Sign Language Detection

MINOR PROJECT
SOFTWARE DESIGN DOCUMENTATION
(VI Semester)

BACHELOR OF TECHNOLOGY

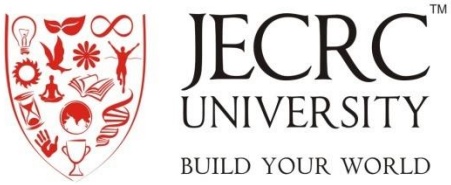
in

COMPUTER SCIENCE AND ENGINEERING

SUBMITTED BY

HIMANSHI CHAINANI (20BCZN015) – Section G
ALOK SHARMA (20BCZN031) – Section G

Project Guide: Mr. Tushar Vyas (Assistant Professor, CSE)



JECRC UNIVERSITY, JAIPUR

FEBRUARY 2023 to JUNE 2023

1. Software Design

1.1 System Architecture

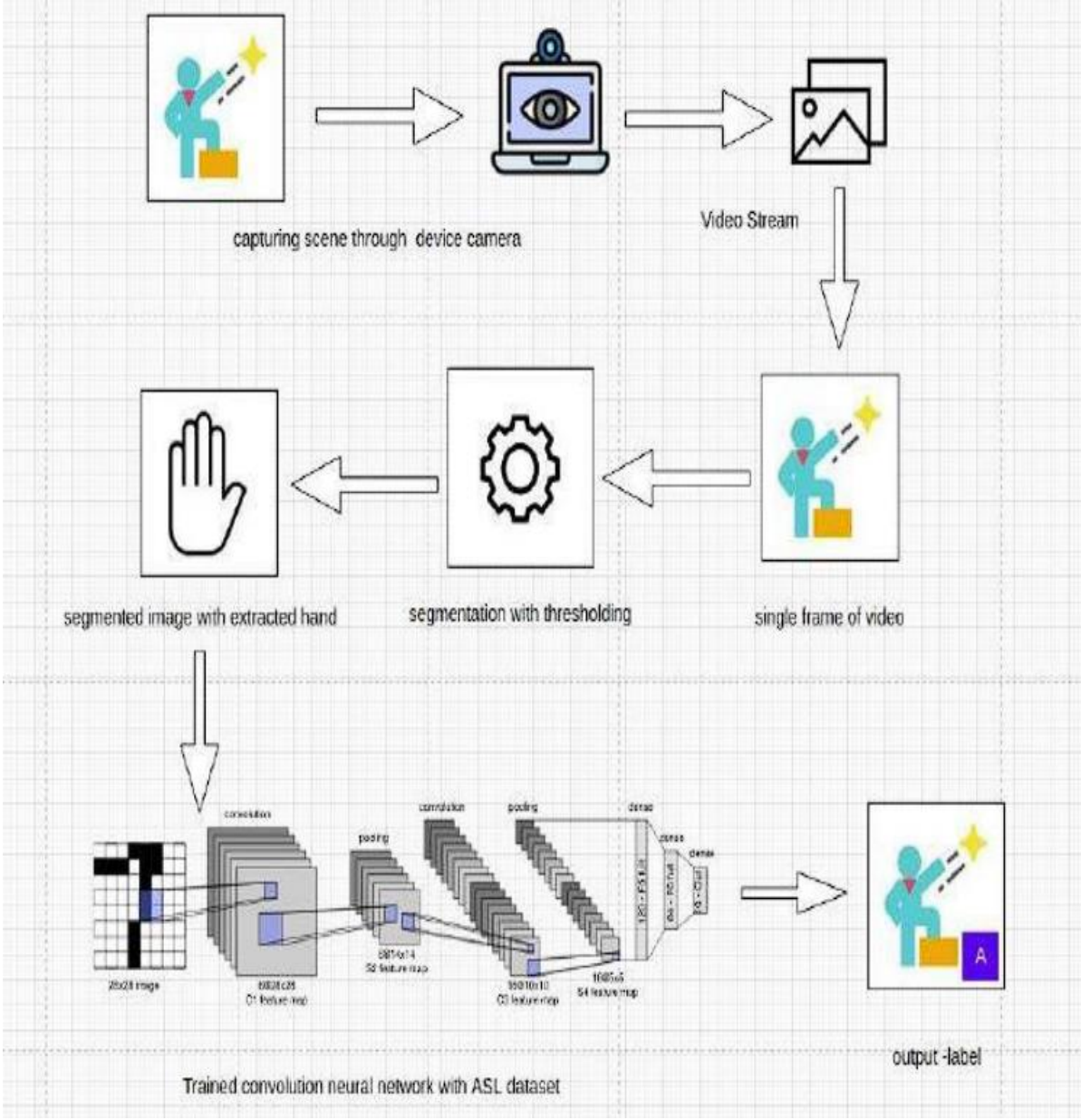


Fig 1.1 Architecture of Sign Language Detection system.

1.2 Architecture Diagram

In the sign language recognition model, the architecture provides a blueprint and ideal techniques to follow so as to developed a well-structured application as per our requirement. This architecture mainly involves three phases:

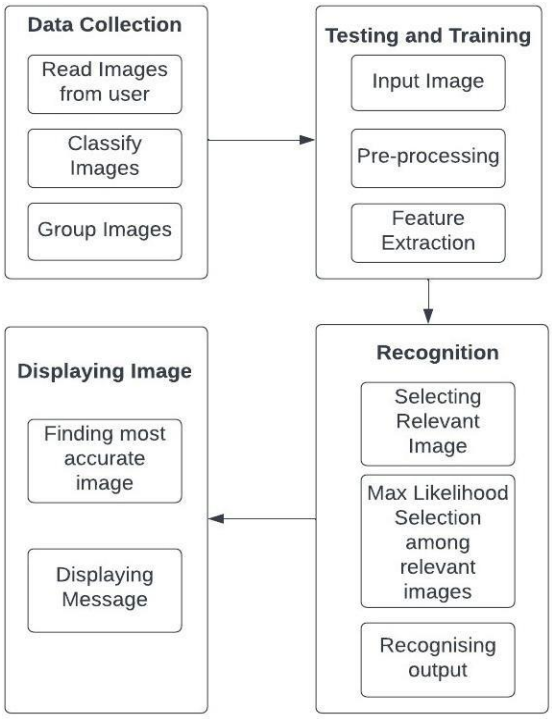


Fig. 1.2 Architecture Model

Phase 1- Data Collection Phase: A model is being built and a sequence of images is being fed to the model. This phase isuseful to further train the model based on the type of symbol.

Phase 2- Training and Testing Phase: This phase involves a set of inputs to the model and a particular output is being expected. Based on the outputs, the accuracy of the model is being identified.

Phase 3- Recognition of output: In this phase, an image is being given as input to the model. Based on the images beingtrained to the model, it matches the image with a particular output and generates a message. The sign being identified in the above phase has a particular meaning and

a message is being assigned in the training phase. The meaning is displayed to the user.

1.3 Flowcharts:

Each one of the phases in the architecture are further explained and sketched graphically in the form of flowcharts. These flowcharts further explain how each of the phases described in the architecture diagram functions.

This helps us in deeply understanding its core features, usability and its relation with the other respective phases present.

1.3.1 Data Collection Phase:

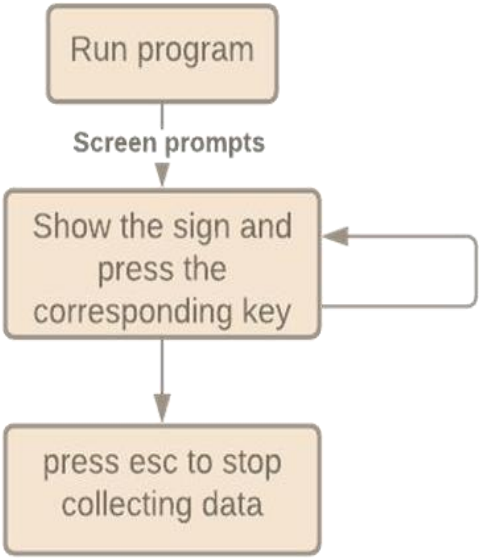


Fig. 1.3 Flow of Data Collection Phase

1.3.2 Training and Testing Phase:

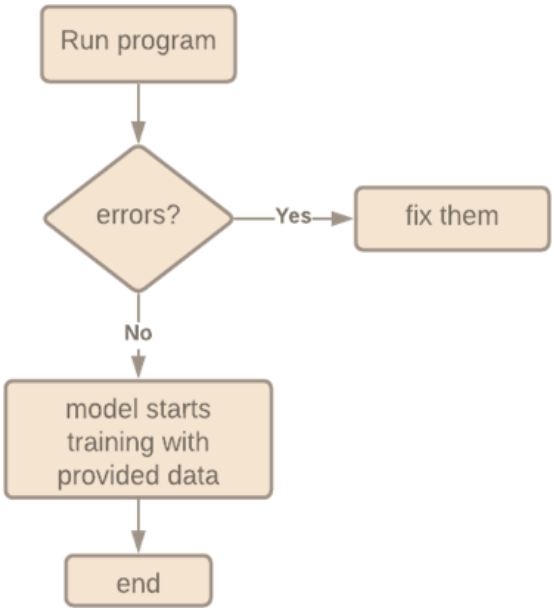


Fig. 1.4 Flow of Training and Testing Phase

1.3.3 Recognition Phase:

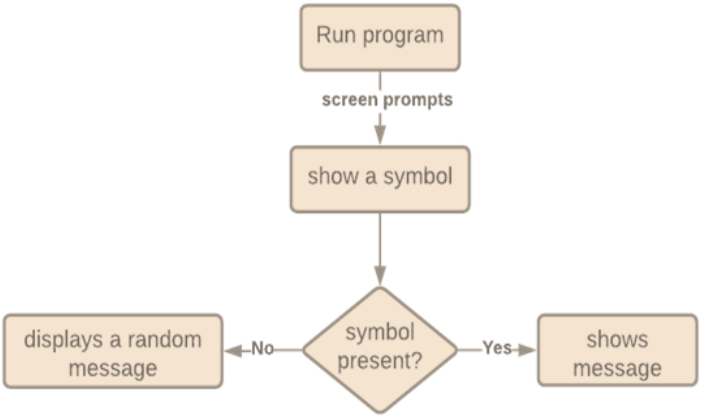


Fig. 1.5 Flow of Recognition Phase

1.4 Building the Convolution Neural Network

The most important part of the project is to implement the Keras CNN model that can be trained to understand the gestures and predict the gesture presented. Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow. For the purpose of this

project, the version of Keras being used is version 2.3.1. The model used here is the sequential model. Building a sequential model in Keras allows to build a model layer by layer. The Keras add() function can be used to add the layers to the model. The first layer in the model is the Conv2D layer. A Keras Conv2D is a 2D Convolution Layer, this layer creates a convolution kernel that is wind with layers input which helps produce a tensor of outputs. The kernel in case of image processing s a convolution matrix or masks which can be used for blurring, sharpening, embossing, edge detection and more by doing a convolution between a kernel and an image. The first layer contains nodes with a kernel size of 2. The activation function used here is ReLU(Rectified Linear Activation), the choice of selecting ReLU here is that the method has proved to work well in neural networks. The first layer also has a parameter called input shape, this defines the shape of the input image. As the dimensions of the image are specified as 50*50, the value passed to the input shape is 50,50,1 where 1 defines that the image is greyscale.

The next layer that is added is the MaxPooling2D layer. This layer is then used to reduce the spatial dimensions of the output volume. A series of Conv2D and MaxPooling2D layers are added to the model.

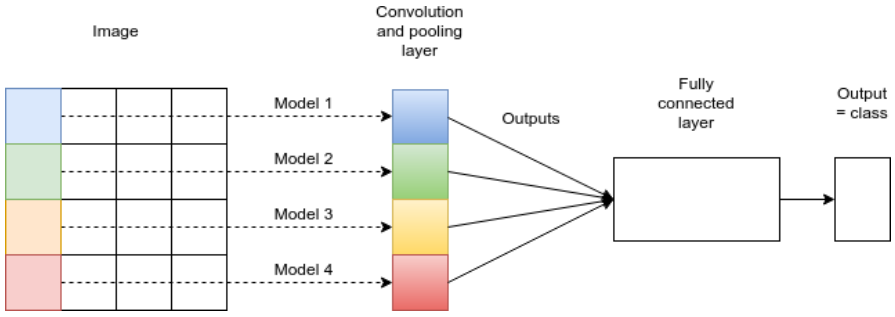


Fig. 1.5 Keras Model

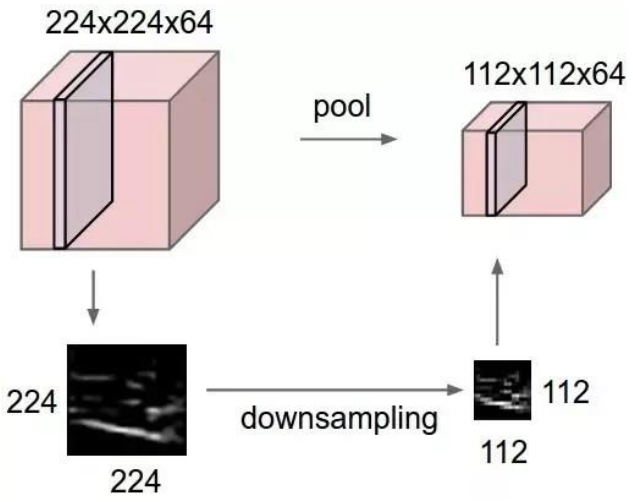


Fig. 1.6 Max Pooling layer

Next a Flatten layer is added that converts the 3 dimensional input layer into a single dimensional output so that the output from the Flatten layer can be used by the Dense Layer.

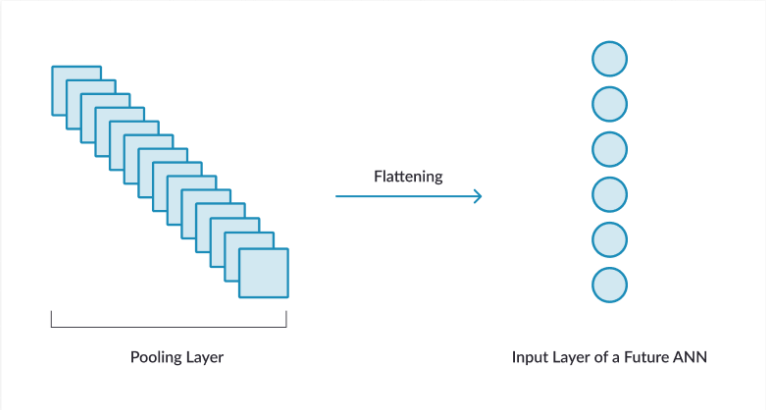


Fig. 1.7 Flatten Layer

Then finally a series of dense layers are added each with a specific number of neurons (the number of neurons must be decided based on the loss / accuracy curves). In order to tackle the over fitting, the Dropout regularization technique is used by adding a dropout layer. Dropout consists of randomly setting a fraction of input units to 0 at each update during training time, which helps prevent over fitting.

Once the model is defined, all the training dataset which comprises of train

images andtrain labels is sent to the Keras model. The model is then fit using these train data andlabels and the evaluation score for the model is generated using the accuracy parameter.

Algorithm 3: Keras CNN Model

```
Train Images : = Loading the train images data;
Train Labels : = Loading the train labels data;
Validation Images : = Loading the validation images data;
Validation Labels : = Loading the validation labels data;
Creating the Keras CNN Model;
Input Layer : = Adding the Convolution Layer with Input shape;
Maxpooling Layer : = Adding a sequence of Maxpooling Layers;
Flatten Layer : = Performing the Flattening of the Layer to 1-d;
Dense Layer : = Adding a sequence of Dense layers with different number of neurons;
Regularization := Adding the Dropout Regularization to handle over-fitting;
return 'Trained Model';
```

Fig. 1.8 Keras CNN Model

2. Design

2.1 Dataflow Diagram

The DFD is also known as bubble chart. It is a simple graphical formalism that can be used to represent a system in terms of the input data to the system, various processing carried out on these data, and the output data is generated by the system. It maps out the flow of information for any process or system, how data is processed in terms of inputs and outputs. It uses defined symbols like rectangles, circles and arrows to show data inputs, outputs, storage points and the routes between each destination. They can be used to analyse an existing system or model of a new one. A DFD can often visually "say" things that would be hard to explain in words and they work for both technical and non- technical.

There are four components in DFD:

1. External Entity

2. Process

3. Data Flow

4. Data Store

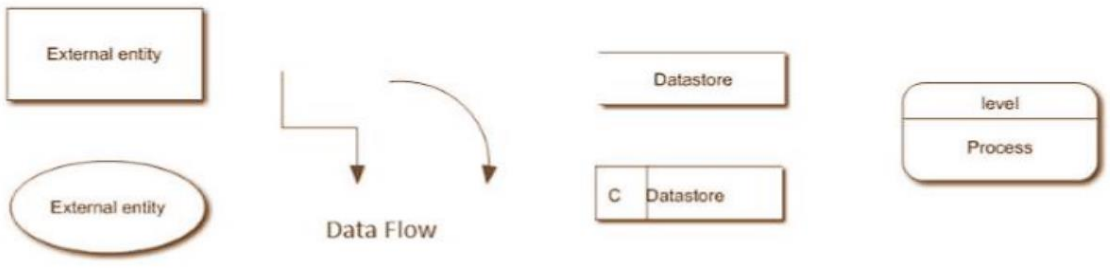


Fig. 2.1 Data Flow Diagram Symbols

2.1.1 External Entity:

It is an outside system that sends or receives data, communicating with the system. They are the sources and destinations of information entering and leaving the system. They might be an outside organization or person, a computer system or a business system. They are known as terminators,

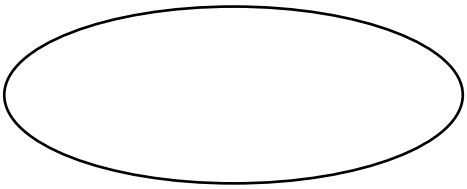
sources and sinks or actors. They are typically drawn on the edges of the diagram. These are sources and destinations of the system's input and output.

Representation:

2.1.2 Process:

It is just like a function that changes the data, producing an output. It might perform computations for sort data based on logic or direct the dataflow based on business rules.

Representation:



2.1.3 Data Flow:

A dataflow represents a package of information flowing between two objects in the data-flow diagram, Data flows are used to model the flow of information into the system, out of the system and between the elements within the system.

Representation:

2.1.4 Data Store:

These are the files or repositories that hold information for later use, such as a database table or a membership form. Each data store receives a simple label.

Representation:

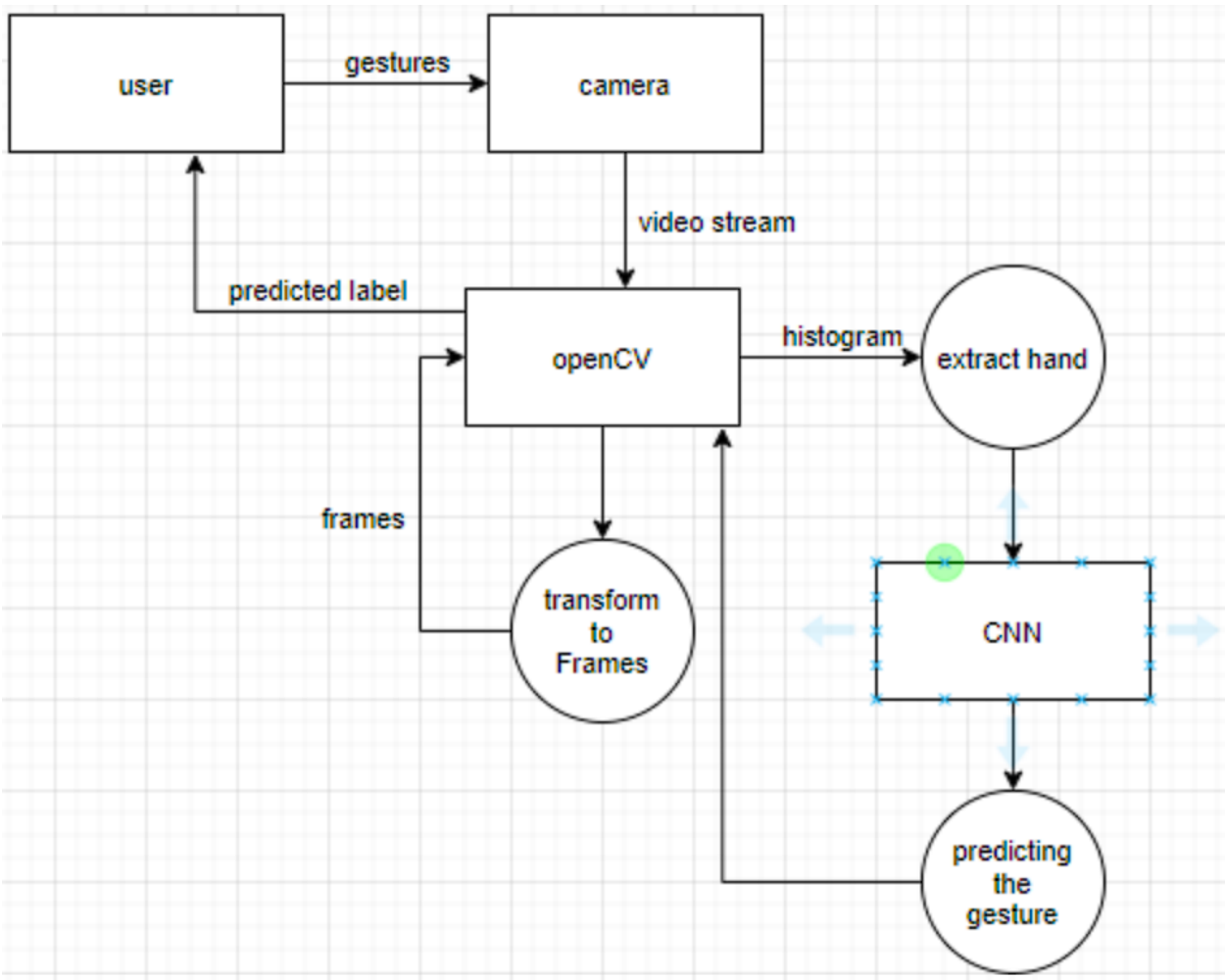


Fig. 2.2 Dataflow Diagram for Sign Language Detection.

Level 0: Context Diagram

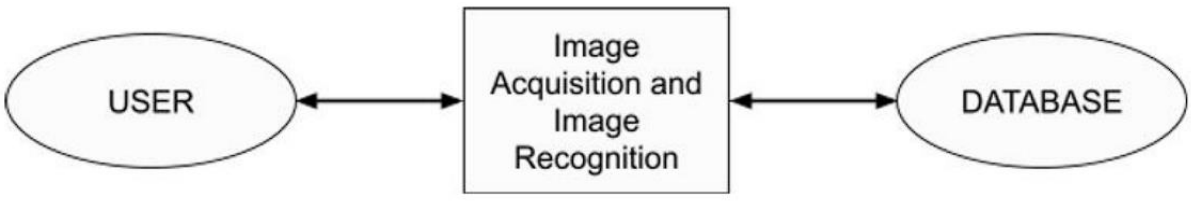


Fig. 2.3 Level Zero DFD

Level 1: Data Flow Diagram

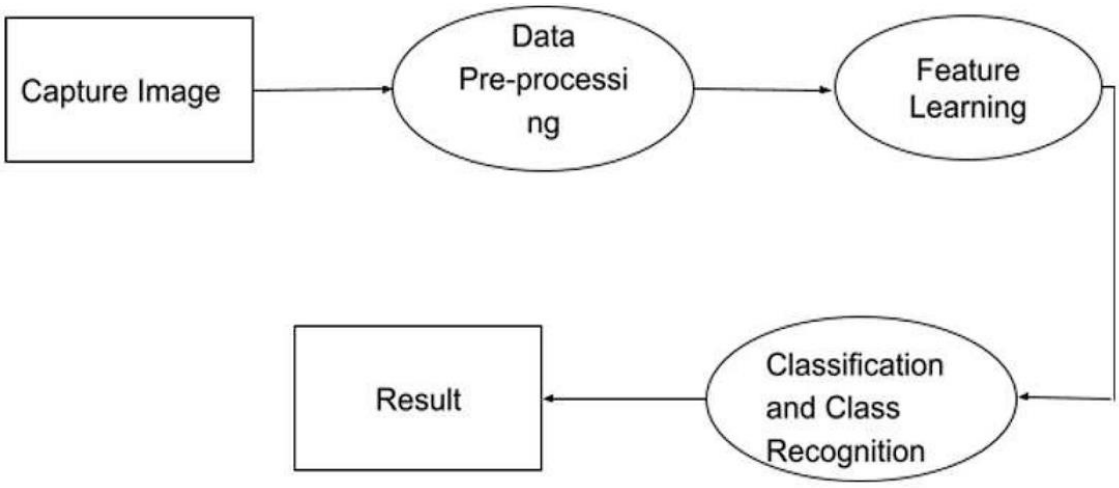


Fig. 2.4 Level One DFD

2.2 UML Diagrams

UML stands for Unified Modeling Language. Taking SRS document of analysis as input to the design phase drawn UML diagrams. The UML is only language so is just one part of the software development method. The UML is process independent, although optimally it should be used in a process that should be driven, architecture-centric, iterative, and incremental. The UML is language for visualizing, specifying, constructing, documenting the articles in a software-intensive system. It is based on diagrammatic representations of software components.

A modeling language is a language whose vocabulary and rules focus on the conceptual and physical representation of the system. A modeling language such as the UML is thus a standard language for software blueprints. The UML is a graphical language, which consists of all interesting systems. There are also different structures that can transcend what can be represented in a programming language.

These are different diagrams in UML.

2.2.1 Use Case Diagram

Use Case during requirement elicitation and analysis to represent the functionality of the system. Use case describes a function by the system that yields a visible result for an actor. The identification of actors and use cases result in the definitions of the boundary of the system i.e., differentiating the tasks accomplished by the system and the tasks accomplished by its environment. The actors are outside the boundary of the system, whereas the use cases are inside the boundary of the system. Use case describes the behavior of the system as seen from the actor's point of view. It describes the function provided by the system as a set of events that yield a visible result for the actor.

2.2.1.1 Purpose of Use Case Diagrams

The purpose of use case diagram is to capture the dynamic aspect of a system. However, this definition is too generic to describe the purpose, as other four diagrams (activity, sequence, collaboration, and Statechart) also have the same purpose. We will look into some specific purpose, which will distinguish it from other four diagrams.

Use case diagrams are used to gather the requirements of a system including internal and external influences. These requirements are mostly design requirements. Hence, when a system is analyzed to gather its functionalities, use cases are prepared and actors are identified.

When the initial task is complete, use case diagrams are modelled to present the outside view.

In brief, the purposes of use case diagrams can be said to be as follows:

1. Used to gather the requirements of a system.
2. Used to get an outside view of a system.
3. Identify the external and internal factors influencing the system.
4. Show the interaction among the requirements are actors.

2.2.1.2 How to Draw a Use Case Diagram?

Use case diagrams are considered for high level requirement analysis of a system. When the requirements of a system are analyzed, the functionalities are captured in use cases.

We can say that use cases are nothing but the system functionalities written in an organized manner. The second thing which is relevant to use cases are the actors. Actors can be defined as something that interacts with the system.

Actors can be a human user, some internal applications, or may be some external applications. When we are planning to draw a use case diagram, we should have the following items identified.

2.2.1.3 Functionalities to be represented as use case:

- Actors
- Relationships among the use cases and actors.

Use case diagrams are drawn to capture the functional requirements of a system. After identifying the above items, we have to use the following guidelines to draw an efficient use case diagram.

The name of a use case is very important. The name should be chosen in such a way so that it can identify the functionalities performed.

Give a suitable name for actors.

Show relationships and dependencies clearly in the diagram.

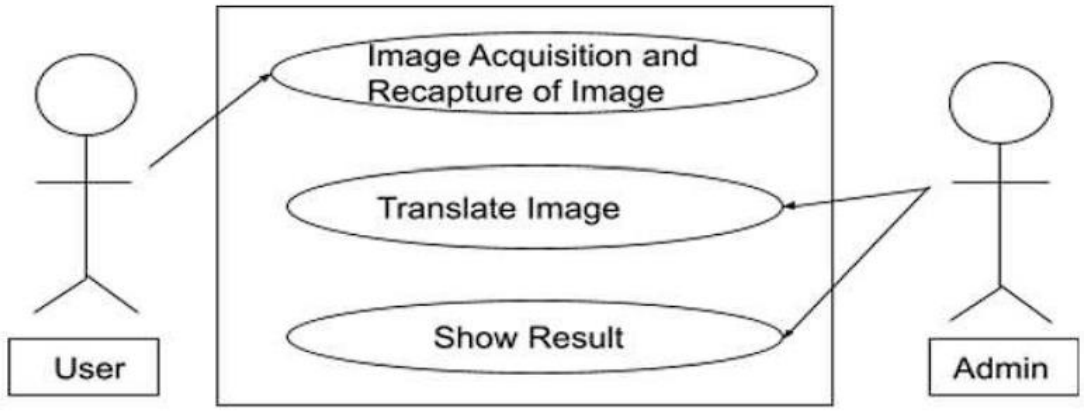
Do not try to include all types of relationships, as the main purpose of the diagram is to identify the requirements.

Use notes whenever required to clarify some important points.

Fig. 2.5 Use Case Symbols.

2.2.1.4 Association between Actors and Use Cases

A use case describes specific functionality that the system provides to its users. The functionality is triggered by an actor. Actors are connected to the use case through binary association. The association indicates that the actors and the use cases communicates through message passing. An actor must be associated with at least one use case. Similarly, a given use case must be associated with the at least one actor.



2.2.1.5 Use Case Description

1. Image Acquisition and Recapture Image:

1.1. Brief Description

This use case describes the capturing of the live images made by the users and to recapture in case they make a wrong sign.

1.2. Actors

The following actor interacts and participates in this use case: User

1.3. Flow of Event

The use case starts when the actor(user) make any sign of alphabet and press the option as per his requirement.

1.4. Special Requirement

None

1.5. Pre-Condition

The system should fulfill all the software and hardware requirements before running the application.

1.6. Post-Condition

If the use case if successful, it will go for feature learning.

1.7. Extension Points

None

2. Translate Image:

2.1. Brief Description

This use case describes the pre-processing and feature learning is done by CNN algorithm.

2.2. Actors

The following actor interacts and participates in this use case: Admin

2.3. Flow of Event

I. Receive image acquired through the camera.

II. Pre-processing and feature learning is done by CNN itself.

2.4. Special Requirement

None

2.5 Pre-Condition

I. The system should fulfill all the software and hardware requirements before running the application.

II. Connection with the database should be proper.

2.6 Post-Condition

If the use case if successful, the detected and recognized alphabet will be shown.

2.7 Extension Points

None

3. Show Results:

3.1. Brief Description

This use case describes the case in which result is provided by admin(database) in the form of alphabet.

3.2. Actors

The following actor interacts and participates in this use case: Admin(Database)

3.3. Flow of Event

I. The admin gives the alphabet of the acquired image drawn by the user at the beginning.

II. The user interprets the alphabet.

3.4 Special Requirement

None

3.5 Pre-Condition

I. The system should fulfill all the software and hardware requirements before running the application.

II. Connection with the database should be proper.

III. User must draw a valid sign.

3.6 Post-Condition

If the use case if successful, details regarding the recognized alphabet will be shown as image on half of the interface.

3.7 Extension Points

None

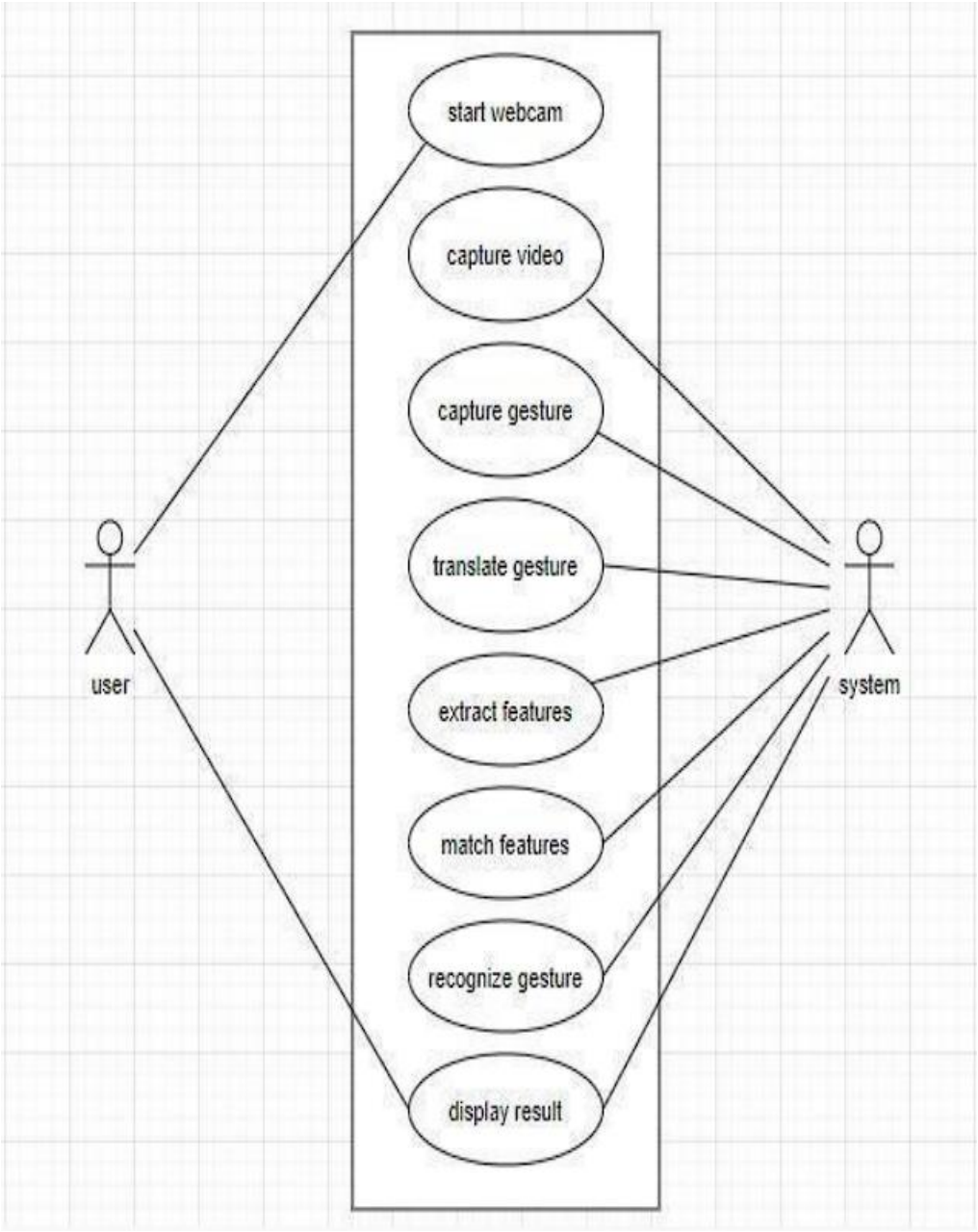


Fig. 2.6 Use Case diagram for Sign Language Detection system.

Table 2.1 Use Case scenario for Sign Language Detection system.

Use Case name	Sign language recognition
Participating actors	User, system
Flow of events	Start the system small Capturing video(s) Capture gesture(s) Translate gesture(s) Extract feature(s) Match feature(s) Recognize gesture(s) Display result
Entry condition	Run the code
Exit condition	Displaying the label
Quality Requirements	Cam pixels clarity, Good light condition

2.2.2 Class Diagram

Class diagrams model class structure and contents using design elements such as classes, packages and objects. Class diagram describe the different perspective when designing a system-conceptual, specification and implementation. Classes are composed of three things: name, attributes, and operations. Class diagram also display relationships such as containment, inheritance, association etc. The association relationship is most common relationship in a class diagram. The association shows the relationship between instances of classes.

2.2.2.1 How to Draw a Class Diagram?

Class diagrams are the most popular UML diagrams used for construction of software applications. It is very important to learn the drawing procedure of class diagram.

Class diagrams have a lot of properties to consider while drawing but here the diagram will be considered from a top-level view.

Class diagram is basically a graphical representation of the static view of the system and represents different aspects of the application. A collection of class diagrams represents the whole system.

The following points should be remembered while drawing a class diagram –

The name of the class diagram should be meaningful to describe the aspect of the system.

Each element and their relationships should be identified in advance.

Responsibility (attributes and methods) of each class should be clearly identified.

For each class, minimum number of properties should be specified, as unnecessary properties will make the diagram complicated.

Use notes whenever required to describe some aspect of the diagram. At the end of the drawing it should be understandable to the developer/coder.

Finally, before making the final version, the diagram should be drawn on plain paper and reworked as many times as possible to make it correct.

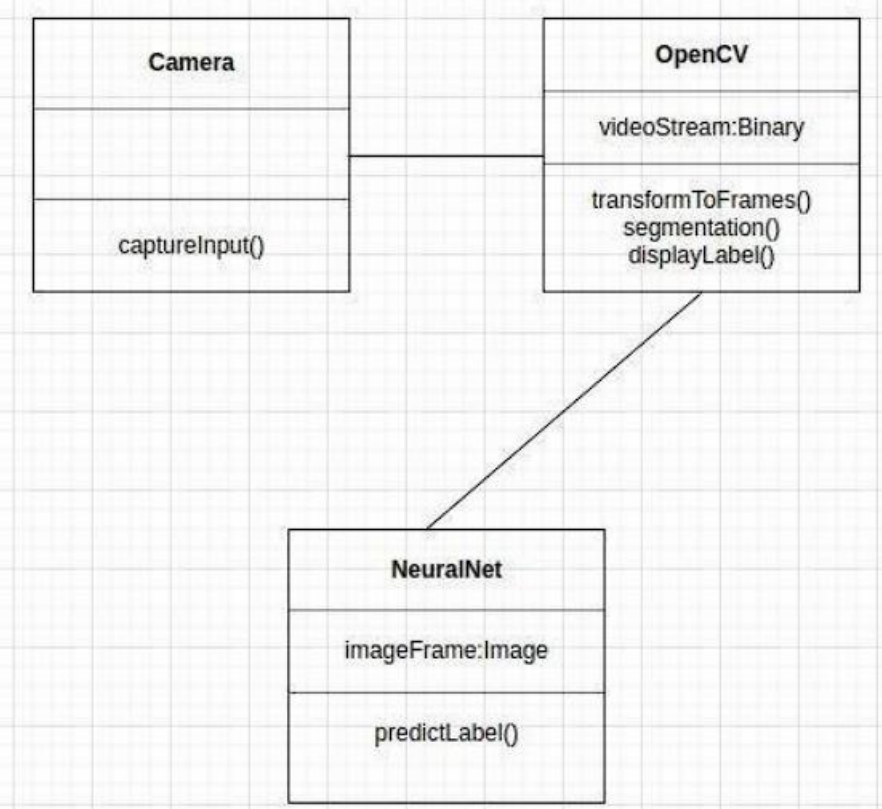


Fig. 2.7 Class diagram of Sign Language Detection system.

2.2.3 Sequence Diagram

Sequence diagram displays the time sequence of the objects participating in the interaction. This consists of the vertical dimension(time) and horizontal dimension (different objects).

Objects: Object can be viewed as an entity at a particular point in time with specific value and as a holder of identity.

A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development. Sequence diagrams are sometimes called event diagrams or event scenarios.

A sequence diagram shows, as parallel vertical lines (lifelines), different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.

If the lifeline is that of an object, it demonstrates a role. Leaving the instance name blank can represent anonymous and unnamed instances.

Messages, written with horizontal arrows with the message name written above them, display interaction. Solid arrow heads represent synchronous calls, open arrow heads represent asynchronous messages, and dashed lines represent reply messages. If a caller sends a synchronous message, it must wait until the message is done, such as invoking a subroutine. If a caller sends an asynchronous message, it can continue processing and doesn’t have to wait for a response. Asynchronous calls are present in multithreaded applications, event-driven applications and in message-oriented middleware. Activation boxes, or method-call boxes, are opaque rectangles drawn on top of lifelines to represent that processes are being performed in response to the message (Execution Specifications in UML).

Objects calling methods on themselves use messages and add new activation boxes on top of any others to indicate a further level of processing. If an object is destroyed (removed from memory),

an X is drawn on bottom of the lifeline, and the dashed line 48 ceases to be drawn below it. It should be the result of a message, either from the object itself, or another.

A message sent from outside the diagram can be represented by a message originating from a filled-in circle (found message in UML) or from a border of the sequence diagram (gate in UML).

UML has introduced significant improvements to the capabilities of sequence diagrams. Most of these improvements are based on the idea of interaction fragments which represent smaller pieces of an enclosing interaction. Multiple interaction fragments are combined to create a variety of combined fragments, which are then used to model interactions that include parallelism, conditional branches, optional interactions.

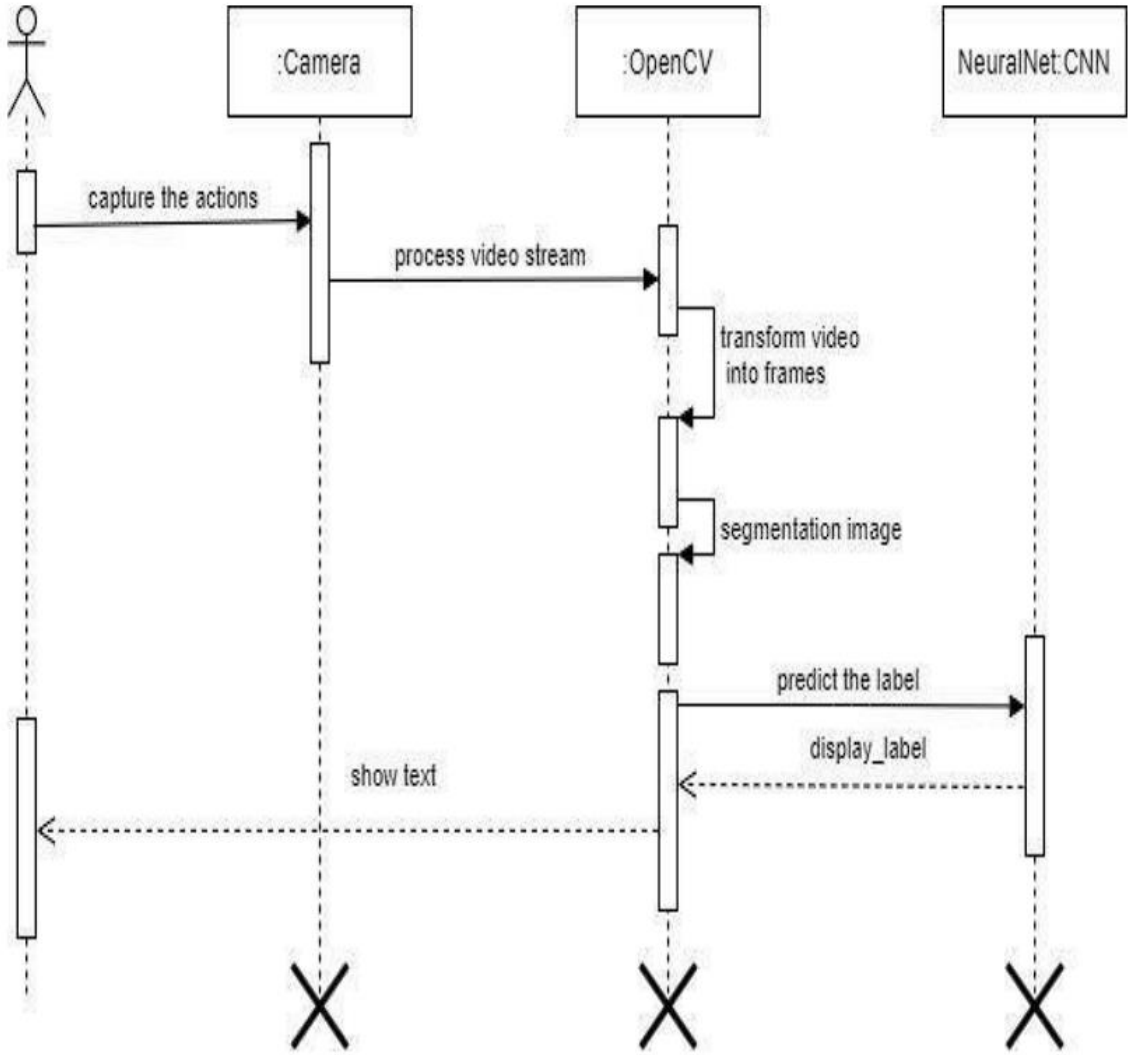


Fig. 2.8 Sequence diagram of Sign Language Detection system.

2.2.4 State Chart

A state chart diagram describes a state machine which shows the behavior of classes. It shows the actual changes in state not processes or commands that create those changes and is the dynamic behavior of objects over time by modeling the life cycle of objects of each class.

It describes how an object is changing from one state to another state. There are mainly two states in State Chart Diagram:

1. Initial State
2. Final-State.

Some of the components of State Chart Diagram are:

1. State: It is a condition or situation in life cycle of an object during which it's satisfies same condition or performs some activity or waits for some event.
2. Transition: It is a relationship between two states indicating that object in first state performs some actions and enters into the next state or event.
3. Event: An event is specification of significant occurrence that has a location in time and space.

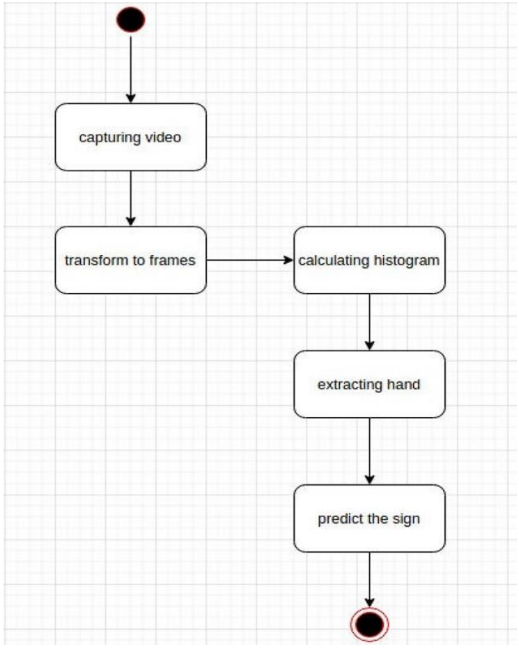


Fig. 2.9 State Chart diagram of Sign Language Detection system.

2.2.5 ER Diagram

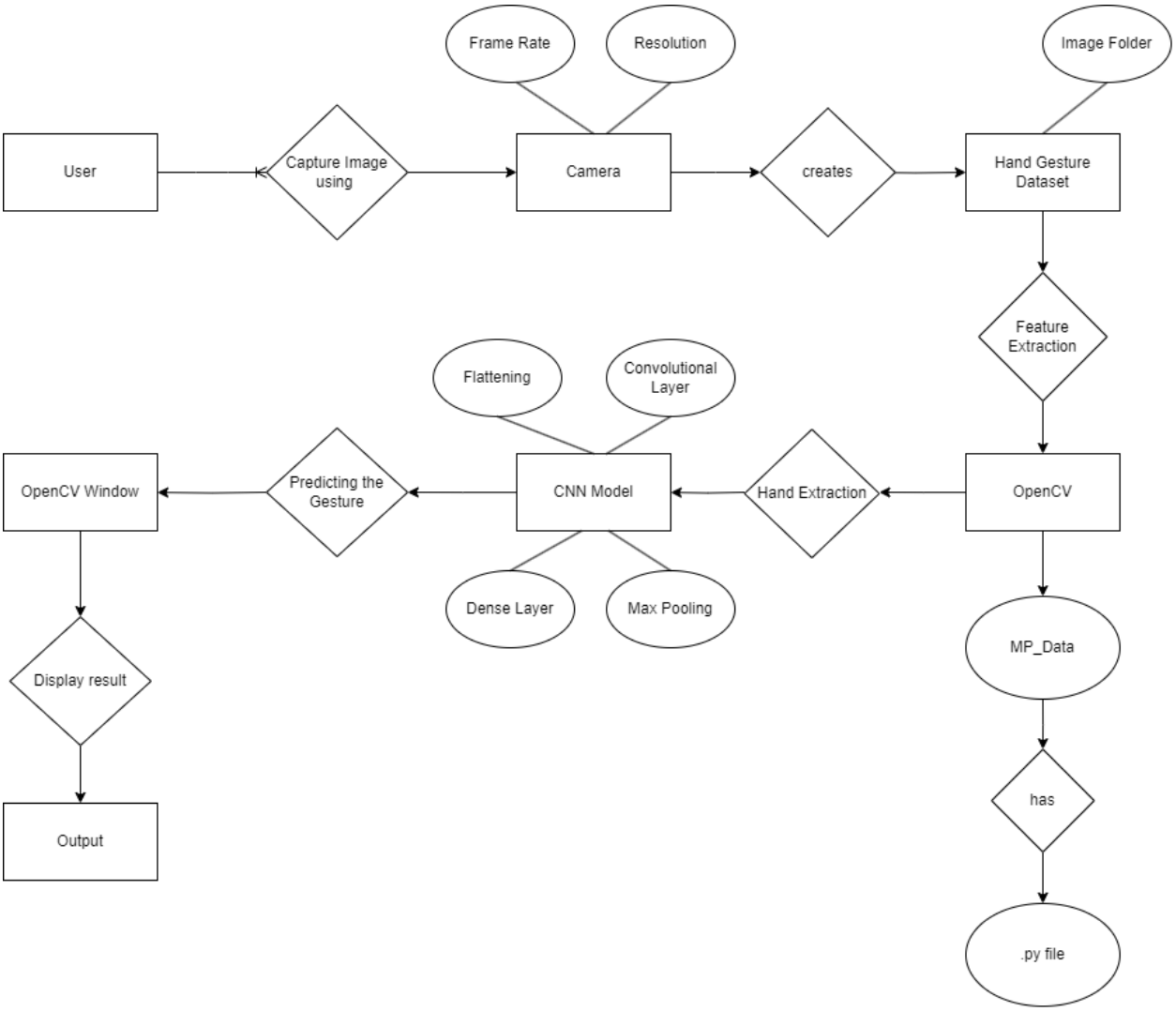


Fig. 2.10 ER Diagram for Sign Language Detection System