Smartphone user Identification ~ Kaggle Competition

Rares Patrascu

Project Documentation

To begin working on the project, I first acquired the necessary data from Kaggle using its API.

1. Preprocessing

   After studying the data, I determined that the grids are unequal, but their mean length is roughly 149.8838 for the train data and 149.9034 for the test data, hence each column of each grid should be cut or padded to 150. Padding was accomplished by concatenating 0 for each column that was less than 150 and giving up the surplus for the larger ones. I also investigated scalling the data, but given that the first two of my models were reasonably accurate, I concluded that I could enhance the model rather than the data.

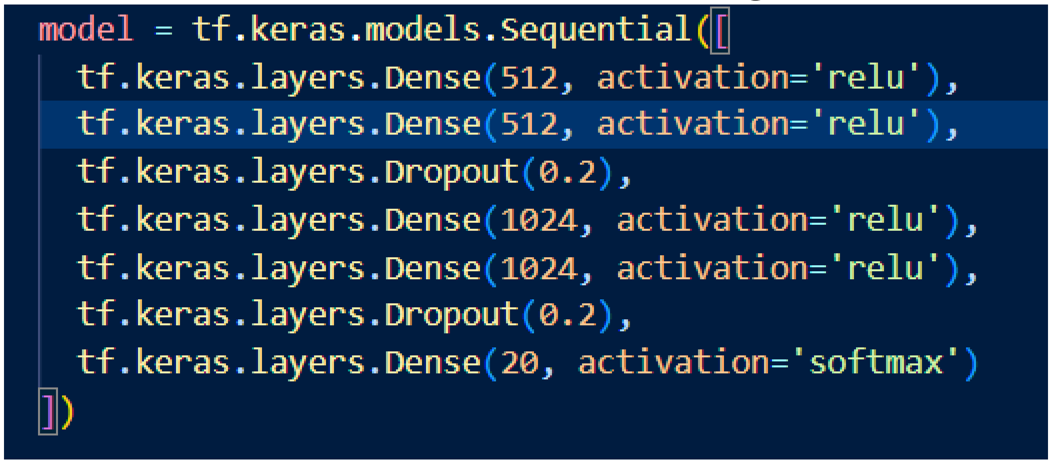
2. Used Models

   The two used types of machine learning models tried we're the Multilayer Perceptron(MLP) and the Convolutional Neural Network (CNN). As we can see all of the utilized models are part of the supervised learning procedure.

   - Multilayer Perceptron

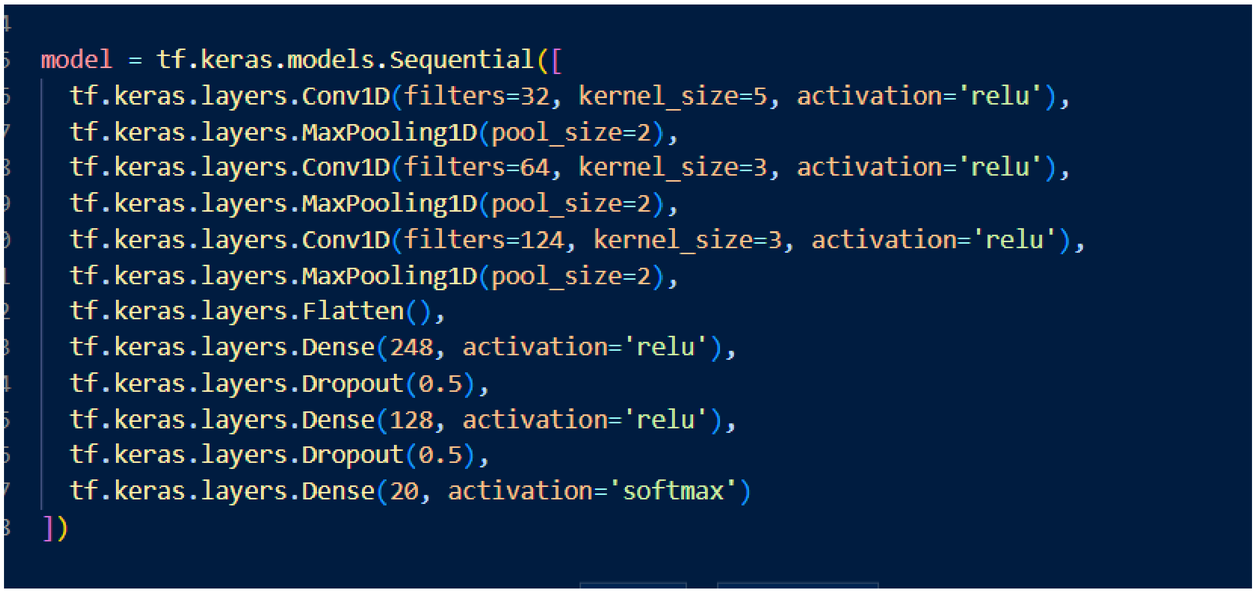
   This was the initial model selected since it is simple to apply and is well known for learning good mapping from inputs to outputs when the input data can be reduced to a flattened array. This being said the input was transformed to be a 450 flattened array.

The first tried architecture was the following one:



```
model = tf.keras.models.Sequential([
  tf.keras.layers.Dense(512, activation='relu'),
  tf.keras.layers.Dense(512, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(1024, activation='relu'),
  tf.keras.layers.Dense(1024, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(20, activation='softmax')
])
```

The model was trained with the last 2000 values as validation data using a SGD optimizer and 0.001 learning rate for 100 epochs. The accuracy result was 0.78 so I decided to use an architecture that has Conv1d Layers.



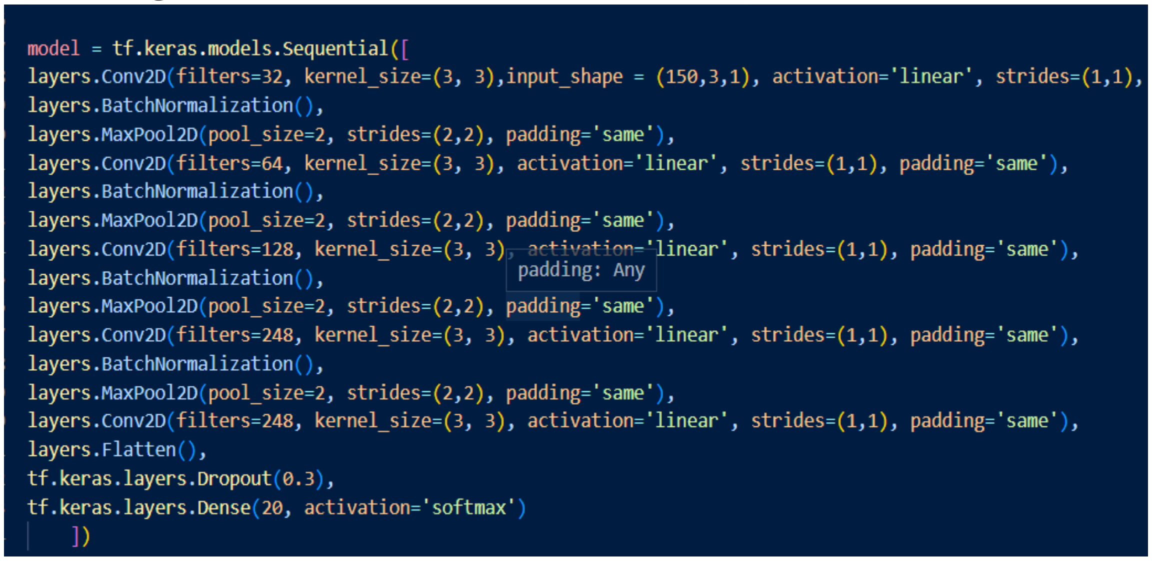
```
model = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=32, kernel_size=5, activation='relu'),
  tf.keras.layers.MaxPooling1D(pool_size=2),
  tf.keras.layers.Conv1D(filters=64, kernel_size=3, activation='relu'),
  tf.keras.layers.MaxPooling1D(pool_size=2),
  tf.keras.layers.Conv1D(filters=124, kernel_size=3, activation='relu'),
  tf.keras.layers.MaxPooling1D(pool_size=2),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(248, activation='relu'),
  tf.keras.layers.Dropout(0.5),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.5),
  tf.keras.layers.Dense(20, activation='softmax')
])
```

After observing that this model improved accuracy with the same hyperparameters, I decided to begin utilizing CNN.

- Convolutional Neural Network

  This type of model is usually perfect to take into consideration the local spatial coherence that grid like data has. Therefore the input data was preproccesed to be sized as (150,3,1).

  Thus being said I have starting tackling this problem with the following CNN model:



```
model = tf.keras.models.Sequential([
layers.Conv2D(filters=32, kernel_size=(3, 3),input_shape = (150,3,1), activation='linear', strides=(1,1),
layers.BatchNormalization(),
layers.MaxPool2D(pool_size=2, strides=(2,2), padding='same'),
layers.Conv2D(filters=64, kernel_size=(3, 3), activation='linear', strides=(1,1), padding='same'),
layers.BatchNormalization(),
layers.MaxPool2D(pool_size=2, strides=(2,2), padding='same'),
layers.Conv2D(filters=128, kernel_size=(3, 3), activation='linear', strides=(1,1), padding='same'),
layers.BatchNormalization(),
layers.MaxPool2D(pool_size=2, strides=(2,2), padding='same'),
layers.Conv2D(filters=248, kernel_size=(3, 3), activation='linear', strides=(1,1), padding='same'),
layers.BatchNormalization(),
layers.MaxPool2D(pool_size=2, strides=(2,2), padding='same'),
layers.Conv2D(filters=248, kernel_size=(3, 3), activation='linear', strides=(1,1), padding='same'),
layers.Flatten(),
tf.keras.layers.Dropout(0.3),
tf.keras.layers.Dense(20, activation='softmax')
    ])
```

This model was trained with 2000 validation data a SGD optimizer 0.001 learning rate for 100 epochs and a “sparse categorical crossentropy” ass the loss function (because the labels we’re not one hot encoded) and got a 9.13 accuracy on the test data.

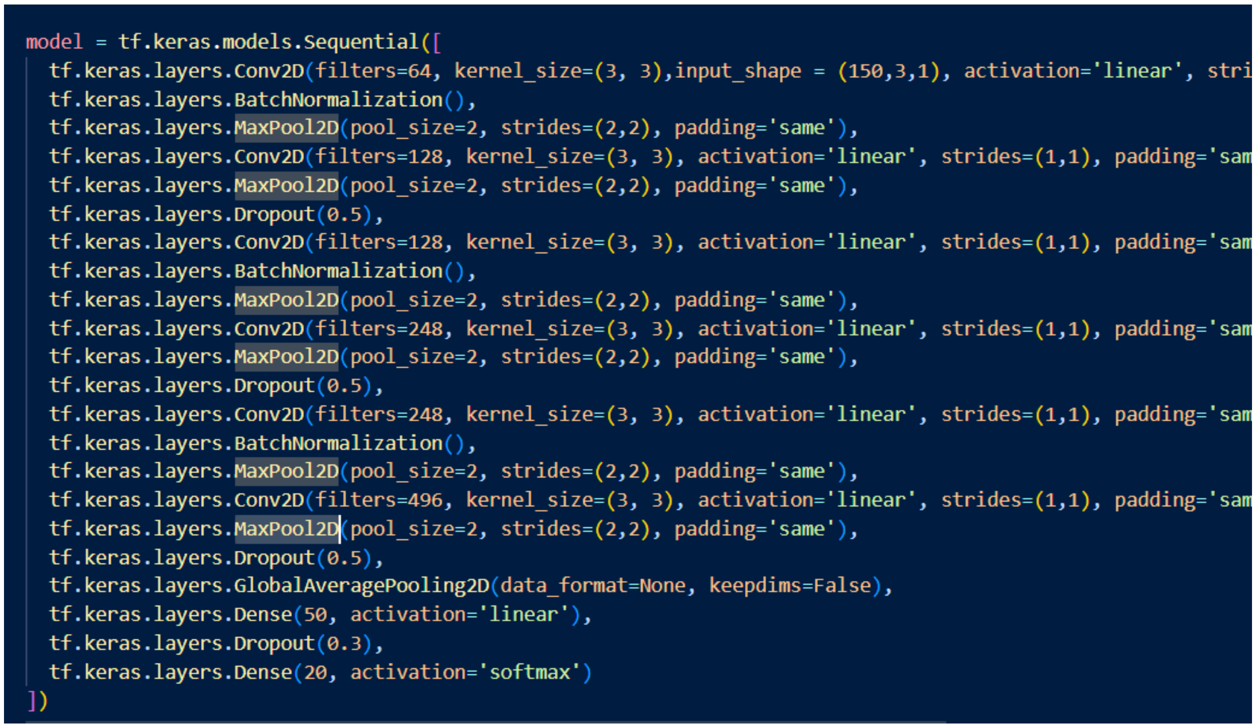
Giving that no normalization was done at all used batch_normalization layers and that proved to improve the accuracy.

After deciding to work only with cnn’s I started testing the following:

- Tested both SGD and Adam as optimizers and concluded that SGD usually get’s better results.
- Added or decreased the number of convolutional layers.

- Started training using cross-validation with 5 folds and saving the best model for each fold.
- Change activation functions between Relu, Linear and LeakyRell. The best one turned out to be LeakyRelly bi improving accuracy with a small percent of 002.

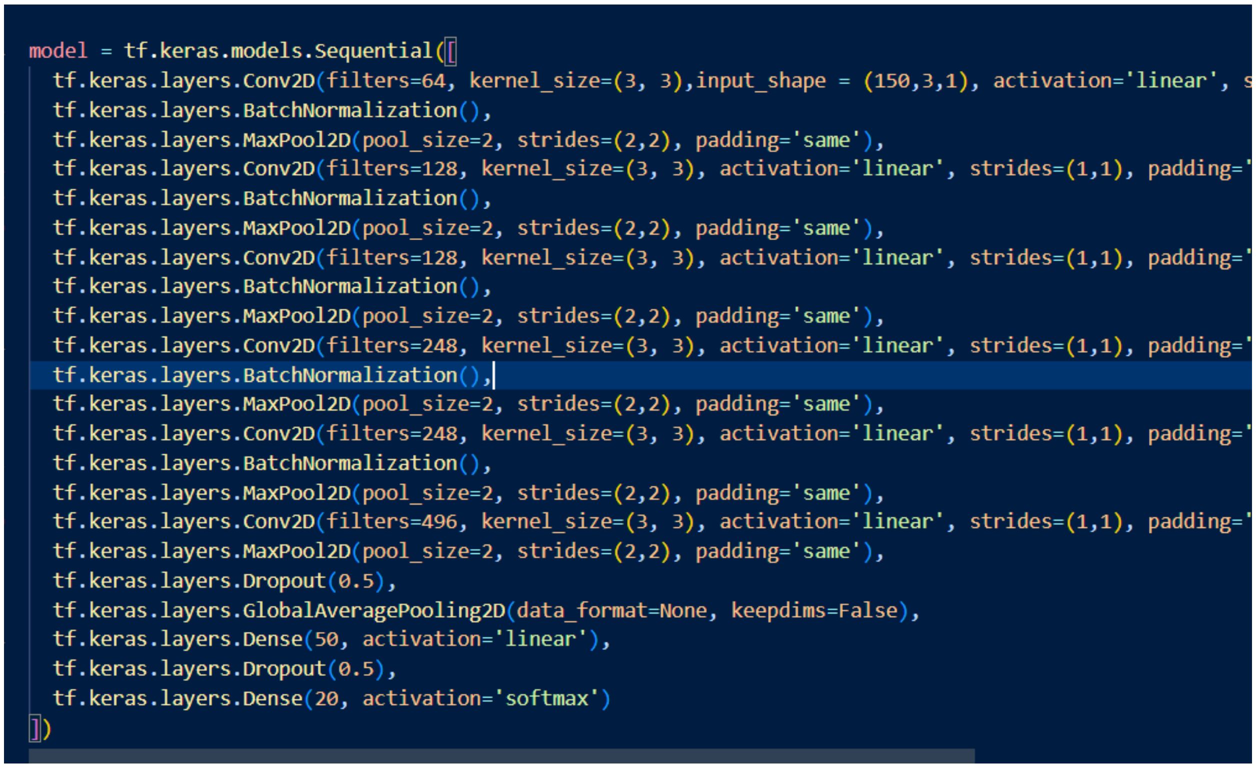
Using these methods I got the following better model that had an accuracy on the test data of 0.94.



```
model = tf.keras.models.Sequential([
  tf.keras.layers.Conv2D(filters=64, kernel_size=(3, 3),input_shape = (150,3,1), activation='linear', stri
  tf.keras.layers.BatchNormalization(),
  tf.keras.layers.MaxPool2D(pool_size=2, strides=(2,2), padding='same'),
  tf.keras.layers.Conv2D(filters=128, kernel_size=(3, 3), activation='linear', strides=(1,1), padding='sam
  tf.keras.layers.MaxPool2D(pool_size=2, strides=(2,2), padding='same'),
  tf.keras.layers.Dropout(0.5),
  tf.keras.layers.Conv2D(filters=128, kernel_size=(3, 3), activation='linear', strides=(1,1), padding='sam
  tf.keras.layers.BatchNormalization(),
  tf.keras.layers.MaxPool2D(pool_size=2, strides=(2,2), padding='same'),
  tf.keras.layers.Conv2D(filters=248, kernel_size=(3, 3), activation='linear', strides=(1,1), padding='sam
  tf.keras.layers.MaxPool2D(pool_size=2, strides=(2,2), padding='same'),
  tf.keras.layers.Dropout(0.5),
  tf.keras.layers.Conv2D(filters=248, kernel_size=(3, 3), activation='linear', strides=(1,1), padding='sam
  tf.keras.layers.BatchNormalization(),
  tf.keras.layers.MaxPool2D(pool_size=2, strides=(2,2), padding='same'),
  tf.keras.layers.Conv2D(filters=496, kernel_size=(3, 3), activation='linear', strides=(1,1), padding='sam
  tf.keras.layers.MaxPool2D(pool_size=2, strides=(2,2), padding='same'),
  tf.keras.layers.Dropout(0.5),
  tf.keras.layers.GlobalAveragePooling2D(data_format=None, keepdims=False),
  tf.keras.layers.Dense(50, activation='linear'),
  tf.keras.layers.Dropout(0.3),
  tf.keras.layers.Dense(20, activation='softmax')
])
```

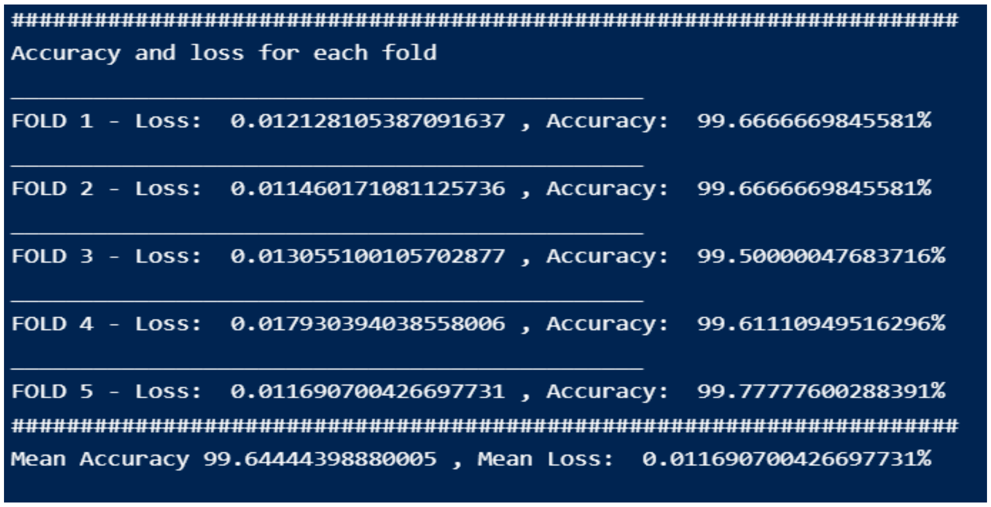
After reading a lot of data regarding the "good" order of the BatchNorm, DropOut, and Activation layers, I started exploring different combinations of these layers to attain the highest possible accuracy.

And after all the combination I came out with the following model, that was trained with a SGD optimizer a scheduled loss that decreased exponentially at the 50,60, and 90 epochs, linear activations on the convolutional layers using cross fold validation and saving the best checkpoint out of all epochs of all folds.

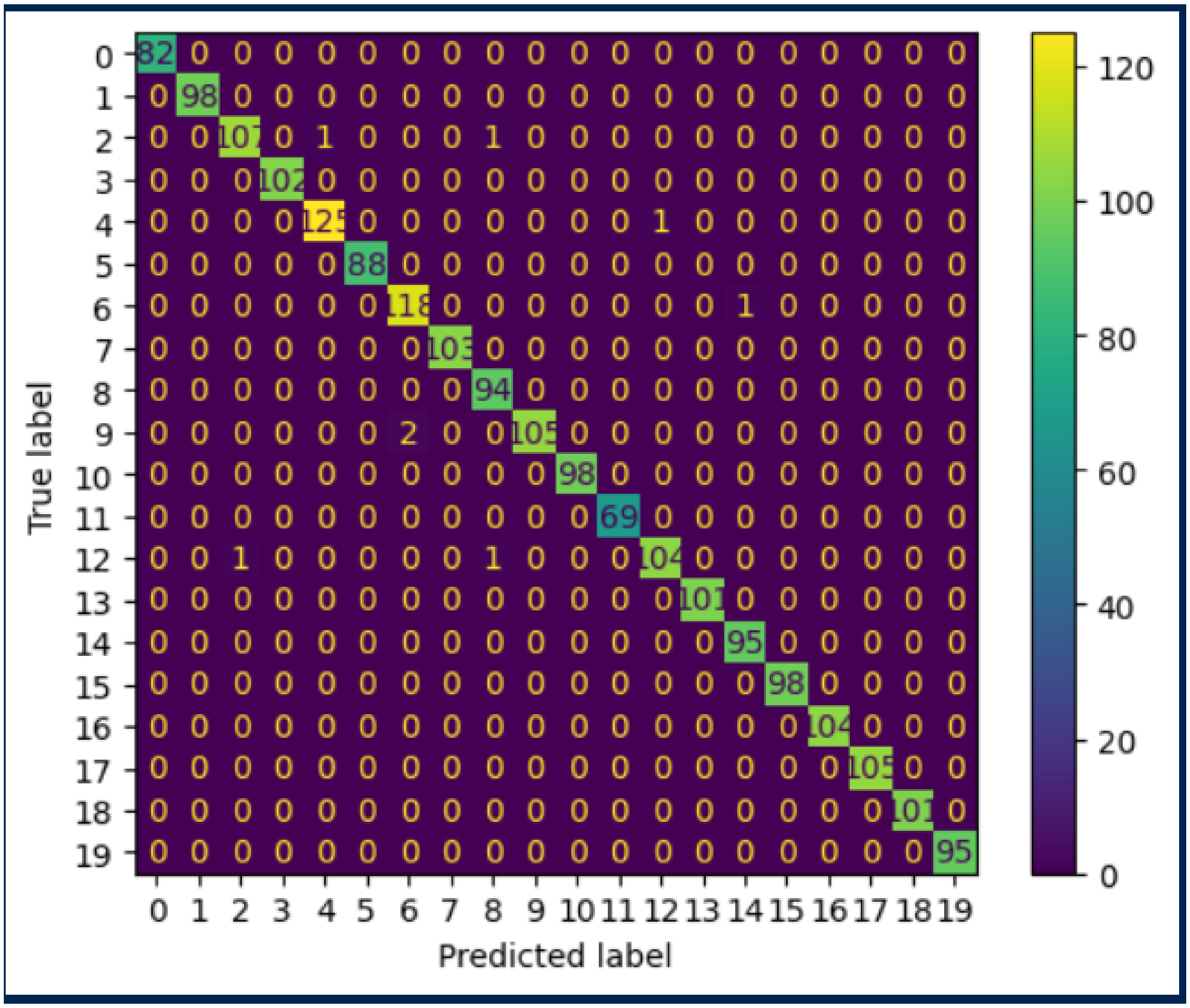


```
model = tf.keras.models.Sequential([
  tf.keras.layers.Conv2D(filters=64, kernel_size=(3, 3),input_shape = (150,3,1), activation='linear', s
  tf.keras.layers.BatchNormalization(),
  tf.keras.layers.MaxPool2D(pool_size=2, strides=(2,2), padding='same'),
  tf.keras.layers.Conv2D(filters=128, kernel_size=(3, 3), activation='linear', strides=(1,1), padding='
  tf.keras.layers.BatchNormalization(),
  tf.keras.layers.MaxPool2D(pool_size=2, strides=(2,2), padding='same'),
  tf.keras.layers.Conv2D(filters=128, kernel_size=(3, 3), activation='linear', strides=(1,1), padding='
  tf.keras.layers.BatchNormalization(),
  tf.keras.layers.MaxPool2D(pool_size=2, strides=(2,2), padding='same'),
  tf.keras.layers.Conv2D(filters=248, kernel_size=(3, 3), activation='linear', strides=(1,1), padding='
  tf.keras.layers.BatchNormalization(),
  tf.keras.layers.MaxPool2D(pool_size=2, strides=(2,2), padding='same'),
  tf.keras.layers.Conv2D(filters=248, kernel_size=(3, 3), activation='linear', strides=(1,1), padding='
  tf.keras.layers.BatchNormalization(),
  tf.keras.layers.MaxPool2D(pool_size=2, strides=(2,2), padding='same'),
  tf.keras.layers.Conv2D(filters=496, kernel_size=(3, 3), activation='linear', strides=(1,1), padding='
  tf.keras.layers.MaxPool2D(pool_size=2, strides=(2,2), padding='same'),
  tf.keras.layers.Dropout(0.5),
  tf.keras.layers.GlobalAveragePooling2D(data_format=None, keepdims=False),
  tf.keras.layers.Dense(50, activation='linear'),
  tf.keras.layers.Dropout(0.5),
  tf.keras.layers.Dense(20, activation='softmax')
])
```

The results of the cross fold validation are the following:



```
########################################################################
Accuracy and loss for each fold
________________________________________________
FOLD 1 - Loss:  0.012128105387091637 , Accuracy:  99.6666669845581%
________________________________________________
FOLD 2 - Loss:  0.011460171081125736 , Accuracy:  99.6666669845581%
________________________________________________
FOLD 3 - Loss:  0.013055100105702877 , Accuracy:  99.50000047683716%
________________________________________________
FOLD 4 - Loss:  0.017930394038558006 , Accuracy:  99.61110949516296%
________________________________________________
FOLD 5 - Loss:  0.011690700426697731 , Accuracy:  99.77777600288391%
########################################################################
Mean Accuracy 99.64444398880005 , Mean Loss:  0.011690700426697731%
```



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	82	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	98	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	107	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	102	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	125	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
5	0	0	0	0	0	88	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	118	0	0	0	0	0	0	0	1	0	0	0	0	0
7	0	0	0	0	0	0	0	103	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	94	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	2	0	0	105	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	98	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	69	0	0	0	0	0	0	0	0
12	0	0	1	0	0	0	0	0	1	0	0	0	104	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	101	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	95	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	98	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	104	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	105	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	101	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	95