorite	0.40	0.67	0.50
Gabbro	0.40	1.00	0.57
Diorite	0.00	0.00	0.00
Average	**0.33**	**0.54**	**0.39**

Table X
RESULTS OF THE CONVOLUTIONAL NEURAL NETWORK

Class	Precision	Recall	F1-score
Granite	0.00	0.00	0.00
Granodiorite	0.42	0.83	0.56
Gabbro	0.40	1.00	0.57
Diorite	0.00	0.00	0.00
Average	**0.20**	**0.46**	**0.28**

B. Results

~~This section cover the creation process of a mobile application that classifies four types of plutonic rocks with the dominant colors extracted from images taken on iOS devices.~~

The KNN model was the model deployed in the iOS mobile application to make the classification on real-time. The Xcode IDE, the Xcode command-line tools, a valid Apple Developer ID, and the CocoaPods dependency manager were the main requeriments to develop it. The application is written mostly in Swift and uses the following two libraries for performing dominant colors extraction and image classification:

- The DominantColor[10] dependency is an open source library written in Swift and created by Indragie Karunaratne. It finds the dominant colors of an image using the CIE LAB color space and the k-means clustering algorithm. In this project an adaptation of this library was implemented exporting the 7 main files from the repository into the application project.
- The TensorFlowLite Swift[11] library is TensorFlow's lightweight solution for Swift developers. It enables low-latency inference of on-device machine learning models with a small binary size and fast performance supporting hardware acceleration. For the application, TensorFlowLiteSwift pod name was added into the project's Podfile, and from command line the library was resolved into the Xcode project by the CocoaPods dependency manager.

Three source codes of iOS applications were taken from the TensorFlow Lite's repository on github as the example guides to built our application for rock classification:

- The image classification[12] is an example application for TensorFlow Lite on iOS. It uses Image classification to continuously classify whatever it sees from the device's back camera, using a quantized MobileNet model.
- The image segmentation[13] iOS sample aplication implements a state-of-art deep learning model for semantic image segmentation, where the goal is to assign semantic labels (e.g. person, dog, cat) to every pixel in the input image.
- The digit classifier[14] iOS sample application uses a digit classifier model built with TensorFlow 2.0 (Keras API), and trained on MNIST to classify handwritten digits drawn in a sketch.

[10]https://github.com/indragiek/DominantColor
[11]https://github.com/tensorflow/tensorflow/tree/master/tensorflow/lite/swift
[12]http://bit.ly/3oSuxQK
[13]http://bit.ly/3nJM6RD
[14]http://bit.ly/3srLkMx

Shown below is the application process (see Fig. 4). The source code of the application is available online[15].

C. Discussion

Five machine learning algorithms were trained with just the four dominant colors extracted from rock images. The best algorithm in this experiment was K-Nearest Neighbors with an accuracy of 82%. Its values for precision, recall, and F1 score were 84%, 89%, and 83%. After obtain the best suited algorithm for rock classification with dominant colors, an iOS mobile application was developed to classify rock images into four classes of plutonic rocks (granite, granodioorite, gabbro and diorite). This application also allows the user to visualize the dominant colors and the average color in new images and detect changes in rock colors.

IV. CONCLUSIONS AND FUTURE WORK

REFERENCES

[1] M. Williams, *Igneous rocks: How are they formed?* English, Access date: 12/16/2020, Universe Today, Dec. 2015. [Online]. Available: https://www.universetoday.com/82009/how-are-igneous-rocks-formed/.

[2] National Geographic Society, *Igneous rocks*, English, ENCYCLOPEDIC ENTRY, Access date: 12/16/2020, National Geographic, Oct. 2019. [Online]. Available: https://www.nationalgeographic.org/encyclopedia/igneous-rocks/.

[3] R. Gillaspy, *Volcanic vs plutonic igneous rocks: Definition and differences*, English, Access date: 12/16/2020, Study.com, Nov. 2013. [Online]. Available: https://study.com/academy/lesson/volcanic-vs-plutonic-igneous-rocks-definition-and-differences.html.

[4] Natural Resources Conservation Service, "Part 631: Geology," in *National Engineering Handbook*, 210-VI, Access date: 12/16/2020, 2012, ch. 4, p. 7. [Online]. Available: https://directives.sc.egov.usda.gov/OpenNonWebContent.aspx?content=31848.wba.

[5] A. Caté, L. Perozzi, E. Gloaguen, and M. Blouin, "Machine learning as a tool for geologists," *The Leading Edge*, vol. 36, pp. 215–219, Mar. 2017. DOI: 10.1190/tle36030215.1.

[6] f. y. n., *RACEFEN Glosario de geología*, Spanish, Access date: 01/04/2021, España: Real academia de ciencias exactas, físicas y naturales. [Online]. Available: http://www.ugr.es/~agcasco/personal/rac_geologia/0_rac.htm.

[7] Encyclopædia Britannica, *Diorite*, English, Access date: 12/15/2020, Encyclopædia Britannica, Jan. 2009. [Online]. Available: https://www.britannica.com/science/diorite.

[8] P. Harrington, *Machine Learning in Action*. MANNING PUBN, Apr. 1, 2012, 384 pp., ISBN: 1617290181. [Online]. Available: https://www.ebook.de/de/product/15619827/peter_harrington_machine_learning_in_action.html.

[15]https://github.com/sarah-hs/Rock-Classifer-iOS

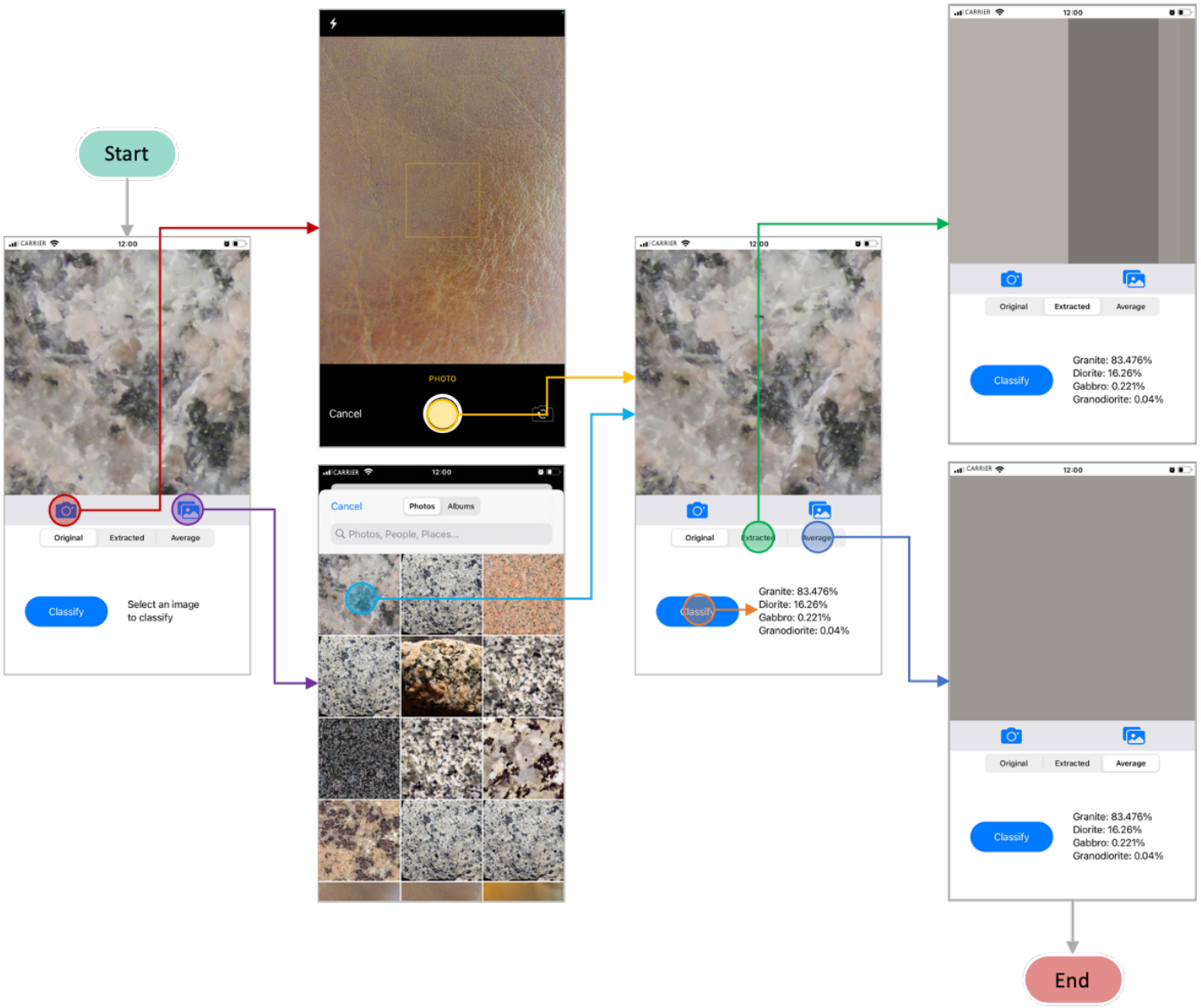

Figure 4. Application workflow

[9] A. Géron, *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly UK Ltd., Oct. 1, 2019, 819 pp., ISBN: 1492032646. [Online]. Available: https://www.ebook.de/de/product/33315532/aurelien_geron_hands_on_machine_learning_with_scikit_learn_keras_and_tensorflow.html.

[10] Scikit-learn, *Nearest Neighbors*, English, Access date: 12/15/2020, Scikit-learn. [Online]. Available: https://scikit-learn.org/stable/modules/neighbors.html#id5.

[11] C. Reinders, H. Ackermann, M. Y. Yang, and B. Rosenhahn, "Chapter 4 - Learning Convolutional Neural Networks for Object Detection with Very Little Training Data," in *Multimodal Scene Understanding*, M. Y. Yang, B. Rosenhahn, and V. Murino, Eds., Academic Press, 2019, pp. 65–100, ISBN: 978-0-12-817358-9. DOI: https://doi.org/10.1016/B978-0-12-817358-9.00010-X. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B978012817358900010X.

[12] L. Tan, "Chapter 17 - Code Comment Analysis for Improving Software Quality," in *The Art and Science of Analyzing Software Data*, C. Bird, T. Menzies, and T. Zimmermann, Eds., This chapter contains figures, tables, and text copied from the author's PhD dissertation and the papers that the author of this chapter coauthored [[3], [1], [35], [7]]. Sections 17.2.3, 17.4.3, 17.5, and 17.6 are new, and the other sections are augmented, reorganized, and improved., Boston: Morgan Kaufmann, 2015, pp. 493–517, ISBN: 978-0-12-411519-4. DOI: https://doi.org/10.1016/B978-0-12-411519-4.00017-3. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780124115194000173.

[13] S. Wallace, *Perl Graphics Programming: Creating Svg, SWF (Flash), JPEG and PNG Files with Perl.*

OREILLY MEDIA, Dec. 1, 2002, 478 pp., ISBN: 059600219X. [Online]. Available: https://www.ebook.de/de/product/3251138/shawn_wallace_perl_graphics_programming_creating_svg_swf_flash_jpeg_and_png_files_with_perl.html.

[14] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, no. 85, pp. 2825–2830, 2011. [Online]. Available: http://jmlr.org/papers/v12/pedregosa11a.html.

[15] Michelle, *What exactly is TensorFlow?* English, Access date: 12/15/2020, Medium, Oct. 2018. [Online]. Available: https://medium.com/datadriveninvestor/what-exactly-is-tensorflow-80a90162d5f1.

[16] TensorFlow, *TensorFlow Lite guide*, English, comp. software, Access date: 12/15/2020, TensorFlow, Mar. 2020. [Online]. Available: https://www.tensorflow.org/lite/guide.

[17] Apple, *Swift. the powerful programming language that is also easy to learn.* English, Tech. Rep., Access date: 12/15/2020, Apple. [Online]. Available: https://developer.apple.com/swift/.

[18] C. Bohon, *Apple's Swift programming language*, English, Access date:12/15/2020, TechRepublic, Sep. 2020. [Online]. Available: https://www.techrepublic.com/article/apples-swift-programming-language-the-smart-persons-guide/.

[19] Apple Insider, *Xcode*, English, Tech. Rep., Access date: 12/15/2020, Apple insider, Dec. 2020. [Online]. Available: https://appleinsider.com/inside/xcode.

[20] Apple, *Introducing Xcode 12*, English, tech report, Access date: 12715/2020, Apple. [Online]. Available: https://developer.apple.com/xcode/.

[21] CocoaPods, *What is CocoaPods*, English, comp. software, Access date: 01/15/2021, CocoaPods. [Online]. Available: https://guides.cocoapods.org/using/getting-started.html.

[22] D. J. Lary, G. K. Zewdie, X. Liu, D. Wu, E. Levetin, R. J. Allee, N. Malakar, A. Walker, H. Mussa, A. Mannino, and D. Aurin, "Machine Learning Applications for Earth Observation," in *Earth Observation Open Science and Innovation*, C. Mathieu Pierre-Philippeand Aubrecht, Ed., Cham: Springer International Publishing, 2018, pp. 165–218, ISBN: 978-3-319-65633-5. DOI: 10.1007/978-3-319-65633-5_8. [Online]. Available: https://doi.org/10.1007/978-3-319-65633-5_8.

[23] X. Ran, L. Xue, Y. Zhang, Z. Liu, X. Sang, and J. He, "Rock Classification from Field Image Patches Analyzed Using a Deep Convolutional Neural Network," *Mathematics*, vol. 7, no. 8, p. 755, 2019. DOI: 10.3390/math7080755.

[24] G. Cheng and W. Guo, "Rock images classification by using deep convolution neural network," *Journal of Physics: Conference Series*, vol. 887, p. 012 089, 2017. DOI: 10.1088/1742-6596/887/1/012089.

[25] G. Fan, F. Chen, D. Chen, and Y. Dong, "Recognizing Multiple Types of Rocks Quickly and Accurately Based on Lightweight CNNs Model," *IEEE Access*, vol. 8, pp. 55 269–55 278, 2020. DOI: 10.1109/access.2020.2982017.

[26] G. Fan, F. Chen, D. Chen, Y. Li, and Y. Dong, "A Deep Learning Model for Quick and Accurate Rock Recognition with Smartphones," *Mobile Information Systems*, vol. 2020, pp. 1–14, 2020. DOI: 10.1155/2020/7462524.

[27] Y. Zhang, M. Li, S. Han, Q. Ren, and J. Shi, "Intelligent Identification for Rock-Mineral Microscopic Images Using Ensemble Machine Learning Algorithms," *Sensors*, vol. 19, no. 18, p. 3914, 2019. DOI: 10.3390/s19183914.

[28] J. Maitre, K. Bouchard, and L. P. Bédard, "Mineral grains recognition using computer vision and machine learning," *Computers & Geosciences*, vol. 130, pp. 84–93, 2019. DOI: 10.1016/j.cageo.2019.05.009.

[29] P. Dangeti, *Statistics for Machine Learning*. Packt Publishing, Jul. 21, 2017, 442 pp., ISBN: 1788295757. [Online]. Available: https://www.ebook.de/de/product/29829428/pratap_dangeti_statistics_for_machine_learning.html.

[30] Scikit learn, *Support Vector Classification*, English, Access date: 01/15/2021, Scikit-learn. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html?highlight=svc#sklearn.svm.SVC.

[31] G. Bonaccorso, A. Fandango, and R. Shanmugamani, *Python*. Packt Publishing, Dec. 19, 2018, 764 pp., ISBN: 1789957214. [Online]. Available: https://www.ebook.de/de/product/35139551/giuseppe_bonaccorso_armando_fandango_rajalingappaa_shanmugamani_python.html.

[32] A. Mishra, *Metrics to evaluate your machine learning algorithm*, English, Access date: 12/22/2020, Medium, Feb. 2018. [Online]. Available: https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234.

Table I
COMPARISON TABLE OF THE ARTICLES DESCRIBED IN THE RELATED WORK

Author	Year	Dataset	Algorithm of extraction	Extracted features	Dataset for training	Best trained model	Results
Zhang et al. [27]	2019	481 images of four minerals (K-feldspar, perthite, plagioclase, and quartz)	Inception-v3 CNN model	High-level features (such as chromatic aberration and texture)	90% of extracted features	Stacking model (LR, SVM, and MLP)	Model accuracy: 90.9% -
Maitre et al. [28]	2019	3,192 sub-images of the view taken to a surface with 27 mineral grains species	Simple Linear Iterative Clustering (SLIC) algorithm	Color intensity and Peak intensity of computed histograms from superpixels	70% of extracted features	Random Forest (RF)	Model accuracy: 89% -
Ran et al. [23]	2019	14,589 patches from 2,290 images of six rock types (granite, limestone, conglomerate, sandstone, shale, mylonite)	-	-	60% of the patches	3-layer structure CNN	Model accuracy: 97.76% -
Cheng and Guo [24]	2017	4,200 images of three feldspar sandstones rock types (coarse, medium granular, and fine)	-	-	75% of the images	6-layer structure CNN	Model accuracy: 98.5% -
Fan et al. [25]	2020	3,208 images of 28 rock lithology categories	-	-	80% of the images	SqueezeNet and MobileNet	Android application accuracies: 94.55% and 93.27% File sizes: 9.2 MB and 17.6 MB Execution times: 736 to 366 and 1,218 to 523 millisecond
Fan et al. [26]	2020	3,795 images of 30 different rock types	-	-	80% of the images	ShuffleNet	Android application accuracy: 97.65% Occupation space on PC: 18.2M Classification time: 786 millisecond
E. Vázquez and H. Alférez []	2021	846 patches from 71 images of six plutonic rock types (diorite, gabbro, granite, granodiorite, monzodiorite, and tonalite)	-	-	50% of the patches	4-layer structure CNN	Model accuracy: 95% Android application accuracy results: 70% for gabbro, 28.5% for diorite, 100% for granodiorite, and 28.5% for granite