Real - Time Communication System Powered by Al for Specially Abled

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 PROJECT OVERVIEW

In our society, we have people with disabilities. The technology is developing day by day but no significant developments are undertaken for the betterment of these people. Communications between deaf-mute and a normal person has always been a challenging task. It is very difficult for mute people to convey their message to normal people. Since normal people are not trained on hand sign language. In emergency times conveying their message is very difficult. The human hand has remained a popular choice to convey information in situations where other forms like speech cannot be used. Voice Conversion System with Hand Gesture Recognition and translation will be very useful to have a proper conversation between a normal person and an impaired person in any language.

The project aims to develop a system that converts the sign language into a human hearing voice in the desired language to convey a message to normal people, as well as convert speech into understandable sign language for the deaf and dumb. We are making use of a convolution neural network to create a model that is trained on different hand gestures. An app is built which uses this model. This app enables deaf and dumb people to convey their information using signs which get converted to human-understandable language and speech is given as output.

1.2 PURPOSE

Dumb people are usually face some problems on normal communication with other people in society. It has been observed that they sometimes find it difficult to interact with normal people with their gestures. Because people with hearing problems or deaf people cannot speak like normal people, they have to depend on a kind of visual communication in most cases. To overcome these problems, we have proposed a system that uses cameras to capture and convert videos of hand gestures from dumb people who turn into speech for understanding normal people. The primary application for addressing the sign language is the improvement of the sign language. Computer recognition of the sign language is an important research problem for communication with the hearing impaired. The system does not require that the hand is perfectly aligned to the camera. The project uses the image processing system to identify, especially the English alphabetical character language used by the deaf to communicate. The system proposed to develop and build an intelligent system that uses image processing, machine learning and artificial intelligence concepts to make visual inputs of hand gestures of sign language and to create an easily recognizable form of outputs.

CHAPTER 2

LITERATURE SURVEY

2.1 EXISTING PROBLEM

One of the most precious gifts to a human being is an ability to see, listen, speak and respond according to the situations. But there are some unfortunate ones who are deprived of this. Making a single compact device for people with Visual, Hearing and Vocal impairment is a tough job. Communication between deaf-dumb and normal person have been always a challenging task. This paper proposes an innovative communication system framework for deaf, dumb and blind people in a single compact device. We provide a technique for a blind person to read a text and it can be achieved by capturing an image through a camera which converts a text to speech (TTS). It provides a way for the deaf people to read a text by speech to text (STT) conversion technology. Also, it provides a technique for dumb people using text to voice conversion. The system is provided with four switches and each switch has a different function. The blind people can be able to read the words using by Tesseract OCR (Online Character Recognition), the dumb people can communicate their message through text which will be read out by speak, the deaf people can be able to hear others speech from text. All these functions are implemented by the use of Raspberry Pi.

2.2 REFERENCE

Prof. P.G. Ahire, K.B. Tilekary,T.A. Jawake, P.B. Warale, “Two Way Communicator between Deaf and Dumb People and Normal People", 978-1-4799-6892-3/15 31.00 c 2015 IEEE. [2] Shreyashi Narayan Sawant, "Sign Language recognition System to aid Deaf- dumb People Using PCA", IJCSET ISSN : 2229-3345 Vol. 5 No. 05 May 2014. [3] Amitkumar Shined, Frames Kagalkar, ”Sign Language to Text and Vice Versa Recognition using Computer Vision in Marathi”, International Journal of Computer Applications (0975 – 8887) National Conference on Advances in Computing (NCAC 2015) [4] Setiawardhana, Risky Yuniar Hakkun, Achmad Baharuddin, "Sign Language Learning based on Android For Deaf and Speech Impaired People", 978-1-4673-9345- 4/15/31.00 c 2015 IEEE [5] M. Ebrahim Al-Ahdal & Nooritawati Md Tahir,’’ Review in Sign Language Recognition Systems’’ Symposium on Computer & Informatics(ISCI),pp:52-57, IEEE ,2012 [6] Archana S. Ghotkar, Rucha Khatal, Sanjana Khupase, Surbhi Asati & Mithila Hadap,’’ Hand Gesture Recognition for Indian Sign Language’’ International Conference on Computer Communication and Informatics (ICCCI), pp:1- 4.IEEE,Jan 2012. [7] Iwan Njoto Sandjaja, Nelson Marcos,’’ Sign Language Number Recognition’’ Fifth International Joint Conference on INC, IMS and IDC, IEEE 2009

2.3 PROBLEM STATEMENT DEFINITION

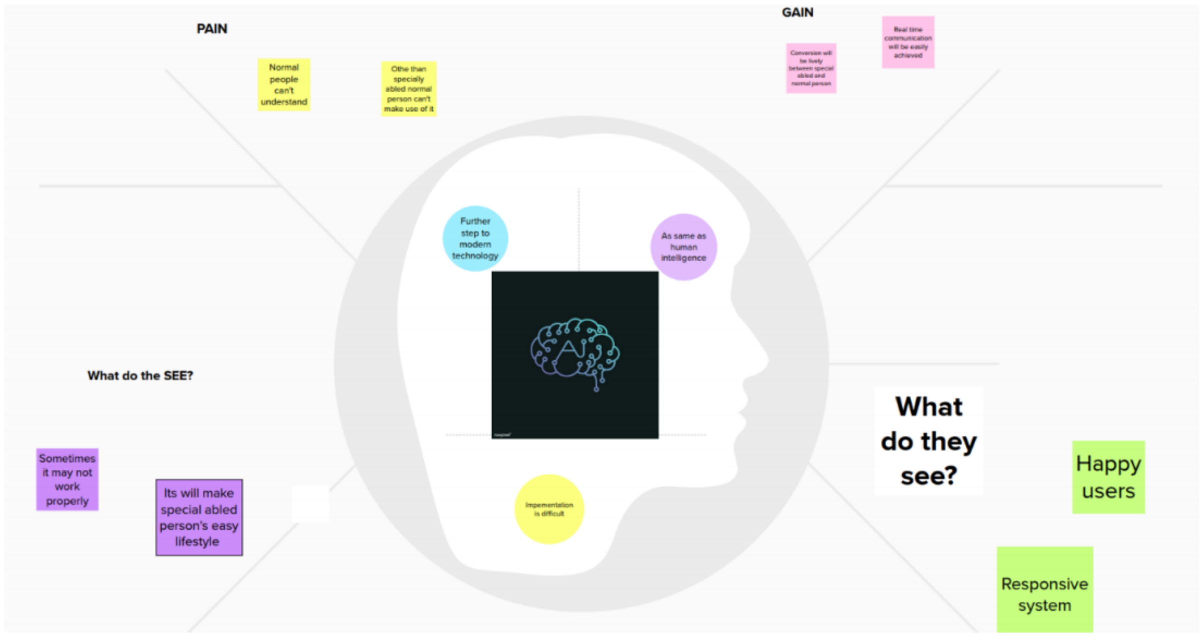
The project aims to develop a system that converts the sign language into a human hearing voice in the desired language as well as to convert speech into understandable sign language for the deaf and dumb. A convolution neural network is used to create a model that will be trained on different hand gestures. A web application to use the model will be built. This application will enable the deaf and dumb people to convey their information using signs which get converted to human-understandable language and speech is given as output.

CHAPTER 3

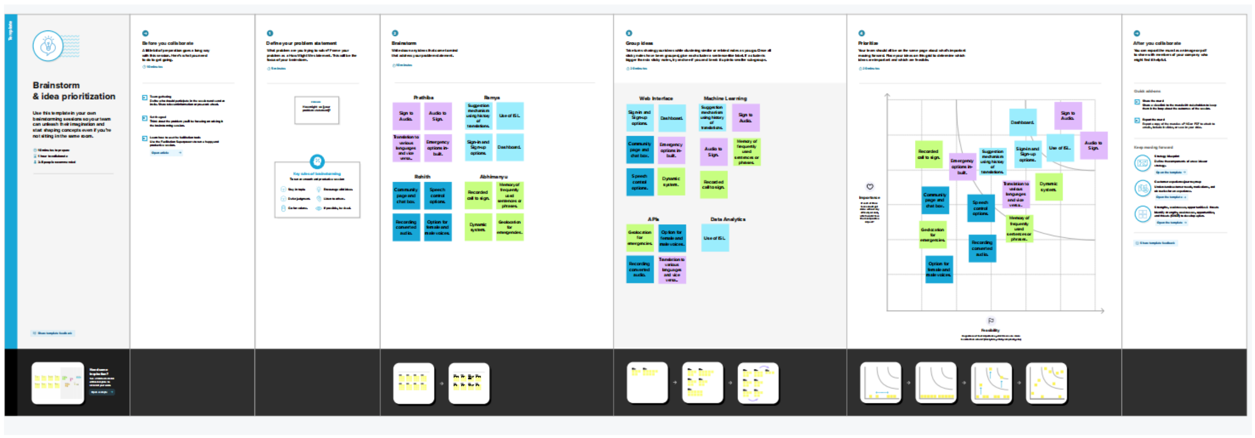
IDEATION & PROPOSED SOLUTION

3.1 Empathy Map Canvas

An Empathy Map Is A Simple,Easy-To-Digest Visual That Captures Knowledge About a User’S Behaviors And Attitudes.


PAIN
Normal people can't understand
Othe than specially abled normal person can't make use of it
GAIN
Conversion will be lively between special abled and normal person
Real time communication will be easily achieved
Further step to modern technology
As same as human intelligence
AI
What do the SEE?
Sometimes it may not work properly
Its will make special abled person's easy lifestyle
Impementation is difficult
What do they see?
Happy users
Responsive system

3.2 Ideation & Brainstorming


Brainstorm
& idea prioritization
Before you collaborate
Define your problem statement
Brainstorm
Prathiba
Ramya
Rohith
Abhimanyu
Sign to Audio.
Audio to Sign.
Translation to various languages and vice versa.
Emergency options in-built.
Suggestion mechanism using history of translations.
Use of ISL.
Sign-in and Sign-up options.
Dashboard.
Community page and chat box.
Speech control options.
Recording converted audio.
Option for female and male voices.
Recorded call to sign.
Memory of frequently used sentences or phrases.
Dynamic system.
Geolocation for emergencies.
Group ideas
Web Interface
Machine Learning
APIs
Data Analytics
Prioritize
Importance
Feasibility

3.3 Proposed Solution

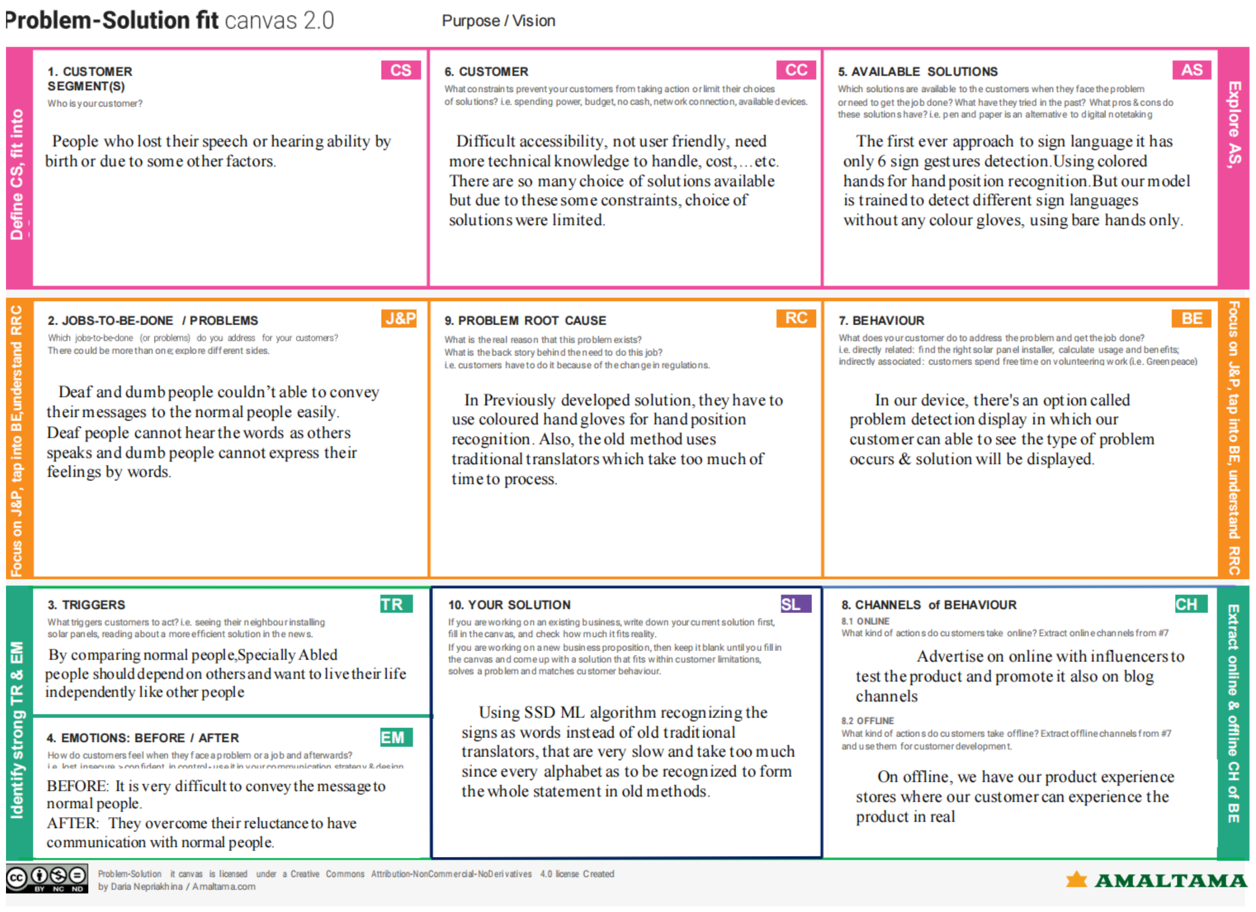
Project team shall fill the following information in proposed solution template.

S.No.	Parameter	Description
1	Problem Statement (Problem to be solved)	The project aims to develop a system that converts the sign language into a human hearing voice in the desired language as well as to convert speech into understandable sign language for the deaf and dumb. A convolution neural network is used to create a model that will be trained on different hand gestures. A web application to use the model will be built.

2	Idea / Solution description	The idea is to develop a system that converts the sign language into a human hearing voice in the desired language to convey a message to normal people, as well as convert speech into understandable sign language for the deaf and dumb. We are making use of a convolution neural network to create a model that is trained on different hand gestures. An app is built which uses this model. This app enables deaf and dumb people to convey their information using signs which get converted to human-understandable language and speech is given as output.
3	Novelty/ Uniqueness	➢ Two-way communication. ➢ An application with embedded features: ○ Translation into various languages. ○ Suggestion mechanism based on the history of translations. ○ In-built emergency options.

4	Social Impact / Customer Satisfaction	➢ Understanding specially-abled people in a better way. ➢ Equal opportunities ➢ Break the social barrier
5	Business Model (Revenue Model)	Not many models of this kind are available in the society, so educating people about the uses and technology of this model will surely create a huge market for this product. This can be done by means of mass communication and advertisements. Instead of promoting it in business point of view, promoting it in a service point of view will expand its reach.
6	Scalability of the Solution	If a basic model of this solution is created, then expanding it, needs some minimal man power with sufficient knowledge regarding this. So, with the increase in its demand, we can scale its production comfortably

3.4 Problem Solution fit



Problem-Solution fit canvas 2.0 Purpose / Vision

Define CS, fit into

1. CUSTOMER SEGMENT(S) CS

Who is your customer?

People who lost their speech or hearing ability by birth or due to some other factors.

6. CUSTOMER CC

What constraints prevent your customers from taking action or limit their choices of solutions? i.e. spending power, budget, no cash, network connection, available devices.

Difficult accessibility, not user friendly, need more technical knowledge to handle, cost,…etc. There are so many choice of solutions available but due to these some constraints, choice of solutions were limited.

5. AVAILABLE SOLUTIONS AS

Which solutions are available to the customers when they face the problem or need to get the job done? What have they tried in the past? What pros & cons do these solutions have? i.e. pen and paper is an alternative to digital notetaking

The first ever approach to sign language it has only 6 sign gestures detection.Using colored hands for hand position recognition.But our model is trained to detect different sign languages without any colour gloves, using bare hands only.

Explore AS,

Focus on J&P, tap into BE,understand RRC

2. JOBS-TO-BE-DONE / PROBLEMS J&P

Which jobs-to-be-done (or problems) do you address for your customers? There could be more than one; explore different sides.

Deaf and dumb people couldn't able to convey their messages to the normal people easily. Deaf people cannot hear the words as others speaks and dumb people cannot express their feelings by words.

9. PROBLEM ROOT CAUSE RC

What is the real reason that this problem exists? What is the back story behind the need to do this job? i.e. customers have to do it because of the change in regulations.

In Previously developed solution, they have to use coloured hand gloves for hand position recognition. Also, the old method uses traditional translators which take too much of time to process.

7. BEHAVIOUR BE

What does your customer do to address the problem and get the job done? i.e. directly related: find the right solar panel installer, calculate usage and benefits; indirectly associated: customers spend free time on volunteering work (i.e. Greenpeace)

In our device, there's an option called problem detection display in which our customer can able to see the type of problem occurs & solution will be displayed.

Focus on J&P, tap into BE, understand RRC

Identify strong TR & EM

3. TRIGGERS TR

What triggers customers to act? i.e. seeing their neighbour installing solar panels, reading about a more efficient solution in the news.

By comparing normal people,Specially Abled people should depend on others and want to live their life independently like other people

4. EMOTIONS: BEFORE / AFTER EM

How do customers feel when they face a problem or a job and afterwards? i.e. lost, insecure > confident, in control - use it in your communication strategy & design

BEFORE: It is very difficult to convey the message to normal people.
AFTER: They overcome their reluctance to have communication with normal people.

10. YOUR SOLUTION SL

If you are working on an existing business, write down your current solution first, fill in the canvas, and check how much it fits reality.
If you are working on a new business proposition, then keep it blank until you fill in the canvas and come up with a solution that fits within customer limitations, solves a problem and matches customer behaviour.

Using SSD ML algorithm recognizing the signs as words instead of old traditional translators, that are very slow and take too much since every alphabet as to be recognized to form the whole statement in old methods.

8. CHANNELS of BEHAVIOUR CH

8.1 ONLINE
What kind of actions do customers take online? Extract online channels from #7

Advertise on online with influencers to test the product and promote it also on blog channels

8.2 OFFLINE
What kind of actions do customers take offline? Extract offline channels from #7 and use them for customer development.

On offline, we have our product experience stores where our customer can experience the product in real

Extract online & offline CH of BE

AMALTAMA

CHAPTER 4

REQUIREMENT ANALYSIS

4.1 Functional requirements

1. **Blob Detection:** This algorithm helps to draw rectangles around the defective part. The methods aim to detect areas in a digital image that differ in properties, such as brightness or color, compared to surrounding regions. Independent detection of corresponding regions in scaled versions of the same image. A blob is a region of an image in which some properties are constant or approximately constant, all points in a blob can be viewed in a certain sense to be similar to one another.

2. **Skin color recognition:** Skin detection is the process of finding skin colored pixels and regions in an image or video. This process is typically used as a preprocessing step to find areas that may have human faces and limbs in images.

3**. Template Matching:** Template matching is a technique in digital image processing to find small portions of an image that match a template image. It can be used in as part of quality control,one way to navigate a mobile robot,or as a way to detect edges in images.

4.2 Non-Functional requirements

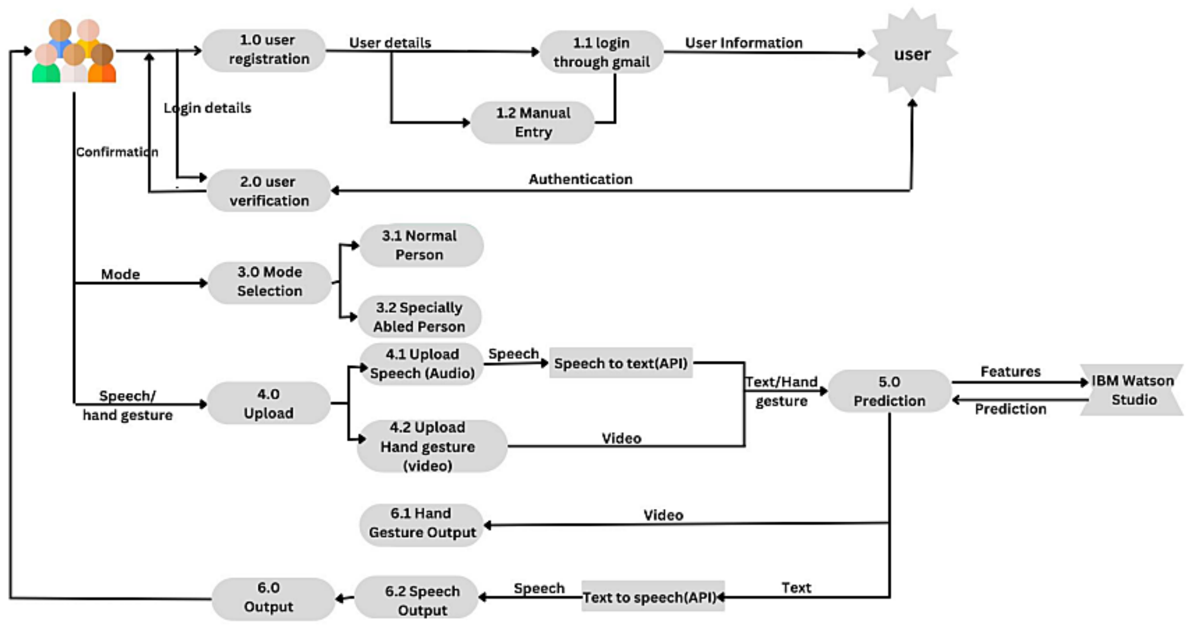
PRODUCT PERSPECTIVE: · To implement a system for recognizing sign language hand configurations as described which will additionally provide the facility to each individual to define and upload his own sign language into the system since every country or even regional group uses its own set of signs. · To develop a tool which will help deaf people in communication. To develop a Sign language, can be translated into text or sound based on images, videos. Signs can be converted to Speech so that there is a two - way communication.

PRODUCT FUNCTION: It’s a Desktop application. · User will start video from camera. · User will be able to register different signs for further recognition using camera. · When user will start recognition activity and give various hand gestures in front of camera, sign will be

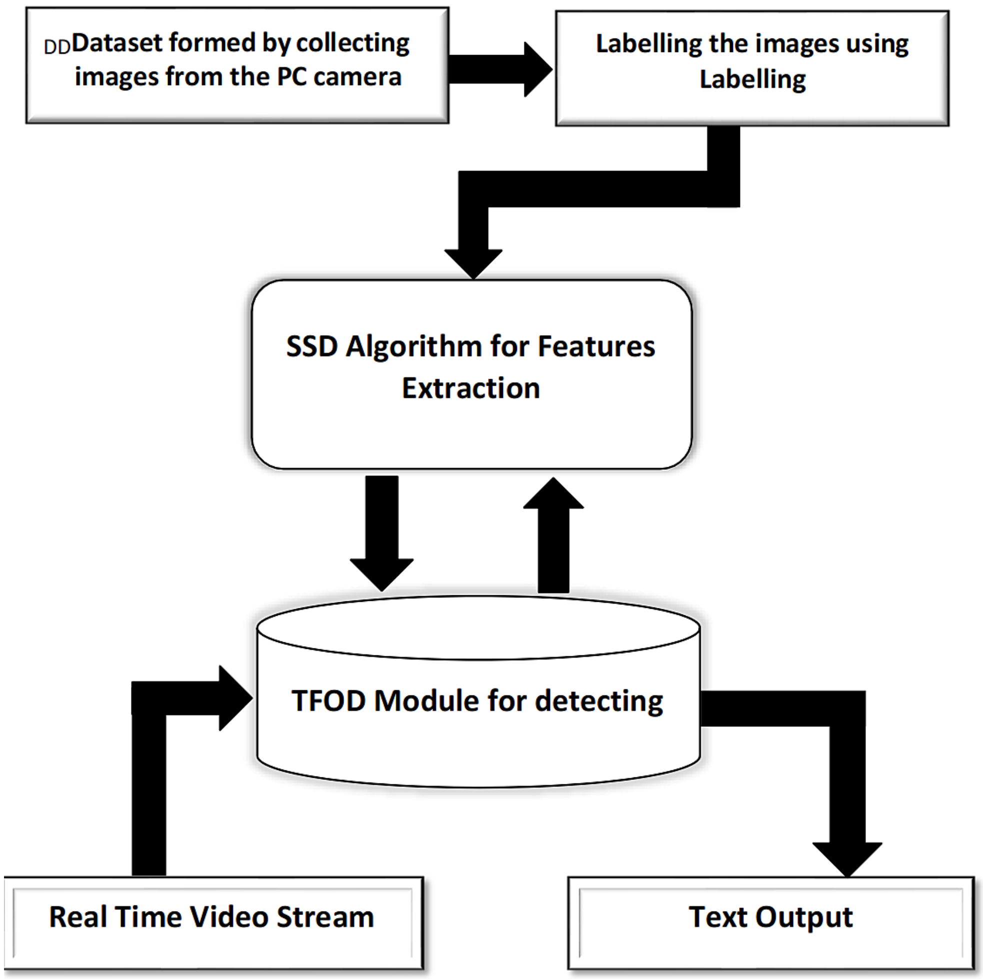
CHAPTER 5

5. PROJECT DESIGN

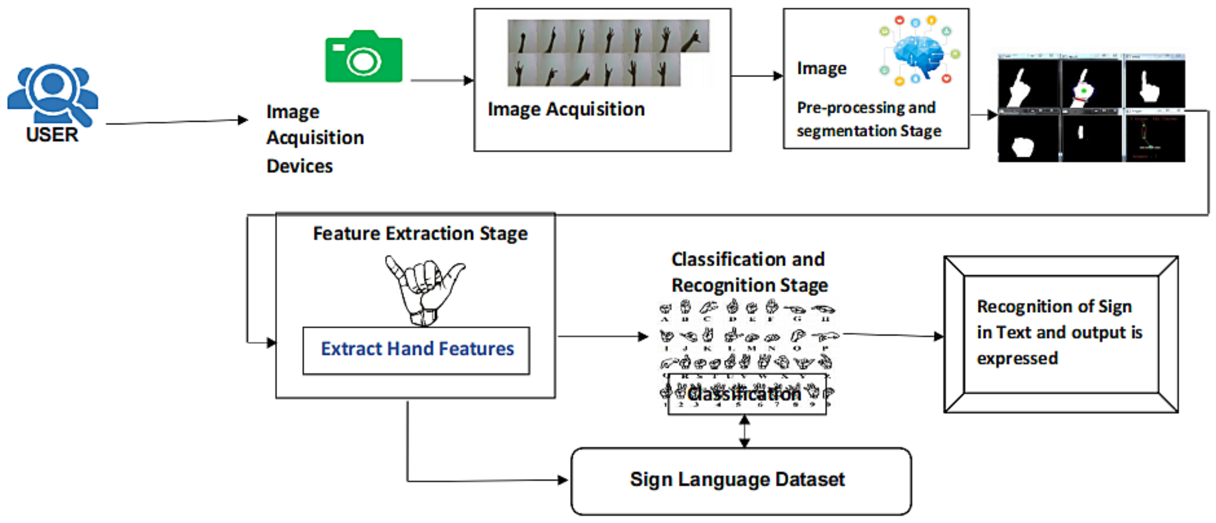
5.1 Data Flow Diagrams



5.2 Solution & Technical Architecture


DDDataset formed by collecting images from the PC camera
Labelling the images using Labelling
SSD Algorithm for Features Extraction
TFOD Module for detecting
Real Time Video Stream
Text Output

SYSTEM ARCHITECTURE



USER
Image Acquisition Devices
Image Acquisition
Image Pre-processing and segmentation Stage
Feature Extraction Stage
Extract Hand Features
Classification and Recognition Stage
Classification
Recognition of Sign in Text and output is expressed
Sign Language Dataset

Table-1 : Components & Technologies:

S.No	Component	Description	Technology
1.	User Interface	Customer have to login through their respective website or phone number. Then interaction will happen with the User interface.	javascript, CSS,HTML
2.	Application Logic-1	It requires various types libraries, frameworks to develop the project	Java / Python
3.	Application Logic-2	Helps to converting the human gestures/actions into written words.	Machine learning
4.	Application Logic-3	Provides helpful,feasible answers after recognising the human gestures.	ANN,CNN
5.	Database	Data could be numbers or words.	MySQL, Rational database
6.	Cloud Database	Providing customer to use host database without buying additional hardware..	Deep learning and neural networks
7.	File Storage	File storage could be fast, reliable and flexible..	Local file system
8.	External API-1	Used to access the information in the cloud	Weather API
9.	External API-2	Used to access the information for data driven decision making...	Aadhar API
10.	Machine Learning Model	Machine learning interact with various algorithms that are required for implementation.	Image acquisation
11.	Infrastructure (Server / Cloud)	Application deployment on local system /local cloud server configuration. Install the windows version and execute the installer..	Local, Cloud Foundry, Kubernetes, etc.

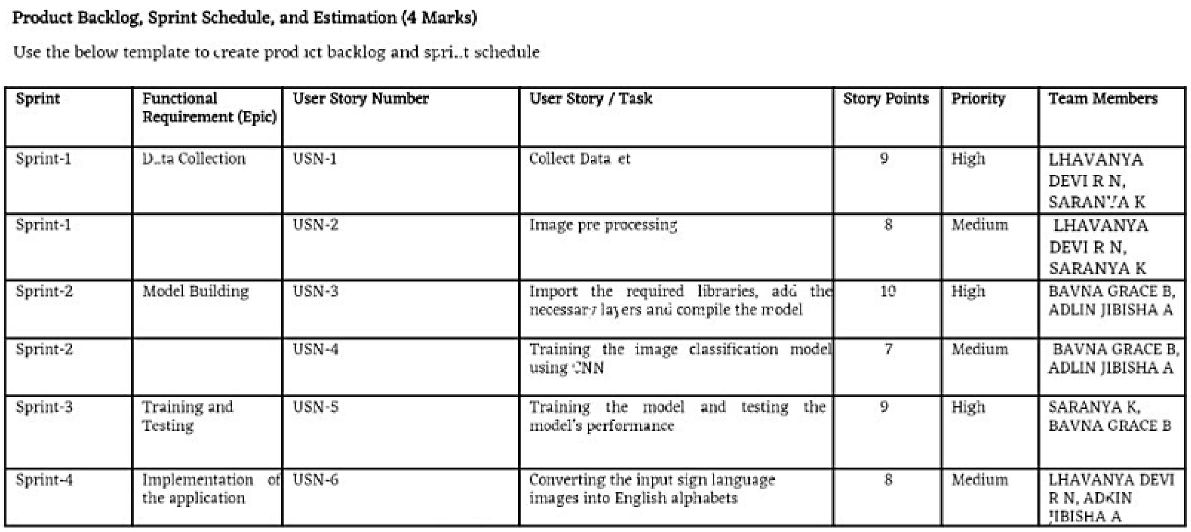
5.3 User Stories

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Specially Abled Person)	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	I can access my account and select the mode of usage	High	Sprint-1
		USN-2	As a user, I can register for the application through Gmail		High	Sprint-1
	Confirmation	USN-3	As a user, I will receive confirmation email once I have registered for the application	I can receive confirmation email & click confirm	Medium	Sprint-2
	Login	USN-4	As a user, I can log into the application by entering email & password	Credentials has to be matched	Medium	Sprint-2
	Mode Selection	USN-5	As a user, I will be prompted to select the mode of communication and I will select the specially abled mode (Gesture to Speech)	Either of the modes has to be chosen for further processing	High	Sprint-3
	Video Capturing	USN-6	As a user of this mode I will capture my hand gesture as video	Minimum video quality criteria has to be met	High	Sprint-1
	Gesture interpretation	USN-7	As a user of this mode, I will be able to receive and interpret the translated gestures from the other end.	Must be a valid gesture	Low	Sprint-1
Customer (Normal Person)	Registration	USN-8	As a user, I can register for the application by entering my email, password, and confirming my password.	I can access my account and select the mode of usage	High	Sprint-1
		USN-9	As a user, I can register for the application through Gmail		High	Sprint-1
	Confirmation	USN-10	As a user, I will receive confirmation email once I have registered for the application	I can receive confirmation email & click confirm	Medium	Sprint-2
	Login	USN-11	As a user, I can log into the application by entering email & password	Credentials has to be matched	Medium	Sprint-2
	Mode Selection	USN-12	As a user, I will be prompted to select the mode of communication and I will select the specially abled mode (Gesture to Speech)	Either of the modes has to be chosen for further processing	High	Sprint-3
	Speech Recording	USN-13	As a user of this mode I will record the speech in order to convert it into gesture	Minimum audio quality criteria have to be met	High	Sprint-1
	Speech recognition	USN-14	As a user of this mode, I will be able to receive and interpret the translated speech from the other end.	The words must be a recognizable	Low	Sprint-1
Administrator	Application monitoring and controlling	USN-15	As an admin, I will be responsible for controlling the user activities and further upgradations of the application	Admin level privilege	Medium	Sprint-3

CHAPTER 6

6. PROJECT PLANNING & SCHEDULING

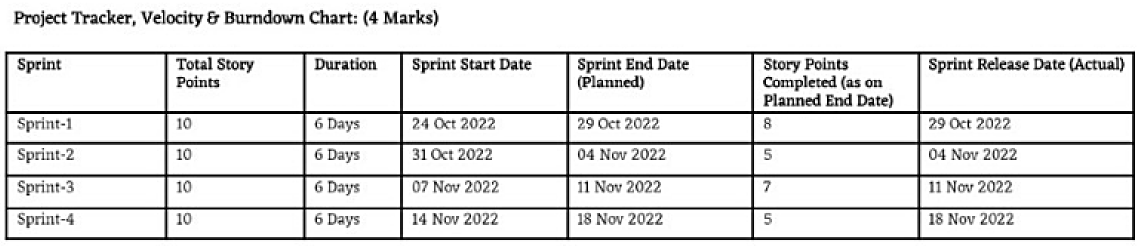
6.1 Sprint Planning & Estimation



Product Backlog, Sprint Schedule, and Estimation (4 Marks)

Use the below template to create product backlog and sprint schedule

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Data Collection	USN-1	Collect Dataset	9	High	LHAVANYA DEVI R N, SARANYA K
Sprint-1		USN-2	Image pre processing	8	Medium	LHAVANYA DEVI R N, SARANYA K
Sprint-2	Model Building	USN-3	Import the required libraries, add the necessary layers and compile the model	10	High	BAVNA GRACE B, ADLIN JIBISHA A
Sprint-2		USN-4	Training the image classification model using CNN	7	Medium	BAVNA GRACE B, ADLIN JIBISHA A
Sprint-3	Training and Testing	USN-5	Training the model and testing the model's performance	9	High	SARANYA K, BAVNA GRACE B
Sprint-4	Implementation of the application	USN-6	Converting the input sign language images into English alphabets	8	Medium	LHAVANYA DEVI R N, ADLIN JIBISHA A



Project Tracker, Velocity & Burndown Chart: (4 Marks)

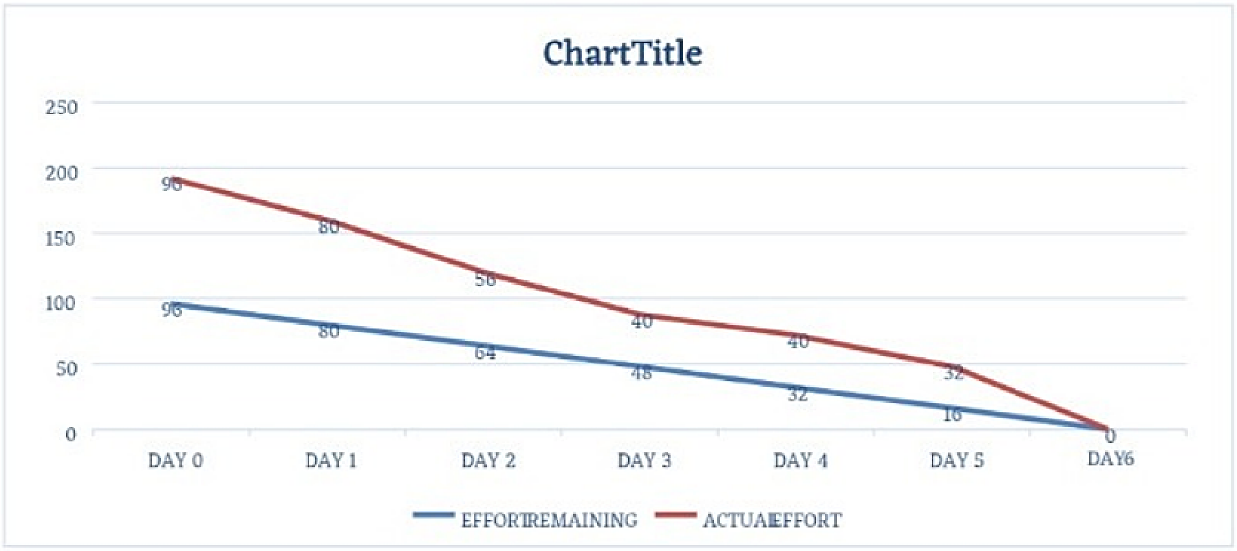
Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	10	6 Days	24 Oct 2022	29 Oct 2022	8	29 Oct 2022
Sprint-2	10	6 Days	31 Oct 2022	04 Nov 2022	5	04 Nov 2022
Sprint-3	10	6 Days	07 Nov 2022	11 Nov 2022	7	11 Nov 2022
Sprint-4	10	6 Days	14 Nov 2022	18 Nov 2022	5	18 Nov 2022

Velocity:

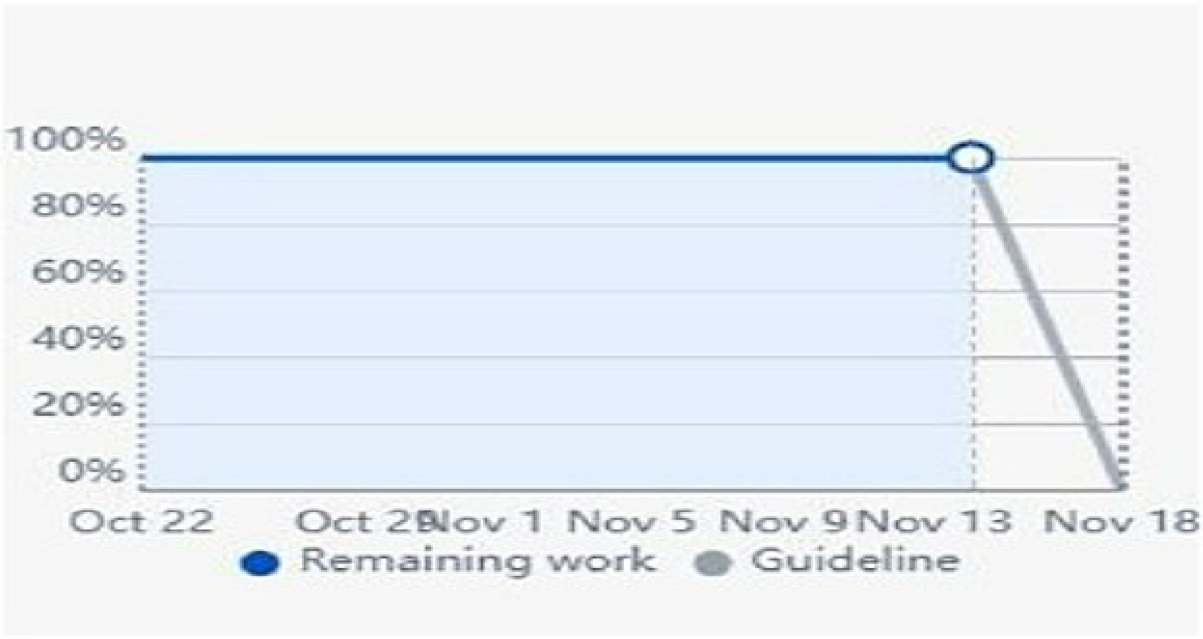
$$AV = \frac{sprint\ duration}{velocity}$$

AV = 6/10 = 0.6

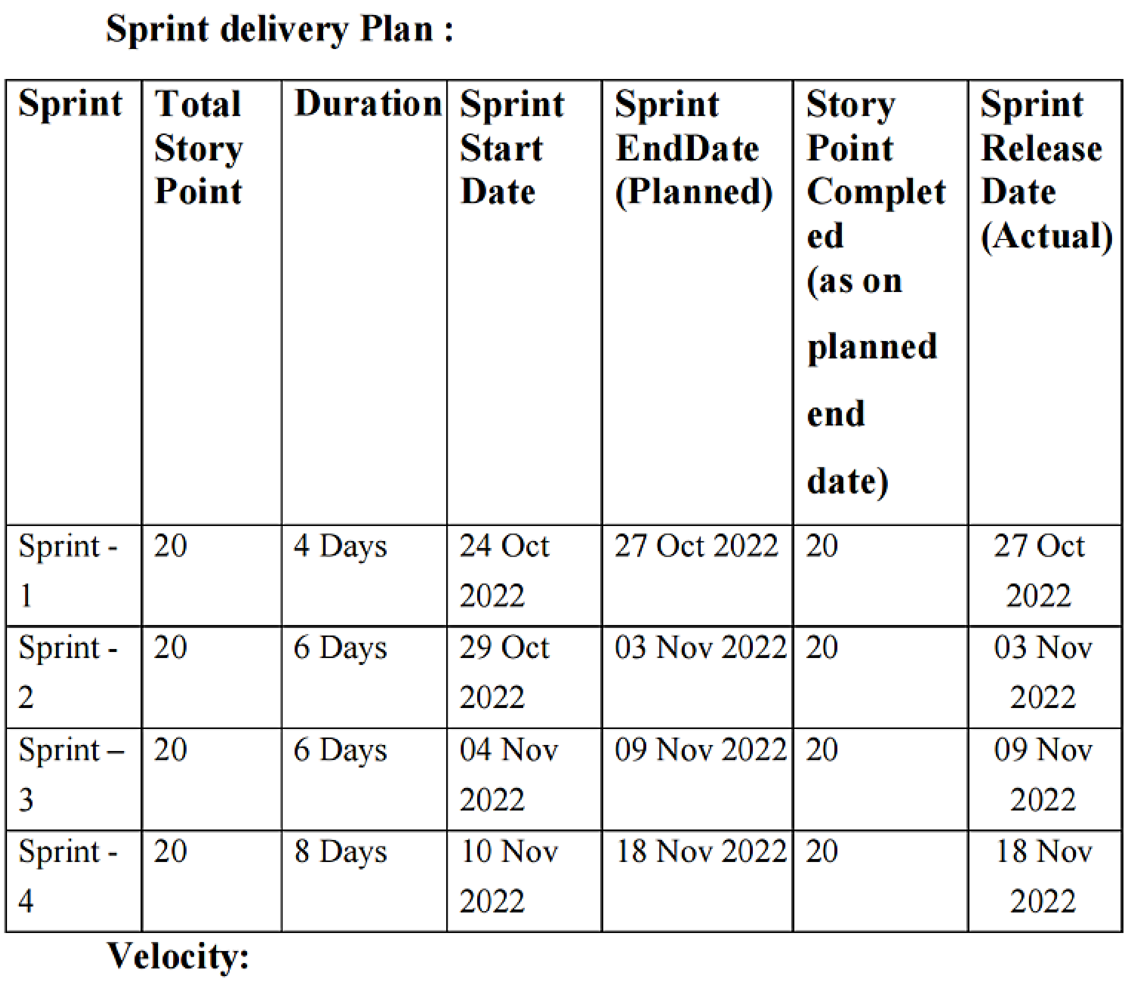
BURNDOWN CHART



SPRINT BURNDOWN CHART



6.2 Sprint Delivery Schedule



Sprint delivery Plan :

Sprint	Total Story Point	Duration	Sprint Start Date	Sprint EndDate (Planned)	Story Point Completed (as on planned end date)	Sprint Release Date (Actual)
Sprint - 1	20	4 Days	24 Oct 2022	27 Oct 2022	20	27 Oct 2022
Sprint - 2	20	6 Days	29 Oct 2022	03 Nov 2022	20	03 Nov 2022
Sprint – 3	20	6 Days	04 Nov 2022	09 Nov 2022	20	09 Nov 2022
Sprint - 4	20	8 Days	10 Nov 2022	18 Nov 2022	20	18 Nov 2022

Velocity:

Imagine we have a 10 day sprint duration and the velocity of the team is 20.

AV = Sprint

Duration = 20 = 2Velocity

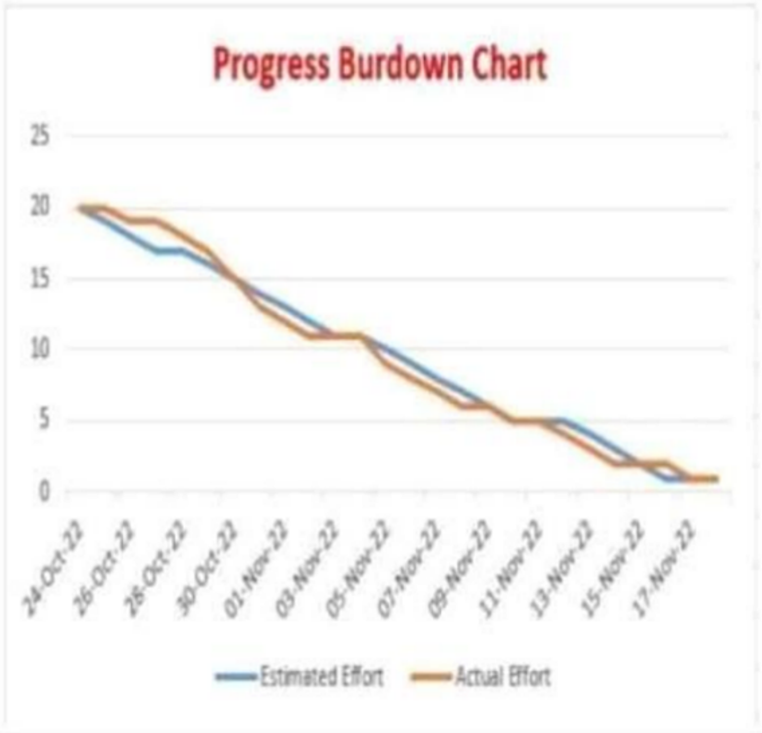
Burndown Chart:

A Burn down chart is a graphical representation of work left to do versus time. Itis often used in agile software development methodologies such as Scrum.
However, burn down charts can be applied to any project containing measurableprogress over time.

BURNDOWN CHART

Sprint	**Date**	**Estimated Effort**	**Actual Effort**
Sprint - 1	24 – Oct - 2022	20	20
	25 – Oct - 2022	19	20
	26 – Oct - 2022	18	19
	27 – Oct - 2022	17	19
	28 – Oct - 2022	17	18
Sprint - 2	29 – Oct - 2022	16	17
	30 – Oct - 2022	15	15
	31 – Oct - 2022	14	13
	01 – Nov - 2022	13	12
	02 – Nov - 2022	12	11
	03 – Nov - 2022	11	11
Sprint - 3	04 – Nov - 2022	11	11
	05 – Nov - 2022	10	9
	06 – Nov - 2022	9	8
	07 – Nov - 2022	8	7

	08 – Nov - 2022	7	6
	09 – Nov - 2022	6	6
Sprint - 4	10 – Nov - 2022	5	5
	11 – Nov - 2022	5	5
	12 – Nov - 2022	5	4
	13 – Nov - 2022	4	3
	14 – Nov - 2022	3	2
	15 – Nov - 2022	2	2
	16 – Nov - 2022	1	2
	17– Nov - 2022	1	1
	18 – Nov - 2022	1	1



BURN DOWN CHART

6.3 Report From JIRA

CHAPTER 7

7. CODING & SOLUTIONING (Explain the features added in the project along with code)

7.1IMAGE PREPROCESSING

- Image pre-processing includes zooming, shearing,flipping to increase-the robustness of the model after it is built. Keras package is used for pre-processing images.
- Importing ImageDataGenerator Library to create an instance-for which include shearing, rescale,zooming,etc to make the model robust with different types of images.

{"nbformat":4,"nbformat_minor":0,"metadata":{"colab":{"provenance":[],"mount_file_id":"1ZRob3ICs7Dd5ULPtd23qQEaAJid-il1-","authorship_tag":"ABX9TyPd4P7aF075C4dY8i3HvhjK"},"kernelspec":{"name":"python3","display_name":"Python 3"},"language_info":{"name":"python"}},"cells":[{"cell_type":"markdown","source":["Image Preprocessing"],"metadata":{"id":"jMxfkhLsFIlq"}},{"cell_type":"markdown","source":["Import ImageDataGenerator Library And Configure It"],"metadata":{"id":"fWt446aBFMD-"}},{"cell_type":"code","source":["from tensorflow.keras.preprocessing.image import ImageDataGenerator"],"metadata":{"id":"Y3PrgbiYFPBK","executionInfo":{"status":"ok","timestamp":1668782574866,"user_tz":-330,"elapsed":546,"user":{"displayName":"Lhavanya Devi Nagarajan","userId":"14108198786569258516"}}},"execution_count":28,"outputs":[]},{"cell_type":"code","source":["# Training Datagen\n","train_datagen = ImageDataGenerator(rescale=1/255,zoom_range=0.2,horizontal_flip=True,vertical_flip=False)\n","# Testing Datagen\n","test_datagen =

ImageDataGenerator(rescale=1/255)"],"metadata":{"id":"SXrlcw42FT5y","executionInfo":{"status":"ok","timestamp":1668782575685,"user_tz":-330,"elapsed":17,"user":{"displayName":"Lhavanya Devi
Nagarajan","userId":"14108198786569258516"}}},"execution_count":29,"outputs":[]},{"cell_type":"code","source":["import tensorflow as tf\n","import os\n","from tensorflow.keras.models import Sequential\n","from tensorflow.keras.layers import Dense, Conv2D, Flatten, Dropout, MaxPooling2D\n","from tensorflow.keras.preprocessing.image import ImageDataGenerator\n","import numpy as np\n","import matplotlib.pyplot as plt\n","import IPython.display as display\n","from PIL import Image\n","import
pathlib"],"metadata":{"id":"SMCF4sG8FXqI","executionInfo":{"status":"ok","timestamp":1668782575686,"user_tz":-330,"elapsed":17,"user":{"displayName":"Lhavanya Devi
Nagarajan","userId":"14108198786569258516"}}},"execution_count":30,"outputs":[]},{"cell_type":"markdown","source":["Apply ImageDataGenerator Functionality To Train And Test
set"],"metadata":{"id":"8RbLZEqBFbcw"}},{"cell_type":"code","source":["from google.colab import drive"],"metadata":{"id":"iXoMSVM4Fa14","executionInfo":{"status":"ok","timestamp":1668782575687,"user_tz":-330,"elapsed":18,"user":{"displayName":"Lhavanya Devi
Nagarajan","userId":"14108198786569258516"}}},"execution_count":31,"outputs":[]},{"cell_type":"code","source":["!unzip
'/content/drive/MyDrive/training_set.zip'"],"metadata":{"colab":{"base_uri":"https://localhost:8080/"},"id":"odiyWuJuFhjN","executionInfo":{"status":"ok","timestamp":1668782575687,"user_tz":-330,"elapsed":18,"user":{"displayName":"Lhavanya Devi
Nagarajan","userId":"14108198786569258516"}},"outputId":"2fea67fb-ab21-4ccc-d2bd-86f084105b98"},"execution_count":32,"outputs":[{"output_type":"stream","name":"stdout","text":["unzip: cannot find or open /content/drive/MyDrive/training_set.zip,
/content/drive/MyDrive/training_set.zip.zip or
/content/drive/MyDrive/training_set.zip.ZIP.\n"]}]},{"cell_type":"code","source":["from tensorflow.keras.preprocessing.image import ImageDataGenerator\n","print(\"This dataset has been created and uploaded by IBM-TeamID-IBM-Project-45753-1660732074\")"],"metadata":{"colab":{"base_uri":"https://localhost:8080/"},"id":"hiyzDLzwGqxc","executionInfo":{"status":"ok","timestamp":1668782575688,"user_tz":-330,"elapsed":14,"user":{"displayName":"Lhavanya Devi
Nagarajan","userId":"14108198786569258516"}},"outputId":"f354d308-1e4f-4649-8803-137b9ca6ad80"},"execution_count":33,"outputs":[{"output_type":"stream","name":"stdout","text":["This dataset has been created and uploaded by IBM-TeamID-IBM-Project-45753-1660732074\n"]}]},{"cell_type":"code","source":["train_datagen = ImageDataGenerator(rescale=1./255,zoom_range=0.2,horizontal_flip=True, vertical_flip=False)\n"],"metadata":{"id":"AIwHQVQsG03J","executionInfo":{"status":"ok","timestamp":1668782575689,"user_tz":-330,"elapsed":13,"user":{"displayName":"Lhavanya Devi
Nagarajan","userId":"14108198786569258516"}}},"execution_count":34,"outputs":[]},{"cell_type":"code","source":["test_datagen=
ImageDataGenerator(rescale=1./255)"],"metadata":{"id":"WozX5V7DHBUK","executionInfo":{"statu

s":"ok","timestamp":1668782575690,"user_tz":-330,"elapsed":14,"user":{"displayName":"Lhavanya Devi Nagarajan","userId":"14108198786569258516"}}},"execution_count":35,"outputs":[]},{"cell_type":"code","source":["x_train = train_datagen.flow_from_directory('/content/drive/MyDrive/dataset/training_set',target_size=(64,64), batch_size=300,\n","   class_mode='categorical', color_mode = \"grayscale\")"],"metadata":{"id":"o7cX0hvgJxpb"},"execution_count":null,"outputs":[]},{"cell_type":"markdown","source":["Found 15750 images belonging to 9 classes."],"metadata":{"id":"cRlIkbF3MY2I"}},{"cell_type":"code","source":["x_test = test_datagen.flow_from_directory('/content/drive/MyDrive/dataset/test_set',target_size=(64,64), batch_size=300,\n","   class_mode='categorical', color_mode = \"grayscale\")"],"metadata":{"id":"qbFpdxHHMW0n"},"execution_count":null,"outputs":[]},{"cell_type":"markdown","source":["Found 2250 images belonging to 9 classes."],"metadata":{"id":"-fxMy9suMqQk"}},{"cell_type":"code","source":["x_train.class_indices"],"metadata":{"id":"r5df74-hMtcU"},"execution_count":null,"outputs":[]},{"cell_type":"markdown","source":["{'A': 0, 'B': 1, 'C': 2, 'D': 3, 'E': 4, 'F': 5, 'G': 6, 'H': 7, 'I': 8}"],"metadata":{"id":"V7KpEEiFMw0e"}},{"cell_type":"code","source":["x_test.class_indices"],"metadata":{"id":"ioE-dGXJMzlu"},"execution_count":null,"outputs":[]},{"cell_type":"markdown","source":["{'A': 0, 'B': 1, 'C': 2, 'D': 3, 'E': 4, 'F': 5, 'G': 6, 'H': 7, 'I': 8}"],"metadata":{"id":"Zt2oAEarM2AL"}}]}

● Applying ImageDataGenerator Functionality To Train And Test Set

```
{
 "nbformat": 4,
 "nbformat_minor": 0,
 "metadata": {
  "colab": {
   "provenance": []
  },
  "kernelspec": {
   "name": "python3",
    "display_name": "Python 3"
   },
   "language_info": {
    "name": "python"
   }
  },
  "cells": [
   {
    "cell_type": "markdown",
    "source": [
     "Image Preprocessing"
    ],
    "metadata": {
     "id": "Zm1YQ0LCQRQ6"
    }
   },
   {
    "cell_type": "markdown",
    "source": [
     "Import ImageDataGenerator Library And Configure It"
    ],
    "metadata": {
     "id": "mP7jaBGgQQ8D"
    }
   },
  {
   "cell_type": "code",
   "source": [
    "from tensorflow.keras.preprocessing.image import ImageDataGenerator"
   ],
   "metadata": {
    "id": "Sadth9M-Qh6U"
   },
   "execution_count": 1,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "# Training Datagen\n",
    "train_datagen =
ImageDataGenerator(rescale=1/255,zoom_range=0.2,horizontal_flip=True,vertical_flip=
False)\n",
    "# Testing Datagen\n",
    "test_datagen = ImageDataGenerator(rescale=1/255)"
   ],
   "metadata": {
    "id": "EFDv-PMYQv_S"
   },
   "execution_count": 2,
   "outputs": []
```

```
    },
    {
      "cell_type": "code",
      "source": [
        "import tensorflow as tf\n",
        "import os\n",
        "from tensorflow.keras.models import Sequential\n",
        "from tensorflow.keras.layers import Dense, Conv2D, Flatten, Dropout,
MaxPooling2D\n",
        "from tensorflow.keras.preprocessing.image import ImageDataGenerator\n",
        "import numpy as np\n",
        "import matplotlib.pyplot as plt\n",
        "import IPython.display as display\n",
        "from PIL import Image\n",
        "import pathlib"
      ],
      "metadata": {
        "id": "vCuYI3MfQ0Lt"
      },
      "execution_count": 3,
      "outputs": []
    }
  ]
}
\
```

7.2.MODEL BUILDING

```
{
  "nbformat": 4,
  "nbformat_minor": 0,
  "metadata": {
    "colab": {
      "provenance": []
    },
    "kernelspec": {
      "name": "python3",
      "display_name": "Python 3"
    },
    "language_info": {
      "name": "python"
    }
  },
  "cells": [
    {
      "cell_type": "markdown",
      "source": [
        "**Model Building**"
      ],
      "metadata": {
        "id": "PRsJ3ijtgnn0"
      }
```

```
},
{
  "cell_type": "markdown",
  "source": [
    "Adding The Convolution Layer"
  ],
  "metadata": {
    "id": "miFQeQpZgnaA"
  }
},
{
  "cell_type": "code",
  "execution_count": 1,
  "metadata": {
    "id": "zziyZDbTghKJ"
  },
  "outputs": [],
  "source": [
    "import numpy as np\n",
    "import matplotlib.pyplot as plt"
  ]
},
{
  "cell_type": "code",
  "source": [
      "from tensorflow.keras.preprocessing.image import ImageDataGenerator"
     ],
     "metadata": {
      "id": "DqMpPtNfg2Xo"
     },
     "execution_count": 2,
     "outputs": []
    },
    {
     "cell_type": "code",
     "source": [
      "# Training Datagen\n",
      "train_datagen =
ImageDataGenerator(rescale=1/255,zoom_range=0.2,horizontal_flip=True,vertical_flip=
False)\n",
      "# Testing Datagen\n",
      "test_datagen = ImageDataGenerator(rescale=1/255)"
     ],
     "metadata": {
      "id": "GOFehXMFg2UL"
     },
     "execution_count": 3,
     "outputs": []
    },
    {
     "cell_type": "code",
    "source": [
     "# Training Dataset\n",
"x_train=train_datagen.flow_from_directory(r'/content/drive/MyDrive/Dataset/training
_set',target_size=(64,64), class_mode='categorical',batch_size=900)\n",
     "# Testing Dataset\n",
"x_test=test_datagen.flow_from_directory(r'/content/drive/MyDrive/Dataset/test_set',tar
get_size=(64,64), class_mode='categorical',batch_size=900)\n"
    ],
    "metadata": {
     "colab": {
      "base_uri": "https://localhost:8080/"
     },
     "id": "LKPcW8pEg2E9",
     "outputId": "60ed574c-16dc-4d5c-aa96-8708a69626de"
    },
    "execution_count": 4,
    "outputs": [
     {
      "output_type": "stream",
      "name": "stdout",
      "text": [
       "Found 4964 images belonging to 9 classes.\n",
       "Found 0 images belonging to 9 classes.\n"
      ]
     }
```

```
  ]
 },
 {
  "cell_type": "code",
  "source": [
   "# let img1 be an image with no features\n",
   "img1 = np.array([np.array([200, 200]), np.array([200, 200])])\n",
   "img2 = np.array([np.array([200, 200]), np.array([0, 0])])\n",
   "img3 = np.array([np.array([200, 0]), np.array([200, 0])])\n",
   "  \n",
   "kernel_horizontal = np.array([np.array([2, 2]), np.array([-2, -2])])\n",
   "print(kernel_horizontal, 'is a kernel for detecting horizontal edges')\n",
   "  \n",
   "kernel_vertical = np.array([np.array([2, -2]), np.array([2, -2])])\n",
   "print(kernel_vertical, 'is a kernel for detecting vertical edges')"
  ],
  "metadata": {
   "colab": {
    "base_uri": "https://localhost:8080/"
   },
   "id": "a17jSFg6g1_c",
   "outputId": "5a4d48eb-8ac4-46e6-e4aa-1e547675df0f"
  },
  "execution_count": 5,
  "outputs": [
    {
      "output_type": "stream",
      "name": "stdout",
      "text": [
        "[[ 2  2]\n",
        " [-2 -2]] is a kernel for detecting horizontal edges\n",
        "[[ 2 -2]\n",
        " [ 2 -2]] is a kernel for detecting vertical edges\n"
      ]
    }
  ]
},
{
  "cell_type": "code",
  "source": [
    "# We will apply the kernels on the images by\n",
    "# elementwise multiplication followed by summation\n",
    "def apply_kernel(img, kernel):\n",
    "    return np.sum(np.multiply(img, kernel))\n",
    "  \n",
    "# Visualizing img1\n",
    "plt.imshow(img1)\n",
    "plt.axis('off')\n",
    "plt.title('img1')\n",
    "plt.show()\n",
    "\n",
    "# Checking for horizontal and vertical features in image1\n",
    "print('Horizontal edge confidence score:', apply_kernel(img1, \n",
    "                                    kernel_horizontal))\n",
    "print('Vertical edge confidence score:', apply_kernel(img1, \n",
    "                                    kernel_vertical))"
   ],
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/",
     "height": 299
    },
    "id": "H3ZgqJtxg14s",
    "outputId": "660f2a40-bcc2-45ad-bc6e-7b658ac3b8fd"
   },
   "execution_count": 6,
   "outputs": [
    {
     "output_type": "display_data",
     "data": {
      "text/plain": [
       "<Figure size 432x288 with 1 Axes>"
      ],
      "image/png":
"iVBORw0KGgoAAAANSUhEUgAAAOcAAAD3CAYAAADmIkO7AAAABHNCSVQI
CAgIfAhkiAAAAAlwSFlzAAALEgAACxIB0t1+/AAAADh0RVh0U29mdHdhcmUAbW
F0cGxvdGxpYiB2ZXJzaW9uMy4yLjIsIGh0dHA6Ly9tYXRwbG90bGliLm9yZy+WH4y
```

JAAAExUlEQVR4nO3bzculcxzH8c93QpLylBIRWc7KYhYjyUbZyEJWKMWGnX+AS
Rb+AQuzk5JpSrKbsVFsSCYLZXYjSgpLz+Zncd90L+4mM3PPnM/cXq86dc7Vda6+p3
Pe/a7zNGutAH0ObHoAYHfihFLihFLihFLihFLihFLiLDUzX87MQ5ueg80RZ6m11sG11
od7fdyZOTozp2fm7Mw8s9fHZ++I8//niyQvJPl804NwbldtegB2NzNnkjyX5IEkB5P8luSx
JGeSPL59eXF7+7NrrZZPb97snyZtJ7kvySZLTSW5Yaz2VJGut17f3+/XyPRouhJXzyvBo
kreS3JTkVJIT2Xru7kjySpI3duz7dpJPk9yS5EiSpy/noOwdcV4ZPlprnVhr/ZnkeJJbk7y
21vojyTtJ7p6ZG2fmriSHkry01vp9rfVxkvc3NzYXQ5xXhu93XP8lyQ9rrb923E6S65Pcn
uSntdbPO/b/5jLMxyUgzv3luyQ3z8x1O7bceb7ZmYOZ+v96
r+2t1+bZJJcPTPXzozXQSFPyv7zZJLDSX5M8mqSY9n6RPcfJ7N1Knx/kqPb1x+8zDPy
H4w/W+9vM3MsyVdrrZc3PQvnx8q5z8zMoZm5d2YOzMwj2fpu9L1Nz8X58yOE/ee2J
O9m63vOb5M8v9Y6tdmRuBBOa6GU01oodc7T2ocPPGFZhUvsg7PHZ7ftVk4oJU4oJU
4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU
4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU
4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU
4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU
4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU
4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU
4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU
4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU
4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU
4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU
4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU
4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oNWutTc8A7MLKCaXECaXECa
XECaXECaXECaX+BlePefRt35pgAAAAAElFTkSuQmCC\n"

},

"metadata": {

"needs_background": "light"

}

},

{

"output_type": "stream",

"name": "stdout",

"text": [

"Horizontal edge confidence score: 0\n",

"Vertical edge confidence score: 0\n"

```
      ]
    }
  ]
},
{
  "cell_type": "code",
  "source": [
    "# Visualizing img2\n",
    "plt.imshow(img2)\n",
    "plt.axis('off')\n",
    "plt.title('img2')\n",
    "plt.show()\n",
    "\n",
    "# Checking for horizontal and vertical features in image2\n",
    "print('Horizontal edge confidence score:', apply_kernel(img2, \n",
    "                                            kernel_horizontal))\n",
    "print('Vertical edge confidence score:', apply_kernel(img2, \n",
    "                                            kernel_vertical))"
  ],
  "metadata": {
    "colab": {
      "base_uri": "https://localhost:8080/",
      "height": 299
    },
    "id": "_IFvDkyhhWh_",
    "outputId": "838ef263-75b2-4126-cf17-1240ce3c876d"
   },
   "execution_count": 7,
   "outputs": [
    {
     "output_type": "display_data",
     "data": {
      "text/plain": [
       "<Figure size 432x288 with 1 Axes>"
      ],
      "image/png":
"iVBORw0KGgoAAAANSUhEUgAAAOcAAAD3CAYAAADmIkO7AAAABHNCSVQI
CAgIfAhkiAAAAAlwSFlzAAALEgAACxIB0t1+/AAAADh0RVh0U29mdHdhcmUAbW
F0cGxvdGxpYiB2ZXJzaW9uMy4yLjIsIGh0dHA6Ly9tYXRwbG90bGliLm9yZy+WH4y
JAAAFB0lEQVR4nO3azYvucxjH8c91koU4niNPJWFhw+IsRLIRG0mWWIiNpbKwI/E
vKEuR09kIO2xsLEhZKVI6ohx5WEhkwdfiHjWdxsnoOPfHeL3qrpnf/OZ3X3fTu+t+mF
lrBehzaNsDAHsTJ5QSJ5QSJ5QSJ5QSJ5QSZ6mZ+Xhm7tz2HGyPOEuttW5aa717Oq8
5MzfMzBsz8+3M/DAzb83MjafzPjGSN5PcmOSyJB8keWOrE/GXxn8IdZqZ40
keS3J7kpuS/JrkviTHkzywc3ti5/ija623d37v2iQvJbk3yLyftJPk1y/lrroT3u46Ik3ye5ZK31/b/7
iNgvm/O/4d4kLye5MMlHSd7K5m93ZZJnk7y469xxXs9mIFyd5JsnDp7juHUlOCLOTzV
nqpM1521rrrp3j9yY5ms02/G1mzkvyYzbhHk7yeZLDa62fd85/JUlO3pwzc1U2m/XJtdb
RM/Kg2Beb87/hm11f/5Lku7XWb7u+T5Jzk1yR5Ic/w9zx5ckXm5lLk7yd5AVh9hLnwfJ1
kotm5pxdx67efcLMXJhNmG+utZ4/k8OxP+I8QNZaXyT5MMkzM3M3P2zNyazevVJMnM
HM7m9ep7a62ntjQmf5M4D54Hk9yazbuwzyU5ls07uklyf5IjSR6ZmZ923a7ZzqicijeEDri
ZOZbkk7XW09uehf2xOQ+YmTkyM9fNzKGZuSebz0Zf3/Zc7N9Z2x6x6A0+7yJK9l8znnV
0keX2t9tN2R+Cc8rYVSntZCqVM+rf39xPXWKvzLDl3+2ex5/EwPAvw94oRS4oRS4oRS
4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4o
RS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS
4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4o
RS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS
4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4o
RS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS
4oRS4oRS4oRS4oRS4oRSZ53qh3dfcfOZmgP+t975fe/jNieUEieUEieUEieUEieUEieUEie
UEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEie
UEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEie
UEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEie
UEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEie
```

UEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEie
UEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUmrXWtmcA9mBzQilxQilxQilxQ
ilxQilxQqk/AHW+mUBIaBO/AAAAAElFTkSuQmCC\n"

```
        },
        "metadata": {
          "needs_background": "light"
        }
      },
      {
        "output_type": "stream",
        "name": "stdout",
        "text": [
          "Horizontal edge confidence score: 800\n",
          "Vertical edge confidence score: 0\n"
        ]
      }
    ]
  },
  {
    "cell_type": "code",
    "source": [
      "# Visualizing img3\n",
      "plt.imshow(img3)\n",
      "plt.axis('off')\n",
      "plt.title('img3')\n",
      "plt.show()\n",
    "  \n",
    "# Checking for horizontal and vertical features in image3\n",
    "print('Horizontal edge confidence score:', apply_kernel(img3, \n",
    "                                    kernel_horizontal))\n",
    "print('Vertical edge confidence score:', apply_kernel(img3, \n",
    "                                    kernel_vertical))"
   ],
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/",
     "height": 299
    },
    "id": "Rsp88WaahWah",
    "outputId": "fd9c9485-a74a-4fa1-f201-7930d72f4c0e"
   },
   "execution_count": 8,
   "outputs": [
    {
     "output_type": "display_data",
     "data": {
      "text/plain": [
       "<Figure size 432x288 with 1 Axes>"
      ],
      "image/png":
"iVBORw0KGgoAAAANSUhEUgAAAOcAAAD3CAYAAADmIkO7AAAABHNCSVQI
CAgIfAhkiAAAAAlwSFlzAAALEgAACxIB0t1+/AAAADh0RVh0U29mdHdhcmUAbW
F0cGxvdGxpYiB2ZXJzaW9uMy4yLjIsIGh0dHA6Ly9tYXRwbG90bGliLm9yZy+WH4y
JAAAFOklEQVR4nO3awYudZxnG4fuJtaZFK41iS4qIFt100W6ESrJwobVQQnGt7tz4F
2hXBv8EwYVQSktDQwgUcSe6EKGUFiHQILQUSkqK1WBEFLRgk7eLmcAQh4SJ05
470+tazXznO988h+HH+57vnFlrBehzaNMDALsTJ5QSJ5QSJ5QSJ5QSJ5QSZ6mZ+dP
MfHPTc7A54iy11nporfX7/bzmzHx+Zl6amcsz84+ZeXlmju3n32D/jC8hfHzMzOEkX0ry
ZpKV5MkkzyT5wlrr/U3Oxv+ycpaamQsz862ZOTkzZ2fm1Mz8a2bOz8zXZuapmbk0Mx
dn5rEdz/vyzPxh+9zfzcwvZuZUkqy13ltrvbHWuppkklxJcm+SI5t5ldyIOG8PJ5I8n62Qzi
X5Tbb+dw8k+VmSX+4494Ukryb5XJKTSX5w/cVm5rUk7yX5dZKn11qXPsTZuUW2ta
Vm5kKSHyY5nuTYWuvb28dPJDmd5LNrrSsz85kk/8xWuPckeSvJPWutf2+ff23V/P511
z+c5LtJ7lxrPfeRvCj2xMp5e/jrjp//k+Rva60rO35Pkk8nOZrk79fC3HZxtwub/3FPJ/nJzD
y83wPz/xPnwfJukiMzc/eOY1+8CpxHiBrrbeT/DHJyZm5c2a+ka33q0
mSmXl0Zo5vP3bXzPw4yX1JXtnQyNzAHZsegH33vSTPJrmcrRtDZ5J8Yvux
Wyv8mOZ/kibXWnz/6MbkZN4QOuJk5k+T1tdZPNz0Le2Nbe8DMzNdn5sGZOTQzj2f
riwa/2vRc7J1t7cFzf5IXs/U55ztJfrTWOrfZkbgVtrvVQyrYWSt1wW3v1L1+1rN5mvnP0k
U2PwB799urZ2e24lRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRN
KiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRN
KiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRN
KiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRN
KiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRN
KiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRN
KiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRN
KiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRN
KiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRN
KiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRN
KiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRN
KiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRN
KiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRN
KiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRN
KiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKzVpr0zMAu7ByQilxQilxQil
xQilxQilxQqkPACw7n6oaSSoWAAAAAElFTkSuQmCC\n"

     },

     "metadata": {

      "needs_background": "light"

     }

    },

    {

     "output_type": "stream",

     "name": "stdout",
      "text": [
        "Horizontal edge confidence score: 0\n",
        "Vertical edge confidence score: 800\n"
      ]
    }
  ]
},
{
  "cell_type": "code",
  "source": [
    "print(\"Len x-train : \", len(x_train))\n",
    "print(\"Len x-test : \", len(x_test))\n"
  ],
  "metadata": {
    "colab": {
      "base_uri": "https://localhost:8080/"
    },
    "id": "4stxf4JRhWXK",
    "outputId": "050f779a-bf39-47ed-ad95-4cb750f1ae41"
  },
  "execution_count": 9,
  "outputs": [
    {
      "output_type": "stream",
      "name": "stdout",
    "text": [
     "Len x-train :  6\n",
     "Len x-test :  0\n"
    ]
   }
  ]
 },
 {
  "cell_type": "code",
  "source": [
   "# The Class Indices in Training Dataset\n",
   "x_train.class_indices"
  ],
  "metadata": {
   "colab": {
    "base_uri": "https://localhost:8080/"
   },
   "id": "aOKiD6_AhWMd",
   "outputId": "1b9e68ce-aaaa-4817-ac39-03538bec4446"
  },
  "execution_count": 10,
  "outputs": [
   {
    "output_type": "execute_result",
    "data": {
        "text/plain": [
          "{'A': 0, 'B': 1, 'C': 2, 'D': 3, 'E': 4, 'F': 5, 'G': 6, 'H': 7, 'I': 8}"
        ]
      },
      "metadata": {},
      "execution_count": 10
    }
  ]
},
{
  "cell_type": "markdown",
  "source": [
    "**Model Creation**"
  ],
  "metadata": {
    "id": "5nj6XMhFhkXt"
  }
},
{
  "cell_type": "code",
  "source": [
    "# Importing Libraries\n",
    "from tensorflow.keras.models import Sequential\n",
    "from tensorflow.keras.layers import Convolution2D,MaxPooling2D,Flatten,Dense"
  ],
  "metadata": {
    "id": "qomXO6KjhtlA"
  },
  "execution_count": 11,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "# Creating Model\n",
    "model=Sequential()"
  ],
  "metadata": {
    "id": "PdifbnNzht4N"
  },
  "execution_count": 12,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "# Adding Layers\n",
    "model.add(Convolution2D(32,(3,3),activation='relu',input_shape=(64,64,3)))"
  ],
  "metadata": {
      "id": "dO93GJdphtsd"
    },
    "execution_count": 13,
    "outputs": []
  }
 ]
}
```

● Initializing The Model

```
{
 "nbformat": 4,
 "nbformat_minor": 0,
 "metadata": {
  "colab": {
   "provenance": []
  },
  "kernelspec": {
   "name": "python3",
   "display_name": "Python 3"
  },
  "language_info": {
   "name": "python"
  }
 },
 "cells": [
  {
   "cell_type": "markdown",
   "source": [
    "**Model Building**"
   ],
   "metadata": {
    "id": "0u4hq4-nef62"
   }
  },
  {
   "cell_type": "markdown",
   "source": [
    "Initializing The Model"
   ],
   "metadata": {
    "id": "0n7kyxv5egEt"
   }
  },
  {
   "cell_type": "code",
   "source": [
    "from tensorflow.keras.preprocessing.image import ImageDataGenerator"
   ],
   "metadata": {
    "id": "bcX2Eug2eqsM"
   },
    "execution_count": 1,
    "outputs": []
   },
   {
    "cell_type": "code",
    "source": [
     "spatial_dropout=0.05\n",
     "recurrent_dropout=0.1"
    ],
    "metadata": {
     "id": "bS-OSz0oetQ8"
    },
    "execution_count": 2,
    "outputs": []
   },
   {
    "cell_type": "code",
    "source": [
     "# Training Datagen\n",
     "train_datagen = 
ImageDataGenerator(rescale=1/255,zoom_range=0.2,horizontal_flip=True,vertical_flip=
False)\n",
     "# Testing Datagen\n",
     "test_datagen = ImageDataGenerator(rescale=1/255)"
    ],
    "metadata": {
      "id": "OEL-eZU4etKm"
    },
    "execution_count": null,
    "outputs": []
   },
   {
    "cell_type": "code",
    "source": [
      "# Training Dataset\n",
"x_train=train_datagen.flow_from_directory('/content/drive/MyDrive/Dataset/training_set',target_size=(64,64), class_mode='categorical',batch_size=900)\n",
      "# Testing Dataset\n",
"x_test=test_datagen.flow_from_directory('/content/drive/MyDrive/Dataset/test_set',target_size=(64,64), class_mode='categorical',batch_size=900)\n"
    ],
    "metadata": {
      "id": "ub_PFFHfexaV"
    },
    "execution_count": null,
    "outputs": []
   },
   {
    "cell_type": "markdown",
    "source": [
      "Found 15760 images belonging to 9 classes.\n",
    "Found 2250 images belonging to 9 classes."
   ],
   "metadata": {
    "id": "jAQBE0SufUr-"
   }
  },
  {
   "cell_type": "code",
   "source": [
    "print(\"Len x-train : \", len(x_train))\n",
    "print(\"Len x-test : \", len(x_test))"
   ],
   "metadata": {
    "id": "vvdUsVNnfXNV"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "markdown",
   "source": [
    "Len x-train :  18\n",
    "Len x-test :  3"
   ],
   "metadata": {
    "id": "a6a0JI4Mfadu"
  }
 },
 {
  "cell_type": "code",
  "source": [
    "# The Class Indices in Training Dataset\n",
    "x_train.class_indices"
  ],
  "metadata": {
    "id": "dubajLk0fd47"
  },
  "execution_count": null,
  "outputs": []
 },
 {
  "cell_type": "markdown",
  "source": [
    "**Model Creation**"
  ],
  "metadata": {
    "id": "Tfd_00cOfhg-"
  }
 },
 {
    "cell_type": "code",
    "source": [
     "# Importing Libraries\n",
     "from tensorflow.keras.models import Sequential\n",
     "from tensorflow.keras.layers import Convolution2D,MaxPooling2D,Flatten,Dense"
    ],
    "metadata": {
     "id": "XAuf7RsPflE0"
    },
    "execution_count": 6,
    "outputs": []
   },
   {
    "cell_type": "code",
    "source": [
     "# Creating Model\n",
     "model=Sequential()"
    ],
    "metadata": {
     "id": "fu7ASeDIfn6E"
    },
    "execution_count": 7,
    "outputs": []
   }
  ]
```

}

● Adding The Convolution Layer

```
{
  "nbformat": 4,
  "nbformat_minor": 0,
  "metadata": {
    "colab": {
      "provenance": []
    },
    "kernelspec": {
      "name": "python3",
      "display_name": "Python 3"
    },
    "language_info": {
      "name": "python"
    }
  },
  "cells": [
    {
      "cell_type": "markdown",
      "source": [
        "**Model Building**"
      ],
      "metadata": {
        "id": "PRsJ3ijtgnn0"
      }
    },
    {
      "cell_type": "markdown",
      "source": [
        "Adding The Convolution Layer"
      ],
      "metadata": {
        "id": "miFQeQpZgnaA"
      }
    },
    {
      "cell_type": "code",
      "execution_count": 1,
      "metadata": {
```

```
      "id": "zziyZDbTghKJ"
     },
     "outputs": [],
     "source": [
      "import numpy as np\n",
      "import matplotlib.pyplot as plt"
     ]
    },
    {
     "cell_type": "code",
     "source": [
      "from tensorflow.keras.preprocessing.image import ImageDataGenerator"
     ],
     "metadata": {
      "id": "DqMpPtNfg2Xo"
     },
     "execution_count": 2,
     "outputs": []
    },
    {
     "cell_type": "code",
     "source": [
      "# Training Datagen\n",
      "train_datagen =
ImageDataGenerator(rescale=1/255,zoom_range=0.2,horizontal_flip=True,vertical_flip=False)\n",
      "# Testing Datagen\n",
      "test_datagen = ImageDataGenerator(rescale=1/255)"
     ],
     "metadata": {
      "id": "GOFehXMFg2UL"
     },
     "execution_count": 3,
     "outputs": []
    },
    {
     "cell_type": "code",
     "source": [
      "# Training Dataset\n",

"x_train=train_datagen.flow_from_directory(r'/content/drive/MyDrive/Dataset/training_set',target
_size=(64,64), class_mode='categorical',batch_size=900)\n",
        "# Testing Dataset\n",
"x_test=test_datagen.flow_from_directory(r'/content/drive/MyDrive/Dataset/test_set',target_siz
e=(64,64), class_mode='categorical',batch_size=900)\n"
      ],
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "LKPcW8pEg2E9",
        "outputId": "60ed574c-16dc-4d5c-aa96-8708a69626de"
      },
      "execution_count": 4,
      "outputs": [
        {
          "output_type": "stream",
          "name": "stdout",
          "text": [
            "Found 4964 images belonging to 9 classes.\n",
            "Found 0 images belonging to 9 classes.\n"
          ]
        }
      ]
    },
    {
      "cell_type": "code",
      "source": [
        "# let img1 be an image with no features\n",
        "img1 = np.array([np.array([200, 200]), np.array([200, 200])])\n",
        "img2 = np.array([np.array([200, 200]), np.array([0, 0])])\n",
        "img3 = np.array([np.array([200, 0]), np.array([200, 0])])\n",
        "  \n",
        "kernel_horizontal = np.array([np.array([2, 2]), np.array([-2, -2])])\n",
        "print(kernel_horizontal, 'is a kernel for detecting horizontal edges')\n",
        "  \n",
        "kernel_vertical = np.array([np.array([2, -2]), np.array([2, -2])])\n",
        "print(kernel_vertical, 'is a kernel for detecting vertical edges')"
      ],
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
```

```
   },
   "id": "a17jSFg6g1_c",
   "outputId": "5a4d48eb-8ac4-46e6-e4aa-1e547675df0f"
  },
  "execution_count": 5,
  "outputs": [
   {
    "output_type": "stream",
    "name": "stdout",
    "text": [
     "[[ 2  2]\n",
     " [-2 -2]] is a kernel for detecting horizontal edges\n",
     "[[ 2 -2]\n",
     " [ 2 -2]] is a kernel for detecting vertical edges\n"
    ]
   }
  ]
 },
 {
  "cell_type": "code",
  "source": [
   "# We will apply the kernels on the images by\n",
   "# elementwise multiplication followed by summation\n",
   "def apply_kernel(img, kernel):\n",
   "    return np.sum(np.multiply(img, kernel))\n",
   "  \n",
   "# Visualizing img1\n",
   "plt.imshow(img1)\n",
   "plt.axis('off')\n",
   "plt.title('img1')\n",
   "plt.show()\n",
   "\n",
   "# Checking for horizontal and vertical features in image1\n",
   "print('Horizontal edge confidence score:', apply_kernel(img1, \n",
   "                                            kernel_horizontal))\n",
   "print('Vertical edge confidence score:', apply_kernel(img1, \n",
   "                                            kernel_vertical))"
  ],
  "metadata": {
   "colab": {
    "base_uri": "https://localhost:8080/",
          "height": 299
        },
        "id": "H3ZgqJtxg14s",
        "outputId": "660f2a40-bcc2-45ad-bc6e-7b658ac3b8fd"
      },
      "execution_count": 6,
      "outputs": [
        {
          "output_type": "display_data",
          "data": {
            "text/plain": [
              "<Figure size 432x288 with 1 Axes>"
            ],
            "image/png":
"iVBORw0KGgoAAAANSUhEUgAAAOcAAAD3CAYAAADmIkO7AAAABHNCSVQICAgIfAhkiAAAA
AIwSFlzAAALEgAACxIB0t1+/AAAADh0RVh0U29mdHdhcmUAbWF0cGxvdGxpYiB2ZXJzaW9uM
y4yLjIsIGh0dHA6Ly9tYXRwbG90bGliLm9yZy+WH4yJAAAExUlEQVR4nO3bzculcxzH8c93QpLylBI
RWc7KYhYjyUbZyEJWKMWGnX+ASRb+AQuzk5JpSrKbsVFsSCYLZXYjSgpLz+Zncd90L+4mM3P
PnM/cXq86dc7Vda6+p3Pe/a7zNGutAH0ObHoAYHfihFLihFLihFLihFLihFLiLDUzX87uQ5ueg80R
Z6m11sG11od11od7fdyZOTozp2fm7Mw8s9fHZ++I8//niyQPJPl8P1jtdtNnkjB5P8B8lu
SxJGeSPL59eXF7+7NrZPSW5Yaz2VJGut17f3+/XyPRouhJXzyvBokreS3JT
kVJIT2Xru7kjySpI3duz7dpJPk9yS5EiSpy/noOwdcV4ZPlrprnVhr/ZnkeJJbk7y21voj
yTtJ7p6ZG2f
mriSHkry01vp9rfVxkvc3NzYXQ5xXhu93XP8IyQ9rrb923E6S65PcnuSntdbPO/b/5jLMxyUgzv3luy
Q3z8x1O7bdudualhuDji3EfWWl8n+SzJZkZm5ZmYOZ+v96r+2t1+bZJJcPTPXozXQSFPyv7zZJLDSX
5M8mqSY9n6RPcfJ7N1Knx/kqPb1x+8zDPyH4w/W+9vM3MsyvdrrZc3c3PQvnvx8q5z8zOE/ee2JO9m63vOb1T2ocPPGFZhU
YOzMwj2fpu9L1Nz8X58yOE/ee2JO9m63vOb5M8v9Y6tdmRuBBBOa6GU01oodc7T2ocPPGFZhU
vsg7PHZ7ftVk4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4o
JU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJ
U4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJ
U4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJ
U4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJ
U4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJ
U4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJ
U4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJ
U4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJ
U4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJ
U4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJ
U4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJU4oJ
U4oJU4oJU4oNWutTc8A7MLKCaXECaXECaXECaXECaXECaX+BlePefRt35pgAAAAAElFTkSuQ
mCC\n"
          },
      "metadata": {
        "needs_background": "light"
      }
    },
    {
      "output_type": "stream",
      "name": "stdout",
      "text": [
        "Horizontal edge confidence score: 0\n",
        "Vertical edge confidence score: 0\n"
      ]
    }
  ]
},
{
  "cell_type": "code",
  "source": [
    "# Visualizing img2\n",
    "plt.imshow(img2)\n",
    "plt.axis('off')\n",
    "plt.title('img2')\n",
    "plt.show()\n",
    "\n",
    "# Checking for horizontal and vertical features in image2\n",
    "print('Horizontal edge confidence score:', apply_kernel(img2, \n",
    "                                          kernel_horizontal))\n",
    "print('Vertical edge confidence score:', apply_kernel(img2, \n",
    "                                          kernel_vertical))"
  ],
  "metadata": {
    "colab": {
      "base_uri": "https://localhost:8080/",
      "height": 299
    },
    "id": "_IFvDkyhhWh_",
    "outputId": "838ef263-75b2-4126-cf17-1240ce3c876d"
  },
  "execution_count": 7,
  "outputs": [
    {
      "output_type": "display_data",
          "data": {
           "text/plain": [
            "<Figure size 432x288 with 1 Axes>"
           ],
           "image/png":
"iVBORw0KGgoAAAANSUhEUgAAAOcAAAD3CAYAAADmIkO7AAAABHNCSVQICAgIfAhkiAAAA
AlwSFlzAAALEgAACxIB0t1+/AAAADh0RVh0U29mdHdhcmUAbWF0cGxvdGxpYiB2ZXJzaW9uM
y4yLjIsIGh0dHA6Ly9tYXRwbG90bGliLm9yZy+WH4yJAAAFB0lEQVR4nO3azYvucxjH8c91koU4ni
NPJWFhw+IsRLIRG0mWWIiNpbKwI/EvKEuR09kIO2xsLEhZKVI6ohx5WEhkwdfiHjWdxsnoOPfHe
L3qrpnf/OZ3X3fTu+t+mFlrBehzaNsDAHsTJ5QSJ5QSJ5QSJ5QSJ5QSZ6mZ+Xhm7tz2HGyPOEu
ttW5aa717Oq85MzfMzBsz8+3M/DAzb83MjafzPjh9PcmOrE/GXXn8/rE/GXxn8IdZ
qZ40keS3J7kpuS/JrkviTHkzywc3ti5/ija623d37v2iQvJbklyftJPk1y/IrroT3u46Ik3ye5ZK31/b/7iNg
vm/O/4d4kLye5MMlHSd7K5m93ZZJnk7ycm9JsnDp7juHUlOCLOTzVnqpM1521rrrp
3j9yY5ms02/G1mzkvyYzbhHk7yeZLDa62fd85/JUlO3pwzc1U2m/XJtdbRM/Kg2Beb87/hm11f/5
Lku7XWb7u+T5Jzk1yR5Ic/w9zx5ckXm5lLk7yd5AVh9hLnwfJ1kotm5pxdx67efcLMXJhNmG+utZ
4/k8OxP+I8QNZaXyT5MMkzM3P2zNyazevVJMnMHM7m9ep7a62ntjQmf5M4D54Hk9yazbuwzy
U5ls07ukIyf5IjSR6ZmZ923a7ZzqicijeEDriZOZbkk7XW09uehf2xOQ+YmTkyM9fNzKGZuSU
/Zc7N9Z2x6A0+7yJK9l8znnV0keX2t9tN2R+Cc8rYVSntZCqVM+rf39xPXWKvzLDl3+2ex5/EwPAv
w94oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4o
RS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4o
RS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4o
RS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4o
RS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4o
RS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRS4oRSZ5
3qh3dfcfOZmgP+t975fe/jNieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEi
eUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEie
UEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieU
EieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUE
ieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEieUEie
UEieUEieUEieUEieUEieUEieUEieUmrXWtmcA9mBzQiIxQiIxQiIxQiIxQiIxQqk/AHW+mUBIaBO/AAA
AAElFTkSuQmCC\n"
          },
          "metadata": {
           "needs_background": "light"
          }
         },
         {
          "output_type": "stream",
          "name": "stdout",
          "text": [
           "Horizontal edge confidence score: 800\n",
            "Vertical edge confidence score: 0\n"
          ]
        }
      ]
    },
    {
      "cell_type": "code",
      "source": [
        "# Visualizing img3\n",
        "plt.imshow(img3)\n",
        "plt.axis('off')\n",
        "plt.title('img3')\n",
        "plt.show()\n",
        "  \n",
        "# Checking for horizontal and vertical features in image3\n",
        "print('Horizontal edge confidence score:', apply_kernel(img3, \n",
        "                                            kernel_horizontal))\n",
        "print('Vertical edge confidence score:', apply_kernel(img3, \n",
        "                                            kernel_vertical))"
      ],
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/",
          "height": 299
        },
        "id": "Rsp88WaahWah",
        "outputId": "fd9c9485-a74a-4fa1-f201-7930d72f4c0e"
      },
      "execution_count": 8,
      "outputs": [
        {
          "output_type": "display_data",
          "data": {
            "text/plain": [
              "<Figure size 432x288 with 1 Axes>"
            ],
            "image/png":
"iVBORw0KGgoAAAANSUhEUgAAAOcAAAD3CAYAAADmIkO7AAAABHNCSVQICAgIfAhkiAAAA
AlwSFlzAAALEgAACxIB0t1+/AAAADh0RVh0U29mdHdhcmUAbWF0cGxvdGxpYiB2ZXJzaW9uM
y4yLjIsIGh0dHA6Ly9tYXRwbG90bGliLm9yZy+WH4yJAAAF0klEQVR4nO3awYudZxnG4fuJtaZFK
41iS4qIFt100W6ESrJwobVQnGt7tz4F2hXBv8EwYVQSktDQwgUcSe6EKGUFiHQILQUSkqK1WB
```

```
EFLRgk7eLmcAQh4SJ05470+tazXznO988h+HH+57vnFlrBehzaNMDALsTJ5QSJ5QSJ5QSJ5QSJ
5QSZ6mZ+dPMfHPTc7A54iy11nporfX7/bzmzHx+ZI6amcsz84+ZeXlmju3n32D/jC8hfHzMzOEkX
0ryZpKV5MkkzyT5wlrr/U3Oxv+ycpaamQsz862ZOTkzZ2fm1Mz8a2bOz8zXZuapmbk0Mxdn5rEd
z/vyzPxh+9zfzcwvZuZUkqy13ltrvbHWuppkklxJcm+SI5t5ldyIOG8PJ5I8n62QziX5Tbb+dw8k+Vm
SX+4494Ukryb5XJKTSX5w/cVm5rUk7yX5dZKn11qXPsTZuUW2taVm5kKSHyY5nuTYWuvb28dP
JDmd5LNrrSsz85kk/8xWuPckeSvJPWutf2+ff23V/P511z+c5LtJ7lxrPfeRvCj2xMp5e/jrjp//k+Rva
60rO35Pkk8nOZrk79fC3HZxtwtub3FPJ/nJzDy83wPz/xPnwfJukiMzc/eOY1+8yXM+meQrH95I3
CpxHiBrrbeT/DHJyZm5c2a+ka33q0mSmXl0Zo5vP3bXzPw4yX1JXtnQyNzAHZsegH33vSTPJrm
crRtDZ5J8YvuxTyX5ebZWyv8mOZ/kibXWnz/6MbkZN4QOuJk5k+T1tdZPNz0Le2Nbe8DMzNdn5
sGZOTQzj2friwa/2vRc7J1t7cFzf5IXs/U55ztJfrTWOrfZkbgVtrVQyrYWSt1wW3v1L1+1rN5mvnP0
kU2PwB799urZ2e24lRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKi
RNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiR
NKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRN
KiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNK
iRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKi
RNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiR
NKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRN
KiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNK
iRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKi
RNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiR
NKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRN
KiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKiRNKzVpr0zMAu7ByQiIxQiIxQiIxQiIxQiIxQqkPACw7n6o
aSSoWAAAAAElFTkSuQmCC\n"
         },
         "metadata": {
           "needs_background": "light"
         }
       },
       {
         "output_type": "stream",
         "name": "stdout",
         "text": [
           "Horizontal edge confidence score: 0\n",
           "Vertical edge confidence score: 800\n"
         ]
       }
     ]
   },
   {
     "cell_type": "code",
     "source": [
    "print(\"Len x-train : \", len(x_train))\n",
    "print(\"Len x-test : \", len(x_test))\n"
   ],
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/"
    },
    "id": "4stxf4JRhWXK",
    "outputId": "050f779a-bf39-47ed-ad95-4cb750f1ae41"
   },
   "execution_count": 9,
   "outputs": [
    {
     "output_type": "stream",
     "name": "stdout",
     "text": [
      "Len x-train :  6\n",
      "Len x-test :  0\n"
     ]
    }
   ]
  },
  {
   "cell_type": "code",
   "source": [
    "# The Class Indices in Training Dataset\n",
    "x_train.class_indices"
   ],
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/"
    },
    "id": "aOKiD6_AhWMd",
    "outputId": "1b9e68ce-aaaa-4817-ac39-03538bec4446"
   },
   "execution_count": 10,
   "outputs": [
    {
     "output_type": "execute_result",
     "data": {
      "text/plain": [
        "{'A': 0, 'B': 1, 'C': 2, 'D': 3, 'E': 4, 'F': 5, 'G': 6, 'H': 7, 'I': 8}"
       ]
      },
      "metadata": {},
      "execution_count": 10
     }
    ]
   },
   {
    "cell_type": "markdown",
    "source": [
     "**Model Creation**"
    ],
    "metadata": {
     "id": "5nj6XMhFhkXt"
    }
   },
   {
    "cell_type": "code",
    "source": [
     "# Importing Libraries\n",
     "from tensorflow.keras.models import Sequential\n",
     "from tensorflow.keras.layers import Convolution2D,MaxPooling2D,Flatten,Dense"
    ],
    "metadata": {
     "id": "qomXO6KjhtlA"
    },
    "execution_count": 11,
    "outputs": []
   },
   {
    "cell_type": "code",
    "source": [
     "# Creating Model\n",
     "model=Sequential()"
    ],
    "metadata": {
     "id": "PdifbnNzht4N"
    },
    "execution_count": 12,
    "outputs": []
    },
    {
      "cell_type": "code",
      "source": [
        "# Adding Layers\n",
        "model.add(Convolution2D(32,(3,3),activation='relu',input_shape=(64,64,3)))"
      ],
      "metadata": {
        "id": "dO93GJdphtsd"
      },
      "execution_count": 13,
      "outputs": []
    }
  ]
}
```

● Adding The Pooling Layer

```
{
  "nbformat": 4,
  "nbformat_minor": 0,
  "metadata": {
    "colab": {
      "provenance": []
    },
    "kernelspec": {
      "name": "python3",
      "display_name": "Python 3"
    },
    "language_info": {
      "name": "python"
    }
  },
  "cells": [
    {
      "cell_type": "markdown",
      "source": [
        "**Model Building**"
      ],
      "metadata": {
        "id": "gmIm1Lx2VUma"
      }
    },
{
  "cell_type": "markdown",
  "source": [
    "Adding The Pooling Layer"
  ],
  "metadata": {
    "id": "tu2okTIqVUi1"
  }
},
{
  "cell_type": "code",
  "execution_count": 1,
  "metadata": {
    "id": "3XR6V3haVPwa"
  },
  "outputs": [],
  "source": [
    "from tensorflow.keras.preprocessing.image import ImageDataGenerator"
  ]
},
{
  "cell_type": "code",
  "source": [
    "import numpy as np\n",
    "from keras.models import Sequential\n",
    "from keras.layers import MaxPooling2D"
  ],
  "metadata": {
    "id": "yNRuXUZYVlFP"
  },
  "execution_count": 2,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "# define input image\n",
    "image = np.array([[2, 2, 7, 3],\n",
    "\t\t\t\t[9, 4, 6, 1],\n",
    "\t\t\t\t[8, 5, 2, 4],\n",
    "\t\t\t\t[3, 1, 2, 6]])\n",
    "image = image.reshape(1, 4, 4, 1)"
   ],
   "metadata": {
    "id": "20trB49HVk7-"
   },
   "execution_count": 3,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "# define model containing just a single max pooling layer\n",
    "model = Sequential(\n",
    "\t[MaxPooling2D(pool_size = 2, strides = 2)])\n",
    "\n",
    "# generate pooled output\n",
    "output = model.predict(image)"
   ],
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/"
    },
    "id": "EI1YpATIVk4e",
    "outputId": "bb29d4ff-b33a-4b48-9be7-cf75a3938f99"
   },
   "execution_count": 4,
   "outputs": [
    {
     "output_type": "stream",
     "name": "stdout",
     "text": [
      "1/1 [==============================] - 0s 199ms/step\n"
     ]
    }
   ]
  },
  {
   "cell_type": "code",
   "source": [
    "# print output image\n",
    "output = np.squeeze(output)\n",
        "print(output)"
      ],
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "uGCJkIfMVk0X",
        "outputId": "d5b8fa53-fff8-4724-b9f5-2302a7e68797"
      },
      "execution_count": 5,
      "outputs": [
        {
          "output_type": "stream",
          "name": "stdout",
          "text": [
            "[[9 7]\n",
            " [8 6]]\n"
          ]
        }
      ]
    },
    {
      "cell_type": "code",
      "source": [
        "# Training Datagen\n",
        "train_datagen =
ImageDataGenerator(rescale=1/255,zoom_range=0.2,horizontal_flip=True,vertical_flip=False)\n",
        "# Testing Datagen\n",
        "test_datagen = ImageDataGenerator(rescale=1/255)"
      ],
      "metadata": {
        "id": "Q4nl30xpVktU"
      },
      "execution_count": 6,
      "outputs": []
    },
    {
      "cell_type": "code",
      "source": [
        "# Training Dataset\n",
"x_train=train_datagen.flow_from_directory(r'/content/drive/MyDrive/Dataset/training_set',target_size=(
64,64), class_mode='categorical',batch_size=900)\n",
    "# Testing Dataset\n",

"x_test=test_datagen.flow_from_directory(r'/content/drive/MyDrive/Dataset/test_set',target_size=(64,64),
class_mode='categorical',batch_size=900)\n"
   ],
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/"
    },
    "id": "BgNvlgUEVxLh",
    "outputId": "2a2172e0-65ee-48f4-b1aa-5a8a668d841d"
   },
   "execution_count": 9,
   "outputs": [
    {
     "output_type": "stream",
     "name": "stdout",
     "text": [
      "Found 857 images belonging to 9 classes.\n",
      "Found 0 images belonging to 9 classes.\n"
     ]
    }
   ]
  },
  {
   "cell_type": "code",
   "source": [
    "print(\"Len x-train : \", len(x_train))\n",
    "print(\"Len x-test : \", len(x_test))"
   ],
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/"
    },
    "id": "eHqeJmQVWesy",
    "outputId": "67366357-48f6-4c48-9403-2910c1d09518"
   },
   "execution_count": 10,
   "outputs": [
    {
     "output_type": "stream",
     "name": "stdout",
     "text": [
      "Len x-train :  1\n",
      "Len x-test :  0\n"
     ]
    }
   ]
  },
  {
   "cell_type": "code",
   "source": [
    "# The Class Indices in Training Dataset\n",
    "x_train.class_indices"
   ],
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/"
    },
    "id": "w8CwFim-WiBP",
    "outputId": "e5a58c8e-d165-4707-f2ff-10a433a227b4"
   },
   "execution_count": 11,
   "outputs": [
    {
     "output_type": "execute_result",
     "data": {
      "text/plain": [
       "{'A': 0, 'B': 1, 'C': 2, 'D': 3, 'E': 4, 'F': 5, 'G': 6, 'H': 7, 'I': 8}"
      ]
     },
     "metadata": {},
     "execution_count": 11
    }
   ]
  },
  {
   "cell_type": "markdown",
   "source": [
    "**Model Creation**"
  ],
  "metadata": {
    "id": "LmizRi4fWoo3"
  }
},
{
  "cell_type": "code",
  "source": [
    "# Importing Libraries\n",
    "from tensorflow.keras.models import Sequential\n",
    "from tensorflow.keras.layers import Convolution2D,MaxPooling2D,Flatten,Dense"
  ],
  "metadata": {
    "id": "lEepIZLuWwab"
  },
  "execution_count": 12,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "# Creating Model\n",
    "model=Sequential()\n"
  ],
  "metadata": {
    "id": "FuoajD7tWzVm"
  },
  "execution_count": 13,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "# Adding Layers\n",
    "model.add(Convolution2D(32,(3,3),activation='relu',input_shape=(64,64,3)))"
  ],
  "metadata": {
    "id": "F5Hvy6KNW2iT"
  },
  "execution_count": 14,
  "outputs": []
```

```
    },
    {
      "cell_type": "code",
      "source": [
        "model.add(MaxPooling2D(pool_size=(2,2)))"
      ],
      "metadata": {
        "id": "UtsAg3YVW2VZ"
      },
      "execution_count": 15,
      "outputs": []
    }
  ]
}
```

● Adding The Flatten Layer

```
{
  "nbformat": 4,
  "nbformat_minor": 0,
  "metadata": {
    "colab": {
      "provenance": []
    },
    "kernelspec": {
      "name": "python3",
      "display_name": "Python 3"
    },
    "language_info": {
      "name": "python"
    }
  },
  "cells": [
    {
      "cell_type": "markdown",
      "source": [
        "**Model Building**"
      ],
      "metadata": {
        "id": "ecz8Y_zNjQ6y"
      }
    },
    {
```

```
  "cell_type": "markdown",
  "source": [
    "Adding The Flatten Layer"
  ],
  "metadata": {
    "id": "fVsCaDSujRCS"
  }
},
{
  "cell_type": "code",
  "source": [
    "# importing numpy as np\n",
    "import numpy as np"
  ],
  "metadata": {
    "id": "LqkxZCmZjYK7"
  },
  "execution_count": 1,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "# declare flatten np\n",
    "gfg = np.array([[6, 9, 12], [8, 5, 2], [18, 21, 24]])\n",
    "\n",
    "# using array.flatten() method\n",
    "flat_gfg = gfg.flatten(order='A')\n",
    "print(flat_gfg)\n"
  ],
  "metadata": {
    "colab": {
      "base_uri": "https://localhost:8080/"
    },
    "id": "3nxMBOwXjYT-",
    "outputId": "32a7d166-ea1e-4f93-8a35-2a511bf3f3e7"
  },
  "execution_count": 2,
  "outputs": [
    {
      "output_type": "stream",
        "name": "stdout",
        "text": [
          "[ 6  9 12  8  5  2 18 21 24]\n"
        ]
      }
    ]
  },
  {
    "cell_type": "code",
    "source": [
      "from tensorflow.keras.preprocessing.image import ImageDataGenerator"
    ],
    "metadata": {
      "id": "40_c72tbjYQk"
    },
    "execution_count": null,
    "outputs": []
  },
  {
    "cell_type": "code",
    "source": [
      "# Training Datagen\n",
      "train_datagen =
ImageDataGenerator(rescale=1/255,zoom_range=0.2,horizontal_flip=True,vertical_flip=False)\n",
      "# Testing Datagen\n",
      "test_datagen = ImageDataGenerator(rescale=1/255)"
    ],
    "metadata": {
      "id": "Y5ObpZxjjf2s"
    },
    "execution_count": null,
    "outputs": []
  },
  {
    "cell_type": "code",
    "source": [
      "# Training Dataset\n",

"x_train=train_datagen.flow_from_directory('/content/drive/MyDrive/Dataset/training_set',target
_size=(64,64), class_mode='categorical',batch_size=900)\n",
      "# Testing Dataset\n",
"x_test=test_datagen.flow_from_directory('/content/drive/MyDrive/Dataset/test_set',target_size
=(64,64), class_mode='categorical',batch_size=900)"
   ],
   "metadata": {
    "id": "HlHErzo7jfpu"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "markdown",
   "source": [
    "Found 15760 images belonging to 9 classes.\n",
    "Found 2250 images belonging to 9 classes."
   ],
   "metadata": {
    "id": "2By_MkHmkZBt"
   }
  },
  {
   "cell_type": "code",
   "source": [
    "print(\"Len x-train : \", len(x_train))\n",
    "print(\"Len x-test : \", len(x_test))\n"
   ],
   "metadata": {
    "id": "dmzLATW4kbKY"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "markdown",
   "source": [
    "Len x-train :  18\n",
    "Len x-test :  3"
   ],
   "metadata": {
    "id": "iMI_Hmiakgjp"
   }
```

```
    },
    {
      "cell_type": "code",
      "source": [
        "# The Class Indices in Training Dataset\n",
        "x_train.class_indices"
      ],
      "metadata": {
        "id": "iIjcVQTzkhBb"
      },
      "execution_count": null,
      "outputs": []
    },
    {
      "cell_type": "markdown",
      "source": [
        "{'A': 0, 'B': 1, 'C': 2, 'D': 3, 'E': 4, 'F': 5, 'G': 6, 'H': 7, 'I': 8}"
      ],
      "metadata": {
        "id": "QACYBi5pkmjL"
      }
    },
    {
      "cell_type": "markdown",
      "source": [
        "**Model Creation**"
      ],
      "metadata": {
        "id": "XSBxj9mQko30"
      }
    },
    {
      "cell_type": "code",
      "source": [
        "model = Sequential()\n",
        "for i, feat in enumerate(args.conv_f):\n",
        "    if i==0:\n",
        "        model.add(Conv2D(feat, input_shape=x[0].shape, kernel_size=3, padding =
'same',use_bias=False))\n",
        "    else:\n",
        "        model.add(Conv2D(feat, kernel_size=3, padding = 'same',use_bias=False))\n",
    "        model.add(BatchNormalization())\n",
    "        model.add(LeakyReLU(alpha=args.conv_act))\n",
    "        model.add(Conv2D(feat, kernel_size=3, padding = 'same',use_bias=False))\n",
    "        model.add(BatchNormalization())\n",
    "        model.add(LeakyReLU(alpha=args.conv_act))\n",
    "        model.add(Dropout(args.conv_do[i]))"
   ],
   "metadata": {
    "id": "7JUMgR3Zksix"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "model.add(Flatten())\n",
    "\n",
    "#Input code here\n",
    "\n",
    "denseArgs = {'use_bias':False}\n",
    "for i,feat in enumerate(args.dense_f):\n",
    "    model.add(Dense(feat,**denseArgs))\n",
    "    model.add(BatchNormalization())\n",
    "    model.add(LeakyReLU(alpha=args.dense_act))\n",
    "    model.add(Dropout(args.dense_do[i]))\n",
    "model.add(Dense(1))"
   ],
   "metadata": {
    "id": "poHAdXlaktxk"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "# Importing Libraries\n",
    "from tensorflow.keras.models import Sequential\n",
    "from tensorflow.keras.layers import Convolution2D,MaxPooling2D,Flatten,Dense"
   ],
  "metadata": {
    "id": "ggqOIIrzktuK"
  },
  "execution_count": 15,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "# Creating Model\n",
    "model=Sequential()"
  ],
  "metadata": {
    "id": "yEQWTNrqktra"
  },
  "execution_count": 16,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "# Adding Layers\n",
    "model.add(Convolution2D(32,(3,3),activation='relu',input_shape=(64,64,3)))"
  ],
  "metadata": {
    "id": "FcHzBbe-kto9"
  },
  "execution_count": 17,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "model.add(MaxPooling2D(pool_size=(2,2)))"
  ],
  "metadata": {
    "id": "e8KZFLIrktmT"
  },
  "execution_count": 18,
  "outputs": []
},
    {
      "cell_type": "code",
      "source": [
        "model.add(Flatten())"
      ],
      "metadata": {
        "id": "f7w13Ydvktjf"
      },
      "execution_count": 19,
      "outputs": []
    },
    {
      "cell_type": "code",
      "source": [
        "# Adding Dense Layers\n",
        "model.add(Dense(300,activation='relu'))\n",
        "model.add(Dense(150,activation='relu'))\n",
        "model.add(Dense(9,activation='softmax'))"
      ],
      "metadata": {
        "id": "z1tLu8dRk9s4"
      },
      "execution_count": 20,
      "outputs": []
    }
  ]
}
```

● Compiling The Model

```
{

  "nbformat": 4,

  "nbformat_minor": 0,

  "metadata": {

    "colab": {

      "provenance": []

    },
  "kernelspec": {
    "name": "python3",
    "display_name": "Python 3"
  },
  "language_info": {
    "name": "python"
  }
},
"cells": [
  {
    "cell_type": "markdown",
    "source": [
      "**Model Building**"
    ],
    "metadata": {
      "id": "5pqakJ7OoRes"
    }
  },
  {
    "cell_type": "markdown",
    "source": [
      "Compile To The Mode"
    ],
    "metadata": {
      "id": "5AT-SfHoobDV"
    }
```

```
},
{
 "cell_type": "code",
 "source": [
  "from tensorflow.keras.preprocessing.image\n",
  "import ImageDataGenerator"
 ],
 "metadata": {
  "id": "1sNYdJoood-1"
 },
 "execution_count": null,
 "outputs": []
},
{
 "cell_type": "code",
 "source": [
  "model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])"
 ],
 "metadata": {
  "id": "Lzy7Bm2doeSr"
 },
 "execution_count": 38,
 "outputs": []
},
{
 "cell_type": "code",
"source": [
  "# Creating sample sourcecode to multiply two variables\n",
  "# x and y.\n",
  "srcCode = 'x = 10\\ny = 20\\nmul = x * y\\nprint(\"mul =\", mul)'\n",
  " \n",
  "# Converting above source code to an executable\n",
  "execCode = compile(srcCode, 'mulstring', 'exec')\n",
  " \n",
  "# Running the executable code.\n",
  "exec(execCode)"
],
"metadata": {
  "colab": {
    "base_uri": "https://localhost:8080/"
  },
  "id": "EoFDtcbUoeI-",
  "outputId": "d7a332b7-9ec1-4f8b-d405-0dc1391cc916"
},
"execution_count": 39,
"outputs": [
  {
    "output_type": "stream",
    "name": "stdout",
    "text": [
      "mul = 200\n"
    ]
```

```
      }
     ]
    },
    {
     "cell_type": "code",
     "source": [
       "# Training Datagen\n",
       "train_datagen =
ImageDataGenerator(rescale=1/255,zoom_range=0.2,horizontal_flip=True,vertical_flip=False)\
n",
       "# Testing Datagen\n",
       "test_datagen = ImageDataGenerator(rescale=1/255)\n"
     ],
     "metadata": {
       "id": "ultCPqevoeFk"
     },
     "execution_count": 40,
     "outputs": []
    },
    {
     "cell_type": "code",
     "source": [
       "# Training Dataset\n",

"x_train=train_datagen.flow_from_directory(r'/content/drive/MyDrive/Dataset/training_set',tar
get_size=(64,64), class_mode='categorical',batch_size=900)\n",
       "# Testing Dataset\n",

"x_test=test_datagen.flow_from_directory(r'/content/drive/MyDrive/Dataset/test_set',target_s
ize=(64,64), class_mode='categorical',batch_size=900)"
   ],
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/"
    },
    "id": "SuloBjQNo_K2",
    "outputId": "de3ceed9-750a-49b6-d950-35e7fc5f3ec6"
   },
   "execution_count": 41,
   "outputs": [
    {
     "output_type": "stream",
     "name": "stdout",
     "text": [
      "Found 8064 images belonging to 9 classes.\n",
      "Found 0 images belonging to 9 classes.\n"
     ]
    }
   ]
  },
  {
   "cell_type": "code",
   "source": [
    "def compile_model_results(model, root=\"./\"):\n",
    "\n",
    "    listing = glob.glob(root + '/models/' + model + '/*/best_pars.pkl')\n",
    "\n",
    "    dic_list = []\n",
    "    for file in listing:\n",
    "        tmp = hyper_parameters_load(file)\n",
    "        dic_list.append(tmp.to_dictionary())\n",
    "\n",
    "    df = pd.DataFrame(dic_list)\n",
    "    df['diff'] = df.test_F1 - df.forecast_F1\n",
    "    df['pci'] = abs(df.test_F1 - df.forecast_F1)\n",
    "\n",
    "    if not os.path.exists(root + '/figures/' +  model ):\n",
    "        os.makedirs(root + '/figures/' +  model )\n",
    "\n",
    "    df.to_csv(root + '/figures/' +  model + '/results.csv', index=False)\n",
    "\n",
    "    return df"
   ],
   "metadata": {
    "id": "A0eM-o0-o_Hd"
   },
   "execution_count": 42,
   "outputs": []
  },
  {
   "cell_type": "code",
"source": [
 "# Set optimizer loss and metrics\n",
 "    opt = Adam(lr=args.initial_lr, beta_1=0.99, beta_2=0.999, decay=1e-6)\n",
 "    if args.net.find('caps') != -1:\n",
 "        metrics = {'out_seg': dice_hard}\n",
 "    else:\n",
 "        metrics = [dice_hard]\n",
 "\n",
 "    loss, loss_weighting = get_loss(root=args.data_root_dir, split=args.split_num,
net=args.net,\n",
 "                                    recon_wei=args.recon_wei, choice=args.loss)\n",
 "\n",
 "    # If using CPU or single GPU\n",
 "    if args.gpus <= 1:\n",
 "        uncomp_model.compile(optimizer=opt, loss=loss, loss_weights=loss_weighting,
metrics=metrics)\n",
 "        return uncomp_model\n",
 "    # If using multiple GPUs\n",
 "    else:\n",
 "        with tf.device(\"/cpu:0\"):\n",
 "            uncomp_model.compile(optimizer=opt, loss=loss, loss_weights=loss_weighting,
metrics=metrics)\n",
 "            model = multi_gpu_model(uncomp_model, gpus=args.gpus)\n",
 "            model.__setattr__('callback_model', uncomp_model)\n",
 "        model.compile(optimizer=opt, loss=loss, loss_weights=loss_weighting,
metrics=metrics)\n",
 "\n",
 "X = array[:,0:8]\n",
      "Y = array[:,8]\n",
      "test_size = 0.33\n",
      "seed = 7\n",
      "X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y, test_size=test_size,
\n",
      "random_state=seed)"
    ],
    "metadata": {
      "id": "5hx19Vc_o_EL"
    },
    "execution_count": null,
    "outputs": []
  },
  {
    "cell_type": "code",
    "source": [
      "print(\"Len x-train : \", len(x_train))\n",
      "print(\"Len x-test : \", len(x_test))"
    ],
    "metadata": {
      "colab": {
        "base_uri": "https://localhost:8080/"
      },
      "id": "twGmaFhho_As",
      "outputId": "fd31865d-9c09-4fc1-9863-8e75b8d79292"
    },
    "execution_count": 46,
  "outputs": [
    {
      "output_type": "stream",
      "name": "stdout",
      "text": [
        "Len x-train :  9\n",
        "Len x-test :  0\n"
      ]
    }
  ]
},
{
  "cell_type": "code",
  "source": [
    "# The Class Indices in Training Dataset\n",
    "x_train.class_indices"
  ],
  "metadata": {
    "colab": {
      "base_uri": "https://localhost:8080/"
    },
    "id": "WyQdx_CKpY1r",
    "outputId": "3377b294-4afa-4b23-a3f5-cc1c87cd2753"
  },
  "execution_count": 47,
  "outputs": [
    {
     "output_type": "execute_result",
     "data": {
      "text/plain": [
       "{'A': 0, 'B': 1, 'C': 2, 'D': 3, 'E': 4, 'F': 5, 'G': 6, 'H': 7, 'I': 8}"
      ]
     },
     "metadata": {},
     "execution_count": 47
    }
   ]
  },
  {
   "cell_type": "markdown",
   "source": [
    "Model Compilation"
   ],
   "metadata": {
    "id": "0k5N7n0hpb0R"
   }
  },
  {
   "cell_type": "code",
   "source": [
    "# Importing Libraries\n",
    "from tensorflow.keras.models import Sequential\n",
   "from tensorflow.keras.layers import Convolution2D,MaxPooling2D,Flatten,Dense"
  ],
  "metadata": {
   "id": "XjSuRetppd5V"
  },
  "execution_count": 48,
  "outputs": []
 },
 {
  "cell_type": "code",
  "source": [
   "# Creating Model\n",
   "model=Sequential()"
  ],
  "metadata": {
   "id": "9Kl-3JXCpeBT"
  },
  "execution_count": 49,
  "outputs": []
 },
 {
  "cell_type": "code",
  "source": [
   "# Adding Layers\n",
   "model.add(Convolution2D(32,(3,3),activation='relu',input_shape=(64,64,3)))"
  ],
  "metadata": {
    "id": "-1WUzPAHpe_g"
  },
  "execution_count": 50,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "model.add(MaxPooling2D(pool_size=(2,2)))\n",
    "model.add(Flatten())"
  ],
  "metadata": {
    "id": "r9SP7FMhpe8r"
  },
  "execution_count": 51,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "# Adding Dense Layers\n",
    "model.add(Dense(300,activation='relu'))\n",
    "model.add(Dense(150,activation='relu'))\n",
    "model.add(Dense(9,activation='softmax'))"
  ],
  "metadata": {
   "id": "yebPmaVQpe0i"
  },
  "execution_count": 52,
  "outputs": []
 },
 {
  "cell_type": "code",
  "source": [
   "# Compiling the Model\n",
   "model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])"
  ],
  "metadata": {
   "id": "Ev1r6v32pexK"
  },
  "execution_count": 53,
  "outputs": []
 },
 {
  "cell_type": "code",
  "source": [
   "# reading code from a file\n",
   "f = open('main.py','r')\n",
   "temp = f.read()\n",
   "f.close()\n",
   "\n",
    "code = compile(temp, 'main.py', 'exec')\n",
    "exec(code)"
  ],
  "metadata": {
    "id": "R1KOtMzXprIf"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "markdown",
  "source": [
    "Saving the Model"
  ],
  "metadata": {
    "id": "wE2PjpJGp_pL"
  }
},
{
  "cell_type": "code",
  "source": [
    "model.save('asl_model_84_54.h5')"
  ],
  "metadata": {
    "id": "eOoxqc4eqAIV"
  },
      "execution_count": 56,
      "outputs": []
    }
  ]
}
```

● Fit And Saving the Model

```
{
  "nbformat": 4,
  "nbformat_minor": 0,
  "metadata": {
    "colab": {
      "provenance": []
    },
    "kernelspec": {
      "name": "python3",
      "display_name": "Python 3"
    },
    "language_info": {
      "name": "python"
    }
  },
  "cells": [
    {
      "cell_type": "markdown",
      "source": [
        "**Model Building**"
      ],
      "metadata": {
        "id": "VnGsFzcZrSiw"
      }
    },
    {
      "cell_type": "markdown",
    "source": [
     "Fit And Save The Model"
    ],
    "metadata": {
     "id": "3ggczOnbrSmH"
    }
   },
   {
    "cell_type": "code",
    "source": [
     "from tensorflow.keras.preprocessing.image import ImageDataGenerator"
    ],
    "metadata": {
     "id": "Xu_P91SUrZjR"
    },
    "execution_count": 57,
    "outputs": []
   },
   {
    "cell_type": "code",
    "source": [
     "# Training Datagen\n",
     "train_datagen =
ImageDataGenerator(rescale=1/255,zoom_range=0.2,horizontal_flip=True,vertical_flip=False)\n
",
     "# Testing Datagen\n",
     "test_datagen = ImageDataGenerator(rescale=1/255)"
    ],
    "metadata": {
     "id": "oALPDP4KrZ3B"
    },
    "execution_count": 58,
    "outputs": []
   },
   {
    "cell_type": "code",
    "source": [
        "# Training Dataset\n",
    "x_train=train_datagen.flow_from_directory(r'/content/drive/MyDrive/Dataset/training_set',targe
t_size=(64,64), class_mode='categorical',batch_size=900)\n",
        "# Testing Dataset\n",
    "x_test=test_datagen.flow_from_directory(r'/content/drive/MyDrive/Dataset/test_set',target_size
=(64,64), class_mode='categorical',batch_size=900)"
      ],
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "85PQcqpcrZyq",
        "outputId": "7c9e527f-8938-415f-e27f-1b692621ebbb"
      },
      "execution_count": 59,
      "outputs": [
        {
          "output_type": "stream",
          "name": "stdout",
          "text": [
            "Found 9031 images belonging to 9 classes.\n",
            "Found 0 images belonging to 9 classes.\n"
          ]
        }
      ]
    },
    {
      "cell_type": "code",
      "source": [
        "# Save Model Using Pickle\n",
        "import pandas\n",
        "from sklearn import model_selection\n",
        "from sklearn.linear_model import LogisticRegression\n",
        "import pickle"
      ],
   "metadata": {
    "id": "Rcp6w2RsrZuP"
   },
   "execution_count": 60,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "url = \"https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians- \n",
    "diabetes.data.csv\"\n",
    "names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']\n",
    "dataframe = pandas.read_csv(url, names=names)\n",
    "array = dataframe.values \n",
    "X = array[:,0:8]\n",
    "Y = array[:,8]\n",
    "test_size = 0.33\n",
    "seed = 7\n",
    "X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,
test_size=test_size, \n",
    "random_state=seed)\n"
   ],
   "metadata": {
    "id": "_G48TUu2rZpi"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "# Fit the model on training set\n",
    "model = LogisticRegression()\n",
    "model.fit(X_train, Y_train)\n",
    "# save the model to disk\n",
    "filename = 'finalized_model.sav'\n",
    "pickle.dump(model, open(filename, 'wb'))\n",
    "\n",
    "# load the model from disk\n",
    "loaded_model = pickle.load(open(filename, 'rb'))\n",
    "result = loaded_model.score(X_test, Y_test)\n",
    "print(result)"
  ],
  "metadata": {
    "id": "kGVR4fDArpOo"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "print(\"Len x-train : \", len(x_train))\n",
    "print(\"Len x-test : \", len(x_test))"
  ],
  "metadata": {
    "colab": {
      "base_uri": "https://localhost:8080/"
    },
    "id": "rgaHvVUQrpLP",
    "outputId": "5869b889-467f-4389-dc80-a22d496ba4b0"
  },
  "execution_count": 64,
  "outputs": [
    {
      "output_type": "stream",
      "name": "stdout",
      "text": [
        "Len x-train :  11\n",
        "Len x-test :  0\n"
      ]
    }
  ]
},
{
  "cell_type": "code",
  "source": [
    "# The Class Indices in Training Dataset\n",
    "x_train.class_indices"
  ],
  "metadata": {
    "colab": {
      "base_uri": "https://localhost:8080/"
    },
    "id": "WJvns00IrpIh",
    "outputId": "58f6b897-a33a-46c5-9ca4-7a49d8636ee3"
  },
  "execution_count": 65,
  "outputs": [
    {
      "output_type": "execute_result",
      "data": {
        "text/plain": [
          "{'A': 0, 'B': 1, 'C': 2, 'D': 3, 'E': 4, 'F': 5, 'G': 6, 'H': 7, 'I': 8}"
        ]
      },
      "metadata": {},
      "execution_count": 65
    }
  ]
},
{
  "cell_type": "markdown",
  "source": [
    "**Model Creation**"
  ],
  "metadata": {
    "id": "UzG_gbiksCGP"
  }
},
{
  "cell_type": "code",
  "source": [
   "# Importing Libraries\n",
   "from tensorflow.keras.models import Sequential\n",
   "from tensorflow.keras.layers import Convolution2D,MaxPooling2D,Flatten,Dense"
  ],
  "metadata": {
   "id": "X7pqeGddr_Am"
  },
  "execution_count": 66,
  "outputs": []
 },
 {
  "cell_type": "code",
  "source": [
   "# Creating Model\n",
   "model=Sequential()"
  ],
  "metadata": {
   "id": "DGcCgT3LsKN5"
  },
  "execution_count": 67,
  "outputs": []
 },
 {
  "cell_type": "code",
  "source": [
   "# Adding Layers\n",
   "model.add(Convolution2D(32,(3,3),activation='relu',input_shape=(64,64,3)))"
  ],
  "metadata": {
   "id": "IEFNeLUHsKJ4"
  },
  "execution_count": 68,
  "outputs": []
 },
 {
   "cell_type": "code",
   "source": [
    "model.add(MaxPooling2D(pool_size=(2,2)))"
   ],
   "metadata": {
    "id": "g0ZBKJkesKHI"
   },
   "execution_count": 69,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "model.add(Flatten())"
   ],
   "metadata": {
    "id": "yalL-B_YsKD0"
   },
   "execution_count": 70,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "# Adding Dense Layers\n",
    "model.add(Dense(300,activation='relu'))\n",
    "model.add(Dense(150,activation='relu'))\n",
    "model.add(Dense(9,activation='softmax'))"
   ],
   "metadata": {
    "id": "dzPq8m2IsKAw"
   },
   "execution_count": 71,
   "outputs": []
  },
  {
   "cell_type": "code",
    "source": [
     "# Compiling the Model\n",
     "model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])"
    ],
    "metadata": {
     "id": "Aq0O0jOysJ9i"
    },
    "execution_count": 72,
    "outputs": []
   },
   {
    "cell_type": "code",
    "source": [
     "# Fitting the Model Generator\n",
"model.fit_generator(x_train,steps_per_epoch=len(x_train),epochs=10,validation_data=x_test,va
lidation_steps=len(x_test))"
    ],
    "metadata": {
     "id": "W2m8rm79sgHk"
    },
    "execution_count": null,
    "outputs": []
   },
   {
    "cell_type": "markdown",
    "source": [
     "Saving the Model"
    ],
    "metadata": {
     "id": "ukQGVIq5sk7p"
    }
   },
   {
    "cell_type": "code",
    "source": [
     "model.save('asl_model_84_54.h5')"
```

```
      ],
      "metadata": {
        "id": "CYiXIQB5sonE"
      },
      "execution_count": null,
      "outputs": []
    }
  ]
}
```

TESTING THE MODEL

● Importing The Packagesand Loading the Saved Model

```
{
  "nbformat": 4,
  "nbformat_minor": 0,
  "metadata": {
    "colab": {
      "provenance": []
    },
    "kernelspec": {
      "name": "python3",
      "display_name": "Python 3"
    },
    "language_info": {
      "name": "python"
    }
  },
  "cells": [
    {
      "cell_type": "markdown",
      "source": [
        "Import The Required Model Building Libraries"
      ],
      "metadata": {
        "id": "uMO1FSxbtg8g"
      }
    },
    {
      "cell_type": "code",
      "source": [],
      "metadata": {
        "id": "zYFxJgIItxcm"
      },
      "execution_count": null,
      "outputs": []
    },
    {
      "cell_type": "code",
      "source": [
        "#import imagedatagenerator\n",
        "from keras.preprocessing.image import ImageDataGenerator"
      ],
      "metadata": {
        "id": "G9N_GDkmtxZ9"
      },
      "execution_count": 1,
      "outputs": []
    },
    {
      "cell_type": "code",
      "source": [
        "#training datagen\n",

"train_datagen=ImageDataGenerator(rescale=1./255,shear_range=0.2,zoom_range=0.2,horizontal_flip=Tr
ue)\n"
      ],
      "metadata": {
        "id": "RzzEykhEtxW0"
      },
      "execution_count": 2,
      "outputs": []
    },
    {
      "cell_type": "code",
  "source": [
    "#testing datagen\n",
    "test_datagen=ImageDataGenerator(rescale=1./255)"
  ],
  "metadata": {
    "id": "5cgZLBrXtxUG"
  },
  "execution_count": 3,
  "outputs": []
 },
 {
  "cell_type": "markdown",
  "source": [
    "IMPORTING tensorflow"
  ],
  "metadata": {
    "id": "TKTGZEH_t6Le"
  }
 },
 {
  "cell_type": "code",
  "source": [
    "import tensorflow as tf\n",
    "import os"
  ],
  "metadata": {
    "id": "H4XIlRnMtxRo"
  },
  "execution_count": 4,
  "outputs": []
 },
 {
  "cell_type": "markdown",
  "source": [
    "Initialize The Model"
  ],
  "metadata": {
    "id": "lov_IFGct_II"
  }
 },
 {
  "cell_type": "code",
  "source": [
   "#create model\n",
   "from keras.models import Sequential\n",
   "from keras.layers import Dense\n",
   "from keras.layers import Convolution2D\n",
   "from keras.layers import MaxPooling2D\n",
   "from keras.layers import Dropout\n",
   "from keras.layers import Flatten \n",
   "from tensorflow.keras.preprocessing.image import ImageDataGenerator\n"
  ],
  "metadata": {
   "id": "xEUnJlnPuCvk"
  },
  "execution_count": 5,
  "outputs": []
 },
 {
  "cell_type": "code",
  "source": [
   "import numpy as np\n",
   "import matplotlib.pyplot as plt #to view graph in colab itself\n",
   "import IPython.display as display\n",
   "from PIL import Image\n",
   "import pathlib"
  ],
  "metadata": {
   "id": "39CNVe86uDOa"
  },
  "execution_count": 6,
  "outputs": []
 },
 {
  "cell_type": "markdown",
  "source": [
   "Unzipping the dataset"
  ],
  "metadata": {
   "id": "ZTuDzmvXvAvT"
  }
 },
  {
    "cell_type": "code",
    "source": [
      "!unzip '/content/drive/MyDrive/dataset/conversation engine for deaf and dumb.zip'"
    ],
    "metadata": {
      "id": "9cxfnmL6vDol"
    },
    "execution_count": null,
    "outputs": []
  },
  {
    "cell_type": "markdown",
    "source": [
      "Applying ImageDataGenerator to training set"
    ],
    "metadata": {
      "id": "fcBQc5Fcvnxt"
    }
  },
  {
    "cell_type": "code",
    "source": [

"x_train=train_datagen.flow_from_directory('/content/Dataset/training_set',target_size=(64,64),batch_siz
e=200,\n",
      "                                          class_mode='categorical',color_mode=\"grayscale\")"
    ],
    "metadata": {
      "colab": {
        "base_uri": "https://localhost:8080/"
      },
      "id": "7LtOcpd2vua7",
      "outputId": "dbd6f5d7-a8cd-400d-976a-bd625c812117"
    },
    "execution_count": 9,
    "outputs": [
      {
        "output_type": "stream",
        "name": "stdout",
        "text": [
          "Found 15750 images belonging to 9 classes.\n"
         ]
        }
      ]
    },
    {
      "cell_type": "markdown",
      "source": [
        "Applying ImageDataGenerator to test set"
      ],
      "metadata": {
        "id": "ZXzkXt3-vu2O"
      }
    },
    {
      "cell_type": "code",
      "source": [

"x_test=test_datagen.flow_from_directory('/content/Dataset/test_set',target_size=(64,64),batch_size=200,\
n",
        "                                          class_mode='categorical',color_mode=\"grayscale\")\n"
      ],
      "metadata": {
        "colab": {
          "base_uri": "https://localhost:8080/"
        },
        "id": "nGTJcVRav3iD",
        "outputId": "d03f53dd-03a3-4e1d-820b-2c0240e9d2df"
      },
      "execution_count": 10,
      "outputs": [
        {
          "output_type": "stream",
          "name": "stdout",
          "text": [
            "Found 2250 images belonging to 9 classes.\n"
          ]
        }
      ]
    },
    {
```

```
  "cell_type": "code",
  "source": [
   "a=len(x_train)\n",
   "b=len(x_test)"
  ],
  "metadata": {
   "id": "nMFZa22jv319"
  },
  "execution_count": 11,
  "outputs": []
 },
 {
  "cell_type": "markdown",
  "source": [
   "Length of training set"
  ],
  "metadata": {
   "id": "z0FR66aXwAT_"
  }
 },
 {
  "cell_type": "code",
  "source": [
   "print(a)"
  ],
  "metadata": {
   "colab": {
    "base_uri": "https://localhost:8080/"
   },
   "id": "iLCqoZDuwCLh",
   "outputId": "899b849c-9a4d-4622-a2cd-08ff12d21a45"
  },
  "execution_count": 12,
  "outputs": [
   {
    "output_type": "stream",
    "name": "stdout",
    "text": [
     "79\n"
    ]
   }
  ]
},
{
  "cell_type": "markdown",
  "source": [
    "Length of test set"
  ],
  "metadata": {
    "id": "2SnxFFVhwFRY"
  }
},
{
  "cell_type": "code",
  "source": [
    "print(b)"
  ],
  "metadata": {
    "colab": {
      "base_uri": "https://localhost:8080/"
    },
    "id": "bBngo2q9wHy9",
    "outputId": "d57fcfe4-23b1-4263-b45f-1a87a28ece73"
  },
  "execution_count": 13,
  "outputs": [
    {
      "output_type": "stream",
      "name": "stdout",
      "text": [
        "12\n"
      ]
    }
  ]
},
{
  "cell_type": "markdown",
  "source": [
    "Add Layers"
  ],
  "metadata": {
    "id": "ddM4dfzowLN-"
  }
 },
 {
  "cell_type": "code",
  "source": [
   "#create model\n",
   "model=Sequential()"
  ],
  "metadata": {
   "id": "m2P302V2wL2Q"
  },
  "execution_count": 14,
  "outputs": []
 },
 {
  "cell_type": "markdown",
  "source": [
   "Add The Convolution Layer"
  ],
  "metadata": {
   "id": "simRxBBWwMP3"
  }
 },
 {
  "cell_type": "code",
  "source": [
   "model.add(Convolution2D(32,(3,3),input_shape=(64,64,1),activation='relu'))\n"
  ],
  "metadata": {
   "id": "8dU9ev4jwMor"
  },
  "execution_count": 16,
  "outputs": []
 },
 {
  "cell_type": "markdown",
  "source": [
   "Add Pooling Layer"
  ],
  "metadata": {
   "id": "fqT1oNDHwNE5"
  }
 },
 {
  "cell_type": "code",
  "source": [
   "model.add(MaxPooling2D(pool_size=(2,2)))"
  ],
  "metadata": {
   "id": "B1dEdZFCwexj"
  },
  "execution_count": 17,
  "outputs": []
 },
 {
  "cell_type": "markdown",
  "source": [
   "Add The Flatten Layer"
  ],
  "metadata": {
   "id": "yoH75jkVwfLb"
  }
 },
 {
  "cell_type": "code",
  "source": [
   "model.add(Flatten())"
  ],
  "metadata": {
   "id": "_mJ7pZqUwmMj"
  },
  "execution_count": 18,
  "outputs": []
 },
 {
  "cell_type": "markdown",
  "source": [
   "Adding The Dense Layers"
  ],
  "metadata": {
   "id": "9tbK4R1kwmqR"
  }
```

```
},
{
 "cell_type": "code",
 "source": [
  "#1st hidden layer\n",
  "model.add(Dense(units=512,activation='relu'))\n",
  "#2nd hidden layer\n",
  "model.add(Dense(units=261,activation='relu'))"
 ],
 "metadata": {
  "id": "PPmVsewnwsBQ"
 },
 "execution_count": 19,
 "outputs": []
},
{
 "cell_type": "code",
 "source": [
  "#output layer\n",
  "model.add(Dense(units=9,activation='softmax'))"
 ],
 "metadata": {
  "id": "YZFX2pr-wsMD"
 },
 "execution_count": 20,
 "outputs": []
},
{
 "cell_type": "markdown",
 "source": [
  "Compile The Model"
 ],
 "metadata": {
  "id": "W_w4S2vqwwmK"
 }
},
{
 "cell_type": "code",
 "source": [
  "model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])"
 ],
   "metadata": {
    "id": "jRAq8fYLwy_E"
   },
   "execution_count": 21,
   "outputs": []
  },
  {
   "cell_type": "markdown",
   "source": [
    "Fit The Model"
   ],
   "metadata": {
    "id": "5g_oVyz_wzR4"
   }
  },
  {
   "cell_type": "code",
   "source": [

"model.fit_generator(x_train,steps_per_epoch=len(x_train),epochs=10,validation_data=x_test,validation_
steps=len(x_test))"
   ],
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/"
    },
    "id": "7-gMM9kTw5tO",
    "outputId": "645ddbfa-c44e-4553-8cc0-130cd9ea4e15"
   },
   "execution_count": 22,
   "outputs": [
    {
     "output_type": "stream",
     "name": "stderr",
     "text": [
      "/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: UserWarning:
`Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which
supports generators.\n",
      "  \"\"\"Entry point for launching an IPython kernel.\n"
     ]
    },
    {
     "output_type": "stream",
     "name": "stdout",
     "text": [
      "Epoch 1/10\n",
      "79/79 [==============================] - 97s 1s/step - loss: 0.4445 - accuracy: 0.8537
- val_loss: 0.2528 - val_accuracy: 0.9440\n",
      "Epoch 2/10\n",
      "79/79 [==============================] - 92s 1s/step - loss: 0.0645 - accuracy: 0.9827
- val_loss: 0.2258 - val_accuracy: 0.9627\n",
      "Epoch 3/10\n",
      "79/79 [==============================] - 89s 1s/step - loss: 0.0256 - accuracy: 0.9925
- val_loss: 0.2369 - val_accuracy: 0.9707\n",
      "Epoch 4/10\n",
      "79/79 [==============================] - 86s 1s/step - loss: 0.0107 - accuracy: 0.9975
- val_loss: 0.2181 - val_accuracy: 0.9751\n",
      "Epoch 5/10\n",
      "79/79 [==============================] - 89s 1s/step - loss: 0.0102 - accuracy: 0.9971
- val_loss: 0.2424 - val_accuracy: 0.9764\n",
      "Epoch 6/10\n",
      "79/79 [==============================] - 85s 1s/step - loss: 0.0051 - accuracy: 0.9987
- val_loss: 0.2771 - val_accuracy: 0.9760\n",
      "Epoch 7/10\n",
      "79/79 [==============================] - 88s 1s/step - loss: 0.0050 - accuracy: 0.9987
- val_loss: 0.2971 - val_accuracy: 0.9724\n",
      "Epoch 8/10\n",
      "79/79 [==============================] - 87s 1s/step - loss: 0.0070 - accuracy: 0.9977
- val_loss: 0.2617 - val_accuracy: 0.9756\n",
      "Epoch 9/10\n",
      "79/79 [==============================] - 88s 1s/step - loss: 0.0022 - accuracy: 0.9993
- val_loss: 0.2976 - val_accuracy: 0.9764\n",
      "Epoch 10/10\n",
      "79/79 [==============================] - 89s 1s/step - loss: 0.0103 - accuracy: 0.9966
- val_loss: 0.2181 - val_accuracy: 0.9760\n"
     ]
    },
    {
     "output_type": "execute_result",
     "data": {
      "text/plain": [
       "<keras.callbacks.History at 0x7f3ef6336c90>"
      ]
     },
     "metadata": {},
     "execution_count": 22
    }
   ]
  },
  {
   "cell_type": "markdown",
   "source": [
    "Save The Model"
   ],
   "metadata": {
    "id": "wusyjXNKw-s2"
   }
  },
  {
   "cell_type": "code",
   "source": [
    "model.save('aslpng2.h5')"
   ],
   "metadata": {
    "id": "O6AVimS-xBMS"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "markdown",
   "source": [
    "Import The Packages And Load The Saved Model"
   ],
   "metadata": {
    "id": "szB6YrPzxDVq"
   }
  },
  {
   "cell_type": "code",
   "source": [
    "from tensorflow.keras.models import load_model\n",
    "import numpy as np\n",
    "import cv2\n",
    "from tensorflow.keras.preprocessing import image\n"
   ],
   "metadata": {
    "id": "iFBFtGIfxH7D"
   },
   "execution_count": 23,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "#load the model\n",
    "model=load_model('aslpng2.h5')"
   ],
   "metadata": {
    "id": "cy1-uXbjxI0H"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "img=image.load_img('/content/Dataset/test_set/A/10.png',target_size=(400,500))\n",
    "img"
   ],
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/",
     "height": 417
    },
    "id": "FBlWpaXIxIUg",
    "outputId": "176b5713-4faf-41b2-b752-8314014ef804"
   },
   "execution_count": 25,
   "outputs": [
    {
     "output_type": "execute_result",
     "data": {
      "text/plain": [
```

"<PIL.Image.Image image mode=RGB size=500x400 at 0x7F3EEE14A910>"
],
"image/png":
"iVBORw0KGgoAAAANSUhEUgAAAfQAAAGQCAIAAADX0QWRAAAHt0lEQVR4nO3dsWqUW
xeA4YkETBqxEUvFQsfKO7ASRFLZKFbW3oCksBSMd2BtZWWnCN6BVboERQnY2thNI/OXp8g35
5ifmfmSN89TbkKyEHlZMJs9kwkAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAD8lY2xB1i+169fD54/f/58Kb//9u3bxw8PDw+X8ssBluLC2AMAsHziDhAk7gBB4g
4QJO4AQeIOELQ59gDLd+nSpZX+/oODg7//4Y2N4GVT4PSzuQMEiTtAkLgDBIk7QJC4AwSdo7sc8/l
8/X/08uXLg+e/f/9e7yDA+WJzBwgSd4AgcQcIEneAIHEHCHJbZhzenAFWyuYOECTuAEHiDhAk7gB
B4g4QFPwmJs6E6XR6op8/PDxc0SSQZHMHCBJ3gCBxBwgSd4AgcQcIEneAoHP0fNWpejhsNpm
9vbw+eP378+Pjhu3fvljLMmzdvBs+fPXv297b2b6/Ozs7O4PnHz9+HDy/cGF4m/nz58/SZ
jrm5s2bg+ffvn1b3R8FmztAkLgDBIk7QJC4AwSJO0CQ2zIwjgcPHyef/369fjhq1evBn/40aN
jq/9evXiX4Pp5/NHSBI3AGCxB0gSNwBgsQdIMhtGWChjY1zmztAkLgDBIk7QJC4AwSJO0CQuAMEiTtAkLgDBIk7QJC4AwSJO0DQ
5tgDLN/du3fHHGEiptPp8cPDw8P1T8Jj2dwBgsQdIEjcAYLEHSBI3AGCxB0gKHgV8smTJ2OPAOPABEH
BwfHD70mdibY3AGCxB0gSNwBgsQdIEjcAYLEHSBI3AGCxB0gSNwBgsQdIEjcAYKCb8sAKzWf
wfPvTlzqtjcAYLEHSBI3AGCxB0gSNwBgoKfbi/6KB9Yqe3t7cHzg4OD9fxRm1WWy25kmY2NwBksQdIEjcAYLEHS
BI3AGCxB0gKPhw2IcPHwbPd3Z21jwJwFhs7gBB4g4QJO4AQeIOECTuAEHB2zI/f/4cewSAkdncAYL
EHSBI3AGCxB0gSNwBgoK3ZYBR7O/vD55Pp9P1DsJkYnMHSBI3AGCxBwgSd4AgcQcIclsGWI5bt2
4Nnh8dHQ2eX79+fXXDYHMHCBJ3gCBxBwgSd4AgcQcIEneAIFchgdW6evXqmNMXJbZhzenAFWyuYOECTuAEHiDhAk7gBxBwg
Sd4AgcQcIEneAIHEHCBJ3gCBxBwgSd4AgcQcIEneAIHEHCBJ3gCBxBwgSd4AgcQcIEneAIHEHCB
J3gCBxBwgSd4AgcQcIEneAIHEHCBJ3gCBxBwgSd4AgcQcIEneAIHEHCBJ3gCBxBwgSd4AgcQcIE
neAIHEHCNocewAgbmtra+wRziObO0CQuAMEiTtAkLgDBIk7QJC4AwSJO0CQuAMEiTtAkLgDBIk
7QJC4AwSJO0CQuAMEiTtAkLgDBIk7QJC4AwRtjD3A8t2/f3/w/NOnT2ueBPgXGxvB/pweNneAIHE
HCBJ3gCBxBwgSd4Cgc/Rp9b179wbPP3/+vOZJgMlk8vbt28Hzp0+frnmSJJs7QJC4AwSJO0CQuAMEi
TtAkLgDBJ2jq5CL7O/vD57Fu/vD57fuXNnvYMAk4kHxZbE5g4QJO4AQeIOECTuAEHiDhDkU+mFDg4OBs+
n0+maJ4FzxW2ZpbC5AwSJO0CQuAMEiTtAkLgDBPlU+sSOjo6Oo6OH167dm3tm8ZzlbMStncAYLEHS
BI3AGCxB0gSNwBgnxavRzz+XzsEeCMVtmpWzuAEHiDhAk7gBB4g4QJO4AQeIOEOQq0mrNZrP
B84sXLq55EkiyuQMEiTtAkLgDBIk7QJC4AwSJO0CQuAMEiTtA0ObYA8ObtbW0NnntojPJ0ZsXIVcKZs7QJC4AwSJO0CQuAMEiTtA0ObYA8ObtbW0NnntojPJ0ZsXIVcKZs7QJC4AwSJO0CQuAMEiTtA0ObYA8ObtbW0NnntojPJ0ZsXIVcKZs7QJC4AwSJO0CQuAMEiTtA
B4g4QJO4AQeIOEOS2zDgWvarhFg2wFDZ3gCBxBwgSd4AgcQcIEneAIN+Ecja4RcPZteh/74ULlssV8
o8LECTuAEHiDhAk7gBB4g4QJO4AQa5Cng03btw4fvj169fj6tTwInteiZPFbK5g4QJO4AQeIOECTuAEH
iDhDka/bOhh8/fhw/9F19nCpfvnwZewTewT+YXMHCBJ3gCBxBwgSd4AgcQcIclsmyC0aRvHy5cuxR+AfN
neAIHEHCBJ3gCBxBwgSd4Ag35CCWzQsx+b/29uD5DZb8yRMbO4AQeIOECTuAEHiDhAk7gBB4g4
Q5CokC7kiyYkerGOUdjcAYLEHSBI3AGCxB0gSNwBgnzNHgvt7e0Np9zd3V3V3/JJwqL168GHsE/pvN
HSBI3AGCxB0gSNwBgsQdIMhbEJyMB2fwhsyZYHMCBJ3gCBxBwgSd4AgcQcIEneAIHEHCBJ3g
CBxBwgSd4AgcQcIEneAIHEHCBJ3gCBxBwgSd4AgcQcIEneAoM2xB+CM2dvbGzzf3d1d8yTv7C5
AwSJO0CQuAMEiTtAkLgDBIk7QJC4AwSJO0CQuAMEiTtAkLgDBIk7QJC4AwSJO0CQuAMEiTtAk
LgDBIk7QNDG2AMQMZ/Pxx6BNdY0I0zwOYOECTuAEHiDhAk7gBB4g4QJO4AQeIOECTuAEHi
DhAk7gBB4g4Q9D8C0qfYvT6GWAAAAABJRU5ErkJggg==\n"
},

```
        "metadata": {},
        "execution_count": 25
      }
    ]
  }
 ]
}
```

➤ The output [6] in the above image represents the index value in the array[‘A’,’B’,’C’,’D’,’E’,’F’,’G’,’H’,’I’].
➤ Thus, the predicted alphabetices

HTML CODING

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1">
<style>
body {font-family: Arial, Helvetica, sans-serif;}

/* Full-width input fields */
input[type=text], input[type=password] {
  width: 100%;
  padding: 12px 20px;
  margin: 8px 0;
  display: inline-block;
  border: 1px solid #ccc;
  box-sizing: border-box;
}

/* Set a style for all buttons */
button {
  background-color: #04AA6D;
  color: white;
  padding: 14px 20px;
  margin: 8px 0;
  border: none;
  cursor: pointer;
  width: 100%;
}

button:hover {
  opacity: 0.8;
}

/* Extra styles for the cancel button */
.cancelbtn {
  width: auto;
  padding: 10px 18px;
  background-color: #f44336;
}

/* Center the image and position the close button */
.imgcontainer {
  text-align: center;
  margin: 24px 0 12px 0;
  position: relative;
}

img.avatar {
  width: 40%;
  border-radius: 50%;
}

.container {
  padding: 16px;
}

span.psw {
  float: right;
  padding-top: 16px;
}
```

```
/* The Modal (background) */
.modal {
  display: none; /* Hidden by default */
  position: fixed; /* Stay in place */
  z-index: 1; /* Sit on top */
  left: 0;
  top: 0;
  width: 100%; /* Full width */
  height: 100%; /* Full height */
  overflow: auto; /* Enable scroll if needed */
  background-color: rgb(0,0,0); /* Fallback color */
  background-color: rgba(0,0,0,0.4); /* Black w/ opacity */
  padding-top: 60px;
}

/* Modal Content/Box */
.modal-content {
  background-color: #fefefe;
  margin: 5% auto 15% auto; /* 5% from the top, 15% from the bottom and centered */
  border: 1px solid #888;
  width: 80%; /* Could be more or less, depending on screen size */
}

/* The Close Button (x) */
.close {
  position: absolute;
  right: 25px;
  top: 0;
  color: #000;
  font-size: 35px;
  font-weight: bold;
}

.close:hover,
.close:focus {
  color: red;
  cursor: pointer;
}

/* Add Zoom Animation */
.animate {
  -webkit-animation: animatezoom 0.6s;
  animation: animatezoom 0.6s
}

@-webkit-keyframes animatezoom {
  from {-webkit-transform: scale(0)}
  to {-webkit-transform: scale(1)}
}

@keyframes animatezoom {
  from {transform: scale(0)}
  to {transform: scale(1)}
}

/* Change styles for span and cancel button on extra small screens */
@media screen and (max-width: 300px) {
  span.psw {
     display: block;
     float: none;
  }
  .cancelbtn {
     width: 100%;
  }
}
</style>
</head>
<body>

<h2>REAL TIME COMMUNICATION SYSTEM POWERED BY AI FOR SPECIALLY ABLED</h2>

<button onclick="document.getElementById('id01').style.display='block'"
style="width:auto;">Login</button>

<div id="id01" class="modal">

  <form class="modal-content animate" action="/action_page.php" method="post">
    <div class="imgcontainer">
      <span onclick="document.getElementById('id01').style.display='none'" class="close"
title="Close Modal">&times;</span>
      <img
src="https://static.vecteezy.com/system/resources/thumbnails/007/407/996/small/user-icon-
```

```
person-icon-client-symbol-login-head-sign-icon-design-vector.jpg">
  </div>

  <div class="container">
    <label for="uname"><b>Username</b></label>
    <input type="text" placeholder="Enter Username" name="uname" required>

    <label for="psw"><b>Password</b></label>
    <input type="password" placeholder="Enter Password" name="psw" required>

    <button type="submit">Login</button>
    <label>
      <input type="checkbox" checked="checked" name="remember"> Remember me
    </label>
  </div>

  <div class="container" style="background-color:#f1f1f1">
    <button type="button" onclick="document.getElementById('id01').style.display='none'"
class="cancelbtn">Cancel</button>
    <span class="psw">Forgot <a href="#">password?</a></span>
  </div>
 </form>
</div>
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport"
      content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0,
minimum-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css">
  <link rel="stylesheet" href="style.css">
  <title>Document</title>
</head>
<body>
<div class="display-cover">
  <video autoplay></video>
  <canvas class="d-none"></canvas>
```

```
    <div class="video-options">
        <select name="" id="" class="custom-select">
            <option value="">Select camera</option>
        </select>
    </div>

    <img class="screenshot-image d-none" alt="">

    <div class="controls">
        <button class="btn btn-danger play" title="Play"><i data-feather="play-circle"></i></button>
        <button class="btn btn-info pause d-none" title="Pause"><i data-
feather="pause"></i></button>
        <button class="btn btn-outline-success screenshot d-none" title="ScreenShot"><i data-
feather="image"></i></button>
    </div>
</div>

<script src="https://unpkg.com/feather-icons"></script>
<script src="script.js"></script>
</body>
<html><head>
</head><body>
    <video src="" ></video>
    <br />
<button id='flipCamera'>Flip</button>
</body>
<script>
  var front = false;
var video = document.querySelector('video');
  document.getElementById('flipCamera').onclick = function() { front = !front; };
  var constraints = { video: { facingMode: (front? "user" : "environment"), width: 640, height: 480  }
};
  navigator.mediaDevices.getUserMedia(constraints)
  .then(function(mediaStream) {
    video.srcObject = mediaStream;
    video.onloadedmetadata = function(e) {
    video.play();
};
})
.catch(function(err) { console.log(err.name + ": " + err.message); })
</script></html>
```

```
</html>
<style>
.screenshot-image {
   width: 150px;
   height: 90px;
   border-radius: 4px;
   border: 2px solid whitesmoke;
   box-shadow: 0 1px 2px 0 rgba(0, 0, 0, 0.1);
   position: absolute;
   bottom: 5px;
   left: 10px;
   background: white;
}

.display-cover {
   display: flex;
   justify-content: center;
   align-items: center;
   width: 70%;
   margin: 5% auto;
   position: relative;
}

video {
   width: 100%;
   background: rgba(0, 0, 0, 0.2);
}

.video-options {
   position: absolute;
   left: 20px;
   top: 30px;
}

.controls {
   position: absolute;
   right: 20px;
   top: 20px;
   display: flex;
}
```

```
.controls > button {
   width: 45px;
   height: 45px;
   text-align: center;
   border-radius: 100%;
   margin: 0 6px;
   background: transparent;
}

.controls > button:hover svg {
   color: white !important;
}

@media (min-width: 300px) and (max-width: 400px) {
   .controls {
      flex-direction: column;
   }

   .controls button {
      margin: 5px 0 !important;
   }
}

.controls > button > svg {
   height: 20px;
   width: 18px;
   text-align: center;
   margin: 0 auto;
   padding: 0;
}

.controls button:nth-child(1) {
   border: 2px solid #D2002E;
}

.controls button:nth-child(1) svg {
   color: #D2002E;
}

.controls button:nth-child(2) {
   border: 2px solid #008496;
}

.controls button:nth-child(2) svg {
   color: #008496;
}

.controls button:nth-child(3) {
   border: 2px solid #00B541;
}

.controls button:nth-child(3) svg {
   color: #00B541;
}

.controls > button {
   width: 45px;
   height: 45px;
   text-align: center;
   border-radius: 100%;
   margin: 0 6px;
   background: transparent;
}

.controls > button:hover svg {
   color: white;
}
</style>

<script>
// Get the modal
var modal = document.getElementById('id01');

// When the user clicks anywhere outside of the modal, close it
window.onclick = function(event) {
   if (event.target == modal) {
      modal.style.display = "none";
   }
}
feather.replace();

const controls = document.querySelector('.controls');
const cameraOptions = document.querySelector('.video-options>select');
const video = document.querySelector('video');
const canvas = document.querySelector('canvas');
const screenshotImage = document.querySelector('img');
const buttons = [...controls.querySelectorAll('button')];
let streamStarted = false;

const [play, pause, screenshot] = buttons;

const constraints = {
  video: {
    width: {
      min: 1280,
      ideal: 1920,
      max: 2560,
    },
    height: {
      min: 720,
      ideal: 1080,
      max: 1440
    },
  }
};
</script>
<script>
const getCameraSelection = async () => {
  const devices = await navigator.mediaDevices.enumerateDevices();
  const videoDevices = devices.filter(device => device.kind === 'videoinput');
  const options = videoDevices.map(videoDevice => {
    return `<option value="${videoDevice.deviceId}">${videoDevice.label}</option>`;
  });
  cameraOptions.innerHTML = options.join('');
};

</script>
<script>

play.onclick = () => {
  if (streamStarted) {
    video.play();
    play.classList.add('d-none');
    pause.classList.remove('d-none');
    return;
  }
  if ('mediaDevices' in navigator && navigator.mediaDevices.getUserMedia) {
    const updatedConstraints = {
      ...constraints,
      deviceId: {
        exact: cameraOptions.value
      }
    };
    startStream(updatedConstraints);
  }
};

const startStream = async (constraints) => {
  const stream = await navigator.mediaDevices.getUserMedia(constraints);
  handleStream(stream);
};

const handleStream = (stream) => {
  video.srcObject = stream;
  play.classList.add('d-none');
  pause.classList.remove('d-none');
  screenshot.classList.remove('d-none');
  streamStarted = true;
};

getCameraSelection();

cameraOptions.onchange = () => {
  const updatedConstraints = {
    ...constraints,
    deviceId: {
      exact: cameraOptions.value
    }
  };
  startStream(updatedConstraints);
};

const pauseStream = () => {
  video.pause();
  play.classList.remove('d-none');
  pause.classList.add('d-none');
};

const doScreenshot = () => {
  canvas.width = video.videoWidth;
  canvas.height = video.videoHeight;
  canvas.getContext('2d').drawImage(video, 0, 0);
  screenshotImage.src = canvas.toDataURL('image/webp');
  screenshotImage.classList.remove('d-none');
};

pause.onclick = pauseStream;
screenshot.onclick = doScreenshot;
</script>

</body>
</html>
```

CHAPTER 8

TESTING

8.1Test Cases

After finishing the development of any computer based system the next complicated time consuming process is system testing.During the time of the testing only development company can know that,how far the user requirements have been met out,and so on.

Software testing is an important element of the software quality assurance and represents the ultimate review of specification, design and coding. The increasing feasibility of software as a system and the cost associated with the software failures are motivated forces for well planned through testing.

These are several rules that can save as testing objectives they are:

1. Testing is a process of executing program with the intent of finding an error.
2. A good test case is one that has a high probability of finding an undiscoverederror.

TEST CASES

Sl. No	Test Case Name	Test Procedure	Pre-Condition	Expected Result	Passed/ failed
1	Data Input	Enter no details and click submit button	Enter no details input	Alert “Select Dataset, Enter Latitude, Longitude”	Passed
2	Data Input	Select datasetand click submit button	Select datasetand click submit button	Alert “Select Dataset, Enter Latitude, Longitude”	Passed
3	Data Input	Select dataset,enter latitude and click submit button	Select dataset,enter latitude and click submit button	Alert “Select Dataset, Enter Latitude, Longitude”	Passed

8.2 USER ACCEPTANCE TESTING:

User Acceptance Testing is the key factor for the success of any system. The system under consideration is tested for user acceptance by constantly keeping in touch with prospective system users at the time of developing and making changes whenever required.

CHAPTER 9
RESULTS

9.1 Performance Metrics

In the implementation phase, developers change several tasks that they were planned to do. They notice that they can build the system without preparing any training and testing images as they were plan. The code is depending on skin color and contour to find the right sign. Moreover, developers narrow the tasks to only one task which is browse websites only. Moreover, the result was precise and accurate aligned with the methodology and testing that was used. This signifies that developing modern technology assists disabled individuals specifically deaf-dumb on interacting among people.

The measurement variables along with the supporting evidence from the methodology concluded that the measures taken to evaluate this study were supported all throughout. Meanwhile, the efficiency and effectiveness of the system provide the utmost benefit of disabled individuals by offering convenience and being able to make their lives easier and better for there are no required training or specificities for them to use the system. Thus, as a result, D-talk allows everyone to determine the hand gestures that are being projected and be able to come up with interpretations on enabled individuals. Hence, communications between deaf-dumb and enabled individuals are way easier and lacks misunderstandings are being prevented this time. This application can catch finger shapes by using the code for Extract skin color and draw lines around the hand. As a result, the system will recognize any element in the frame. The application main screen is shown in figure 9. Thus, users must be careful about what is inside the frame to avoid any other unwanted requests. This system will recognize any element in the box, and the brightness does not matter. D-talk is a dynamic system that includes three gestures in total to browse websites. All that users need to implement this system is WiFi connection and webcam to capture user gestures. The following figures 10 and 11 show the hand gestures that are used for orders that the system can recognize to browse websites. It could be used as a guide for users.

10.ADVANTAGES &DISADVANTAGES

11.CONCLUSION

The proposed communication system between Deaf and Dumb people and ordinary people are aiming for it when bridging the communication gap between two societies. Several work is done earlier in this area, but this paper adds in complete two - sided communication in an efficient manner because the system is implemented as one Handy mobile application. So, it really serves its needs in all aspects. The above strategies prove to be efficient In terms of time and accuracy. Further improvements can be done in the implementation of the communicator with other sign language such as American Sign Language, Accent recognition for different accents throughout Globe, recognition of emotions in sign language and language Translation.

12.FUTURE SCOPE:

Proposed systems scope is related with education of dumb peoples. Dumb people faces many problems when normal person could not understand their language. They were facing communication gap with normal peoples. For communication between deaf person and a second person, a mediator is required to translate sign language of deaf person. But a mediator is required to know the sign language used by deaf person. But this is not always possible since there are multiple sign languages for multiple languages.