RECOGNITION OF INDIAN SIGN LANGUAGE THROUGH DEEP LEARNING MODELS

A Project Report submitted in partial fulfilment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

Submitted by

M. Saai Vinieth , 221910308036
Bhavin Patel , 221910308041
Pranav Reddy G , 221910308044

Under the esteemed guidance of

Mr. K.L.Narasimha Rao
Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING GITAM

(Deemed to be University)

HYDERABAD

NOVEMBER-2022

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING GITAM INSTITUTE OF TECHNOLOGY GITAM

(Deemed to be University)

DECLARATION

I/We, hereby declare that the project report entitled "**RECOGNITION OF INDIAN SIGN LANGUAGE THROUGH DEEP LEARNING MODELS**" is an original work done in the Department of Computer Science and Engineering, GITAM Institute of Technology, GITAM (Deemed to be University) submitted in partial fulfilment of the requirements for the award of the degree of B.Tech. in Computer Science and Engineering. The work has not been submitted to any other college or University for the award of any degree or diploma.

Date:

Registration No(s).	**Name(s)**	**Signature(s)**
221910308036	M. SAAI VINIETH	
221910308041	BHAVIN PATEL	
221910308044	PRANAV REDDY G	

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING GITAM INSTITUTE OF TECHNOLOGY GITAM

(Deemed to be University)

CERTIFICATE

This is to certify that the project report entitled “**RECOGNITION OF INDIAN SIGN LANGUAGE THROUGH DEEP LEARNING MODELS**” is a bonafide record of work carried out by **M. SAAI VINIETH (221910308036), BHAVIN PATEL (221910308041), PRANAV REDDY G (221910308044)** students submitted in partial fulfilment of requirement for the award of degree of Bachelors of Technology in Computer Science and Engineering.

AMC	Panel Member	Panel Member	Project Guide	Head of the Department
Dr. Radha Karampudi	**Dr. D.V.N. Sivakumar**	**Dr. Nisanth Kartheek Mukku**	**Mr. K.L.Narasimha Rao**	**Mr. S.Phani Kumar**
Assistant Professor Dept. of CSE	**Assistant Professor Dept. of CSE**	**Assistant Professor Dept. of CSE**	**Assistant Professor Dept. of CSE**	**Professor & HOD Dept. of CSE**

ACKNOWLEDGEMENT

Our project report would not have been successful without the help of several people. We would like to thank the personalities who were part of our seminar in numerous ways, those who gave us outstanding support from the birth of the seminar.

We are extremely thankful to our honourable Pro-Vice-Chancellor, **Prof. D. Sambasiva Rao**,for providing the necessary infrastructure and resources for the accomplishment of our seminar.We are highly indebted to **Prof. N. Seetharamaiah**, Principal, School of Technology, for his support during the tenure of the seminar.

We are very much obliged to our beloved **Prof. S. Phani Kumar**, Head of the Department of Computer Science & Engineering, for providing the opportunity to undertake this seminar and encouragement in the completion of this seminar.

We hereby wish to express our deep sense of gratitude to **Dr. S Aparna**, Assistant Professor,Department of Computer Science and Engineering, School of Technology and to our guide, **Mr. K. L. Narasimha Rao**, Assistant Professor, Department of Computer Science andEngineering, School of Technology for the esteemed guidance, moral support and invaluable advice provided by them for the success of the project report.

We are also thankful to all the Computer Science and Engineering department staff members who have cooperated in making our seminar a success. We would like to thank all our parents and friends who extended their help, encouragement, and moral support directly or indirectly in our seminar work.

Sincerely,

M. Saai Vinieth (221910308036)

Bhavin Patel (221910308041)

Pranav Reddy G (221910308044)

TABLE OF CONTENTS

LIST OF FIGURES

1. Abstract

Communication and thinking capabilities separate us from animals. Communication is essential to express one's thoughts and feelings. People with hearing impairities lack the ability to communicate efficiently with other abled individuals. Non verbal form of communication is the only solution for people suffering from hearing disability to communicate. With the use of hand signs a channel of communication is established between the abled and the disabled. It is essential to build a model for a standard system like the 'Indian Sign Language' that can incorporate people from both sides and ensure effective communication.

With the improvements in Neural Networks and Deep Learning in recent times, problems that revolve around image classification and object detection can be easily predicted and identified. Models based on Neural Networks and Deep Learning can be implemented in predicting hand signs accurately with very high precision.

In this paper, the segmentation of images will be done based on the human skin-tone and the background will be masked, features will then be extracted from the images and will be mapped to their respective labels. A comparison of performance and accuracy between SVM-CNN classifier and ResNet50v2 , Inceptionv3, VGG16 models will be performed. The hypothetical expected accuracy for the SVM-CNN model is about 90-96%, for the ResNet50v2 model it is about 96-98%, for Inceptionv3 is about 96-98% and for VGG16 model is about 90-95%. The motive of this project is to find the best CNN based architecture in recognition of Indian Sign Language.

2. Introduction

Human life has always depended heavily on communication. The ability to interact with others and express ourselves is a fundamental human requirement. However, our perspective and the way we communicate with others can differ greatly depending on our upbringing, education, society, and so on. Furthermore, it is critical to ensure that we are understood in the way that we intend.

Despite this, normal people have little difficulty interacting with one another and can easily express themselves through speech, gestures, body language, reading, and writing, with speech being the most widely used among them. People with speech impairment, on the other hand, rely solely on sign language, making it more difficult for them to communicate with the rest of the population. This necessitates the use of sign language recognizers capable of recognising and converting sign language into spoken or written language and vice versa.

However, such identifiers are limited, expensive, and time-consuming to use. Now, researchers from various countries are working on these sign language recognizers, which is the primary reason for the development of automatic sign language recognition systems.Despite the fact that India is a diverse country with nearly 17.7% of the world's population, very little work has been done in this research area, which is quite contradictory when compared to other countries. The evidence for this can be attributed to delayed standardisation. The study of Indian Sign Language began in India in 1978. However, because no standard type of ISL existed, its use was limited to short-term courses. Furthermore, the gestures used in most deaf schools differed significantly from one another, and nearly 5% of all deaf people attended these schools. ISL became standardised and attracted the attention of researchers in 2003.

Indian Sign Language (ISL) includes both static and dynamic signs, single and double-handed signs, and many signs for the same alphabet in different parts of India. It makes implementing such a scheme extremely difficult. Furthermore, no standard dataset is available.All of this demonstrates the complexities of Indian sign language.

Recently, researchers have begun to investigate this topic. There are primarily two approaches to sign language recognition that are widely used: sensor-based approaches and vision-based approaches. The sensor-based approach employs gloves or other instruments that recognise finger gestures and translate them into equivalent electrical signals for sign determination, whereas the vision-based approach employs web cameras to capture video or images. Because it requires no specialised hardware, vision-based gesture recognition has the advantage of being spontaneous and is preferred by signers.

2.1 Software Requirements

Operating system - An operating system (OS) is a program that acts as an interface between the system hardware and the user. Moreover, it handles all the interactions between the software and the hardware. All the working of a computer system depends on the OS at the base level.Thus, an efficiently performing operating system is required to run all the deep learning algorithms and to perform the needed tasks

Google Colab - Colaboratory, or "Colab" for short, is a product from Google Research. Colab allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education.Google Colab is a powerful platform for learning and quickly developing machine learning models in Python. It is based on the Jupyter notebook and supports collaborative development. The team members can share and concurrently edit the notebooks, even remotely.

2.2 Hardware Requirements

Operating system acts as an interface between the system and the user and allows one to use the system features. Processor depicts the processing compatibility of the system. Ram is used to store non-volatile data for implementing the algorithms and therefore higher the ram faster is the execution. Hard disk is the main storage unit of the dataset of the given application. Camera is needed to take the input or to collect the dataset.These are the minimum requirements one needs to have in one's respective system to implement the application.

- Operating System : Windows / MAC
- Processor : i5 10th gen or above
- Ram : 4gb or above
- Hard Disk : 50 GB or above
- Camera : System integrated

3. Literature Review

3.1 Indian Sign Language recognition system using SURF with SVM and CNN [1]

Hand signs are an effective form of human-to-human communication that has a number of possible applications. Being a natural means of interaction, they are commonly used for communication purposes by speech impaired people worldwide. In fact, about one percent of the Indian population belongs to this category. This is the key reason why it would have a huge beneficial effect on these individuals to incorporate a framework that would understand Indian Sign Language. In this paper, we present a technique that uses the Bag of Visual Words model (BOVW) to recognize Indian sign language alphabets (A-Z) and digits (0–9) in a live video stream and output the predicted labels in the form of text as well as speech. Segmentation is done based on skin colour as well as background subtraction. SURF (Speeded Up Robust Features) features have been extracted from the images and histograms are generated to map the signs with corresponding labels. The Support Vector Machine (SVM) and Convolutional Neural Networks (CNN) are used for classification. An interactive Graphical User Interface (GUI) is also developed for easy access.

3.2 Optical Flow Hand Tracking and Active Contour Hand Shape Features for Continuous Sign Language Recognition with Artificial Neural Networks[2]

To extract hand tracks and hand shape features from continuous sign language videos for gesture classification using backpropagation neural network. Horn Schunck optical flow (HSOF) extracts tracking features and Active Contours (AC) extract shape features. A feature matrix characterises the signs in continuous sign videos. A neural network object with backpropagation training algorithm classifies the signs into various word sequences in digital format. Digital word sequences are translated into text with matching and the suiting text is voice translated using windows application programmable interface (Win-API). Ten signers, each doing sentences having 30 words long, tests the performance of the algorithm by computing word matching score (WMS). The WMS is varying between 88 and 91 percent when executed on different cross platforms on various processors such as Windows8 with Inteli3, Windows 8.1 with inteli3 and Windows10 with inteli3 running MATLAB13(a).

3.3 Recognition of Indian Sign Language using feature fusion[3]

Sign Language is the most natural and expressive way for the hearing impaired. This paper presents a method for automatic recognition of two handed signs of Indian Sign Language (ISL). The method consists of three phases: Segmentation, Feature Extraction and Recognition. The segmentation is done through Otsu's algorithm. In the feature extraction phase, shape descriptors, HOG descriptors (Histogram of Oriented Gradient) and SIFT (Scale Invariant Feature Transform) feature have been fused to compute a feature vector. In the recognition phase, a multi-class Support Vector Machine (MSVM) is used for training and classifying signs of ISL. The experimental results provide evidence of the effectiveness of the proposed approach with 93% recognition rate.

3.4 Recognition of Indian Sign Language in Live Video[4]

Sign Language Recognition has emerged as one of the important areas of research in Computer Vision. The difficulty faced by the researchers is that the instances of signs vary with both motion and appearance. Thus, in this paper a novel approach for recognizing various alphabets of Indian Sign Language is proposed where continuous video sequences of the signs have been considered. The proposed system comprises three stages: Preprocessing stage, Feature Extraction and Classification. Preprocessing stage includes skin filtering, histogram matching. Eigenvalues and EigenVectors were considered for the feature extraction stage and finally Eigen value weighted Euclidean distance is used to recognize the sign. It deals with bare hands, thus allowing the user to interact with the system in a natural way. We have considered 24 different alphabets in the video sequences and attained a success rate of 96.25%.

4. Problem Identification & Objectives:

4.1 Problem Statement:

The main objective of this project is to implement deep learning models that can accurately predict Indian Sign Language hand signs. To reach this objective we make a comparison between the base paper CNN model , with other proposed deep learning models. Conclusions will be deduced based on their performance metrics.

4.2 Existing System:

Sign language recognition requires efficient and robust data to design a highly accurate system that would be helpful for real-time users. Here the base paper authors have used the custom-built dataset to solve the sign detection and classification problem. Features from the dataset are extracted using SURF (Speeded Up Robust Features), and the classification of the Indian Sign Language is done using SVM and CNN models.

4.3 Flaws and Disadvantages Observed:

- Masking of the images used for training the model were improperly done, which can be improved.
- Time and performance of the existing system is high and outdated.

4.4 Proposed System:

To address this issue, we will be implementing the algorithms from the base paper and extract features using CNN and then compare them with the proposed CNN based architectures which are ResNet50v2, Inceptionv3 and VGG16. The proposed algorithms will improve the system's performance with more accurate predictions.

4.5 Functional Requirements:

- Data Collection
- Data Preprocessing
- Training And Testing
- Modelling
- Predicting

4.6 Non Functional Requirements:

The quality features of a software system are described by non-functional requirements (NFRs). They evaluate software according to standards other than those that take responsiveness, usability, security, portability, and other crucial factors into account.

Google Colab:

- **Ram Usage:** Free-tier Colab will almost always provide ~12 GB of RAM with limited access to high-memory VMs which have 25 GB RAM. Colab Pro increases availability of high-memory VMs (32 GB RAM), while Colab Pro+ extends high memory VMs to 52 GB RAM.

- **GPU Usage:** Most notable is that the majority of free Colab sessions will initialise with a K80 GPU and 12 GB of RAM.

- **TPU Usage:** TPU Tensor Processing Unit is a custom-built integrated circuit developed specifically for machine learning and tailored for TensorFlow, Google's open-source machine learning framework

- **Runtime:** In the version of Colab that is free of charge notebooks can run for at most 12 hours, depending on availability and your usage patterns. Colab Pro, Pro+, and Pay As You Go offer you increased compute availability based on your compute unit balance.

5. System Design

5.1 Proposed System Architecture:

5.1.1 ResNet50v2:

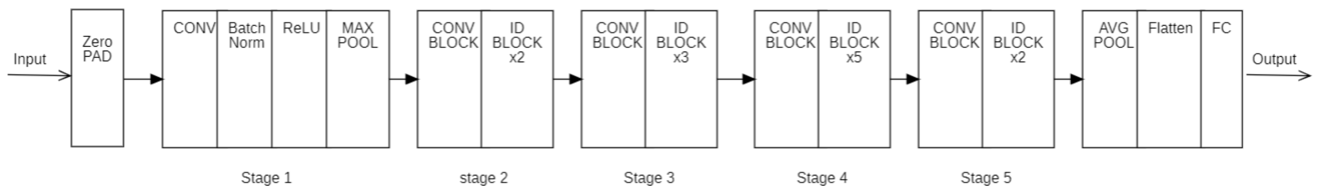


Fig 5.1.1 ResNet50v2

5.1.2 VGG-16:

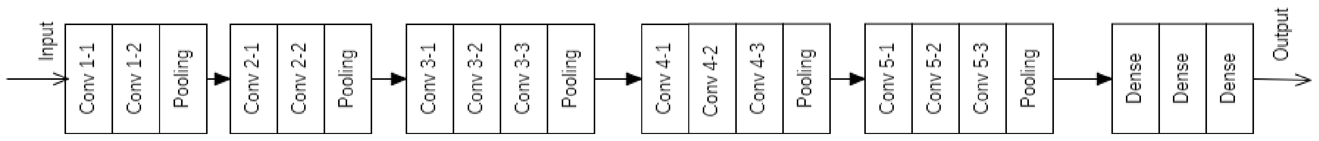


Fig 5.1.2 VGG-16

5.1.3 Inception-v3:

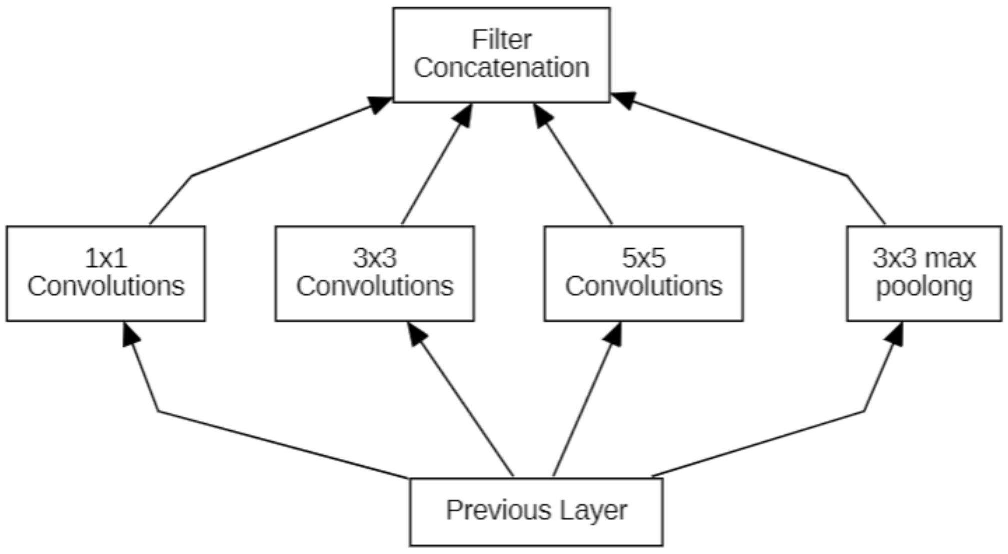


Fig 5.1.3 Inception-v3

5.2 Data Flow Diagrams:

Dataflow is the movement of data through a system comprised of software, hardware or a combination of both. The following is the flow diagram of the proposed approach.

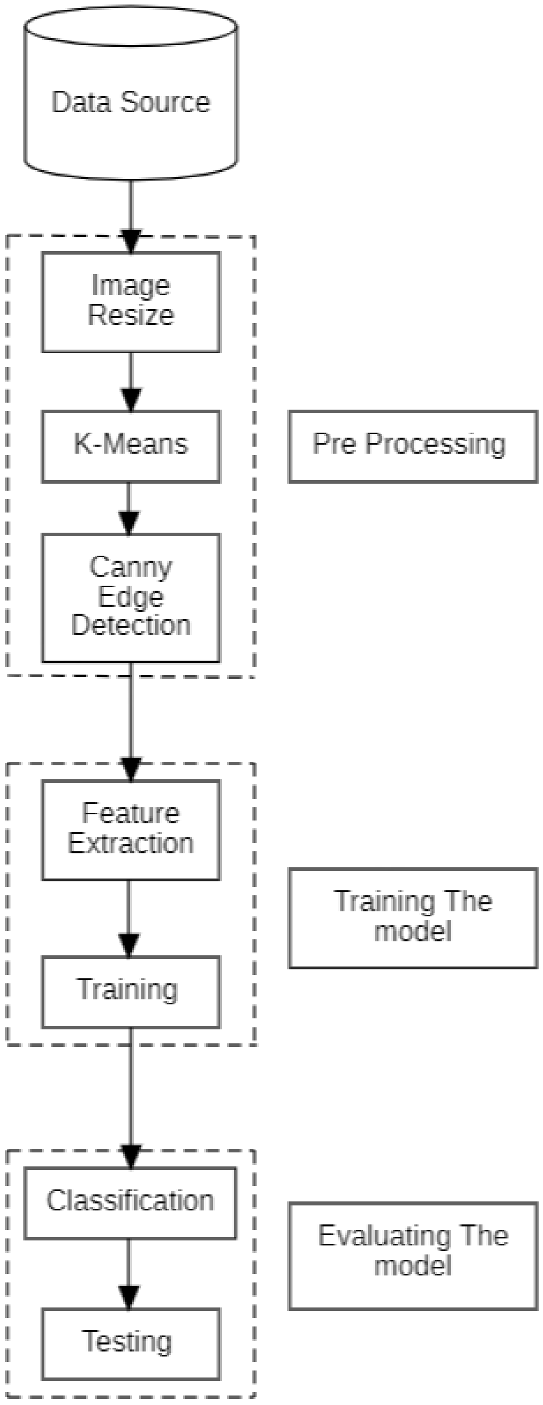


Fig 5.2.1 Data Flow Diagram

5.3 UML Diagrams:

The Unified Modeling Language (UML) is a general-purpose, developmental modelling language in the field of software engineering that is intended to provide a standard way to visualise the design of a system.
It is a modelling language that is most often used for software engineering but has extended its use to business processes and other project workflows. Essentially, UML is visualising software through diagrams, specifically one of the thirteen UML diagrams.

5.3.1 Advantages:

The following are some of the reasons why UML is beneficial for software developers:

- UML helps in communication.
- Drastically saves time.
- Enhances collaboration.
- Maximises a better understanding of a system.
- Unifies design.
- Better structural diagrams.
- Behavioural diagrams.
- Keep it simple and easy to understand for the user.

5.3.2 Use Case Diagram:

In UML, use-case diagrams model the behaviour of a system and help to capture the requirements of the system. Use-case diagrams describe the high-level functions and scope of a system. These diagrams also identify the interactions between the system and its actors.

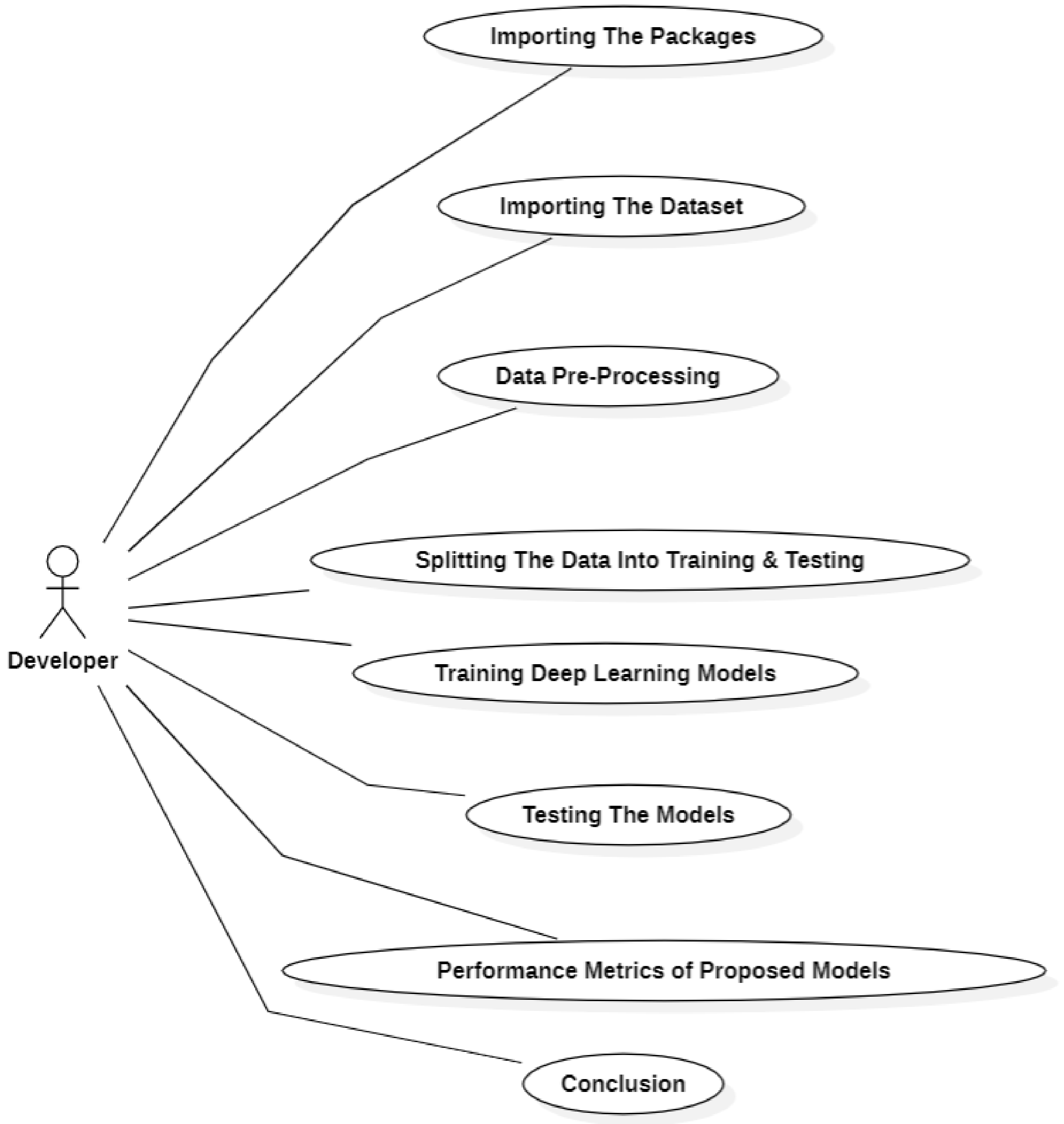

Importing The Packages
Importing The Dataset
Data Pre-Processing
Splitting The Data Into Training & Testing
Developer
Training Deep Learning Models
Testing The Models
Performance Metrics of Proposed Models
Conclusion

Fig 5.3.2.1 Use Case Diagram

5.3.3 Class Diagram:

Class diagrams are the blueprints of your system or subsystem. You can use class diagrams to model the objects that make up the system, to display the relationships between the objects, and to describe what those objects do and the services that they provide.

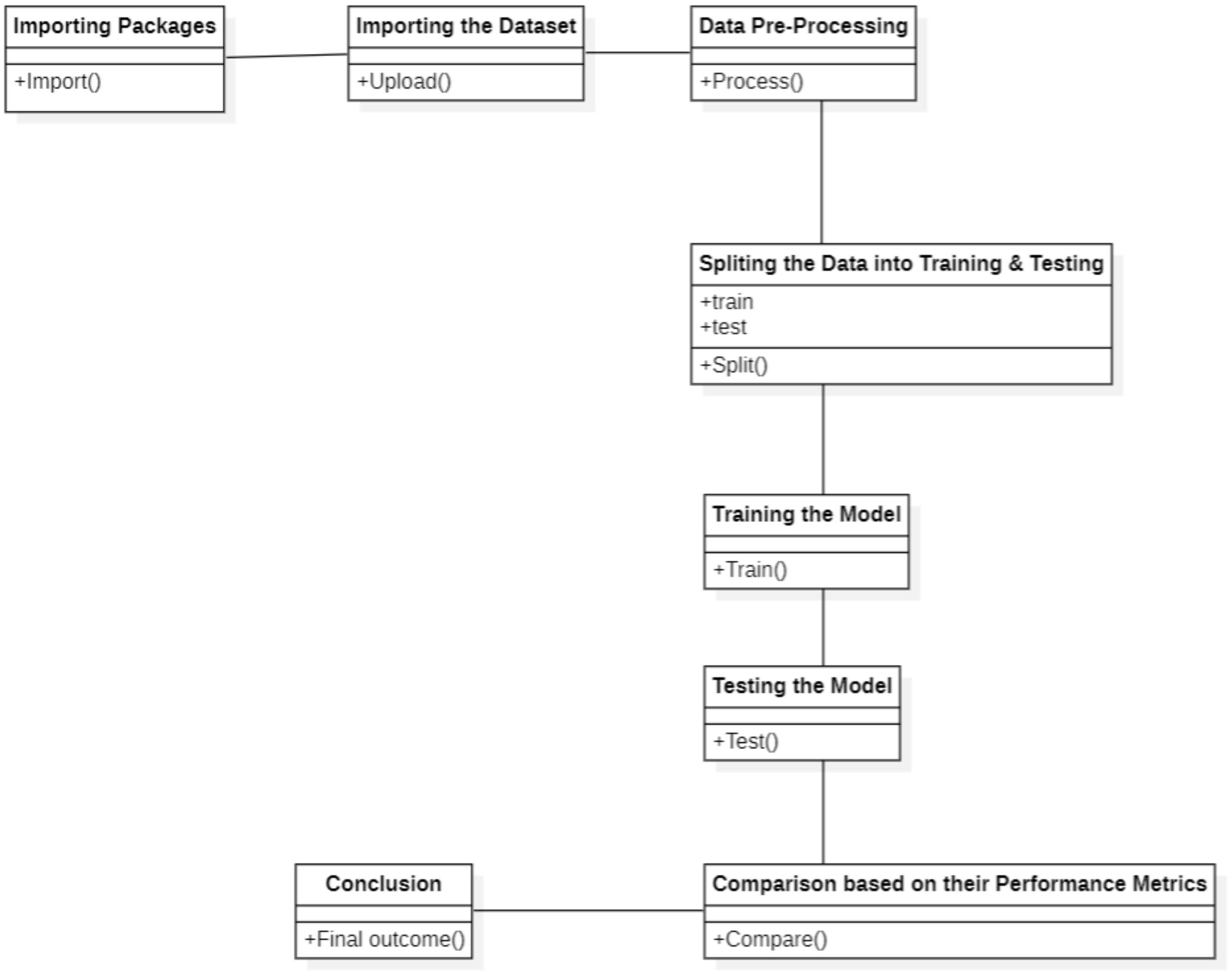


Fig 5.3.3.1 Class Diagram

5.3.4 Activity Diagram:

An activity diagram is a behavioural diagram i.e. it depicts the behaviour of a system. An activity diagram portrays the control flow from a start point to a finish point showing the various decision paths that exist while the activity is being executed.

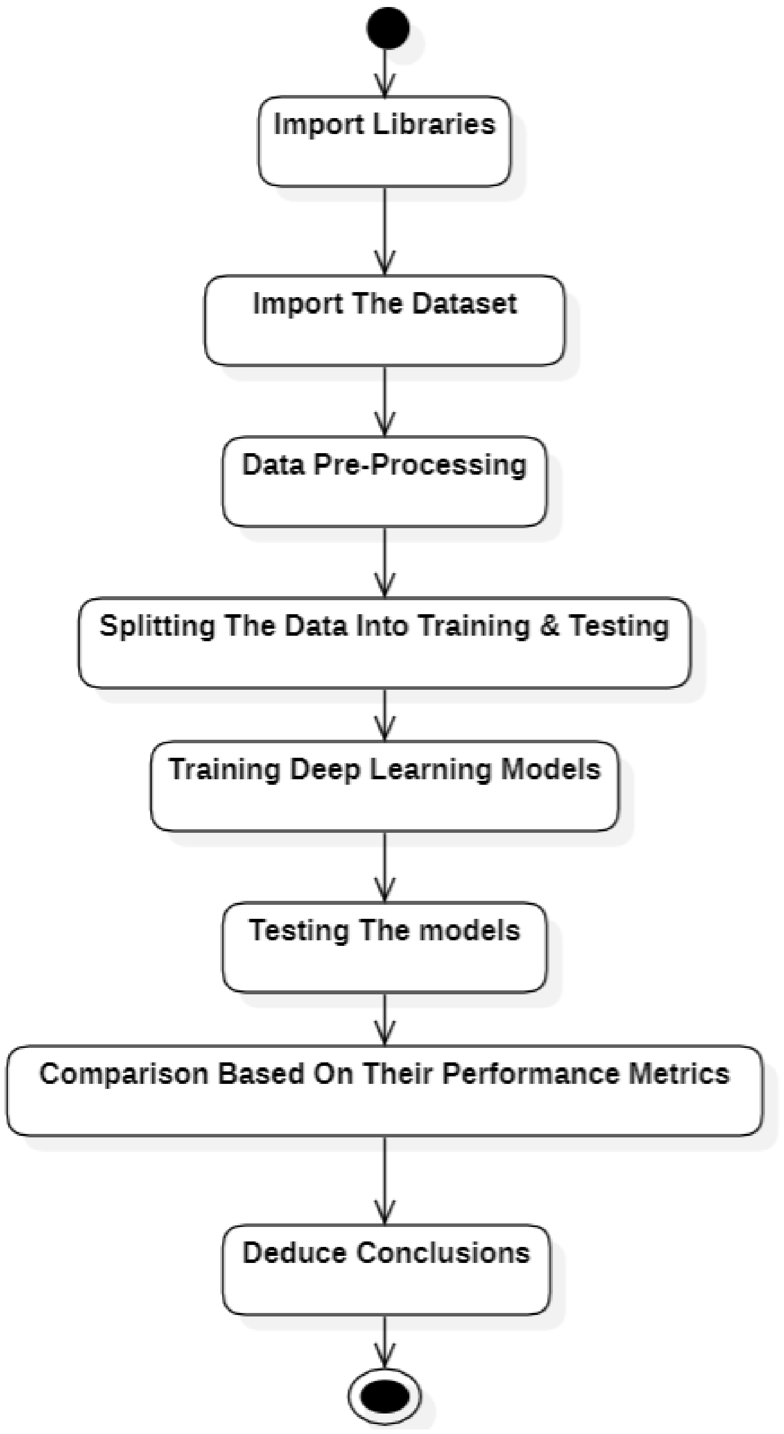

Import Libraries
Import The Dataset
Data Pre-Processing
Splitting The Data Into Training & Testing
Training Deep Learning Models
Testing The models
Comparison Based On Their Performance Metrics
Deduce Conclusions

Fig 5.3.4.1 Activity Diagram

5.3.5 Sequence diagram:

A sequence diagram is a Unified Modeling Language (UML) diagram that illustrates the sequence of messages between objects in an interaction. A sequence diagram consists of a group of objects that are represented by lifelines, and the messages that they exchange over time during the interaction.

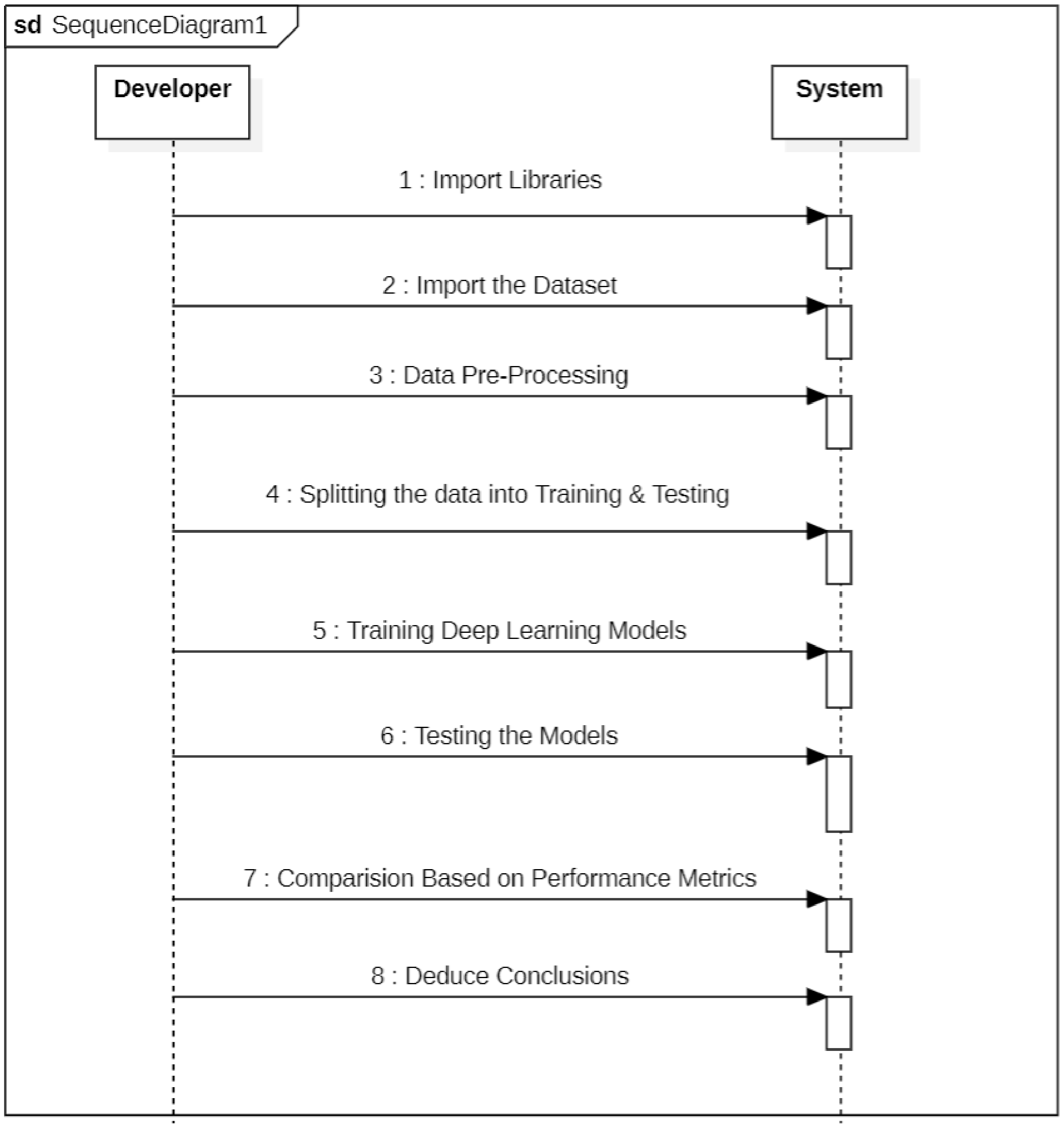


Fig 5.3.5.1 Sequence diagram

6. Overview of Technologies

6.1 Python:

Python is a very popular programming language. He was discovered by Guido Van Rossum. Open source, Python is a language with a large number of built-in functions. It is the language that is the simplest to learn due to its huge libraries and modules. Sklearn, numpy, and pandas are mostly used in the project for output prediction and data structuring.

6.2 Google Colab:

Google Colab is a publicly available online open repository from Google. The implementation of individual cell Python is aided by the Google Collaborations. Working on Google Collaborate is significantly easier to get the best outcome. Each cell is operated separately by the programme. The major advantage is how much easier debugging is. The document is automatically stored, which has the added benefit of simplifying work. Additionally, it offers free GPUs, which we can use to swiftly train the models..

6.3 VS Code:

Microsoft created Visual Studio Code, popularly known as VS Code, a source-code editor for Windows, Linux, and macOS that makes use of the Electron Framework. Among the features are debugging assistance, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git.

7. Implementation

7.1 Workflow :

- **Import Packages:** Import necessary packages and libraries required for the code.
- **Import Dataset:** Download the dataset and import it via mounting google drive.
- **Data Preprocessing:** Preprocess the data.
- **Split Dataset:** Here, split the dataset into training and testing.
- **Building the model:** Use the required architecture, and then train the model.
- **Testing :** Use the test dataset, to check for accuracy and F1 score.

7.2 Libraries Imported :

7.2.1 Tensorflow :

A free and open-source software library called TensorFlow is used for differentiable programming and data flow across a variety of activities. It is a symbolic maths library that is also utilised by neural network applications in machine learning. Google uses it for both research and production. The Google Brain team created TensorFlow for usage within Google. On November 9, 2015, it was made available under the Apache 2.0 open-source licence.

7.2.2 Numpy :

A general-purpose array processing package is called Numpy. It offers a multidimensional array object with outstanding speed as well as capabilities for interacting with these arrays.
It is the cornerstone Python module for scientific computing. It has a number of characteristics, including the following crucial ones: Sophisticated (broadcasting) functions; a strong N-dimensional array object; Effective linear algebra, Fourier transform, and random number capabilities. Tools for integrating C/C++ and Fortran programs.In addition to its apparent scientific applications, Numpy is a powerful multi-dimensional data container. Numpy's ability to establish any data type makes it possible for Numpy to quickly and easily interact with a wide range of databases.

7.2.3 Pandas :

Pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. It is free software released under the three-clause BSD licence.

7.2.4 Scikit – learn :

Through a standardised Python interface, Scikit-learn offers a variety of supervised and unsupervised learning techniques. It is released under several Linux distributions and is available under a liberal simplified BSD licence, which promotes both academic and commercial use.

7.3 Algorithms Implemented:

7.3.1 Gaussian Blur :

Gaussian blurs the image to reduce the amount of noise and remove speckles within the image. It is important to remove the very high frequency components that exceed those associated with the gradient filter used, otherwise, these can cause false edges to be detected.

7.3.2 Kmeans :

K-Means clustering algorithm is an unsupervised algorithm and it is used to segment the interest area from the background. It clusters, or partitions the given data into K-clusters or parts based on the K-centroids. The algorithm is used when you have unlabeled data(i.e. data without defined categories or groups). The goal is to find certain groups based on some kind of similarity in the data with the number of groups represented by K.

7.3.3 Canny Edge Detection :

Canny edge detection is a technique to extract useful structural information from different vision objects and dramatically reduce the amount of data to be processed. It has been widely applied in various computer vision systems.

7.4 Architecture’s Implemented :

7.4.1 ResNet50v2 :

ResNet50V2 [36] is a modified version of ResNet50 that performs better than ResNet50 and ResNet101 on the ImageNet dataset. In ResNet50V2, a modification was made in the propagation formulation of the connections between blocks. ResNet50V2 also achieves a good result on the ImageNet dataset.

7.4.2 VGG16 :

VGG-16 is a convolutional neural network that is 16 layers deep. You can load a pre-trained version of the network trained on more than a million images from the ImageNet database [1]. The pretrained network can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals.

7.4.3 Inceptionv3 :

The Inception V3 model used several techniques for optimising the network for better model adaptation. It has a deeper network compared to the Inception V1 and V2 models, but its speed isn't compromised. It is computationally less expensive. It uses auxiliary Classifiers as regularizers.

7.5 Sample Code Snippets :

7.5.1 Code for Importing Libraries:

```
import numpy as np
from time import time
import pandas as pd
import os
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import cv2
import sklearn
```

```
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import utils
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn import metrics
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import load_img, img_to_array, array_to_img
```

7.5.2 Code for Importing Libraries:

```
#Canny-Edge-Detection
def canny(image):
 Picking_Img = cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
 Gray_Img = cv2.cvtColor(Picking_Img,cv2.COLOR_RGB2GRAY)
 def threshold_vision(img_path):
   _,threshold_Img = cv2.threshold(Gray_Img,90,255,cv2.THRESH_BINARY_INV)
   return threshold_Img
 Threshold_Img = threshold_vision(image)
 canny_img = cv2.Canny(Threshold_Img,10,100)
 return(canny_img)
img = cv2.imread('/content/gdrive/MyDrive/Indian/P/0.jpg')
img = canny(img)
img = cv2.resize(img, (128, 128))
plt.imshow(img,cmap="gray")
```

7.5.3 Code for Training a CNN Model :

```
epochs = 10
learning_rate = 0.0001
adam = Adam(lr=learning_rate)
base_model.compile(optimizer=adam, loss='categorical_crossentropy', metrics=['accuracy'])
history = base_model.fit(
            x_train, y_train,
            epochs=epochs,
            verbose=1,
            validation_data=(x_test, y_test), shuffle=True)
bptime=time()-start
```

7.5.4 Code for Importing an Architecture :

```
from tensorflow.keras.applications import ResNet50V2
resnet50v2arch= ResNet50V2(input_shape=(75, 75, 3), include_top=False, weights="imagenet")
resnet50v2arch.trainable = False
```

7.6 System Testing :

7.6.1. Accuracy :

Accuracy is the most innate performance measure and is simply a ratio of correctly predicted observations to the total observations. In terms of true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN), the formula of the accuracy can be written as

Accuracy = TP + TN / TP + FP + FN + TN

7.6.2. Performance metrics :

Precision is the ratio of correctly predicted positive observations to the total observations which are positive. The recall is the ratio of correctly predicted positive labels to the total no. of labels which are positive whereas the F1 score is the weighted average of precision and recall.

Precision = TP / TP + FP

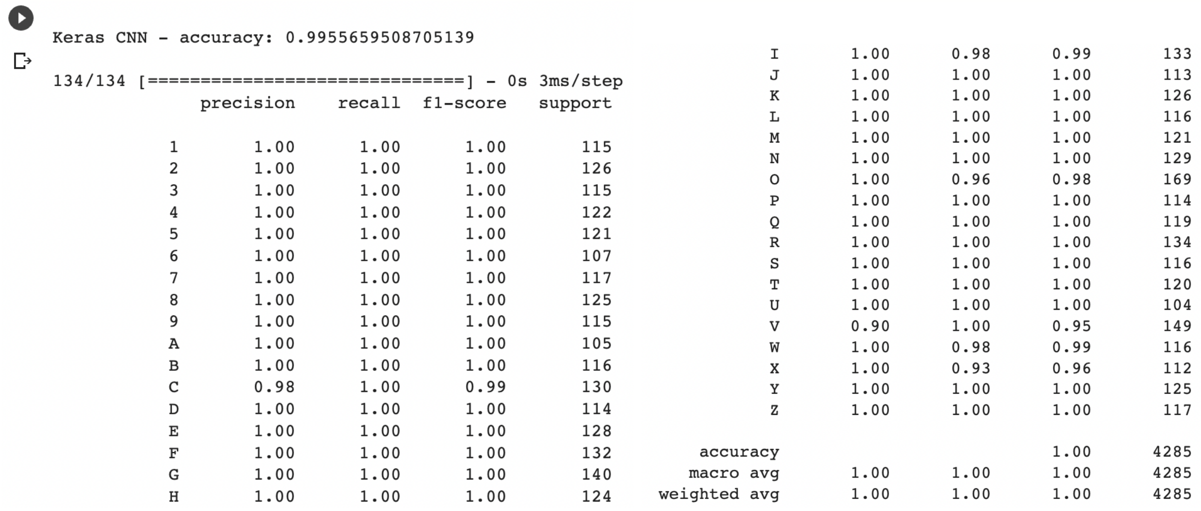
Recall =TP/TP + FN

F1 Score =2*(Recall*Precision)/Recall + Precision

7.6.3 Testing Code Snippet for Performance Metrics

```
score = base_model.evaluate(x_test,y_test, verbose=0)
print('\nKeras CNN - accuracy:', score[1], '\n')
y_pred = base_model.predict(x_test, verbose = 1)
Y_pred_classes = np.argmax(y_pred,axis = 1)
```

```
Y_true = np.argmax(y_test,axis = 1)
report = metrics.classification_report(Y_true, Y_pred_classes, target_names=class_labels)
print(report)
```



```
Keras CNN - accuracy: 0.9955659508705139

134/134 [==============================] - 0s 3ms/step
              precision    recall  f1-score   support

           1       1.00      1.00      1.00       115
           2       1.00      1.00      1.00       126
           3       1.00      1.00      1.00       115
           4       1.00      1.00      1.00       122
           5       1.00      1.00      1.00       121
           6       1.00      1.00      1.00       107
           7       1.00      1.00      1.00       117
           8       1.00      1.00      1.00       125
           9       1.00      1.00      1.00       115
           A       1.00      1.00      1.00       105
           B       1.00      1.00      1.00       116
           C       0.98      1.00      0.99       130
           D       1.00      1.00      1.00       114
           E       1.00      1.00      1.00       128
           F       1.00      1.00      1.00       132
           G       1.00      1.00      1.00       140
           H       1.00      1.00      1.00       124
           I       1.00      0.98      0.99       133
           J       1.00      1.00      1.00       113
           K       1.00      1.00      1.00       126
           L       1.00      1.00      1.00       116
           M       1.00      1.00      1.00       121
           N       1.00      1.00      1.00       129
           O       1.00      0.96      0.98       169
           P       1.00      1.00      1.00       114
           Q       1.00      1.00      1.00       119
           R       1.00      1.00      1.00       134
           S       1.00      1.00      1.00       116
           T       1.00      1.00      1.00       120
           U       1.00      1.00      1.00       104
           V       0.90      1.00      0.95       149
           W       1.00      0.98      0.99       116
           X       1.00      0.93      0.96       112
           Y       1.00      1.00      1.00       125
           Z       1.00      1.00      1.00       117

    accuracy                           1.00      4285
   macro avg       1.00      1.00      1.00      4285
weighted avg       1.00      1.00      1.00      4285
```

Fig 7.6.3.1 Performance Metrics for CNN Model

8. System Results

8.1 Gaussian Blur

```
<matplotlib.image.AxesImage at 0x7fcbf77c0950>
```

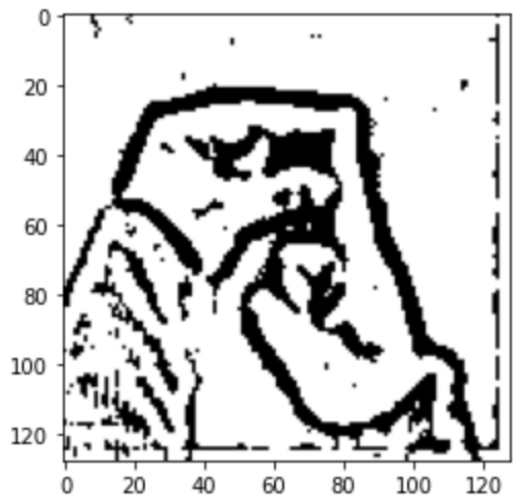


Fig 8.1.1Gaussian Blur

8.2 Canny-Edge Detection

```
<matplotlib.image.AxesImage at 0x7fcbf773b4d0>
```

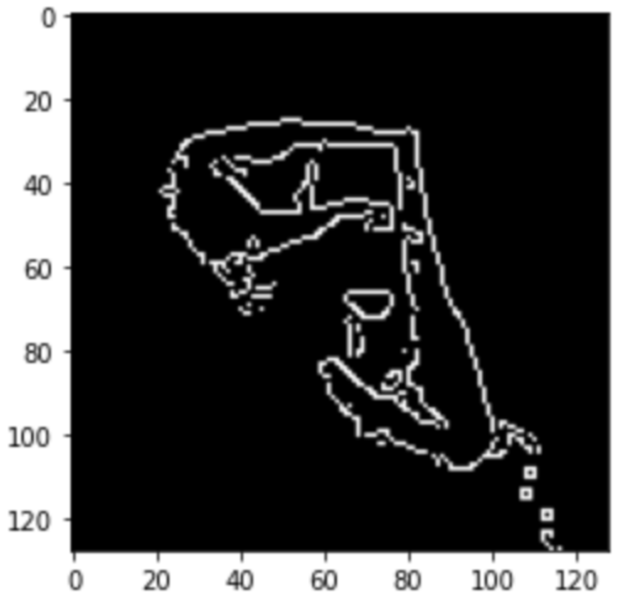


Fig 8.2.1 Canny-Edge Detection

8.3 K-Means

```
<matplotlib.image.AxesImage at 0x7fcbf77276d0>
```

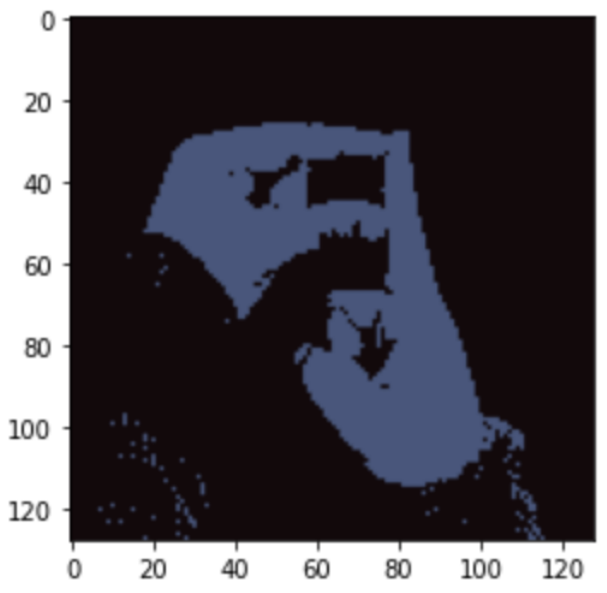
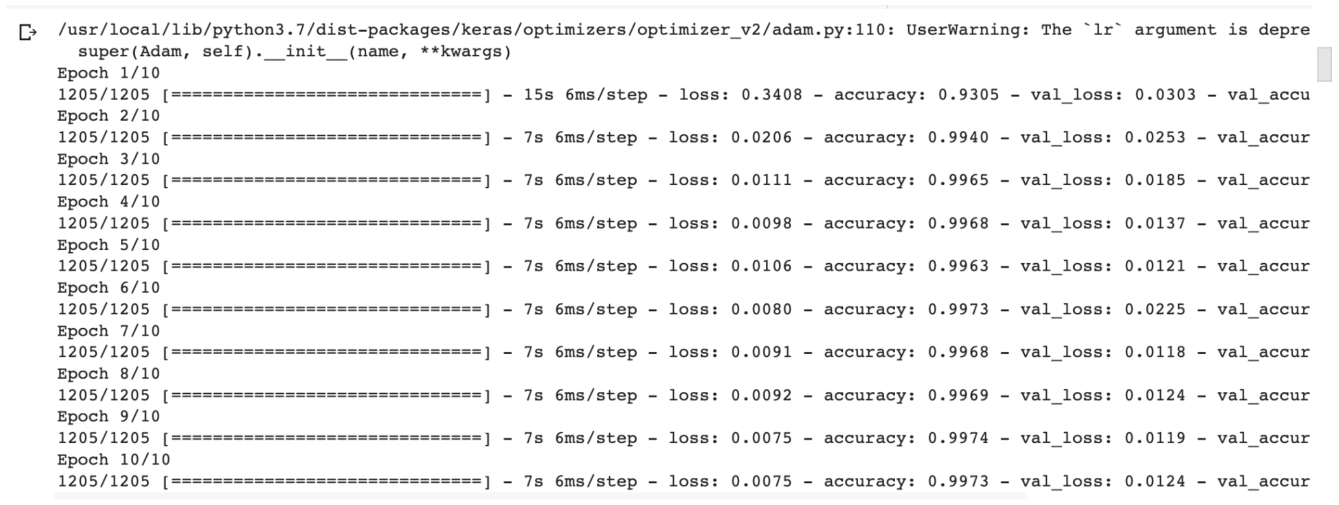


Fig 8.3.1 K-Means

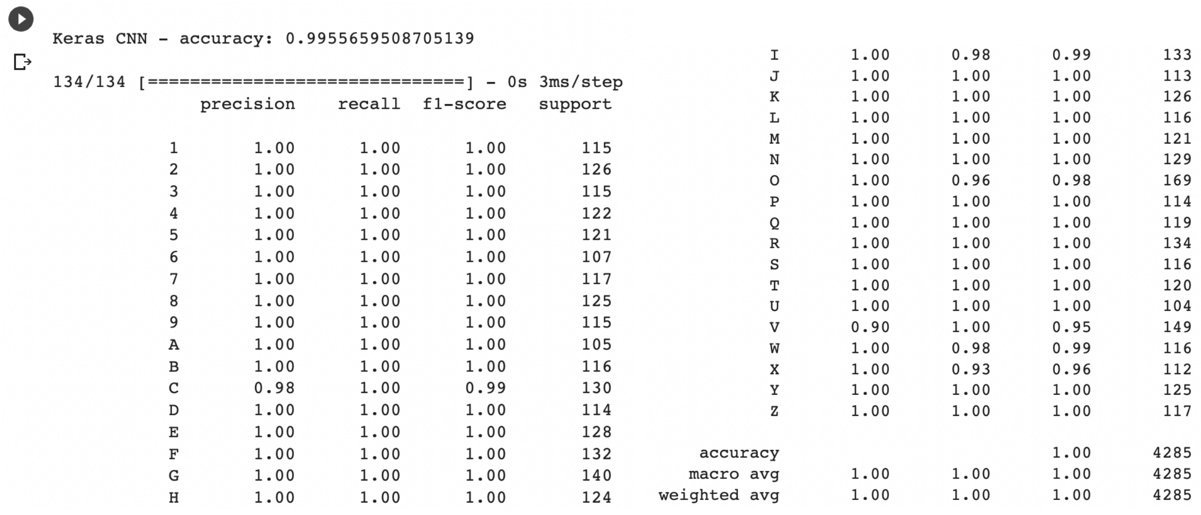
8.4 Training Keras CNN Model



```
/usr/local/lib/python3.7/dist-packages/keras/optimizers/optimizer_v2/adam.py:110: UserWarning: The `lr` argument is depre
  super(Adam, self).__init__(name, **kwargs)
Epoch 1/10
1205/1205 [==============================] - 15s 6ms/step - loss: 0.3408 - accuracy: 0.9305 - val_loss: 0.0303 - val_accu
Epoch 2/10
1205/1205 [==============================] - 7s 6ms/step - loss: 0.0206 - accuracy: 0.9940 - val_loss: 0.0253 - val_accur
Epoch 3/10
1205/1205 [==============================] - 7s 6ms/step - loss: 0.0111 - accuracy: 0.9965 - val_loss: 0.0185 - val_accur
Epoch 4/10
1205/1205 [==============================] - 7s 6ms/step - loss: 0.0098 - accuracy: 0.9968 - val_loss: 0.0137 - val_accur
Epoch 5/10
1205/1205 [==============================] - 7s 6ms/step - loss: 0.0106 - accuracy: 0.9963 - val_loss: 0.0121 - val_accur
Epoch 6/10
1205/1205 [==============================] - 7s 6ms/step - loss: 0.0080 - accuracy: 0.9973 - val_loss: 0.0225 - val_accur
Epoch 7/10
1205/1205 [==============================] - 7s 6ms/step - loss: 0.0091 - accuracy: 0.9968 - val_loss: 0.0118 - val_accur
Epoch 8/10
1205/1205 [==============================] - 7s 6ms/step - loss: 0.0092 - accuracy: 0.9969 - val_loss: 0.0124 - val_accur
Epoch 9/10
1205/1205 [==============================] - 7s 6ms/step - loss: 0.0075 - accuracy: 0.9974 - val_loss: 0.0119 - val_accur
Epoch 10/10
1205/1205 [==============================] - 7s 6ms/step - loss: 0.0075 - accuracy: 0.9973 - val_loss: 0.0124 - val_accur
```

Fig 8.4.1 Keras CNN Model

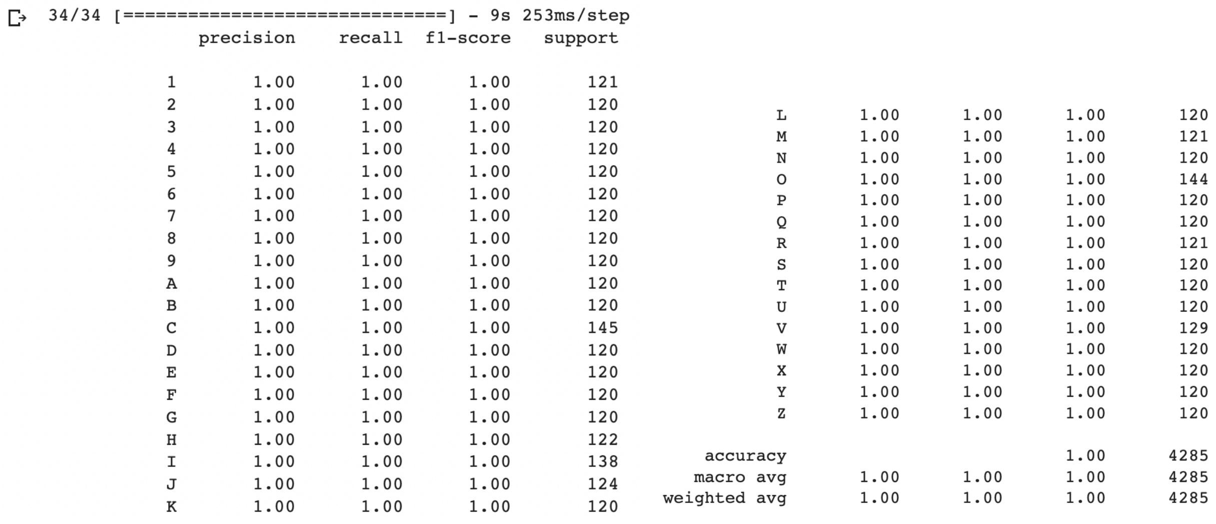
8.5 Performance Metrics for CNN Model



```
Keras CNN - accuracy: 0.9955659508705139

134/134 [==============================] - 0s 3ms/step
              precision    recall  f1-score   support

           1       1.00      1.00      1.00       115
           2       1.00      1.00      1.00       126
           3       1.00      1.00      1.00       115
           4       1.00      1.00      1.00       122
           5       1.00      1.00      1.00       121
           6       1.00      1.00      1.00       107
           7       1.00      1.00      1.00       117
           8       1.00      1.00      1.00       125
           9       1.00      1.00      1.00       115
           A       1.00      1.00      1.00       105
           B       1.00      1.00      1.00       116
           C       0.98      1.00      0.99       130
           D       1.00      1.00      1.00       114
           E       1.00      1.00      1.00       128
           F       1.00      1.00      1.00       132
           G       1.00      1.00      1.00       140
           H       1.00      1.00      1.00       124
           I       1.00      0.98      0.99       133
           J       1.00      1.00      1.00       113
           K       1.00      1.00      1.00       126
           L       1.00      1.00      1.00       116
           M       1.00      1.00      1.00       121
           N       1.00      1.00      1.00       129
           O       1.00      0.96      0.98       169
           P       1.00      1.00      1.00       114
           Q       1.00      1.00      1.00       119
           R       1.00      1.00      1.00       134
           S       1.00      1.00      1.00       116
           T       1.00      1.00      1.00       120
           U       1.00      1.00      1.00       104
           V       0.90      1.00      0.95       149
           W       1.00      0.98      0.99       116
           X       1.00      0.93      0.96       112
           Y       1.00      1.00      1.00       125
           Z       1.00      1.00      1.00       117

    accuracy                           1.00      4285
   macro avg       1.00      1.00      1.00      4285
weighted avg       1.00      1.00      1.00      4285
```

Fig 8.5.1 Performance Metrics for CNN Model

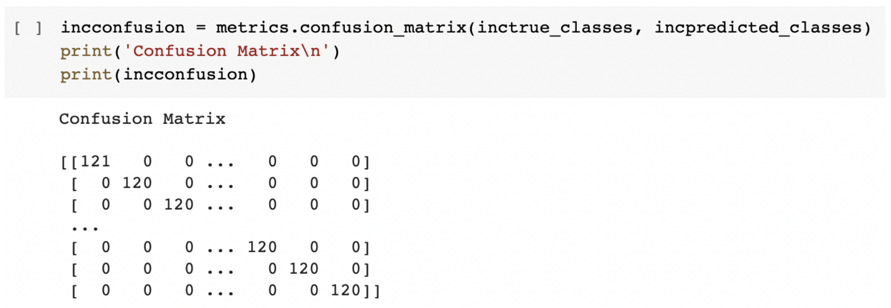
8.6 Performance Metrics for Inceptionv3 Model



```
34/34 [==============================] - 9s 253ms/step
              precision    recall  f1-score   support

           1       1.00      1.00      1.00       121
           2       1.00      1.00      1.00       120
           3       1.00      1.00      1.00       120
           4       1.00      1.00      1.00       120
           5       1.00      1.00      1.00       120
           6       1.00      1.00      1.00       120
           7       1.00      1.00      1.00       120
           8       1.00      1.00      1.00       120
           9       1.00      1.00      1.00       120
           A       1.00      1.00      1.00       120
           B       1.00      1.00      1.00       120
           C       1.00      1.00      1.00       145
           D       1.00      1.00      1.00       120
           E       1.00      1.00      1.00       120
           F       1.00      1.00      1.00       120
           G       1.00      1.00      1.00       120
           H       1.00      1.00      1.00       122
           I       1.00      1.00      1.00       138
           J       1.00      1.00      1.00       124
           K       1.00      1.00      1.00       120
           L       1.00      1.00      1.00       120
           M       1.00      1.00      1.00       121
           N       1.00      1.00      1.00       120
           O       1.00      1.00      1.00       144
           P       1.00      1.00      1.00       120
           Q       1.00      1.00      1.00       120
           R       1.00      1.00      1.00       121
           S       1.00      1.00      1.00       120
           T       1.00      1.00      1.00       120
           U       1.00      1.00      1.00       120
           V       1.00      1.00      1.00       129
           W       1.00      1.00      1.00       120
           X       1.00      1.00      1.00       120
           Y       1.00      1.00      1.00       120
           Z       1.00      1.00      1.00       120

    accuracy                           1.00      4285
   macro avg       1.00      1.00      1.00      4285
weighted avg       1.00      1.00      1.00      4285
```

Fig 8.6.1 Performance Metrics for Inceptionv3 Model

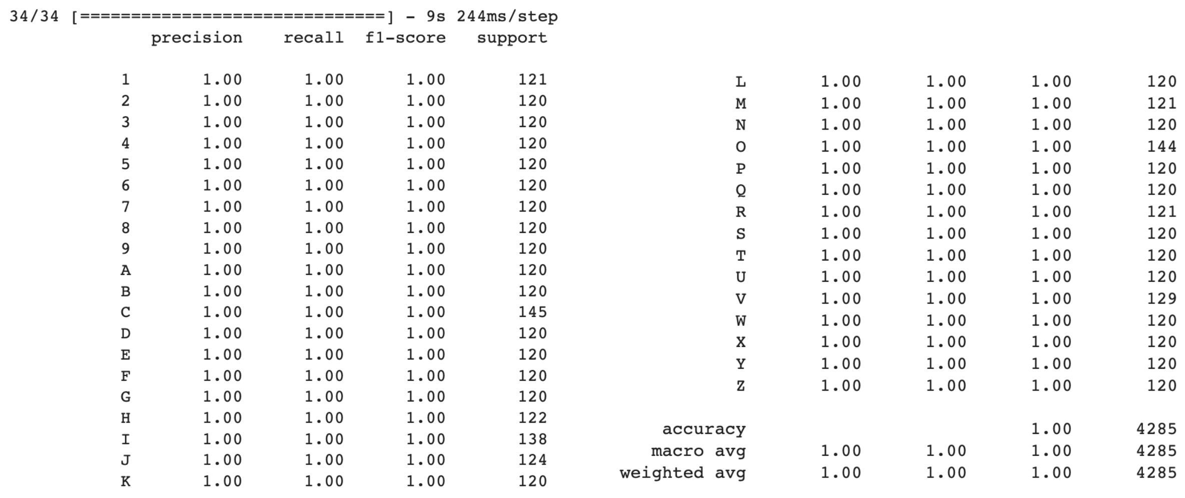
8.7 Confusion Matrix for Inceptionv3 Model



```
[ ]  incconfusion = metrics.confusion_matrix(inctrue_classes, incpredicted_classes)
     print('Confusion Matrix\n')
     print(incconfusion)

     Confusion Matrix

     [[121   0   0 ...   0   0   0]
      [  0 120   0 ...   0   0   0]
      [  0   0 120 ...   0   0   0]
      ...
      [  0   0   0 ... 120   0   0]
      [  0   0   0 ...   0 120   0]
      [  0   0   0 ...   0   0 120]]
```

Fig 8.7.1 Confusion Matrix for Inceptionv3 Model

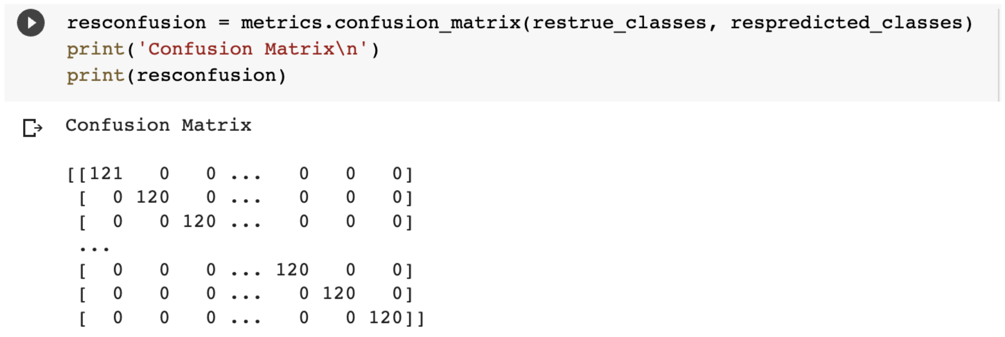
8.8 Performance Metrics for ResNet50v2 Model



```
34/34 [==============================] - 9s 244ms/step
              precision    recall  f1-score   support

           1       1.00      1.00      1.00       121
           2       1.00      1.00      1.00       120
           3       1.00      1.00      1.00       120
           4       1.00      1.00      1.00       120
           5       1.00      1.00      1.00       120
           6       1.00      1.00      1.00       120
           7       1.00      1.00      1.00       120
           8       1.00      1.00      1.00       120
           9       1.00      1.00      1.00       120
           A       1.00      1.00      1.00       120
           B       1.00      1.00      1.00       120
           C       1.00      1.00      1.00       145
           D       1.00      1.00      1.00       120
           E       1.00      1.00      1.00       120
           F       1.00      1.00      1.00       120
           G       1.00      1.00      1.00       120
           H       1.00      1.00      1.00       122
           I       1.00      1.00      1.00       138
           J       1.00      1.00      1.00       124
           K       1.00      1.00      1.00       120
           L       1.00      1.00      1.00       120
           M       1.00      1.00      1.00       121
           N       1.00      1.00      1.00       120
           O       1.00      1.00      1.00       144
           P       1.00      1.00      1.00       120
           Q       1.00      1.00      1.00       120
           R       1.00      1.00      1.00       121
           S       1.00      1.00      1.00       120
           T       1.00      1.00      1.00       120
           U       1.00      1.00      1.00       120
           V       1.00      1.00      1.00       129
           W       1.00      1.00      1.00       120
           X       1.00      1.00      1.00       120
           Y       1.00      1.00      1.00       120
           Z       1.00      1.00      1.00       120

    accuracy                           1.00      4285
   macro avg       1.00      1.00      1.00      4285
weighted avg       1.00      1.00      1.00      4285
```

Fig 8.8.1 Performance Metrics for ResNet50v2 Model

8.9 Confusion Matrix for ResNet50v2 Model

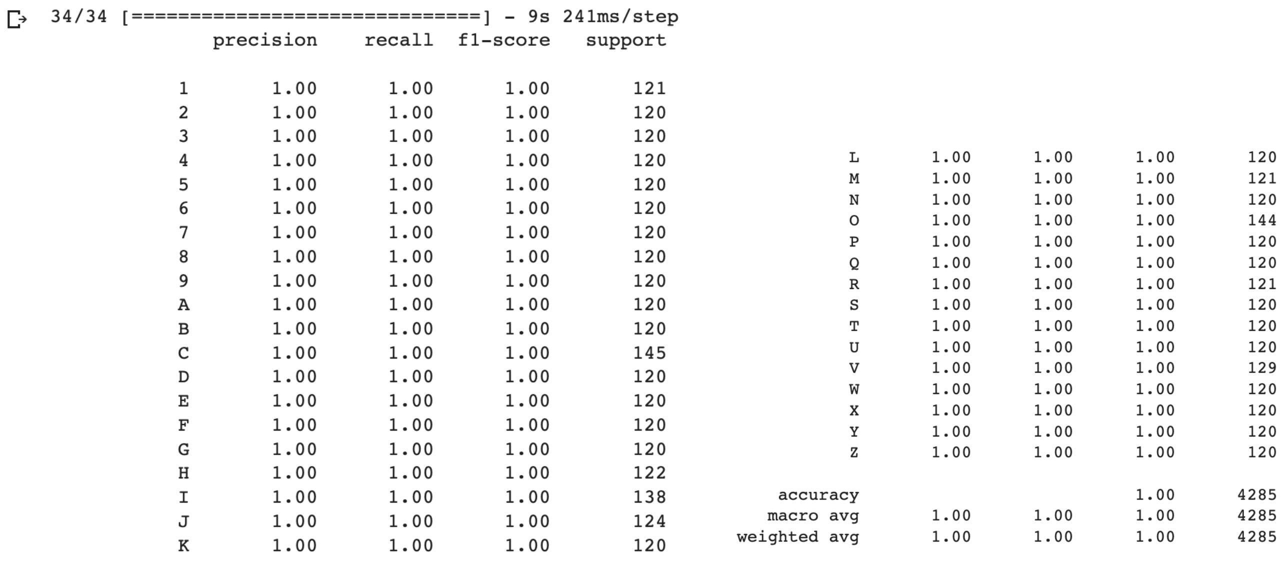


```
resconfusion = metrics.confusion_matrix(restrue_classes, respredicted_classes)
print('Confusion Matrix\n')
print(resconfusion)
```

```
Confusion Matrix

[[121   0   0 ...   0   0   0]
 [  0 120   0 ...   0   0   0]
 [  0   0 120 ...   0   0   0]
 ...
 [  0   0   0 ... 120   0   0]
 [  0   0   0 ...   0 120   0]
 [  0   0   0 ...   0   0 120]]
```

Fig 8.9.1 Confusion Matrix for ResNet50v2 Model

8.10 Performance Metrics for VGG16 Model

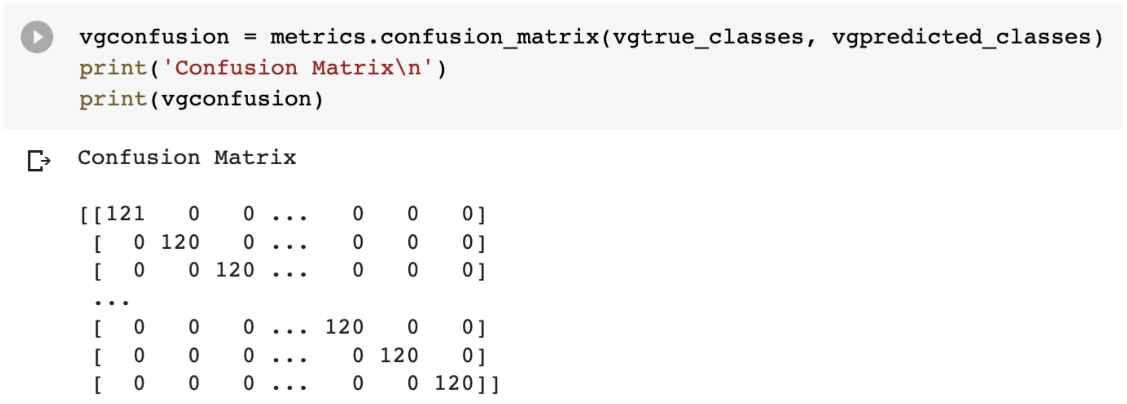


```
34/34 [==============================] - 9s 241ms/step
              precision    recall  f1-score   support

           1       1.00      1.00      1.00       121
           2       1.00      1.00      1.00       120
           3       1.00      1.00      1.00       120
           4       1.00      1.00      1.00       120
           5       1.00      1.00      1.00       120
           6       1.00      1.00      1.00       120
           7       1.00      1.00      1.00       120
           8       1.00      1.00      1.00       120
           9       1.00      1.00      1.00       120
           A       1.00      1.00      1.00       120
           B       1.00      1.00      1.00       120
           C       1.00      1.00      1.00       145
           D       1.00      1.00      1.00       120
           E       1.00      1.00      1.00       120
           F       1.00      1.00      1.00       120
           G       1.00      1.00      1.00       120
           H       1.00      1.00      1.00       122
           I       1.00      1.00      1.00       138
           J       1.00      1.00      1.00       124
           K       1.00      1.00      1.00       120
```

```
           L       1.00      1.00      1.00       120
           M       1.00      1.00      1.00       121
           N       1.00      1.00      1.00       120
           O       1.00      1.00      1.00       144
           P       1.00      1.00      1.00       120
           Q       1.00      1.00      1.00       120
           R       1.00      1.00      1.00       121
           S       1.00      1.00      1.00       120
           T       1.00      1.00      1.00       120
           U       1.00      1.00      1.00       120
           V       1.00      1.00      1.00       129
           W       1.00      1.00      1.00       120
           X       1.00      1.00      1.00       120
           Y       1.00      1.00      1.00       120
           Z       1.00      1.00      1.00       120

    accuracy                           1.00      4285
   macro avg       1.00      1.00      1.00      4285
weighted avg       1.00      1.00      1.00      4285
```

Fig 8.10.1 Performance Metrics for VGG16 Model

8.11 Confusion Matrix for VGG16 Model

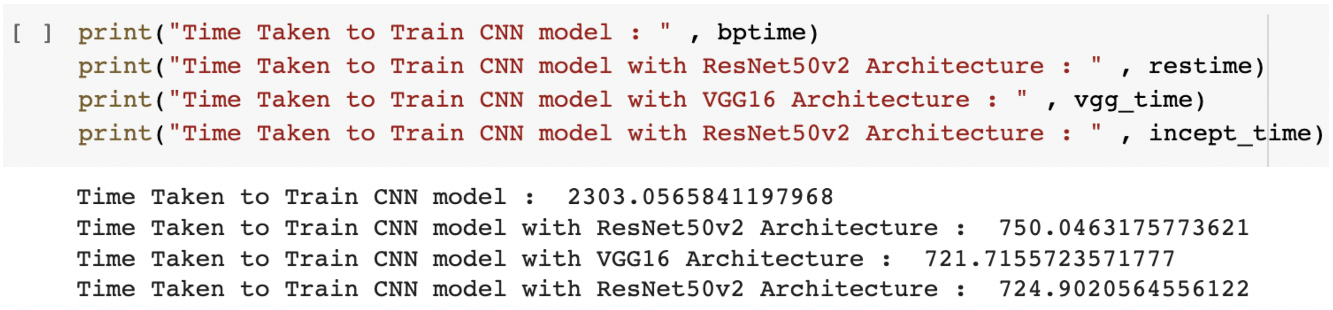


```
vgconfusion = metrics.confusion_matrix(vgtrue_classes, vgpredicted_classes)
print('Confusion Matrix\n')
print(vgconfusion)
```

```
Confusion Matrix

[[121   0   0 ...   0   0   0]
 [  0 120   0 ...   0   0   0]
 [  0   0 120 ...   0   0   0]
 ...
 [  0   0   0 ... 120   0   0]
 [  0   0   0 ...   0 120   0]
 [  0   0   0 ...   0   0 120]]
```

Fig 8.11.1 Confusion Matrix for VGG16 Model

8.12 Time for Training Models



```
print("Time Taken to Train CNN model : " , bptime)
print("Time Taken to Train CNN model with ResNet50v2 Architecture : " , restime)
print("Time Taken to Train CNN model with VGG16 Architecture : " , vgg_time)
print("Time Taken to Train CNN model with ResNet50v2 Architecture : " , incept_time)
```

```
Time Taken to Train CNN model :  2303.0565841197968
Time Taken to Train CNN model with ResNet50v2 Architecture :  750.0463175773621
Time Taken to Train CNN model with VGG16 Architecture :  721.7155723571777
Time Taken to Train CNN model with ResNet50v2 Architecture :  724.9020564556122
```

Fig 8.12.1 Time for Training Models

9. CONCLUSIONS

9.1 Conclusion

The VGG16 model outperformed all the other models with an accuracy and F1 score of 100% and it only took about 721.7 seconds to train and validate the model. Inceptionv3 and ResNet50v2 models both have achieved an accuracy and F1 score of 100% . It took about 724.9 and 750 seconds to train and validate Inceptionv3 and ResNet50v2 models respectively.

The CNN model based on the base paper ,was preprocessed using k-means and canny-edge. The model achieved an accuracy of 99.5% and an F1 score of 100% , but it took 2303 seconds to train and validate the model, which is about 3 times more than the VGG16 model. The traditional way of training a model for image classification is clearly inefficient because of the time consumption and also are usually less accurate when trained for lower number of epochs as compared to the modern architectures like VGG16, ResNet50v2 and Inceptionv3.

9.2 Future Scope

An approach in using a framework to capture live video and use object detection algorithms like YOLOv7 and R CNN, to capture hand gestures and then classify them using transfer learning from models like VGG16 , ResNet50v2 and Inceptionv3.

10. References

[1]https://www.researchgate.net/publication/284626785_Hand_Gesture_Recognition_A_Literature_Review

[2]https://ieeexplore.ieee.org/abstract/document/7544860

[3]https://ieeexplore.ieee.org/abstract/document/6481841

[4]https://research.ijcaonline.org/volume70/number19/pxc3887306.pdf

[5] https://digitalcommons.kennesaw.edu/cs_etd/21/

[6]https://www.scopus.com/record/display.uri?eid=2-s2.0-85060380941&origin=inward

[7]https://www.scopus.com/record/display.uri?eid=2-s2.0-84860850991&origin=inward

[8]https://link.springer.com/chapter/10.1007/978-981-15-7219-7_20

[9] https://keras.io/api/applications/vgg/

[10]https://pytorch.org/vision/stable/models/resnet.html

[11]https://keras.io/api/applications/inceptionv3/