# Satellite - Image Classification

**NOVEMBER 23**

**Authored by:**

**Smit Doshi - 1915123**

## TABLE OF CONTENTS

## Abstract

In this project, we were asked to experiment with a real world dataset, and to explore how Deep Learning Network and Machine Learning Algorithms that can be used to find the patterns in data. After performing the required tasks on a dataset of our choice, herein lies our final report.

Keywords:

- Machine Learning
- Artificial Intelligence
- Computer Vision
- Image Processing
- Deep Learning
- Satellite Image and
- Remote Sensing

## Acknowledgements

Annotated Datasets for Remote Sensing (RS) Image Interpretation.
The interpretation of RS images has been playing an increasingly important role in a large diversity of applications, and thus, has attracted remarkable research attentions. Consequently, various datasets have been built to advance the development of interpretation algorithms for RS images. Covering literature published over the past decade, we perform a systematic review of the existing RS image datasets concerning the current mainstream of RS image interpretation tasks, including scene classification, object detection, semantic segmentation and change detection.

## Overview

The past years have witnessed great progress on remote sensing (RS) image interpretation and its wide applications. With RS images becoming more accessible than ever before, there is an increasing demand for the automatic interpretation of these images. In this context, the benchmark datasets serve as essential prerequisites for developing and testing intelligent interpretation algorithms. After reviewing existing benchmark datasets in the research community of RS image interpretation, this article discusses the problem of how to efficiently prepare a suitable benchmark dataset for RS image interpretation. Specifically, we first analyse the current challenges of developing intelligent algorithms for RS image interpretation with bibliometric investigations. We then present the general guidance on creating benchmark datasets in efficient manners. Following the presented guidance, we also provide an example on building RS image dataset, i.e., Million-AID, a new large-scale benchmark dataset containing a million instances for RS image scene classification. Several challenges and perspectives in RS image annotation are finally discussed to facilitate the research in benchmark dataset construction. We do hope this paper will provide the RS community an overall perspective on constructing large-scale and practical image datasets for further research, especially data-driven ones.

## Introduction

Deep Learning has become the principle driver of numerous new applications and it's an ideal opportunity to truly take a gander at why this is the situation. Deep learning allows computational models to learn by gathering knowledge from experience. Complex concepts can be learnt by deep learning approach due to its hierarchical conceptualization.

In artificial intelligence neural system, deep learning concept considered as a subset, is gaining a lot of data. Likewise how individuals gain as an element of fact, deep learning estimation would work out an undertaking more than once, every time adjusting it a bit to increase the more possible result.

We allude to deep learning in the fact of that the neural networks have various tiers that entitle the learning. If any issue that expects "thought" to make sense of is a problem deep learning can determine out how to illuminate. Deep learning has significantly benefitted for the state-of-the-art in many recurring domains in the modern world. Deep Learning has an immense influence on fields such as computer vision, object detection, object recognition and speech recognition. In this report, a detailed description and introduction to the concept of deep learning is provided.

## *Problems and Issues in Deep Learning*

Before we get started, we must know about how to pick a good Deep Learning Algorithm for the given dataset. To intelligently pick an algorithm to use for Classification task, we must consider the following factors:

1. Size of the training data
2. Accuracy and/or Interpretability of the output
3. Speed or Training time
4. Number of features

Only after considering all these factors can we pick a Classification algorithm that works for the dataset we are working on.

## **Satellite-Image Classification**

Classification and extraction of cover types from satellite/aerial imagery have useful applications in many different areas including defence, mapping, agriculture, disaster response, law enforcement, and environmental monitoring and etc. These applications require the manual identification of objects and facilities in the imagery. Because the geographic expanses to be covered are great and the analysts available to conduct the searches are few, analysing satellite imagery can be a very daunting task especially with multi/hyperspectral imagery and hence, Automation is required.

Typical machine learning methods require a prior feature selection from the imagery and then use a classifier to detect different cover types in the image and hence, traditional object detection and classification algorithms are too inaccurate and unreliable to solve the problem.

Deep learning is a family of machine learning algorithms that have shown promise for the automation of such tasks. It has achieved success in image understanding by means of Convolutional Neural Networks.

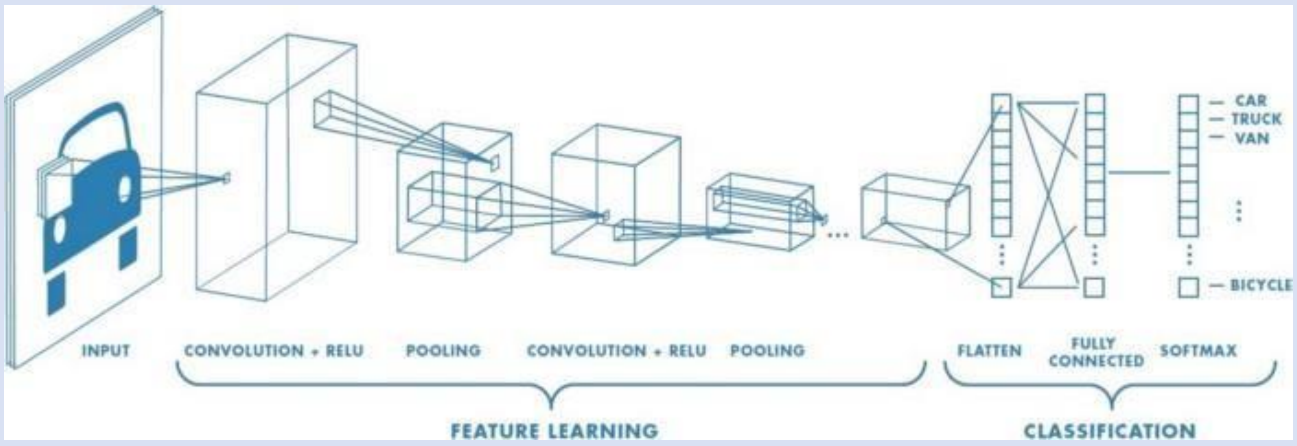
Convolutional Neural Networks (CNNs) circumvent the issue of prior feature selection by incorporating the feature extraction and classification into a single framework. This has made CNNs very attractive as feature selection is delegated to the network which optimally picks out the most relevant features for the classification task.

We describe a Deep Learning System for classifying images from the Satellite – Image Classification Dataset into 4 different classes.

The system consists of an ensemble of convolutional neural networks and additional neural networks that integrate satellite metadata with image features. It is implemented in Python using the Keras and Tensor Flow Deep Learning libraries.

## Convolutional Neural Network

Convolutional neural system is the principal deep neural systems model that has been effectively prepared in the field of PC vision and has accomplished extensive application accomplishments. The convolutional neural systems layer is as yet a vital piece of the deep neural system and broadly applied in the characterization and acknowledgment of the designs and video handling. All the more explicitly, look into around Convolutional Neural Systems (CNN) has demonstrated to be viable in tackling picture grouping and division issues .CNNs empower examination and extraction of the spatial data in pictures, working from fine grained subtleties into more significant level structures. The transient measurement can be added to these systems by adding a third hub to the convolutional bits.



*CNN (but instead of putting car image as an input, we will use a satellite image and instead of the Car, Truck, Van, Bicycle as classification, we will have desert, cloudy, green_area and water as our output)*

## Methodology

### *IMPORT:*

Initially we started by importing all the required Deep Learning libraries required by the project.

i. numpy – For Array Handling

ii. pandas – For Data Manipulation and Analysis

iii. os – Provides functions to interact with operating system

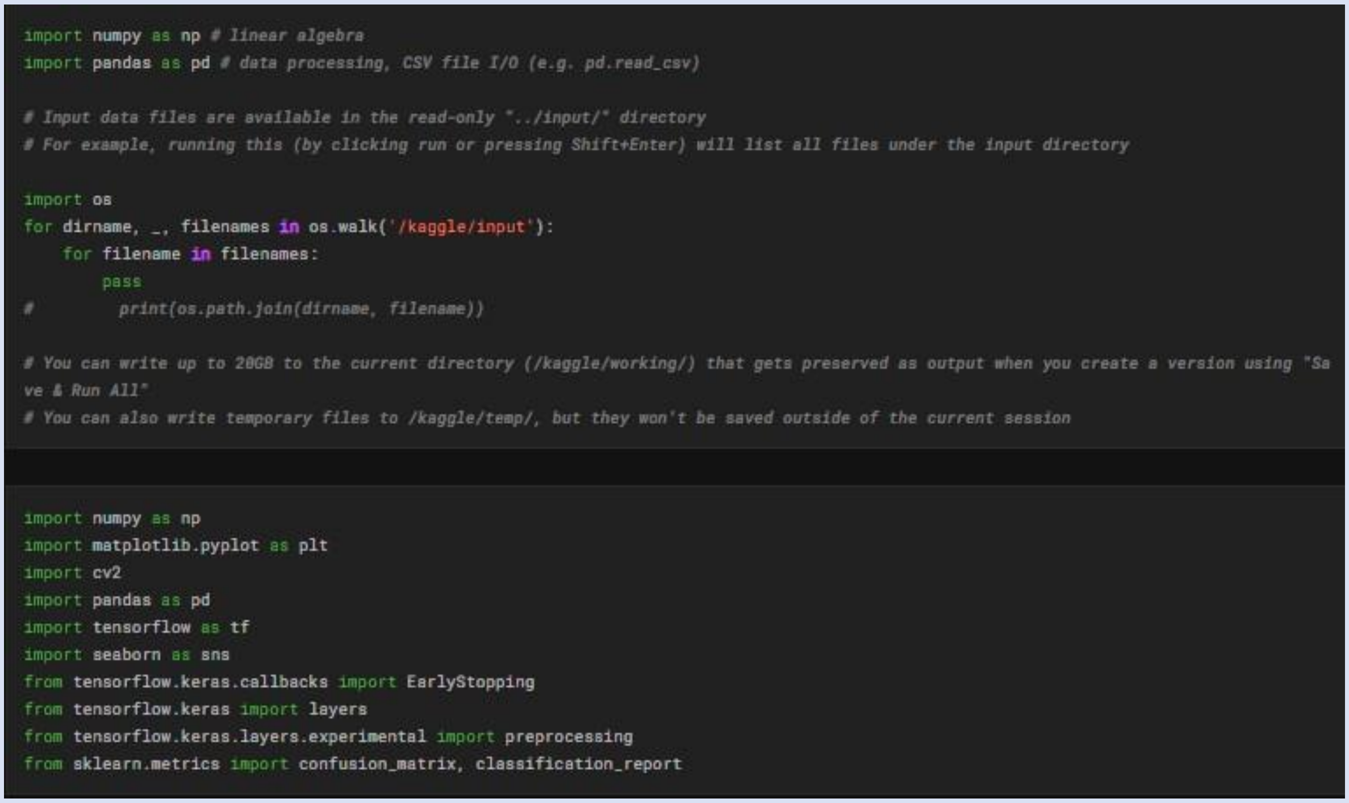
iv. matplotlib – For Plotting graphs

v. keras – To import Dataset

vi. tenserflow – For fast Numerical Computing

vii. sklearn – Provides tools for ML and Statistical Modelling

viii. seaborn - For Drawing Informational Statistical Graphics



```python
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        pass
#        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Sa
ve & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

```python
import numpy as np
import matplotlib.pyplot as plt
import cv2
import pandas as pd
import tensorflow as tf
import seaborn as sns
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras import layers
from tensorflow.keras.layers.experimental import preprocessing
from sklearn.metrics import confusion_matrix, classification_report
```

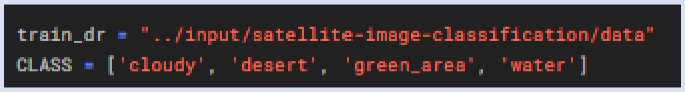
## Dataset:

The dataset used is a sample of images taken from the satellite and has the following feature vectors:

1. desert
2. cloudy
3. green_area
4. water

In the dataset, there are 5631 images with jpg format. This classification task is important because the expert system, when correctly generalized, can tell if the event or a disaster is taking place in that particular region and the Strategic Command Officers can take a look at that case in more detail.

Then we assigned a variable train_dr to the dataset.



```
train_dr = "../input/satellite-image-classification/data"
CLASS = ['cloudy', 'desert', 'green_area', 'water']
```

## Classification model

- To perform a patch-based classification of different land cover types we constructed a Convolutional Neural Network which took in the 256 x 256 x 3 image and outputted the probability of different class assignments.

- The CNN consisted of 6 hidden layers with 3 convolutional and 3 max-pooling layers for feature extraction.

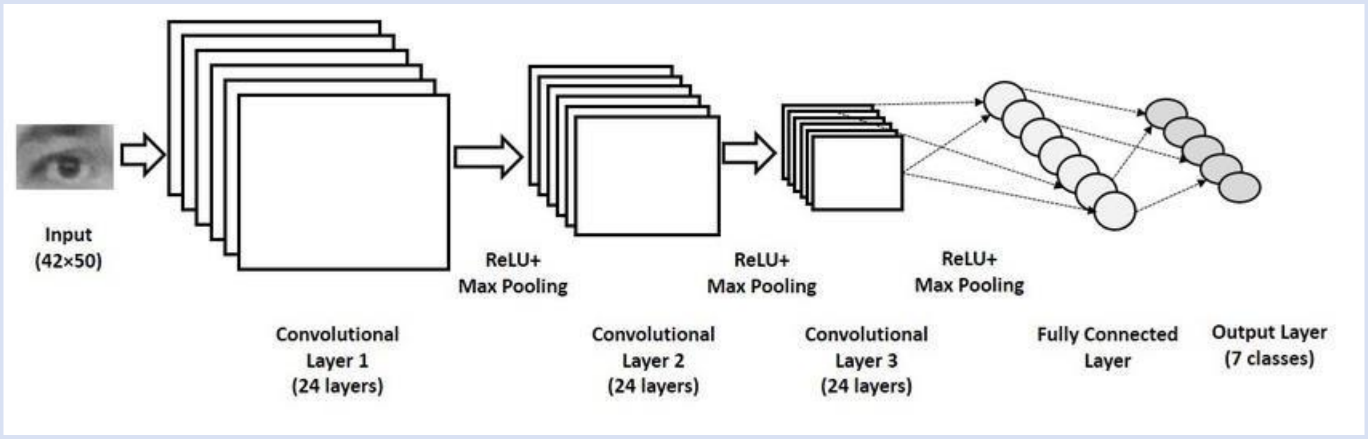
- For the convolutional layers, I used ReLU activation function to speed up training (as the gradient of ReLU is easy and fast to compute during backpropagation). The output layer of my network contained 10 neurons where each neuron represented the probability of class assignment for the input image.

- Adding multiple convolutional layers and pooling layers, the image will be processed for feature extraction.

- The convolutional layers were followed by a fully-connected layer for classification and a dropout layer to adjust for overfitting.

- Flattening is converting the data into a 1-dimensional array for inputting it to the next layer. We flatten the output of the convolutional layers to create a single long feature vector. And it is connected to the final classification model, which is called a fully-connected layer. In other words, we put all the pixel data in one line and make connections with the final layer.

- And there will be fully connected layers heading to the layer for softmax (for a multi-class case) function.



i. **Hidden layer:** are the secret sauce of your network. They allow you to model complex data thanks to their nodes/neurons. They are "hidden" because the true values of their nodes are unknown in the training dataset. In fact, we only know the input and output.

ii. **Max-Pooling:** is a sample-based discretization process. The objective is to down-sample an input representation (image, hidden-layer output matrix, etc.), reducing its dimensionality and allowing for assumptions to be made about features contained in the sub-regions binned. This is done to in part to help over-fitting by providing an abstracted form of the representation. As well, it reduces the computational cost by reducing the number of parameters to learn and provides basic translation invariance to the internal representation. Max pooling is done by applying a max filter to (usually) non-overlapping sub-regions of the initial representation.

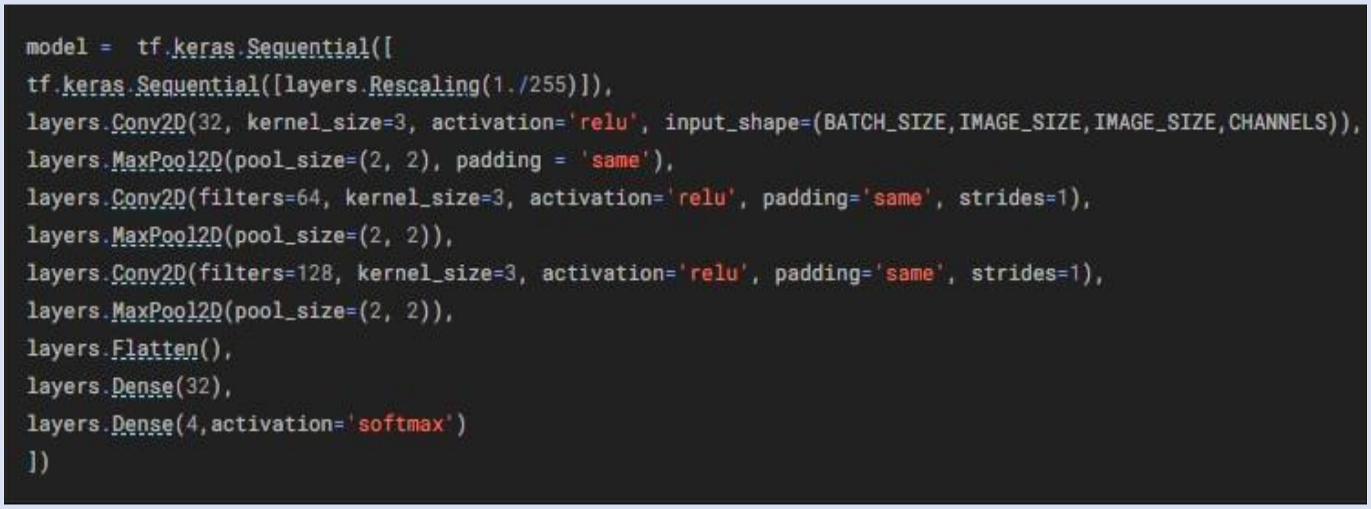
iii. **The rectified linear activation function (ReLU):** is a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero. It has become the default activation function for many types of neural networks because a model that uses it is easier to train and often achieves better performance.

iv. **Feature extraction:** is a process of dimensionality reduction by which an initial set of raw data is reduced to more manageable groups for processing. A characteristic of these large data sets is a large number of variables that require a lot of computing resources to process. Feature extraction is the name for methods that select and /or combine variables into features, effectively reducing the amount of data that must be processed, while still accurately and completely describing the original data set.

v. **Fully Connected Layer:** is simply feed forward neural networks. Fully Connected Layers form the last few layers in the network. The input to the fully connected layer is the output from the final Pooling or Convolutional Layer, which is flattened and then fed into the fully connected layer.

vi. **The Dropout layer:** randomly sets input units to 0 with a frequency of rate at each step during training time, which helps prevent overfitting. Inputs not set to 0 are scaled up by 1/(1 - rate) such that the sum over all inputs is unchanged. Note that the Dropout layer only applies when training is set to True such that no values are dropped during inference. When using model.fit, training will be appropriately set to True automatically, and in other contexts, you can set the kwarg explicitly to True when calling the layer.

vii. **Softmax:** The most common use of the softmax function in applied machine learning is in its use as an activation function in a neural network model. Specifically, the network is configured to output N values, one for each class in the classification task, and the softmax function is used to normalize the outputs, converting them from weighted sum values into probabilities that sum to one. Each value in the output of the softmax function is interpreted as the probability of membership for each class.

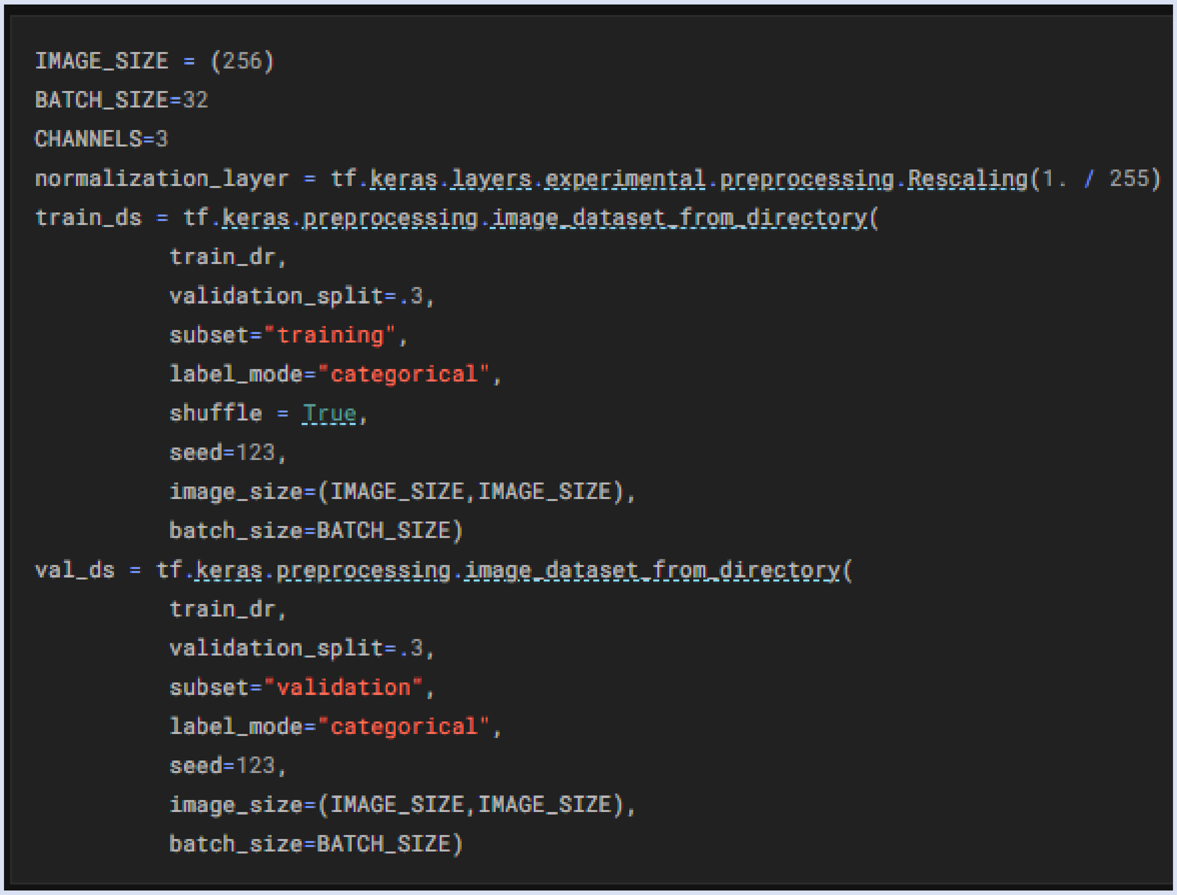


```
model =  tf.keras.Sequential([
tf.keras.Sequential([layers.Rescaling(1./255)]),
layers.Conv2D(32, kernel_size=3, activation='relu', input_shape=(BATCH_SIZE,IMAGE_SIZE,IMAGE_SIZE,CHANNELS)),
layers.MaxPool2D(pool_size=(2, 2), padding = 'same'),
layers.Conv2D(filters=64, kernel_size=3, activation='relu', padding='same', strides=1),
layers.MaxPool2D(pool_size=(2, 2)),
layers.Conv2D(filters=128, kernel_size=3, activation='relu', padding='same', strides=1),
layers.MaxPool2D(pool_size=(2, 2)),
layers.Flatten(),
layers.Dense(32),
layers.Dense(4,activation='softmax')
])
```

## *Batch Normalization*

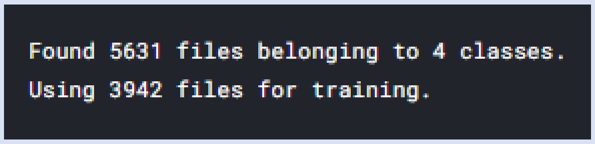
Batch normalization is a technique for training very deep neural networks that standardizes the inputs to a layer for each mini-batch. This has the effect of stabilizing the learning process and dramatically reducing the number of training epochs required to train deep networks.

Here, in the given Dataset, Due to a large number of spectral bands, we had to keep the batch size small as we would frequently run out of memory. So, we finally had to settle on a batch size of 32 images.

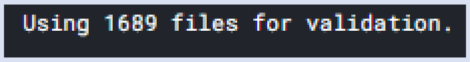


```
IMAGE_SIZE = (256)
BATCH_SIZE=32
CHANNELS=3
normalization_layer = tf.keras.layers.experimental.preprocessing.Rescaling(1. / 255)
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
          train_dr,
          validation_split=.3,
          subset="training",
          label_mode="categorical",
          shuffle = True,
          seed=123,
          image_size=(IMAGE_SIZE,IMAGE_SIZE),
          batch_size=BATCH_SIZE)
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
          train_dr,
          validation_split=.3,
          subset="validation",
          label_mode="categorical",
          seed=123,
          image_size=(IMAGE_SIZE,IMAGE_SIZE),
          batch_size=BATCH_SIZE)
```

The dataset was split 2:1 for training and validation respectively.



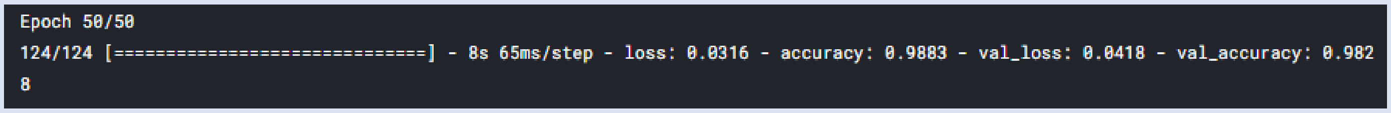
```
Found 5631 files belonging to 4 classes.
Using 3942 files for training.
```



```
Using 1689 files for validation.
```

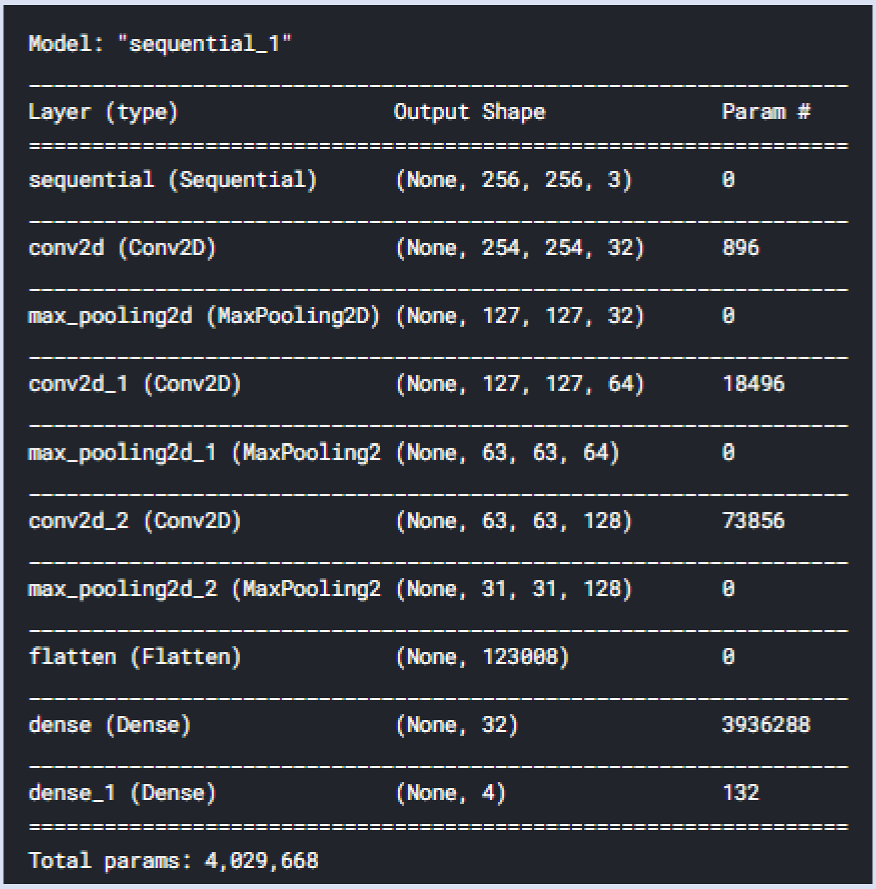
## *Result:*

After executing the above code, we were able to obtain a final training and validation accuracy of 98.83% and 98.28%, with a training and validation loss of 0.0316 and 0.0418, respectively. Since we were able to obtain relatively high accuracy in just 50 epochs, we did not feel the need to train the network any further. You can see the results in the image below.



```
Epoch 50/50
124/124 [==============================] - 8s 65ms/step - loss: 0.0316 - accuracy: 0.9883 - val_loss: 0.0418 - val_accuracy: 0.982
8
```

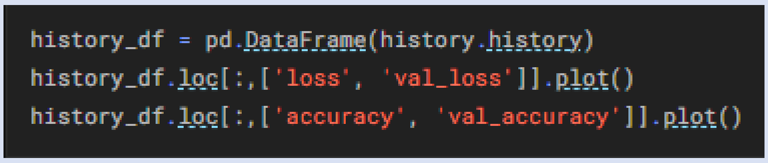
## Summary:



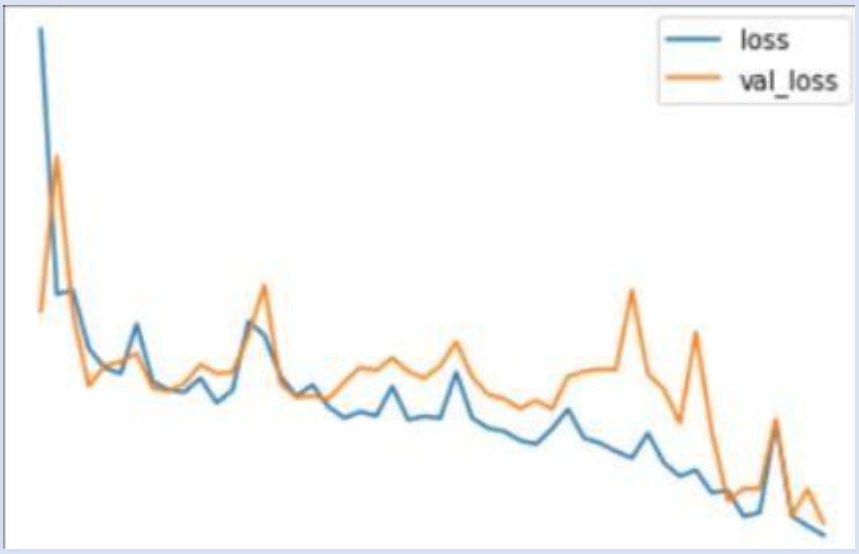
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
sequential (Sequential)      (None, 256, 256, 3)       0
_________________________________________________________________
conv2d (Conv2D)              (None, 254, 254, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 127, 127, 32)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 127, 127, 64)      18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 63, 63, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 63, 63, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 31, 31, 128)       0
_________________________________________________________________
flatten (Flatten)            (None, 123008)            0
_________________________________________________________________
dense (Dense)                (None, 32)                3936288
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 132
=================================================================
Total params: 4,029,668
```

## Graphs:

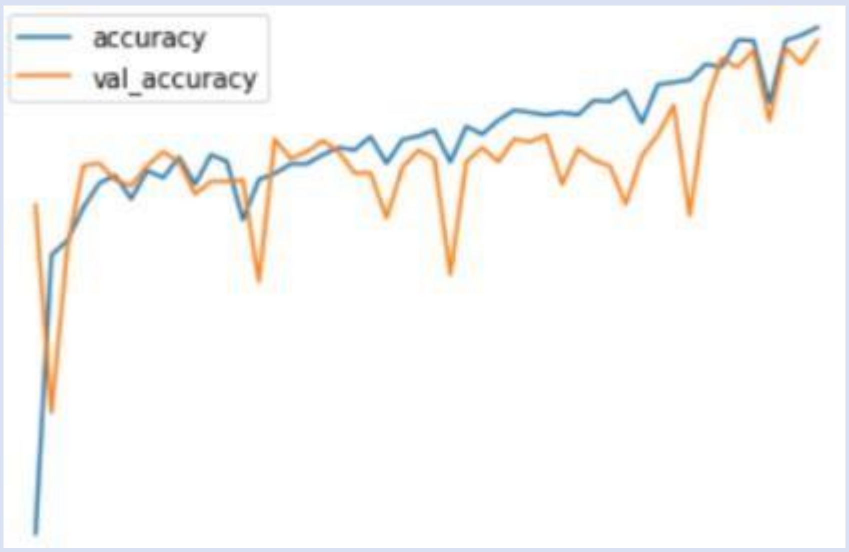
Now that we know the summary, we can now plot the graphs of both, the Loss as well as the Accuracy using the code below.



```
history_df = pd.DataFrame(history.history)
history_df.loc[:,['loss', 'val_loss']].plot()
history_df.loc[:,['accuracy', 'val_accuracy']].plot()
```



*(Training loss vs Validation loss)*



*(Training accuracy vs Validation accuracy)*

## Conclusion:

- We conclude that the dataset is not a complete space, and there are still other feature vectors missing from it. What we were attempting to generalize is a subspace of the actual input space, where the other dimensions are not known. In the future, if similar studies are conducted to generate the dataset used in this report, more feature vectors need to be calculated so that the classifiers can form a better idea of the problem at hand.

- Regions in the satellite images are not understandable every time by eye site. So Satellite image classification is very helpful to gather information about a particular region. Similar structures of different class of images and environmental variations makes the work difficult. Images used for testing is checked to determine the accuracy. In this work, we developed a system for satellite image classification with an accuracy of 98.28%.

- Remote sensing, distant perceiving, image classification, and categorization are considered as challenging research areas in the field of computer vision. The recent focus of research in this domain is to explore the novel deep learning model that can enhance the classification accuracy.

## References

❖ https://www.hindawi.com/journals/mpe/2021/5843816/

❖ https://machinelearningmastery.com

❖ http://statweb.stanford.edu/~tibs/ElemStatLearn/.

❖ Lu, D., & Weng, Q. (2007). A survey of image classification methods and techniques for improving classification performance. International journal of Remote sensing, 28(5), 823-870