# Image Classification with Hand Signs

Theoretical Report

Jessica Christine Erasmus
November 2023

## Table of Contents

## Introduction

Have you ever considered the amazing potential that results from the fusion of technology and human gestures? In the fascinating field of hand sign image classification, state-of-the-art algorithms enable machines to interpret and comprehend the complex language expressed by hand gestures. This field has a lot of promise for a variety of applications, from improving gesture-based human-computer interaction to helping those with hearing impairments.

Hand signals are a form of universal language that use motions to express message. With picture classification algorithms, it is now possible to automatically analyse and comprehend these gestures, creating new opportunities for technology, accessibility, and communication. Accurate hand sign recognition has the potential to change a lot of things about how we engage with technology and each other, from helping those with hearing impairments to improving the usefulness of interactive devices.

## Context

The process of teaching a computer to identify and classify hand signs based on patterns and features extracted from images is referred to as image classification. Many hand gestures exist, and its meaning may differ worldwide. Therefore, for this analysis, a simple dataset of eighteen classifications is used as a beginning for future endeavours.

To enable the model to learn from examples and become more generalised, this entails training a machine learning model with a dataset of hand sign images. The model is a useful tool for real-time applications because it can reliably classify new, unseen hand signs as it gains proficiency.

## Requirements

The task involves developing and improving an image classification model with the use of transfer learning and evaluating the model's performance. This model will be implemented on a dataset that is suitable for performing image classification as well as contains more than 100 000 images. The analysis must be performed in Jupyter Notebook using the dataset as an open data set and python as the main language.

## Image Classification

Within computer vision, image classification is an active and growing field with continuous advancements and applications. Teaching a computer to identify and classify objects or scenes within images is a basic task in computer vision. The objective is to create models and algorithms that can automatically classify or label an input image according to its visual content. It also covers a few more essential viewpoints and components to consider.

## Convolutional Neural Networks

CNNs have proven to be extraordinarily successful at classifying images. The purpose of these neural networks is to extract feature spatial hierarchies automatically and adaptively from images.

CNNs use convolutional, pooling, and fully connected layers as their building blocks to identify complex patterns in images.

## Transfer Learning

In image classification, transfer learning has become a popular technique. Transfer learning is utilizing a pre-trained model on a sizable dataset (like ImageNet) and optimizing it for a particular classification task instead of starting from nothing when training a new model. When there is little labelled data for a given domain, this method works well.

## The Process

A machine is taught to identify and classify objects or scenes inside photographs through a series of critical phases in the image classification process. The quality and variety of the training data, the model architecture selected, and the efficiency of the training procedure are all essential variables in the performance of image classification models.

### Data collection

A labelled dataset of pictures that correspond to the different classes or categories are collected to help the model identify these classes. The matching class label for each image in the collection needs to be specified. The data used in this analysis was collected from Kaggle and contain images classified based on the shown hand sign.

#### Dataset

Typically, hand gesture recognition datasets are used for classification tasks, where a model is trained to identify and classify various hand motions using the provided photos. Convolutional Neural Networks (CNNs) are a popular model because they are good at extracting spatial characteristics from images.

126,763 training photos from the HaGRID (Hand Gesture Recognition Image Dataset) have been altered for image classification purposes rather than object recognition. In this version, sample photos devoid of motions are contained in a separate folder. About 3 GB of 716 GB make up the current dataset of the original dataset.

#### Dataset Link

https://www.kaggle.com/datasets/innominate817/hagrid-classification-512p-127k

#### Dataset Authors

- Alexander Kapitanov
- Andrey Makhlyarchuk
- Karina Kvanchiani

#### Dataset References

- GitHub
- Kaggle Datasets Page

#### Main Dataset Variables

- X – image
- Y – category

#### Categories

- 0: call
- 1: dislike
- 2: fist
- 3: four
- 4: like
- 5: mute
- 6: ok
- 7: one

- 8: palm
- 9: peace
- 10: peace_inverted
- 11: rock
- 12: stop
- 13: stop_inverted
- 14: three
- 15: three2
- 16: two_up
- 17: two_up_inverted

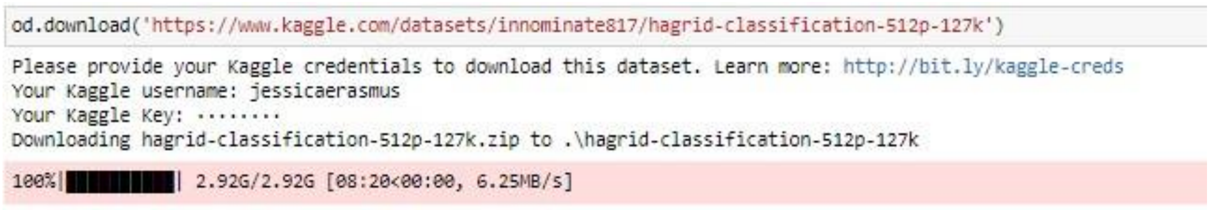
Loading Data

***[Note however that this project has the original dataset, so this section is irrelevant in this case]***

Due to the size restriction on some platforms, the dataset was not included in some projects. Hence, a separate file was created to download the dataset remotely using the Kaggle dataset URL. In fact, downloading the dataset this way took surprisingly faster that manually using the link and unzipping the dataset all together. However, this is stated with storage and processor issues in consideration as well.

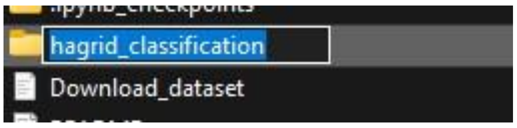
Furthermore, the user must log into their Kaggle account and create an API token in the account settings. A Json file is then downloaded, pertaining the account user's credentials. These credentials are prompted before the dataset can download. Furthermore, click on the following link for more information and better instruction.

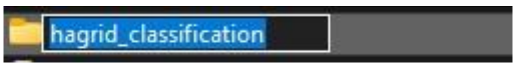
http://bit.ly/kaggle-creds



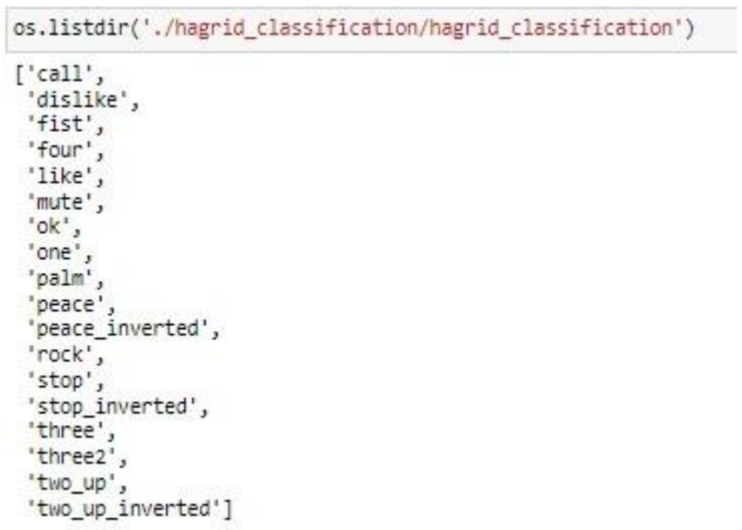
```
od.download('https://www.kaggle.com/datasets/innominate817/hagrid-classification-512p-127k')

Please provide your Kaggle credentials to download this dataset. Learn more: http://bit.ly/kaggle-creds
Your Kaggle username: jessicaerasmus
Your Kaggle Key: ········
Downloading hagrid-classification-512p-127k.zip to .\hagrid-classification-512p-127k

100%|██████████| 2.92G/2.92G [08:20<00:00, 6.25MB/s]
```

Once the dataset has completed downloading, all files and folders were checked to ensure the was no missing information. All hyphenated folders were manually renamed since the OS module gives error on hyphenated naming constructions. The main folder and internal folder need to be renamed.

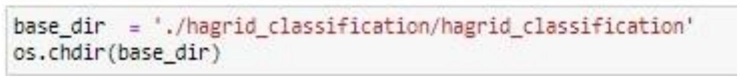




The main analysis notebook was then opened. After importing the necessary libraries, the dataset is located. The base directory is redirected to this location.


```
os.listdir('./hagrid_classification/hagrid_classification')
```

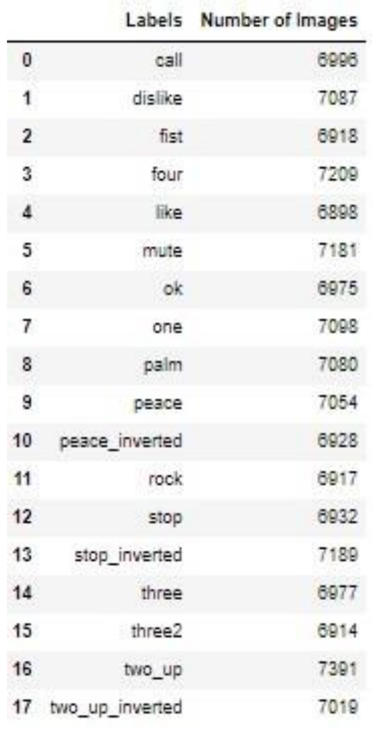
```
['call',
 'dislike',
 'fist',
 'four',
 'like',
 'mute',
 'ok',
 'one',
 'palm',
 'peace',
 'peace_inverted',
 'rock',
 'stop',
 'stop_inverted',
 'three',
 'three2',
 'two_up',
 'two_up_inverted']
```


```
base_dir  = './hagrid_classification/hagrid_classification'
os.chdir(base_dir)
```

***Note however that this project has the original dataset, so this section is irrelevant in this case.***

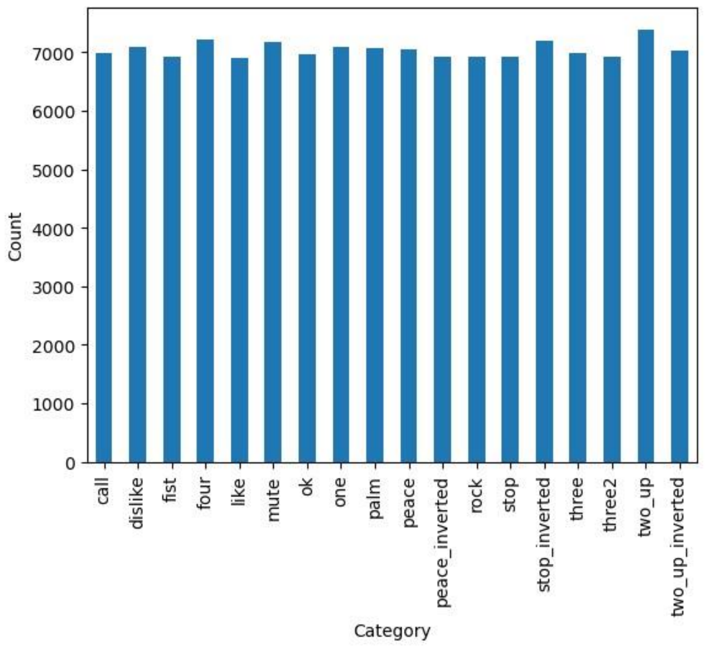
## Exploring the data

The was not much to thoroughly explore in the dataset as the data we need is only 2 variables. However, this section gave enough insight on what the data contains.

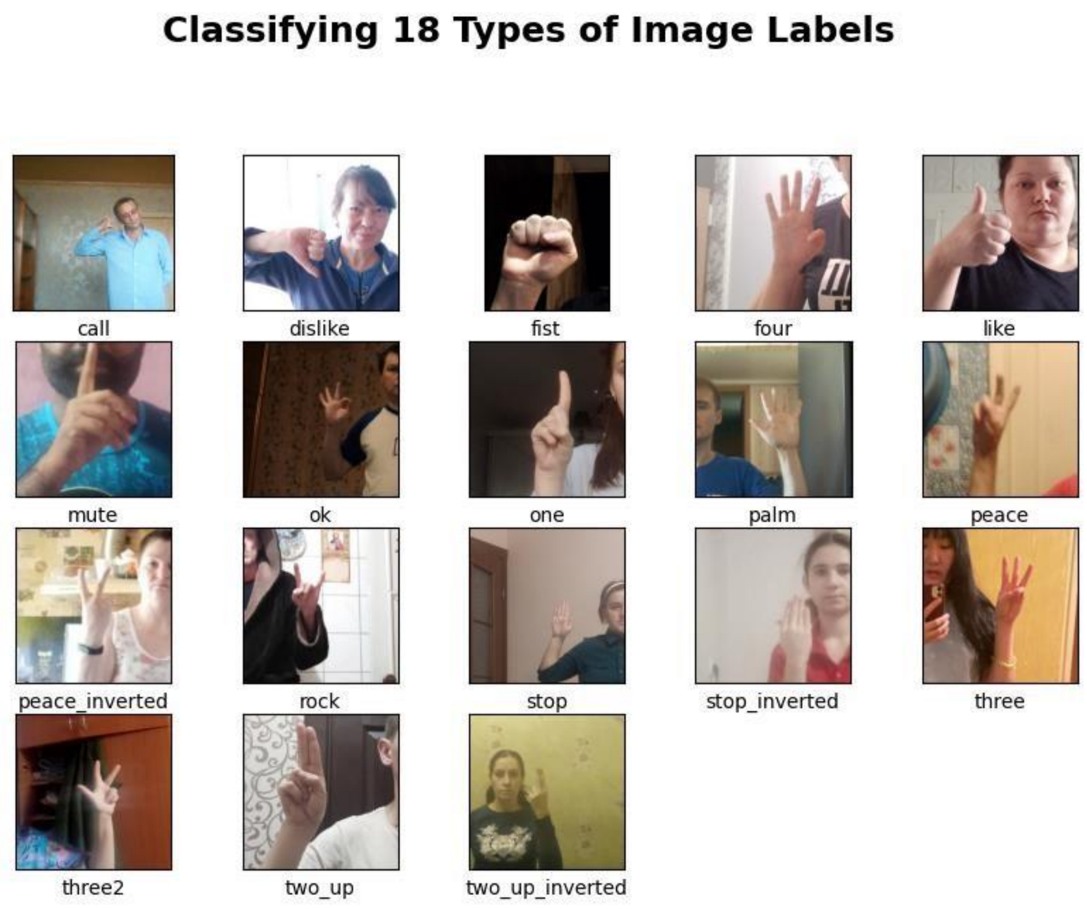


| | Labels | Number of Images |
|---|---|---|
| 0 | call | 6996 |
| 1 | dislike | 7087 |
| 2 | fist | 6918 |
| 3 | four | 7209 |
| 4 | like | 6898 |
| 5 | mute | 7181 |
| 6 | ok | 6975 |
| 7 | one | 7098 |
| 8 | palm | 7080 |
| 9 | peace | 7054 |
| 10 | peace_inverted | 6928 |
| 11 | rock | 6917 |
| 12 | stop | 6932 |
| 13 | stop_inverted | 7189 |
| 14 | three | 6977 |
| 15 | three2 | 6914 |
| 16 | two_up | 7391 |
| 17 | two_up_inverted | 7019 |

As the table shows, there are 18 categories for hand gestures with 6800 – 7400 images. The distribution of these categories is further shown in the following graph.



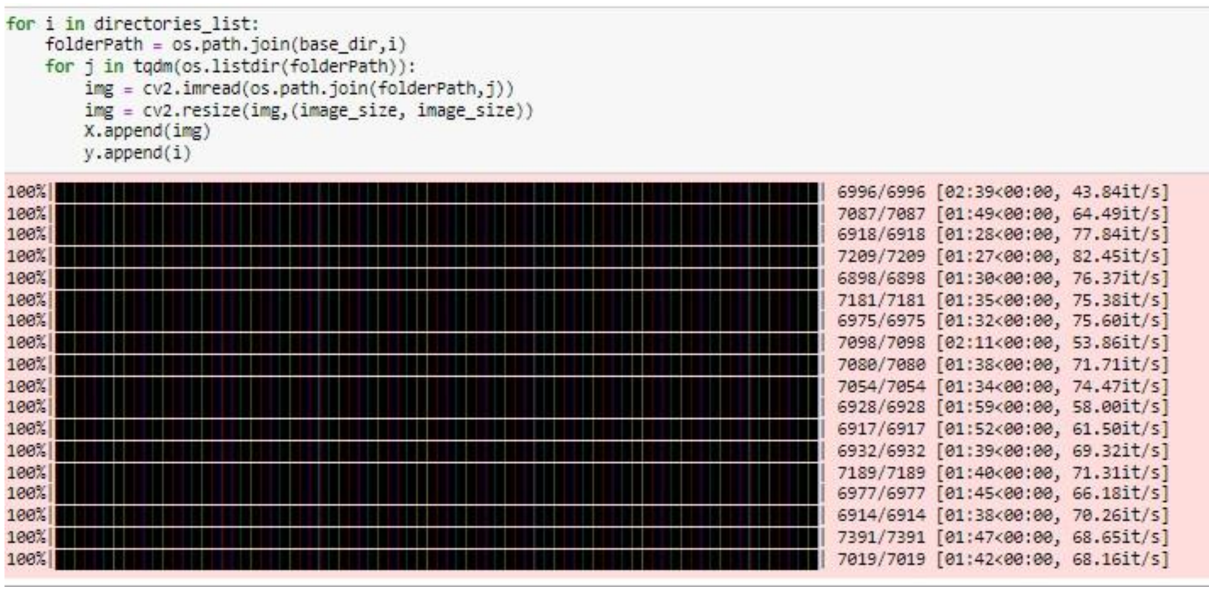
The following figure shows a sample for each category.



## Preprocessing Data

Standardize the image format to prepare the data. This could entail normalizing pixel values, scaling photographs to a consistent resolution, and performing any additional preprocessing that is required. Well-prepared and consistent data guarantees the model's ability to learn.

For this section, the dataset is split into X (image) and Y (category) and appended into arrays. The images have also been resized to 32.



```
for i in directories_list:
    folderPath = os.path.join(base_dir,i)
    for j in tqdm(os.listdir(folderPath)):
        img = cv2.imread(os.path.join(folderPath,j))
        img = cv2.resize(img,(image_size, image_size))
        X.append(img)
        y.append(i)
```

```
100%|██████████| 6996/6996 [02:39<00:00, 43.84it/s]
100%|██████████| 7087/7087 [01:49<00:00, 64.49it/s]
100%|██████████| 6918/6918 [01:28<00:00, 77.84it/s]
100%|██████████| 7209/7209 [01:27<00:00, 82.45it/s]
100%|██████████| 6898/6898 [01:30<00:00, 76.37it/s]
100%|██████████| 7181/7181 [01:35<00:00, 75.38it/s]
100%|██████████| 6975/6975 [01:32<00:00, 75.60it/s]
100%|██████████| 7098/7098 [02:11<00:00, 53.86it/s]
100%|██████████| 7080/7080 [01:38<00:00, 71.71it/s]
100%|██████████| 7054/7054 [01:34<00:00, 74.47it/s]
100%|██████████| 6928/6928 [01:59<00:00, 58.00it/s]
100%|██████████| 6917/6917 [01:52<00:00, 61.50it/s]
100%|██████████| 6932/6932 [01:39<00:00, 69.32it/s]
100%|██████████| 7189/7189 [01:40<00:00, 71.31it/s]
100%|██████████| 6977/6977 [01:45<00:00, 66.18it/s]
100%|██████████| 6914/6914 [01:38<00:00, 70.26it/s]
100%|██████████| 7391/7391 [01:47<00:00, 68.65it/s]
100%|██████████| 7019/7019 [01:42<00:00, 68.16it/s]
```

The following images shows sample images based on the category. Size is small however due to lack of figure formatting constraints from analyst’s side.

The data is then further split into the train and test data, validation data is taken from training data later.

## Model Selection and Training

Select a machine learning model architecture that is suitable for classifying images. Because convolutional neural networks (CNNs) can extract features and spatial hierarchies from images, they are a popular choice in machine learning. Alternatively, pre-trained models for certain tasks can be used with transfer learning.

For this section, there are two models used; Sequential model as the basic model and EfficientNet-b0 is implemented as the transfer learning model.

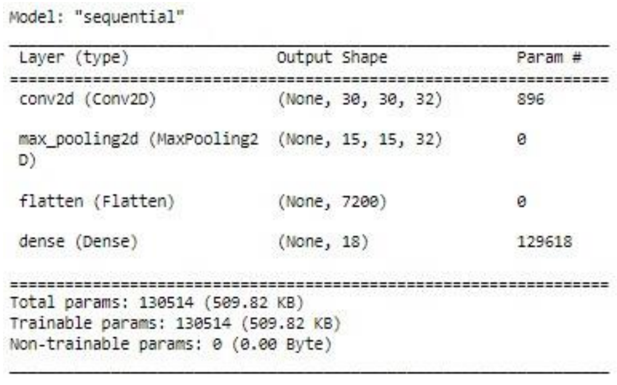
Utilizing the prepared dataset, train the chosen model. In the process of training, the model notices the characteristics and patterns connected to every class in the training set. This entails minimizing the discrepancy between the true and anticipated labels by modifying the model's parameters.

Establish a loss function that measures the discrepancy between the true and predicted labels. To reduce this loss, the model's parameters are adjusted using optimization techniques such stochastic gradient descent. This is a critical step in optimizing the model.

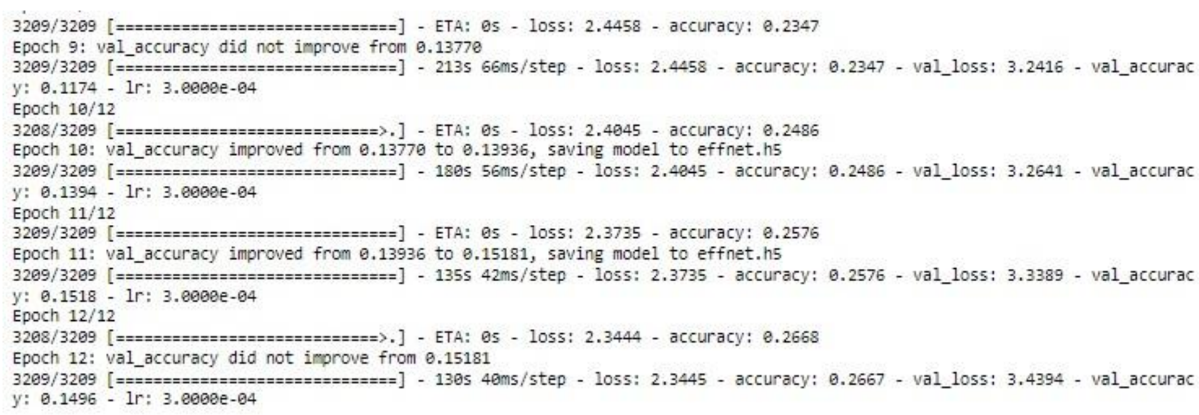
Evaluate the trained model's performance using a different validation dataset that it was not exposed to during training. This stage makes that the model fits new, untested data well and helps detect any overfitting.

### Sequential Model

First the model is set with a few layers and trained.

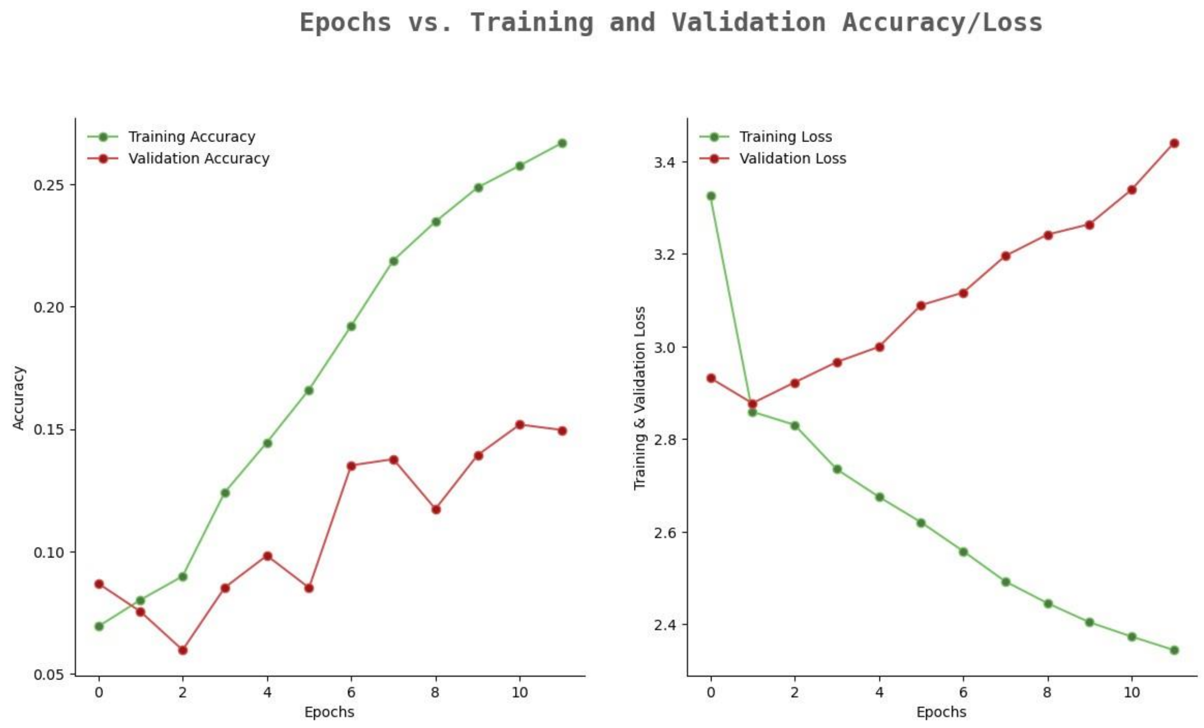


```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 32)        896

 max_pooling2d (MaxPooling2  (None, 15, 15, 32)        0
 D)

 flatten (Flatten)           (None, 7200)              0

 dense (Dense)               (None, 18)                129618

=================================================================
Total params: 130514 (509.82 KB)
Trainable params: 130514 (509.82 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

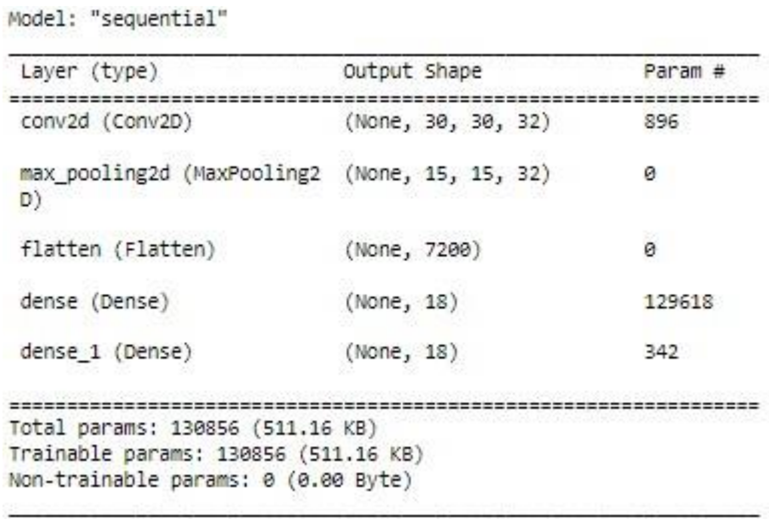


```
3209/3209 [==============================] - ETA: 0s - loss: 2.4458 - accuracy: 0.2347
Epoch 9: val_accuracy did not improve from 0.13770
3209/3209 [==============================] - 213s 66ms/step - loss: 2.4458 - accuracy: 0.2347 - val_loss: 3.2416 - val_accurac
y: 0.1174 - lr: 3.0000e-04
Epoch 10/12
3208/3209 [============================>.] - ETA: 0s - loss: 2.4045 - accuracy: 0.2486
Epoch 10: val_accuracy improved from 0.13770 to 0.13936, saving model to effnet.h5
3209/3209 [==============================] - 180s 56ms/step - loss: 2.4045 - accuracy: 0.2486 - val_loss: 3.2641 - val_accurac
y: 0.1394 - lr: 3.0000e-04
Epoch 11/12
3209/3209 [==============================] - ETA: 0s - loss: 2.3735 - accuracy: 0.2576
Epoch 11: val_accuracy improved from 0.13936 to 0.15181, saving model to effnet.h5
3209/3209 [==============================] - 135s 42ms/step - loss: 2.3735 - accuracy: 0.2576 - val_loss: 3.3389 - val_accurac
y: 0.1518 - lr: 3.0000e-04
Epoch 12/12
3208/3209 [============================>.] - ETA: 0s - loss: 2.3444 - accuracy: 0.2668
Epoch 12: val_accuracy did not improve from 0.15181
3209/3209 [==============================] - 130s 40ms/step - loss: 2.3445 - accuracy: 0.2667 - val_loss: 3.4394 - val_accurac
y: 0.1496 - lr: 3.0000e-04
```

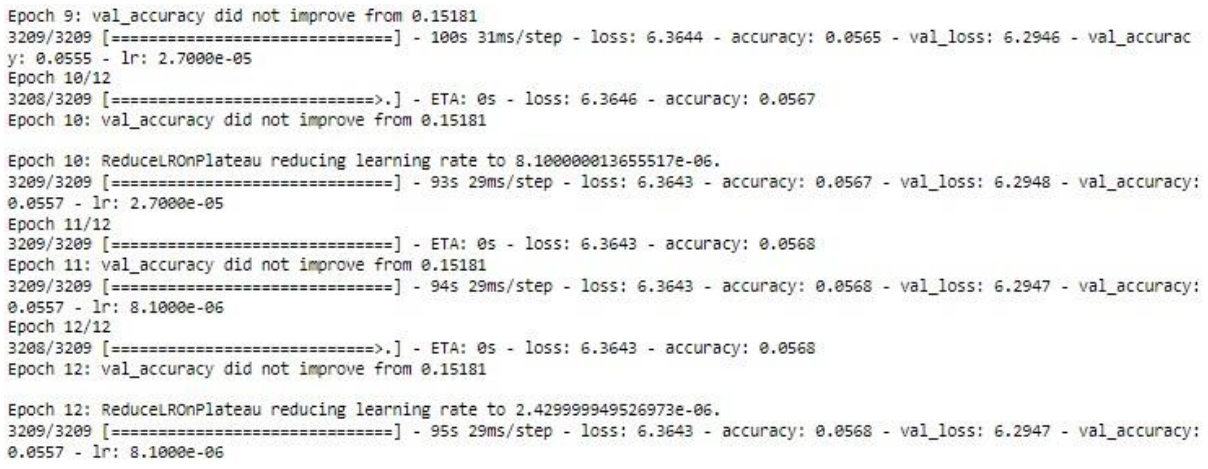
The highest validation accuracy is 15,1% which indicates overfitting since it is too low. The following graph prove the statement.



The training accuracy is significantly higher than the validation accuracy and validation loss is growing tremendously and inversely to training loss. It is assumed the model is missing a regularisation technique. However, this makes the model worse and decreases the learning accuracy exponentially further.

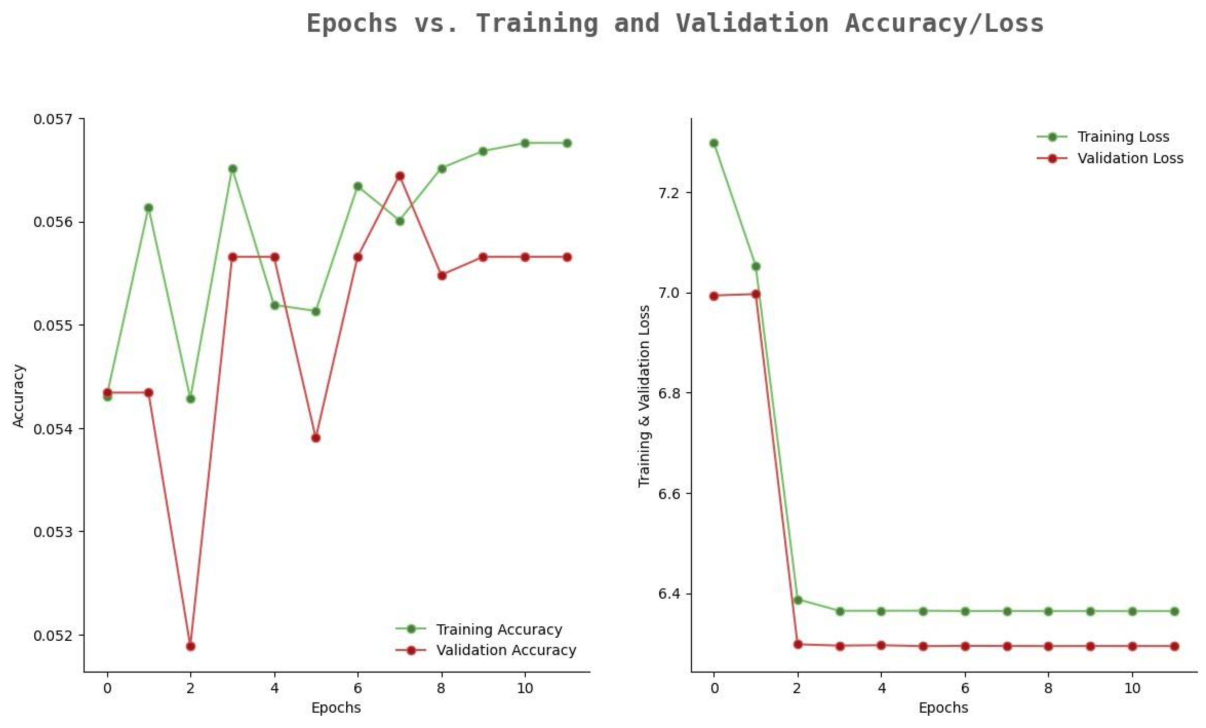


```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 32)        896

 max_pooling2d (MaxPooling2  (None, 15, 15, 32)        0
 D)

 flatten (Flatten)           (None, 7200)              0

 dense (Dense)               (None, 18)                129618

 dense_1 (Dense)             (None, 18)                342

=================================================================
Total params: 130856 (511.16 KB)
Trainable params: 130856 (511.16 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```



```
Epoch 9: val_accuracy did not improve from 0.15181
3209/3209 [==============================] - 100s 31ms/step - loss: 6.3644 - accuracy: 0.0565 - val_loss: 6.2946 - val_accurac
y: 0.0555 - lr: 2.7000e-05
Epoch 10/12
3208/3209 [============================>.] - ETA: 0s - loss: 6.3646 - accuracy: 0.0567
Epoch 10: val_accuracy did not improve from 0.15181

Epoch 10: ReduceLROnPlateau reducing learning rate to 8.100000013655517e-06.
3209/3209 [==============================] - 93s 29ms/step - loss: 6.3643 - accuracy: 0.0567 - val_loss: 6.2948 - val_accuracy:
0.0557 - lr: 2.7000e-05
Epoch 11/12
3209/3209 [==============================] - ETA: 0s - loss: 6.3643 - accuracy: 0.0568
Epoch 11: val_accuracy did not improve from 0.15181
3209/3209 [==============================] - 94s 29ms/step - loss: 6.3643 - accuracy: 0.0568 - val_loss: 6.2947 - val_accuracy:
0.0557 - lr: 8.1000e-06
Epoch 12/12
3208/3209 [============================>.] - ETA: 0s - loss: 6.3643 - accuracy: 0.0568
Epoch 12: val_accuracy did not improve from 0.15181

Epoch 12: ReduceLROnPlateau reducing learning rate to 2.429999949526973e-06.
3209/3209 [==============================] - 95s 29ms/step - loss: 6.3643 - accuracy: 0.0568 - val_loss: 6.2947 - val_accuracy:
0.0557 - lr: 8.1000e-06
```

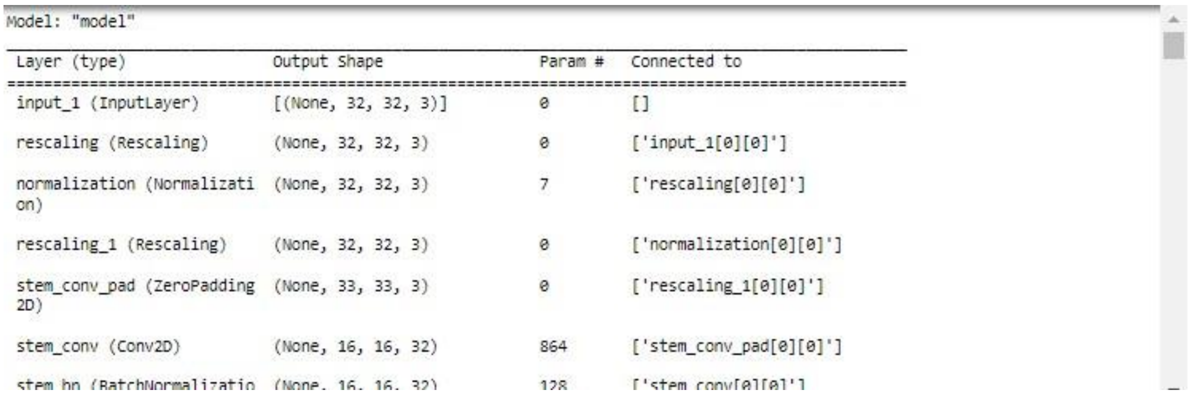
Consider the graphs below.



It is concluded that the model is too complex. Therefore, when another layer was added this decreased the accuracy even further since overfitting could result from model complexity or too many layers. However, the model did not contain many layers. Furthermore, data augmentation may be suggested for future endeavours, however the data is not suitable for this model until  further notice.

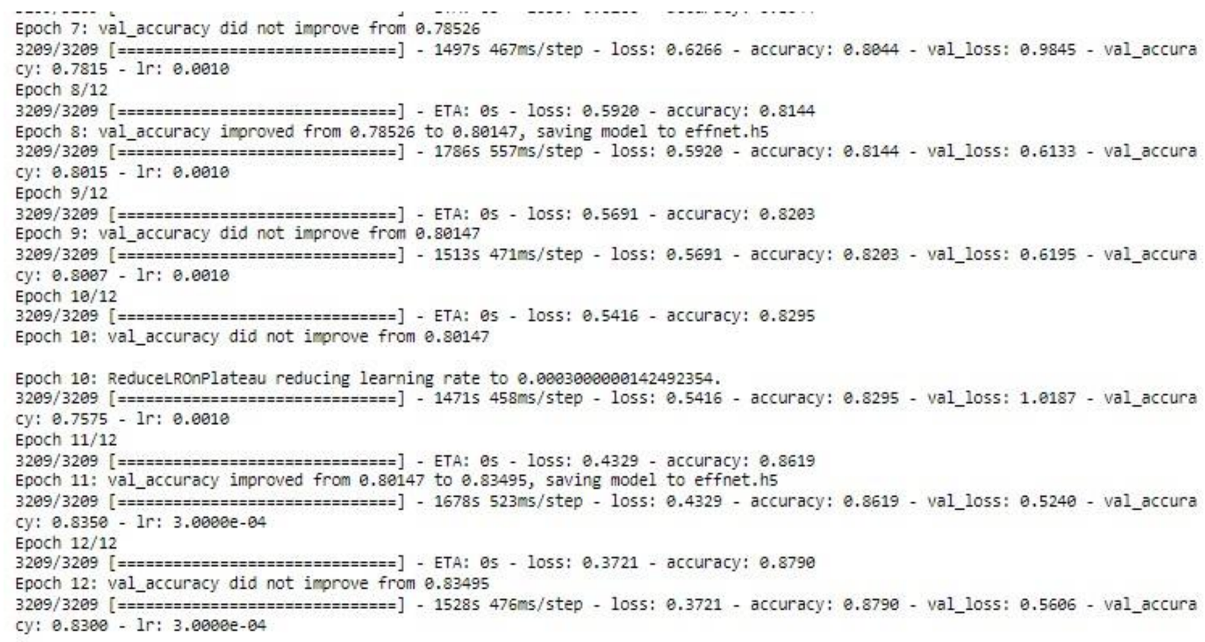
## Transfer learning model

For the transfer learning model, EfficientNet-b0 is implemented. It was originally trained using over a million pictures from the ImageNet collection. The network learns rich feature representations for a large variety of images and is capable of classifying photos into 1000 object categories. It has been demonstrated to perform better on multiple benchmarks than competing models such as ResNet and VGG. However, depending on the task and dataset, different models may perform differently.

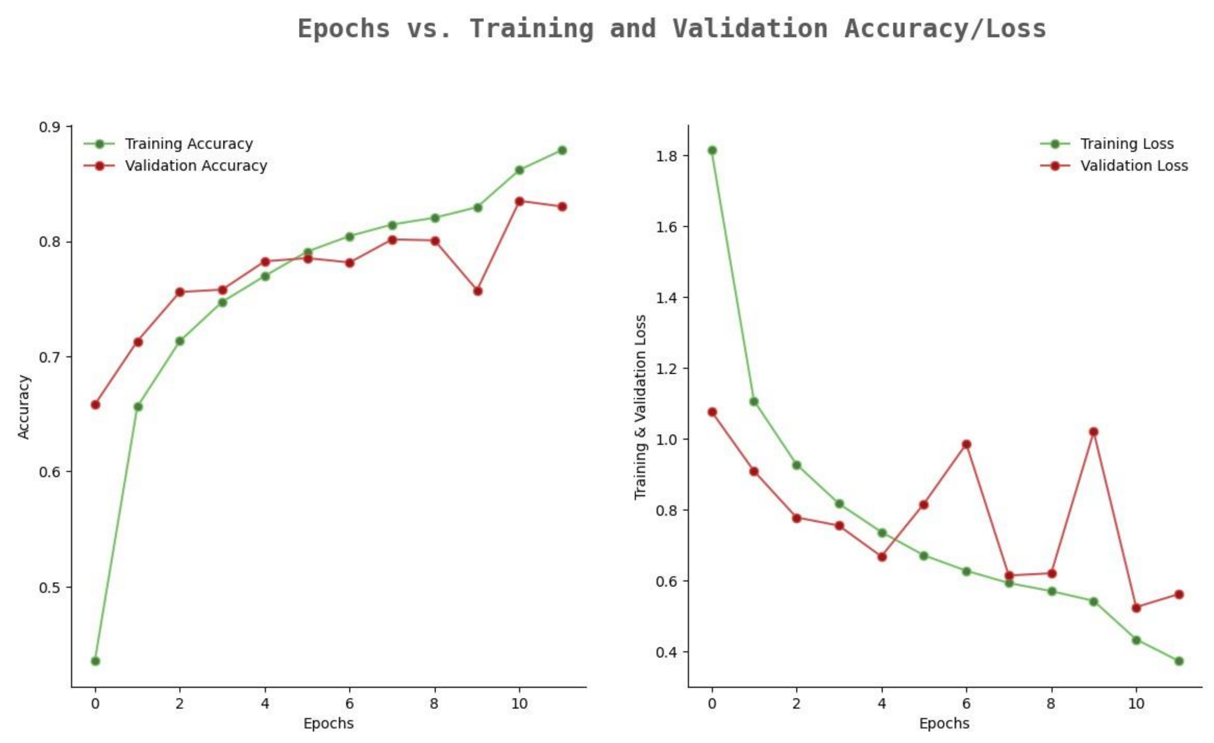


```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                Output Shape                 Param #   Connected to
==================================================================================================
 input_1 (InputLayer)        [(None, 32, 32, 3)]          0         []

 rescaling (Rescaling)       (None, 32, 32, 3)            0         ['input_1[0][0]']

 normalization (Normalizati  (None, 32, 32, 3)            7         ['rescaling[0][0]']
 on)

 rescaling_1 (Rescaling)     (None, 32, 32, 3)            0         ['normalization[0][0]']

 stem_conv_pad (ZeroPadding  (None, 33, 33, 3)            0         ['rescaling_1[0][0]']
 2D)

 stem_conv (Conv2D)          (None, 16, 16, 32)           864       ['stem_conv_pad[0][0]']

 stem_bn (BatchNormalizatio  (None, 16, 16, 32)           128       ['stem_conv[0][0]']
```

The model contains many layers, which in include techniques which help to reduce overfitting. The model validation accuracy does begin to improve, however. After 12 epochs, there is a validation accuracy of 83,96%. For future reference, the aim will be to achieve over 90% accuracy, however this will satisfy for now.



```
Epoch 7: val_accuracy did not improve from 0.78526
3209/3209 [==============================] - 1497s 467ms/step - loss: 0.6266 - accuracy: 0.8044 - val_loss: 0.9845 - val_accura
cy: 0.7815 - lr: 0.0010
Epoch 8/12
3209/3209 [==============================] - ETA: 0s - loss: 0.5920 - accuracy: 0.8144
Epoch 8: val_accuracy improved from 0.78526 to 0.80147, saving model to effnet.h5
3209/3209 [==============================] - 1786s 557ms/step - loss: 0.5920 - accuracy: 0.8144 - val_loss: 0.6133 - val_accura
cy: 0.8015 - lr: 0.0010
Epoch 9/12
3209/3209 [==============================] - ETA: 0s - loss: 0.5691 - accuracy: 0.8203
Epoch 9: val_accuracy did not improve from 0.80147
3209/3209 [==============================] - 1513s 471ms/step - loss: 0.5691 - accuracy: 0.8203 - val_loss: 0.6195 - val_accura
cy: 0.8007 - lr: 0.0010
Epoch 10/12
3209/3209 [==============================] - ETA: 0s - loss: 0.5416 - accuracy: 0.8295
Epoch 10: val_accuracy did not improve from 0.80147

Epoch 10: ReduceLROnPlateau reducing learning rate to 0.0003000000142492354.
3209/3209 [==============================] - 1471s 458ms/step - loss: 0.5416 - accuracy: 0.8295 - val_loss: 1.0187 - val_accura
cy: 0.7575 - lr: 0.0010
Epoch 11/12
3209/3209 [==============================] - ETA: 0s - loss: 0.4329 - accuracy: 0.8619
Epoch 11: val_accuracy improved from 0.80147 to 0.83495, saving model to effnet.h5
3209/3209 [==============================] - 1678s 523ms/step - loss: 0.4329 - accuracy: 0.8619 - val_loss: 0.5240 - val_accura
cy: 0.8350 - lr: 3.0000e-04
Epoch 12/12
3209/3209 [==============================] - ETA: 0s - loss: 0.3721 - accuracy: 0.8790
Epoch 12: val_accuracy did not improve from 0.83495
3209/3209 [==============================] - 1528s 476ms/step - loss: 0.3721 - accuracy: 0.8790 - val_loss: 0.5606 - val_accura
cy: 0.8300 - lr: 3.0000e-04
```

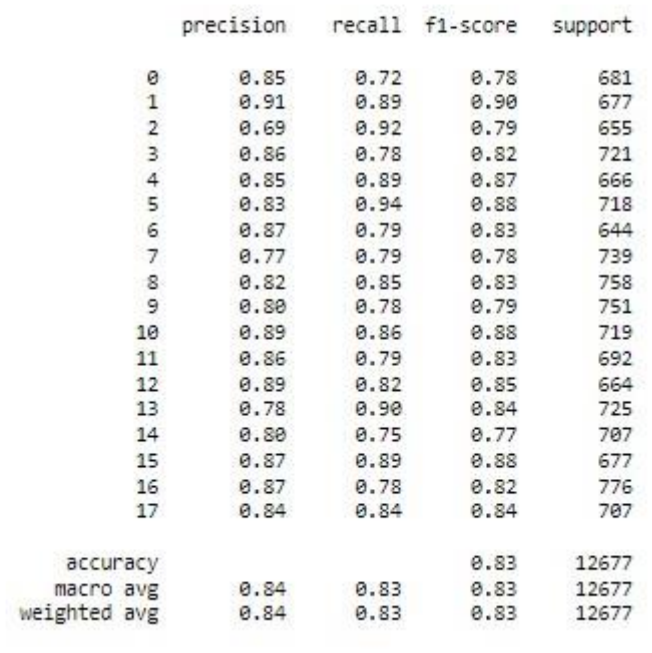


Although the accuracy is increased, there is still a bit of overfitting, but the results have however improved tremendously. This model is selected for predictions and testing.

## Testing

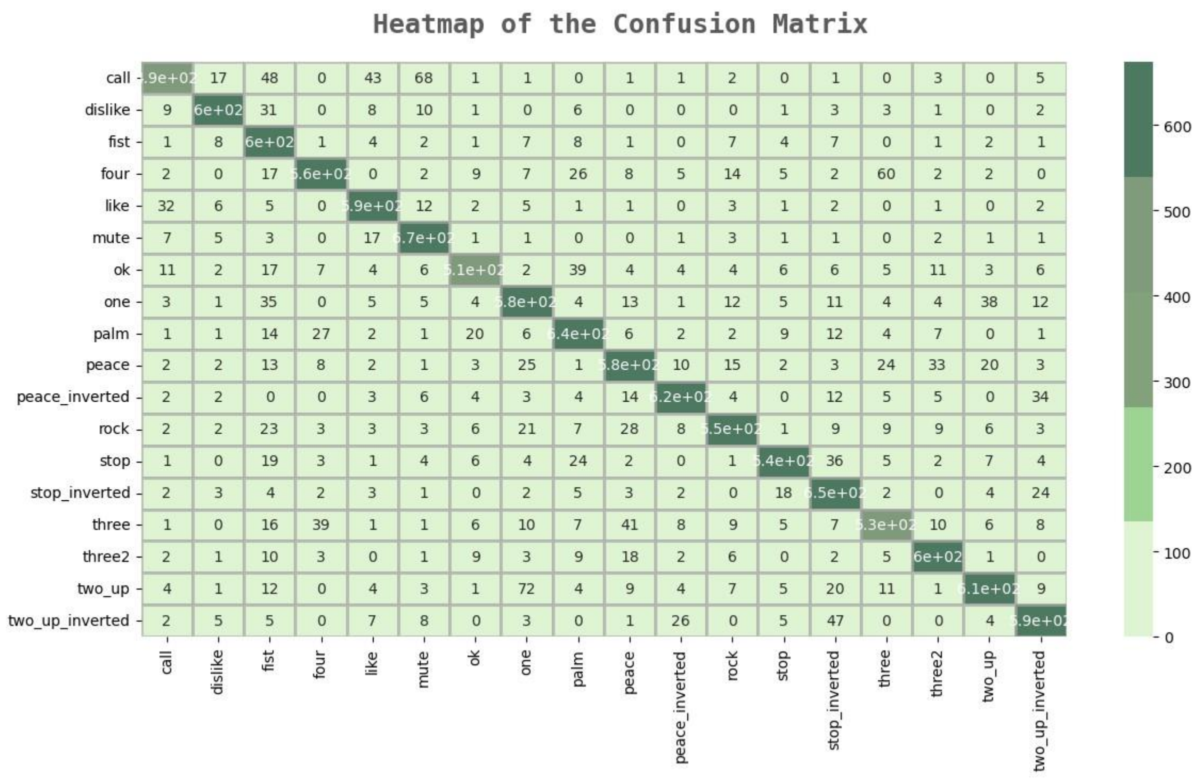
On a test dataset that is distinct from the training and validation sets, assess the model's performance. This stage offers a practical gauge of the model's precision and potency in categorizing fresh pictures.

The prediction accuracy is evaluated, and the following results is gained.



```
              precision    recall  f1-score   support

           0       0.85      0.72      0.78       681
           1       0.91      0.89      0.90       677
           2       0.69      0.92      0.79       655
           3       0.86      0.78      0.82       721
           4       0.85      0.89      0.87       666
           5       0.83      0.94      0.88       718
           6       0.87      0.79      0.83       644
           7       0.77      0.79      0.78       739
           8       0.82      0.85      0.83       758
           9       0.80      0.78      0.79       751
          10       0.89      0.86      0.88       719
          11       0.86      0.79      0.83       692
          12       0.89      0.82      0.85       664
          13       0.78      0.90      0.84       725
          14       0.80      0.75      0.77       707
          15       0.87      0.89      0.88       677
          16       0.87      0.78      0.82       776
          17       0.84      0.84      0.84       707

    accuracy                           0.83     12677
   macro avg       0.84      0.83      0.83     12677
weighted avg       0.84      0.83      0.83     12677
```

The overall accuracy is satisfactory.



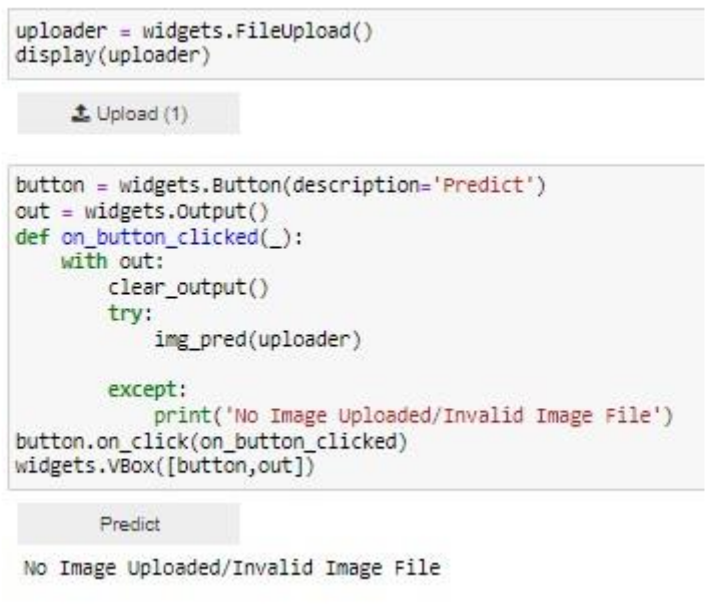
| | call | dislike | fist | four | like | mute | ok | one | palm | peace | peace_inverted | rock | stop | stop_inverted | three | three2 | two_up | two_up_inverted |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| call | .9e+02 | 17 | 48 | 0 | 43 | 68 | 1 | 1 | 0 | 1 | 1 | 2 | 0 | 1 | 0 | 3 | 0 | 5 |
| dislike | 9 | 6e+02 | 31 | 0 | 8 | 10 | 1 | 0 | 6 | 0 | 0 | 0 | 1 | 3 | 3 | 1 | 0 | 2 |
| fist | 1 | 8 | 6e+02 | 1 | 4 | 2 | 1 | 7 | 8 | 1 | 0 | 7 | 4 | 7 | 0 | 1 | 2 | 1 |
| four | 2 | 0 | 17 | 5.6e+02 | 0 | 2 | 9 | 7 | 26 | 8 | 5 | 14 | 5 | 2 | 60 | 2 | 2 | 0 |
| like | 32 | 6 | 5 | 0 | 5.9e+02 | 12 | 2 | 5 | 1 | 1 | 0 | 3 | 1 | 2 | 0 | 1 | 0 | 2 |
| mute | 7 | 5 | 3 | 0 | 17 | 5.7e+02 | 1 | 1 | 0 | 0 | 1 | 3 | 1 | 1 | 0 | 2 | 1 | 1 |
| ok | 11 | 2 | 17 | 7 | 4 | 6 | 5.1e+02 | 2 | 39 | 4 | 4 | 4 | 6 | 6 | 5 | 11 | 3 | 6 |
| one | 3 | 1 | 35 | 0 | 5 | 5 | 4 | 5.8e+02 | 4 | 13 | 1 | 12 | 5 | 11 | 4 | 4 | 38 | 12 |
| palm | 1 | 1 | 14 | 27 | 2 | 1 | 20 | 6 | 5.4e+02 | 6 | 2 | 2 | 9 | 12 | 4 | 7 | 0 | 1 |
| peace | 2 | 2 | 13 | 8 | 2 | 1 | 3 | 25 | 1 | 5.8e+02 | 10 | 15 | 2 | 3 | 24 | 33 | 20 | 3 |
| peace_inverted | 2 | 2 | 0 | 0 | 3 | 6 | 4 | 3 | 4 | 14 | 5.2e+02 | 4 | 0 | 12 | 5 | 5 | 0 | 34 |
| rock | 2 | 2 | 23 | 3 | 3 | 3 | 6 | 21 | 7 | 28 | 8 | 5.5e+02 | 1 | 9 | 9 | 9 | 6 | 3 |
| stop | 1 | 0 | 19 | 3 | 1 | 4 | 6 | 4 | 24 | 2 | 0 | 1 | 5.4e+02 | 36 | 5 | 2 | 7 | 4 |
| stop_inverted | 2 | 3 | 4 | 2 | 3 | 1 | 0 | 2 | 5 | 3 | 2 | 0 | 18 | 5.5e+02 | 2 | 0 | 4 | 24 |
| three | 1 | 0 | 16 | 39 | 1 | 1 | 6 | 10 | 7 | 41 | 8 | 9 | 5 | 7 | 5.3e+02 | 10 | 6 | 8 |
| three2 | 2 | 1 | 10 | 3 | 0 | 1 | 9 | 3 | 9 | 18 | 2 | 6 | 0 | 2 | 5 | 6e+02 | 1 | 0 |
| two_up | 4 | 1 | 12 | 0 | 4 | 3 | 1 | 72 | 4 | 9 | 4 | 7 | 5 | 20 | 11 | 1 | 5.1e+02 | 9 |
| two_up_inverted | 2 | 5 | 5 | 0 | 7 | 8 | 0 | 3 | 0 | 1 | 26 | 0 | 5 | 47 | 0 | 0 | 4 | 5.9e+02 |

It is also good that the is low correlation between the categories since that is what makes the distinguishable.

## Results

Apply the learned model to new, unobserved image prediction. After receiving an input image, the model generates a probability distribution covering all classes that shows how likely it is that the image belongs in each class.

There is a section where the user can upload a picture to predict the category. However, the development is still ongoing. Hence, this section is a beta sample and in development.



```
uploader = widgets.FileUpload()
display(uploader)
```

```
button = widgets.Button(description='Predict')
out = widgets.Output()
def on_button_clicked(_):
    with out:
        clear_output()
        try:
            img_pred(uploader)

        except:
            print('No Image Uploaded/Invalid Image File')
button.on_click(on_button_clicked)
widgets.VBox([button,out])
```

```
No Image Uploaded/Invalid Image File
```

### Fine-Tuning

If the results are not satisfactory, one can adjust hyperparameters or train the model on more data to fine-tune it based on how well it performs on real-world data. The goal of this step is to increase the model's precision and flexibility to fit different application scenarios.

### Deployment

Once the model's performance is acceptable, put it to use in practical applications. During the deployment step, the model is integrated into the platform or system of choice for automatic picture classification.

## Data Augmentation

During training, data augmentation methods are frequently used to increase the stability of image classification models and avoid overfitting. To artificially increase the diversity of the training dataset, these techniques apply random transformations, such as rotation, scaling, and flipping, to the training images.

## Class Imbalance

One of the most common problems in image classification is managing class imbalance. The model may favour the majority classes if some classes have notably fewer examples than others. This problem can be solved by employing strategies like class weights in the loss function or oversampling or under-sampling.

## Interpretability

It is important to know why a model predicts something, particularly in situations where openness is required. Model interpretation techniques are investigated to shed light on how image classification models make decisions. Examples of these techniques include gradient-based methods and attention mechanisms.

## Advancing 2D Images

While 2D images are the focus of traditional image classification, 3D image classification is now being investigated for uses in robotics and medical imaging, among other fields where third-dimensional spatial information is essential.

Real-time applications of image classification include gesture recognition for human computer interaction, facial recognition in security systems, and object detection in video streams. The classification models must be inferred quickly and efficiently for these applications.

## Challenges

The complexity and diversity of gestures, variations in lighting, and the requirement for dependable models that can oversee real-world circumstances are the main obstacles to image classification for hand signs. To increase the precision and effectiveness of hand sign categorization systems, researchers and practitioners in this field are always producing new and creative methods that make use of deep learning architectures, convolutional neural networks (CNNs), and other innovative techniques.

It is frequently necessary to use complex architectures, ensemble techniques, and training strategy innovations to address these issues. However, if a model is already overfit or too complex, this becomes a predicament as well.

## Conclusion

The many phases of data collection, preprocessing, model selection, training, and validation is all part of the process of finding an effective image classification system for hand signs. Convolutional Neural Networks are the keystone that manages the ensemble of pixels into meaningful insights. They are primaries at decoding spatial hierarchies inside images. With the help of transfer learning, a potent technique, the model can overcome its initial limitations by gaining knowledge from prior experiences and adjusting to new tasks.

## References

DataCamp, 2022. *Lasso and Ridge Regression in Python Tutorial.* [Online]
Available at: https://www.datacamp.com/tutorial/tutorial-lasso-ridge-regression
[Accessed 17 Oct 2023].

Dataquest, 2022. *Data Analyst in Python.* [Online]
Available at: https://www.dataquest.io/path/data-analyst/
[Accessed 4 Nov 2023].

Lee, M., 2022. *Crops Image-CNN vs Transfer Learning.* [Online]
Available at: https://www.kaggle.com/code/leekahwin/crops-image-cnn-vs-transfer-learning
[Accessed 5 Nov 2023].

Muller, A. C. & Guido, S., 2018. *Introduction to Machine Learning with Python.* Forth ed. Sebastopol, CA: O'Reilly Media, Inc.

SARAHG, 2017. *Titanic Analysis_Learning to Swim with Python.* [Online]
Available at: https://www.kaggle.com/code/sgus1318/titanic-analysis-learning-to-swim-with-python
[Accessed 15 Oct 2023].

Sherif, A. & Ravinda, A., 2018. *Apache Spark Deep Learning Cookbook.* Birmingham: Packt Publishing.