

Configuration Manual

MSc Research Project
Data Analytics

Bharti Mehatari
Student ID: X20175825

School of Computing
National College of Ireland

Supervisor: Dr. Catherine Mulwa

National College of Ireland
MSc Project Submission Sheet
School of Computing



Student Name: Bharti Mehatari
Student ID: X20175825
Programme: MSc in Data Analytics **Year:** 2021 - 2022
Module: MSc Research Project
Lecturer: Dr. Catherine Mulwa
Submission Due Date: 30/01/2022
Project Title: Hand Gesture Recognition and Classification using Computer Vision and Deep Learning Techniques
Word Count: 1300 **Page Count:** 12

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:



Date: 30/01/2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	<input type="checkbox"/>
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	<input type="checkbox"/>
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	<input type="checkbox"/>

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Bharti Mehatari
X20175825

1 Introduction

The configuration manual includes various different kinds of components required to perform the research project. The components consist of the system configuration (both hardware and software specifications) and the entire environment setup with details code artefact snapshots of steps performed for the following tasks:

- a. Data acquisition and loading of hand images
- b. Data pre-processing
- c. Implementation of models for hand gesture recognition and classification
- d. Evaluation of the models

2 System Configuration

2.1 Hardware

[Table 1](#) mentions the available hardware resources for the implementation of this research project.

Table 1- Hardware Components Information

Machine Model	Dell Inspiron
Processor	Intel(R) Core i5-10210U CPU @ 1.60GHz, 2112 Mhz, 4 Core(s), 8 Logical Processor(s)
RAM	8.00 GB
OS	Windows 10
Graphics	None

2.2 Software

Below mentioned are the tools, libraries and programming language used in this project.

- Python 3.6.9
- Google colaboratory pro with 25 GB RAM
- Tensorflow and keras libraries
- OpenCV and pillow libraries for image processing
- Matplotlib and seaborn libraries for visualisation

3 Project Development

3.1 Data Acquisition and Loading

The dataset for this project was selected from Kaggle¹ which was downloaded into the local machine having a size of about 2 GB. Since the dataset size was very huge, it took too long and at time failed to upload the data on Google Colab directly. Therefore, this was done with the help of Google drive by uploading the zip file on the drive and then mounting it onto colab with the help of below command (Figure 1). After loading the entire dataset, the file structure of all the ten gesture images performed by different individuals looks like Figure 2.

```
[2] from google.colab import drive
drive.mount('/content/gdrive')

Mounted at /content/gdrive

!unzip gdrive/My\ Drive/ResearchProject/data.zip

Streaming output truncated to the last 5000 lines.
inflating: data/gestures_data/07/06_index/frame_07_06_0027.png
inflating: data/gestures_data/07/06_index/frame_07_06_0028.png
inflating: data/gestures_data/07/06_index/frame_07_06_0029.png
inflating: data/gestures_data/07/06_index/frame_07_06_0030.png
inflating: data/gestures_data/07/06_index/frame_07_06_0031.png
inflating: data/gestures_data/07/06_index/frame_07_06_0032.png
```

Figure 1- Data loading

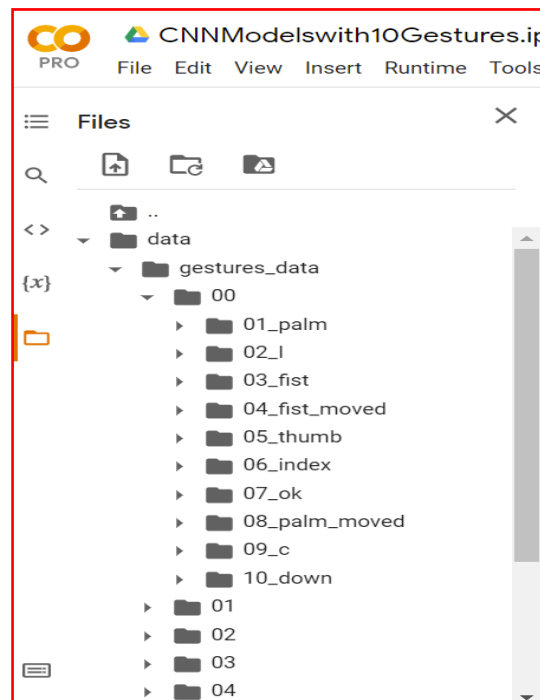


Figure 2- Data structure

¹ <https://www.kaggle.com/gti-upm/leapgestrecog>

All the files are renamed on the basis of initial two characters of the gesture name in order to prepare the images and its corresponding gesture label for Y data. The userfined defined function is created to perform this task as shown below in [Figure 3](#).

```

def rename_file_based_parent_dir(path):
    for directory, subdirectories, files in os.walk(path):
        print(directory)
        print(os.listdir(os.path.abspath(directory)))
        if os.listdir(os.path.abspath(directory)) == []:
            continue
        else:
            if (os.path.isdir(directory)) and len(os.listdir(directory)) >= 11:
                for file in os.listdir(directory):
                    prefix = os.path.basename(directory).split('_')[1][0:2]
                    new_name = os.path.join(os.path.abspath(directory), prefix + os.path.basename(file))
                    os.rename(os.path.join(directory, file), new_name)
    rename_file_based_parent_dir('/content/data')

['frame_05_06_0016.png', 'frame_05_06_0085.png', 'frame_05_06_0160.png', 'frame_05_06_0148.png',
/content/data/gestures_data/05/09_c
['frame_05_09_0084.png', 'frame_05_09_0169.png', 'frame_05_09_0115.png', 'frame_05_09_0101.png',
/content/data/gestures_data/05/01_palm

```

Figure 3- Function to rename files for Y data prep

After running the above function, the file name structure has now changed and the first two characters are appended in the file name as seen below ([Figure 4](#)).



Figure 4- New file names as per renaming functionality

Below ([Figure 5](#)) is a preview of the dataset images consisting of ten hand gestures that are done by cten different subject in order to add variations.

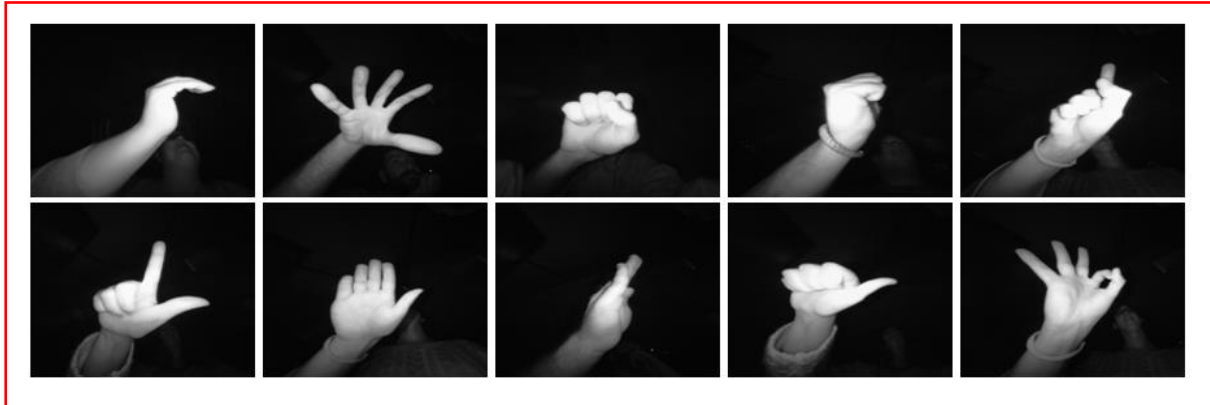


Figure 5- Ten gesture images

3.2 Data Pre-processing

This section contains the main block of code that creates metadata and performs entire pre-processing of 20000 images in a single go. This function begins with locating all the newly renamed files from root directory using regular expressions which are stored as corresponding gesture labels from the stored dictionary for 20000 images in Y data. Then, the commands of OpenCV library accesses the images and performs all the preliminary steps such as image resizing, flipping, converting it into grey scale, image smoothing with the help of Gaussian blur and finally image thresholding provided in [Figure 6](#).

```

X_data = []
y_data = []

root_dir = os.fsencode('/content/data/')

for directory, subdirectories, files in os.walk(root_dir):
    for file in files:
        if not file.startswith(b'.'):
            path = os.path.join(directory, file)
            #gesture_name = int(file.decode('utf8')[10:11])
            gesture_name = re.search('(.*?)frame_(\d{2})_(\d{2})_(\d{4})', os.path.basename(path)).
            if gesture_name:
                gesture_name = gesture_name.group(3)
            if int(gesture_name) in [3, 2, 7, 1, 5, 6, 9, 4, 8, 10]:
                path = os.path.join(directory, file).decode('utf8')
                y_data.append(gestures_map[int(gesture_name)])

                img = cv2.imread(path, cv2.IMREAD_COLOR)
                img = cv2.flip(img, 1)
                gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
                blur = cv2.GaussianBlur(gray, (41, 41), 0) #tuple indicates blur value
                ret, thresh = cv2.threshold(blur, 150, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
                thresh = cv2.resize(thresh, (224, 224))
                thresh = np.array(thresh)
                X_data.append(thresh)
            else:
                continue

```

Figure 6- Code snippet of image processing

Upon the completion of image processing, the final shape of data is printed as per [Figure 7](#).

```

[11] print(f'X_data shape: {X_data.shape}')
      print(f'y_data shape: {y_data.shape}')

X_data shape: (20000, 224, 224)
y_data shape: (20000, 10)

plt.imshow(.5 - X_data[1050])
plt.grid(False)

```

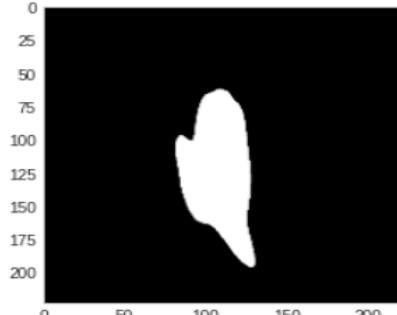


Figure 7- Final Xdata and Ydata shape

Setup of X data and Y data

Figure 8 below illustrates the userdefined functions created to process the core data objects inorder to carry out the gesture recognition operation. As seen from the snippet code artefact, the images are converted to numpy arrays in Python so as to make them as vectors before feeding it into the deep learning network.

```

def process_image(path):
    img = Image.open(path)
    img = img.resize((224, 224))
    img = np.array(img)
    return img

def process_data(X_data, y_data):
    X_data = np.array(X_data, dtype = 'float32')
    #X_data = np.stack((X_data,)*3, axis=-1)
    #X_data /= 255
    y_data = np.array(y_data)
    y_data = to_categorical(y_data)
    return X_data, y_data

def walk_file_tree(relative_path):
    X_data = []
    y_data = []
    for directory, subdirectories, files in os.walk(relative_path):
        for file in files:
            if not file.startswith('.') and (not file.startswith('C_')):
                path = os.path.join(directory, file)
                gesture_name = gestures[file[0:2]]
                y_data.append(gestures_map[gesture_name])
                X_data.append(process_image(path))
            else:
                continue

    X_data, y_data = process_data(X_data, y_data)
    return X_data, y_data

```

Figure 8- Function for X and Y data prep

The gesture labels are stored as dictionaries with labels 0, 1, 2,,8, 9 as seen in Figure 9. Then, all the images are iterated through the pre-processing function and the new label is assigned to each image.

```
gestures = {'L_': 'L',
            'fi': 'Fist',
            'C_': 'C',
            'ok': 'Okay',
            'pa': 'Palm',
            'th': 'Thumb',
            'in': 'Index',
            'fm': 'Fist_Moved',
            'pm': 'Palm_Moved',
            'do': 'Down'
            }

gestures_map = {'Fist' : 0,
                'L': 1,
                'Okay': 2,
                'Palm': 3,
                'Thumb': 4,
                'Index' : 5,
                'C': 6,
                'Fist_Moved': 7,
                'Palm_Moved': 8,
                'Down': 9
                }
```

Figure 9- Gesture labels stored numerically

3.3 Model Implementation

This section mentions all the different kinds of deep learning models that are implemented to solve the research problem of hand gesture recognition and classification. First, all the necessary packages and libraries such as VGG16, Conv2D, confusion matrix, train and test split functions, etc. are imported in [Figure 10](#).

```
import os
import warnings
import cv2
import keras
import matplotlib.pyplot as plt
import matplotlib.style as style
import numpy as np
import pandas as pd
from PIL import Image
from keras import models, layers, optimizers
from keras.applications.vgg16 import VGG16
from keras.callbacks import EarlyStopping, ModelCheckpoint
from keras.layers import Input,Dense, Dropout, Flatten
from keras.models import Model
from keras.preprocessing import image as image_utils
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.utils import to_categorical
import tensorflow as tf
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
import seaborn as sns
from tensorflow.keras.optimizers import SGD

% matplotlib inline
style.use('seaborn-whitegrid')
warnings.filterwarnings(action='once')
```

Figure 10- Importing all the libraries and packages

2D Convolutional Neural Network Model

The implementation is based on 2D Convolutional Neural Networks and [Figure 11](#) demonstrates the how the model is built using three convolutional, three maxpooling and a dense layer. It also involves other regularisation and standardisation techniques such as dropout and batch normalisation. Before building the model, the X and Y data are split into train and test sets in the ratio of 7:3 using the sklearn library.

```
[12] X_train, X_test, y_train, y_test = train_test_split(X_data, y_data, test_size = 0.3, random_state=42, stratify=y_data, )
```

Model Building

```
modelcnn1=models.Sequential()
modelcnn1.add(layers.Conv2D(32, (5, 5), strides=(2, 2), activation='relu', input_shape=(224, 224, 1)))
modelcnn1.add(layers.MaxPooling2D((2, 2)))
modelcnn1.add(BatchNormalization())
modelcnn1.add(layers.Conv2D(64, (3, 3), activation='relu'))
modelcnn1.add(layers.MaxPooling2D((2, 2)))
modelcnn1.add(BatchNormalization())
modelcnn1.add(layers.Conv2D(128, (3, 3), activation='relu'))
modelcnn1.add(layers.MaxPooling2D((2, 2)))
modelcnn1.add(BatchNormalization())

modelcnn1.add(layers.Flatten())
modelcnn1.add(Dropout(0.5))
modelcnn1.add(layers.Dense(128, activation='relu'))
modelcnn1.add(layers.Dense(10, activation='softmax'))
```

Figure 11- 2D CNN Model building

Based on [Figure 12](#) and [Figure 13](#), the models are compiled using both Adam and stochastic gradient descent functions of learning rate = 0.001 with the evaluation metric as accuracy and loss function to be categorical crossentropy since the problem is to classify multiple labels.

```
from tensorflow.keras.optimizers import SGD
opt = SGD(learning_rate=0.001, momentum=0.9)
modelcnn1.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
```

```
[15] hist1 = modelcnn1.fit(X_train, y_train, epochs=15, batch_size=256, verbose=1, validation_data=(X_test, y_test))
```

```
Epoch 1/15
55/55 [=====] - 15s 106ms/step - loss: 1.0608 - accuracy: 0.6773 - val_loss: 0.7734 - val_accuracy: 0.7520
Epoch 2/15
55/55 [=====] - 4s 75ms/step - loss: 0.1449 - accuracy: 0.9627 - val_loss: 0.2040 - val_accuracy: 0.9500
Epoch 3/15
55/55 [=====] - 4s 77ms/step - loss: 0.0691 - accuracy: 0.9838 - val_loss: 0.0626 - val_accuracy: 0.9888
Epoch 4/15
55/55 [=====] - 4s 76ms/step - loss: 0.0449 - accuracy: 0.9919 - val_loss: 0.0298 - val_accuracy: 0.9963
Epoch 5/15
55/55 [=====] - 4s 76ms/step - loss: 0.0331 - accuracy: 0.9939 - val_loss: 0.0176 - val_accuracy: 0.9975
Epoch 6/15
55/55 [=====] - 4s 76ms/step - loss: 0.0253 - accuracy: 0.9958 - val_loss: 0.0117 - val_accuracy: 0.9983
Epoch 7/15
55/55 [=====] - 4s 76ms/step - loss: 0.0194 - accuracy: 0.9972 - val_loss: 0.0082 - val_accuracy: 0.9990
```

Figure 12- Model fitted using SGD optimiser

```
modelcnn.compile(optimizer=tf.keras.optimizers.Adam(), loss='categorical_crossentropy', metrics=['accuracy'])
```

```
hist4 = modelcnn.fit(X_train, y_train, epochs=10, batch_size=64, verbose=1, validation_data=(X_test, y_test))
```

Figure 13- Model fitted using Adam optimiser

Transfer Learning- Pretrained VGG16 Network

Another model is implemented with the help of transfer learning and as per below [Figure 14](#), the VGG16 model which is a pretrained network is depicted. Additional top layers are added over this network as part of the output layer. This model took almost an hour to complete on 30 number of epochs.

```
[14] imageSize = 224
vgg_base = VGG16(weights='imagenet', include_top=False, input_shape=(imageSize, imageSize, 3))
#vgg_base.summary()

base_model = vgg_base # Topless
# Add top layer
x = base_model.output
x = Flatten()(x)
x = Dense(128, activation='relu', name='fc1')(x)
x = Dense(128, activation='relu', name='fc2')(x)
x = Dropout(0.5)(x)
x = Dense(64, activation='relu', name='fc4')(x)
predictions = Dense(7, activation='softmax')(x)

model = Model(inputs=base_model.input, outputs=predictions)

# Train top layers only
for layer in base_model.layers:
    layer.trainable = False

callbacks_list = [keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=3, verbose=1)]

model.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=['accuracy'])

HistVgg1= model.fit(X_train, y_train, epochs=30, batch_size=64, validation_data=(X_train, y_train), verbose=1,
                    callbacks=[early_stopping, model_checkpoint])

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16\_weights\_tf\_dim\_ordering\_tf\_kernels\_notop.h5
58892288/58889256 [=====] - 0s 0us/step
58900480/58889256 [=====] - 0s 0us/step
Epoch 1/30
115/115 [=====] - 49s 314ms/step - loss: 0.7448 - accuracy: 0.7362 - val_loss: 0.0243 - val_accuracy: 0.9944
```

Figure 14- VGG16 pretrained model implementation

Due to the limitations of GPU and RAM memory, the data for the VGG model was reduced to seven gesture labels from ten labels. The dictionary that contained these class labels are also altered ([Figure 15](#))

```
gestures = {'L_': 'L',
            'fi': 'Fist',
            'C_': 'C',
            'ok': 'Okay',
            'pa': 'Palm',
            'th': 'Thumb',
            'in': 'Index'
            }

gestures_map = {'Fist' : 0,
                'L' : 1,
                'Okay' : 2,
                'Palm' : 3,
                'Thumb' : 4,
                'Index' : 5,
                'C' : 6
                }
```

Figure 15- Reduced classes

Early Stopping and Model Checkpoint

The early stopping function as defined in [Figure 16](#) is also added to the models as this helps during the models' processing time and stops when the model is not able to further improve the training. The best weights at this point are saved by putting the model checkpoint method and assigning a location to save it.

```
[12] file_path = '/content/Model1.h5'
      model_checkpoint = ModelCheckpoint(filepath=file_path, save_best_only=True)

      early_stopping = EarlyStopping(monitor='val_accuracy',
                                     min_delta=0,
                                     patience=10,
                                     verbose=1,
                                     mode='auto',
                                     restore_best_weights=True)
```

Figure 16- Additional parameters

Example of early stopping of the VGG16 model is shown below ([Figure 17](#))

```
[ ] Epoch 10/20
58/58 [=====] - 34s 588ms/step - loss: 0.0020 - accuracy: 0.9997 - val_loss: 2.3245e-04 - val_accuracy: 0.9999
Epoch 11/20
58/58 [=====] - 34s 587ms/step - loss: 7.4336e-04 - accuracy: 1.0000 - val_loss: 1.4189e-05 - val_accuracy: 1.0000
Epoch 12/20
58/58 [=====] - 34s 584ms/step - loss: 9.4666e-04 - accuracy: 0.9999 - val_loss: 7.3140e-05 - val_accuracy: 1.0000
Epoch 13/20
58/58 [=====] - 34s 587ms/step - loss: 7.9243e-04 - accuracy: 1.0000 - val_loss: 8.0025e-06 - val_accuracy: 1.0000
Epoch 14/20
58/58 [=====] - 34s 583ms/step - loss: 6.8645e-04 - accuracy: 0.9999 - val_loss: 2.2019e-05 - val_accuracy: 1.0000
Epoch 15/20
58/58 [=====] - 34s 587ms/step - loss: 5.9900e-04 - accuracy: 1.0000 - val_loss: 7.9101e-06 - val_accuracy: 1.0000
Epoch 16/20
58/58 [=====] - ETA: 0s - loss: 4.8883e-04 - accuracy: 0.9999Restoring model weights from the end of the best epoch: 6.
58/58 [=====] - 34s 585ms/step - loss: 4.8883e-04 - accuracy: 0.9999 - val_loss: 2.7848e-05 - val_accuracy: 1.0000
Epoch 00016: early stopping
```

Figure 17- Early stopping

Data Augmentation Model

The data augmentation technique is applied to the model which adds random transformations to the training such as rotation, flips and shifting of the image within the frame, etc. in real-time as seen from [Figure 18](#). This model uses the ImageDataGenerator method which creates such data transformations and the same is fitted to an existing model configuration.

```
[21] datagen = ImageDataGenerator(
      featurewise_center=True,
      featurewise_std_normalization=True,
      rotation_range=45.,
      width_shift_range=0.2,
      height_shift_range=0.2,
      horizontal_flip=True)

[22] augmodel = datagen.fit(X_train)
```

Figure 18- Augmented Data Generation

The model is fitted to the new data generated from the augmentation methods with a batchsize of 128 and number of epochs of 15 as shown in [Figure 19](#).

```

[24] # fits the model on batches with real-time data augmentation:
HistAug = model.fit_generator(datagen.flow(X_train, y_train, batch_size=128),
                             steps_per_epoch=len(X_train) / 128, epochs=15, validation_data=(X_test, y_test))

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: UserWarning: `Model.fit_generator` is deprecated and will be removed
This is separate from the ipykernel package so we can avoid doing imports until

Epoch 1/15
57/57 [=====] - 76s 1s/step - loss: 0.9399 - accuracy: 0.6937 - val_loss: 0.1132 - val_accuracy: 0.9702
Epoch 2/15
57/57 [=====] - 75s 1s/step - loss: 0.2733 - accuracy: 0.9102 - val_loss: 0.2327 - val_accuracy: 0.9229
Epoch 3/15
57/57 [=====] - 75s 1s/step - loss: 0.1863 - accuracy: 0.9438 - val_loss: 0.1536 - val_accuracy: 0.9397
Epoch 4/15
57/57 [=====] - 74s 1s/step - loss: 0.1644 - accuracy: 0.9479 - val_loss: 0.0544 - val_accuracy: 0.9835
Epoch 5/15
57/57 [=====] - 75s 1s/step - loss: 0.1406 - accuracy: 0.9556 - val_loss: 0.1331 - val_accuracy: 0.9486
Epoch 6/15
57/57 [=====] - 74s 1s/step - loss: 0.1040 - accuracy: 0.9698 - val_loss: 0.1052 - val_accuracy: 0.9619
Epoch 7/15
57/57 [=====] - 75s 1s/step - loss: 0.0979 - accuracy: 0.9701 - val_loss: 0.0955 - val_accuracy: 0.9654
Epoch 8/15
57/57 [=====] - 74s 1s/step - loss: 0.0804 - accuracy: 0.9751 - val_loss: 0.1057 - val_accuracy: 0.9673
Epoch 9/15
57/57 [=====] - 75s 1s/step - loss: 0.0865 - accuracy: 0.9740 - val_loss: 0.0938 - val_accuracy: 0.9708
Epoch 10/15
57/57 [=====] - 75s 1s/step - loss: 0.0769 - accuracy: 0.9769 - val_loss: 0.2404 - val_accuracy: 0.9270
Epoch 11/15
57/57 [=====] - 76s 1s/step - loss: 0.0692 - accuracy: 0.9786 - val_loss: 0.0882 - val_accuracy: 0.9759
Epoch 12/15
57/57 [=====] - 75s 1s/step - loss: 0.0688 - accuracy: 0.9790 - val_loss: 0.1212 - val_accuracy: 0.9594
Epoch 13/15
57/57 [=====] - 75s 1s/step - loss: 0.0681 - accuracy: 0.9782 - val_loss: 0.1094 - val_accuracy: 0.9667
Epoch 14/15
57/57 [=====] - 75s 1s/step - loss: 0.0699 - accuracy: 0.9770 - val_loss: 0.1181 - val_accuracy: 0.9629
Epoch 15/15

```

Figure 19- Data Augmented Model run

3.4 Model Evaluation

The evaluation metrics used in this research are the confusion matrix, rate of accuracy and validation loss curve plots. In [Figure 20](#), the first cell prints the overall rate of accuracy of the introduced model. The second cell prints the confusion matrix of all the ten gesture labels with its true versus predicted classification.

```

[16] [loss, acc] = modelcnn1.evaluate(X_test,y_test,verbose=1)
      print("Accuracy: " + str(acc))

188/188 [=====] - 1s 7ms/step - loss: 0.0035 - accuracy: 0.9993
Accuracy:0.9993333220481873

[25] def get_classification_metrics(X_test, y_test):
      pred = modelcnn1.predict(X_test)
      pred = np.argmax(pred, axis=1)
      y_true = np.argmax(y_test, axis=1)
      cnf = confusion_matrix(y_true, pred)
      #print('\n')
      #print(classification_report(y_true, pred))
      plt.figure(figsize = (12,5))
      axs = sns.heatmap(cnf, annot=True, cmap="Blues", fmt="g", linewidths=.5)
      axs.xaxis.set_ticklabels(['Fist', 'L', 'Okay', 'Palm', 'Thumb', 'Index', 'C', 'Fist Moved', 'Palm Moved', 'Down'])
      axs.yaxis.set_ticklabels(['Fist', 'L', 'Okay', 'Palm', 'Thumb', 'Index', 'C', 'Fist Moved', 'Palm Moved', 'Down'])
      plt.yticks(rotation=30)
      plt.show()

```

Figure 20- Code snippets for printing accuracy and confusion matrix

An example of the confusion matrix containing the ten classes is shown in [Figure 21](#).

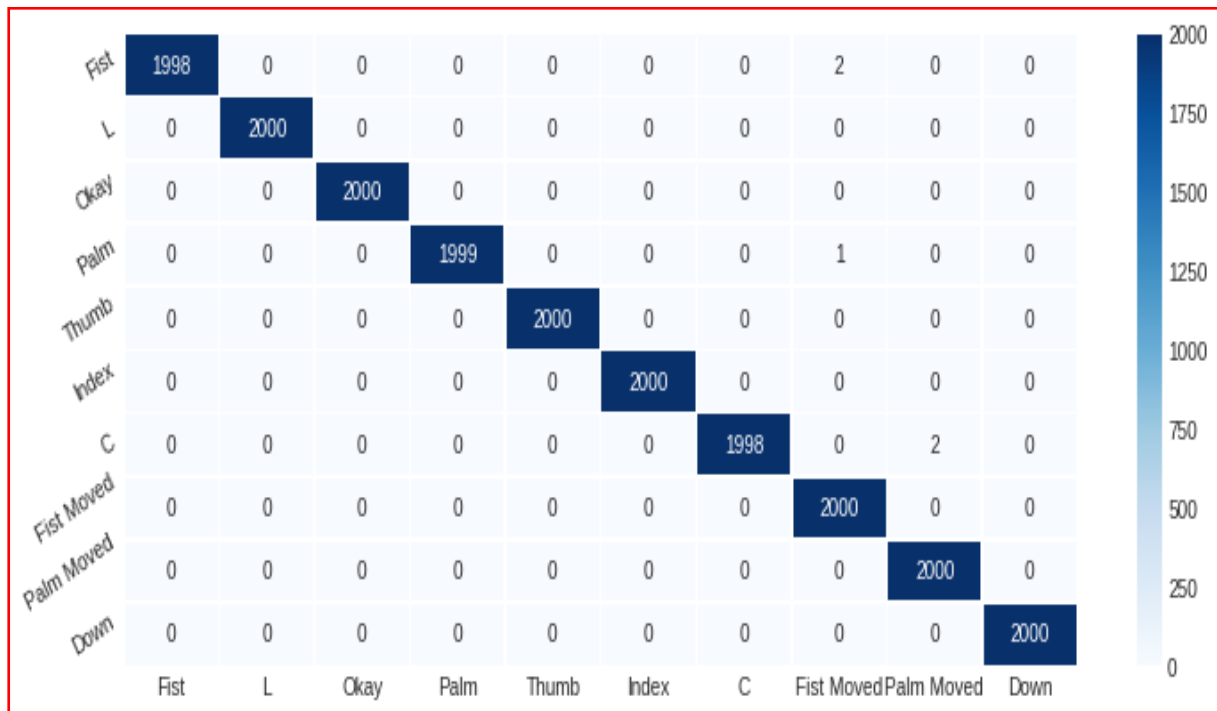


Figure 21- Confusion Matrix of CNN Model

Python’s garbage collector method is used to save up some space in the memory by releasing the unused space ([Figure 22](#))

```
[ ] # Release some unused memory
import gc
gc.collect()

4652
```

Figure 22- Importing garbage collector

Finally, the plot of training verses validation loss is also used to evaluate the model’s performance with the help of matplotlib library as seen generated from below [Figure 23](#).

```
# Plotting the training v/s Validation Loss graph for Model 1
from matplotlib import pyplot
pyplot.plot(hist1.history['loss'], 'g', label='Training_loss')
pyplot.plot(hist1.history['val_loss'], 'b', label='Validation_loss')
pyplot.title('2D CNN Model Training and Validation Loss')
pyplot.xlabel('Epochs')
pyplot.ylabel('Loss')
pyplot.legend()
pyplot.show()
```

Figure 23- Plot for Learning Curves

Example of the loss validation plot is shown below (Figure 24)

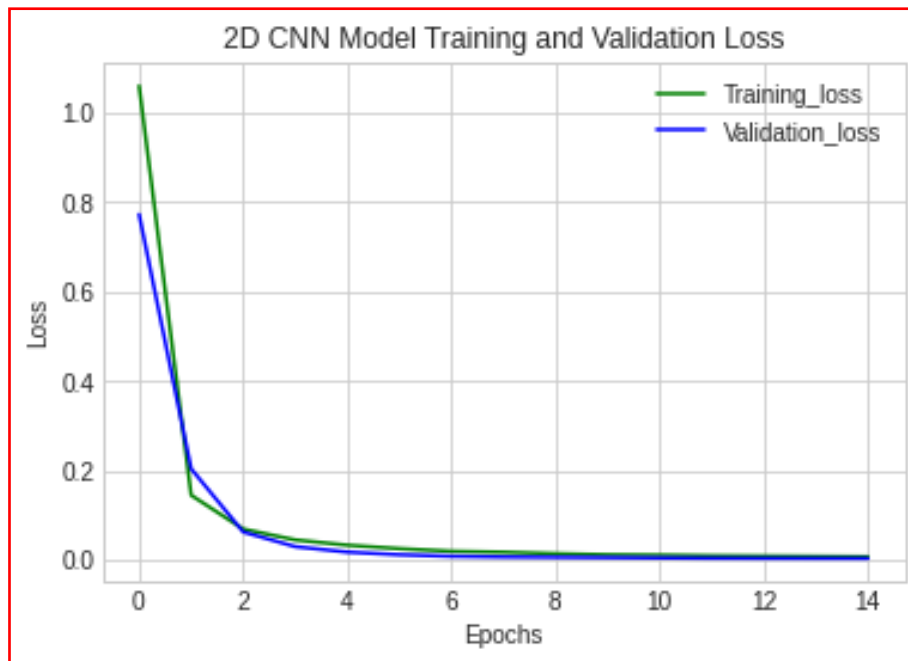


Figure 24- Loss Validation Curve Plot