TRAFFIC SIGN RECOGNITION

TABLE OF CONTENTS

LIST OF FIGURES Figure No Figure Name Page no

LIST OF TABLES

NOMENCLATURE

ADAS	-	Advanced driver-assistance system
GPU	-	Graphics processing unit
CNN	-	Convolutional Neural Network
HDF	-	Hierarchical Data Format
FC	-	Fully connected
OPENCV	-	Open Computer Vision
GTSD	-	German Traffic Sign Dataset

TRAFFIC SIGN RECOGNITION

ABSTRACT:

Detection of small objects is a challenging task. When we consider a Self driving car, it is a “driverless” or autonomous vehicle that operates without human intervention. It uses specially designed hardware or software to move from one destination to another. Even though the self-driving car revolution is unstoppable, it is facing many challenges. The main challenge we face is Traffic rule implementation. In order to detect and classify the traffic signals in street views, we build a deep neural network model that can classify the traffic signs present in the image or a video.

CHAPTER- 1 INTRODUCTION

In the world of Artificial Intelligence and advancement in technologies, many researchers and big companies like Tesla, Uber, Google, Mercedes-Benz, Toyota, Ford, Audi, etc are working on autonomous vehicles and self-driving cars. So, for achieving accuracy in this technology, the vehicles should be able to interpret traffic signs and make decisions accordingly.

There are several different types of traffic signs like speed limits, no entry, traffic signals, turn left or right, children crossing, no passing of heavy vehicles, etc. Traffic signs classification is the process of identifying which class a traffic sign belongs to.

Traffic sign recognition is a technology by which a vehicle is able to recognize the traffic signs which are kept on the road e.g “speed limit” or “turn ahead” or “prohibitions”. This is a part of the features collectively called ADAS. ADAS are group of electronic technologies that assist drivers in driving and parking functions. Through a safe human-machine interface, ADAS increase car and road safety. ADAS use automated technology, such as sensors and cameras, to detect nearby obstacles or drive errors, and respond accordingly.

The technology which is developed to recognize the traffic signs uses image processing techniques to detect the traffic signs. Traffic-Sign recognition first appeared, in the form of speed limit recognition. At that time these systems only detected the round speed limit signs. The objective of the Traffic Sign Recognition project was to identify a traffic sign form a digital photograph.

The below block diagram shows us the overall process of how the traffic signs are recognised. When we give input as an image, a pre-processing algorithm is applied on that image. The pre-processed image is classified and based on that classification the sign is recognized.

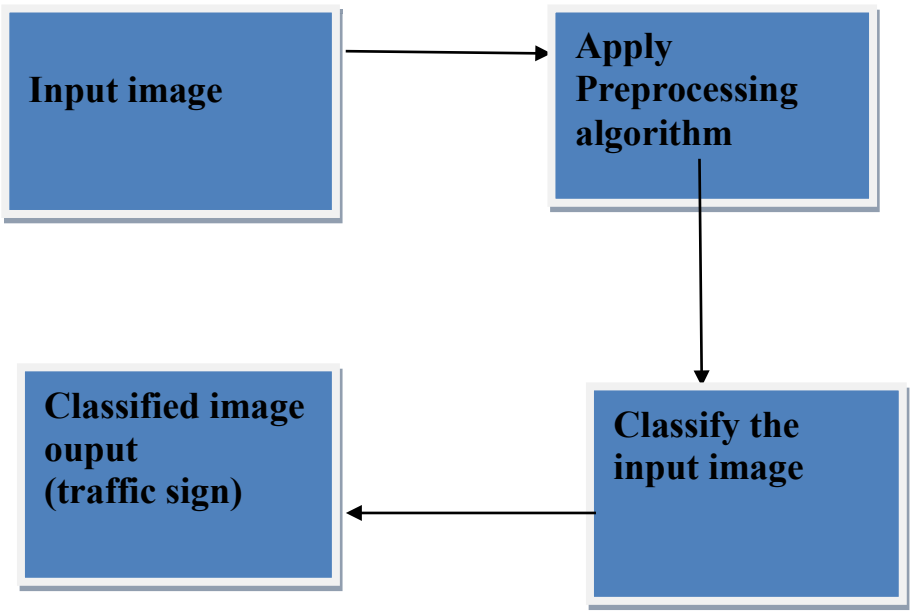


Fig 1.1 Overall Block Diagram

CHAPTER – 2
AIM & SCOPE

Now a days self-driving cars is the booming technology all over the world. Hence, Recognizing Traffic signs is one of the learning objective. We will implement this by using Convoltutional Neural Networks by which we will train our dataset.

2.1 EXISTING SYSTEM

The existing system for recognising traffic signs is implemented by using cnn algorithm which is a deep learning algorithm by using pytorch library.

PyTorch is an open source machine learning library for Python. It is primarily used for applications such as natural language processing. The scope for this build of PyTorch is AMD GPUs with ROCm support, running on Linux.

As discussed above, the drawback arised by using PyTorch library is that it does not support the Intel GPUs. All the users have windows operating system and all windows operating systems are built in by using Intel GPUs. Hence the algorithm which is implemented by using PyTorch library cannot be implemented in all systems.

2.2 PROPOSED SYSTEM

The proposed system for Traffic sign recognition is being implemented by using convolutional neural networks which is a deep learning algorithm by using keras library.

2.3 FEASIBILITY STUDY

A feasibility study is used to evaluate the projects potential for success. The evaluation can be done in three phases. 1.Technical Feasibility

2.Operational Feasibility

3.Economical Feasibility

2.3.1 Technical Feasibility:

Technical Feasibility focuses on the technical resources available. It involves the evaluation of hardware, software, and other technical requirements of the proposed system. Technical feasibility consider the technical requirements of the proposed project. The technical requirements are then compared to the technical capability of the organization.

The Algorithm is implemented in python and tkinter GUI. Python is an interpreted high-level general-purpose programming language. Its language constructs as well as its object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects. Keras library used to build and train the CNN model has predefined functions to add the layers into the model and to specify functionality of the specified layer. 3

2.3.2 Operational Feasibility:

Operational Feasibility examines how a project plan satisfies the requirements identified in the requirements analysis phase of a system development. A system is operationally feasible when it can reduce the cost of developing the system without undermining its quality or product

Traffic sign recognition algorithm can be used as one of the algorithm in autonomous vehicles. Hence, this algorithm does not require any other software installations. The whole algorithm can be implemented when the sensors detects the image. Recognition is the next step after detection, where the image is preprocessed and predicted using the trained CNN model.

2.3.3 Economic Feasibility:

Economic Feasibility serves as an independent project assessment and enhances project credibility-helping decision-makers determine the positive economic benefits to the organization that the proposed project will provide. As this algorithm can be used as a sub algorithm it requires less cost and does not need any equipment.

CHAPTER – 3
CONCEPTS & METHODS

3.1 PROBLEM DESCRIPTION

Detection of small objects is a challenging task. When we consider a Self driving car, it is a "driverless" or autonomous vehicle that operates without human intervention. It uses specially designed hardware or software to move from one destination to another. Even though the self-driving car revolution is unstoppable, it is facing many challenges. The main challenge we face is Traffic rule implementation. In order to detect and classify the traffic signals in street views, we build a deep neural network model that can classify the traffic signs present in the image.

The main important task is recognizing traffic signs to machines when there is no human intervention. Recognizing traffic signs plays a vital role in order to reduce accidents. As the autonomous cars are the booming technology recognizing these traffic signs should be more accurate.

3.2 PROPOSED SOLUTION

Traffic Sign Recognition is implemented by CNN which is a deep learning algorithm . Using CNN, we can easily differentiate a image from other images. CNN algorithm requires less processing time. The CNN algorithm is build by using built in functions in Keras library. Image classification involves the extraction of features from the image to observe some patterns in the dataset. Using an ANN for the purpose of image classification would end up being very costly in terms of computation since the trainable parameters become extremely large.

For example, if we have a 50 X 50 image of a sign, and we want to train our traditional ANN on that image to classify the trainable parameters become –

(50*50) * 100 image pixels multiplied by hidden layer + 100 bias + 2 * 100 output neurons + 2 bias = 2,50,302

We use filters when using CNNs. Filters exist of many different types according to their purpose. Keras is an open source high-level neural network library which is written in python. It is made user-friendly, extensible, and modular for facilitating faster experimentation with deep neural networks. It is a multi backend and supports multi-platform, which helps all the encoders come together for coding.

The dataset we are considering is German Traffic Sign Dataset. The dataset contains more than 50,000 images of different traffic signs. It is further classified into 43 different classes. In dataset some of the classes have many images while some classes have few 5 images. The size of the dataset is around 300 MB. The dataset has a train folder which contains images inside each class and a test folder which we will use for testing our model.

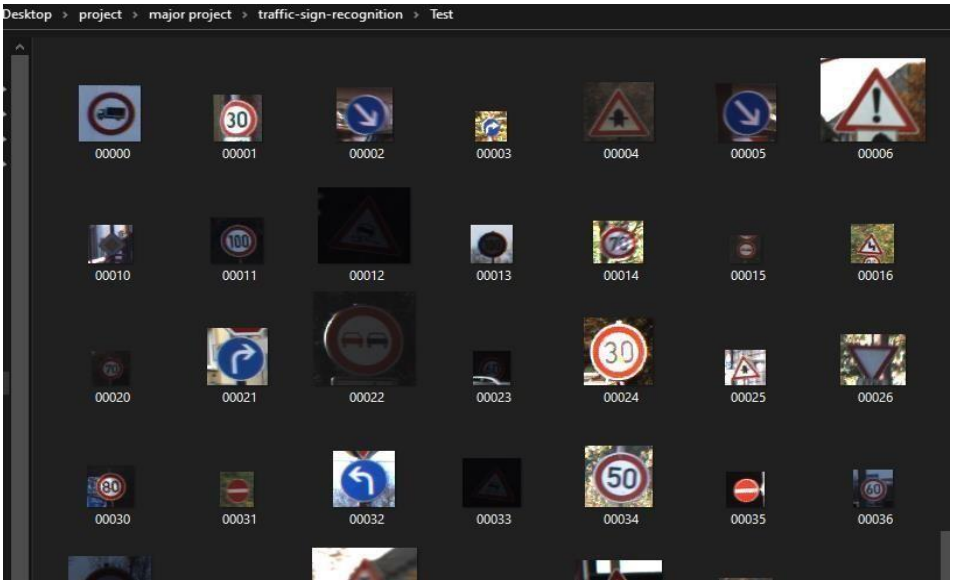


Fig3.1 Test Folder

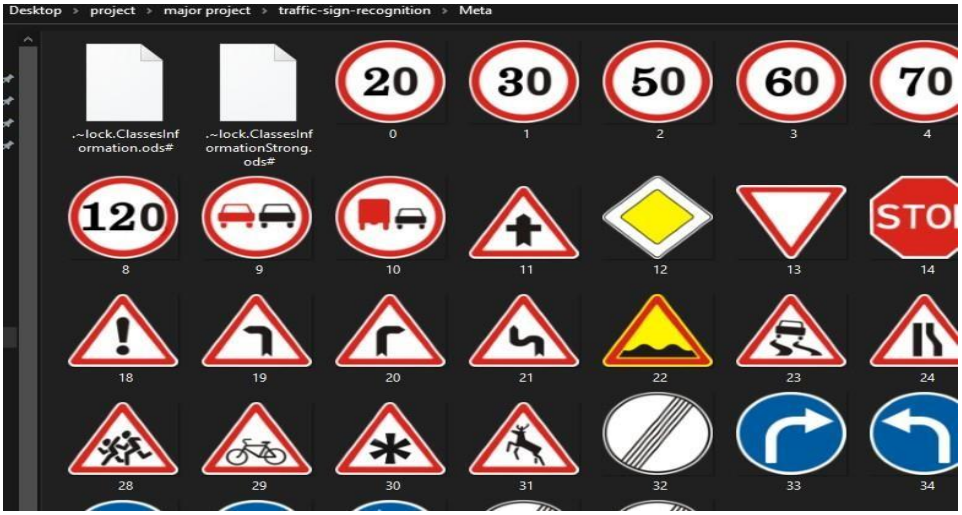


Fig3.2 Meta Folder

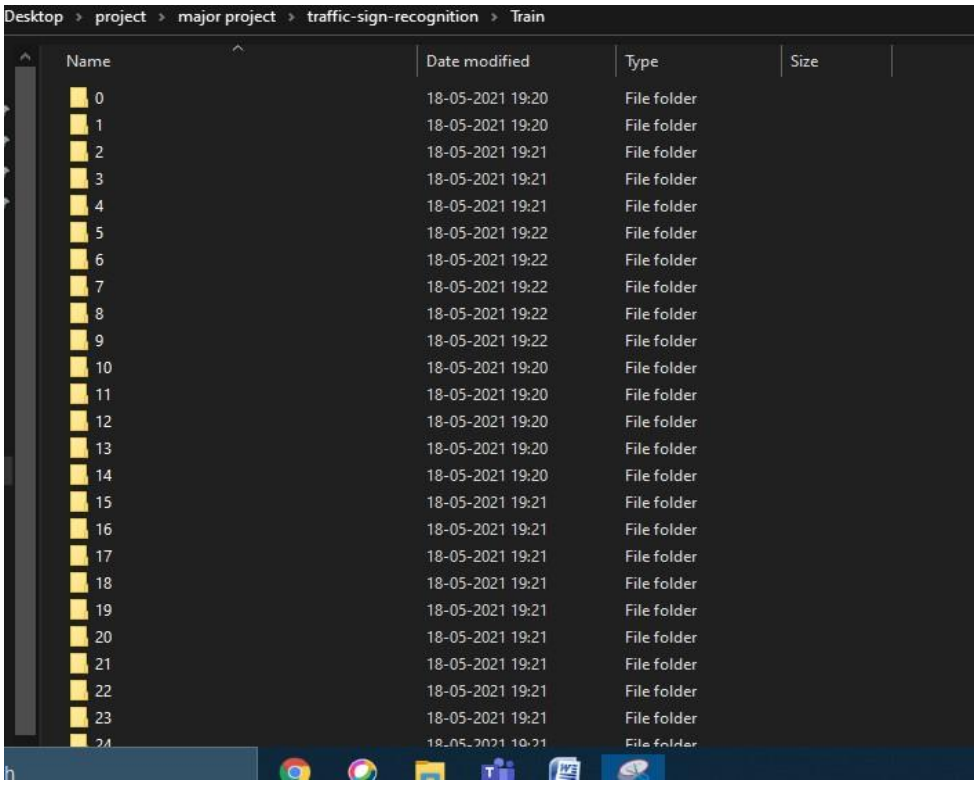


Fig3.3 Train Folder *Architecture*:

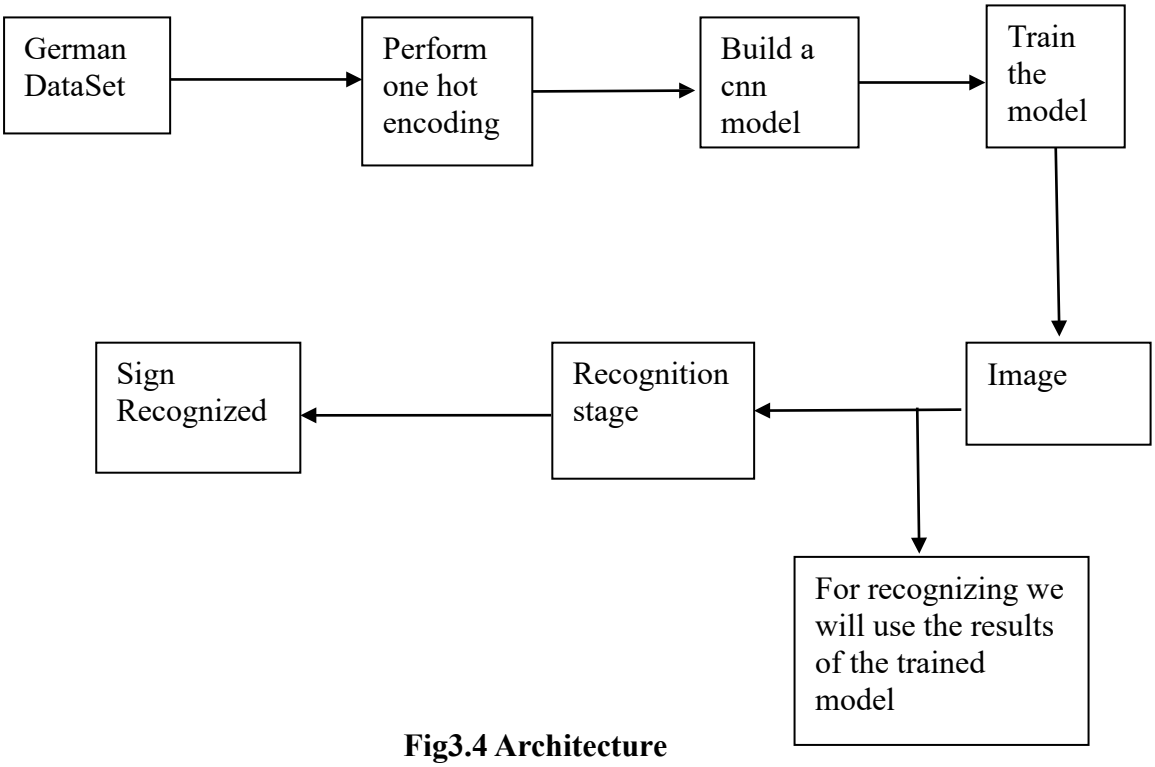


Fig3.4 Architecture

3.3 SYSTEM ANALYSIS METHODS

System Analysis is a process of collecting and interpreting facts, identifying the problems, and decomposition of a system into its components. System analysis is conducted for the purpose of studying a system or its parts in order to identify its objectives.

3.3.1 Use Case Diagram

Use case diagrams are used to gather the requirements of a system including internal and external influences. These requirements are mostly design requirements. The purpose of use case diagram is to capture the dynamic aspect of a system.

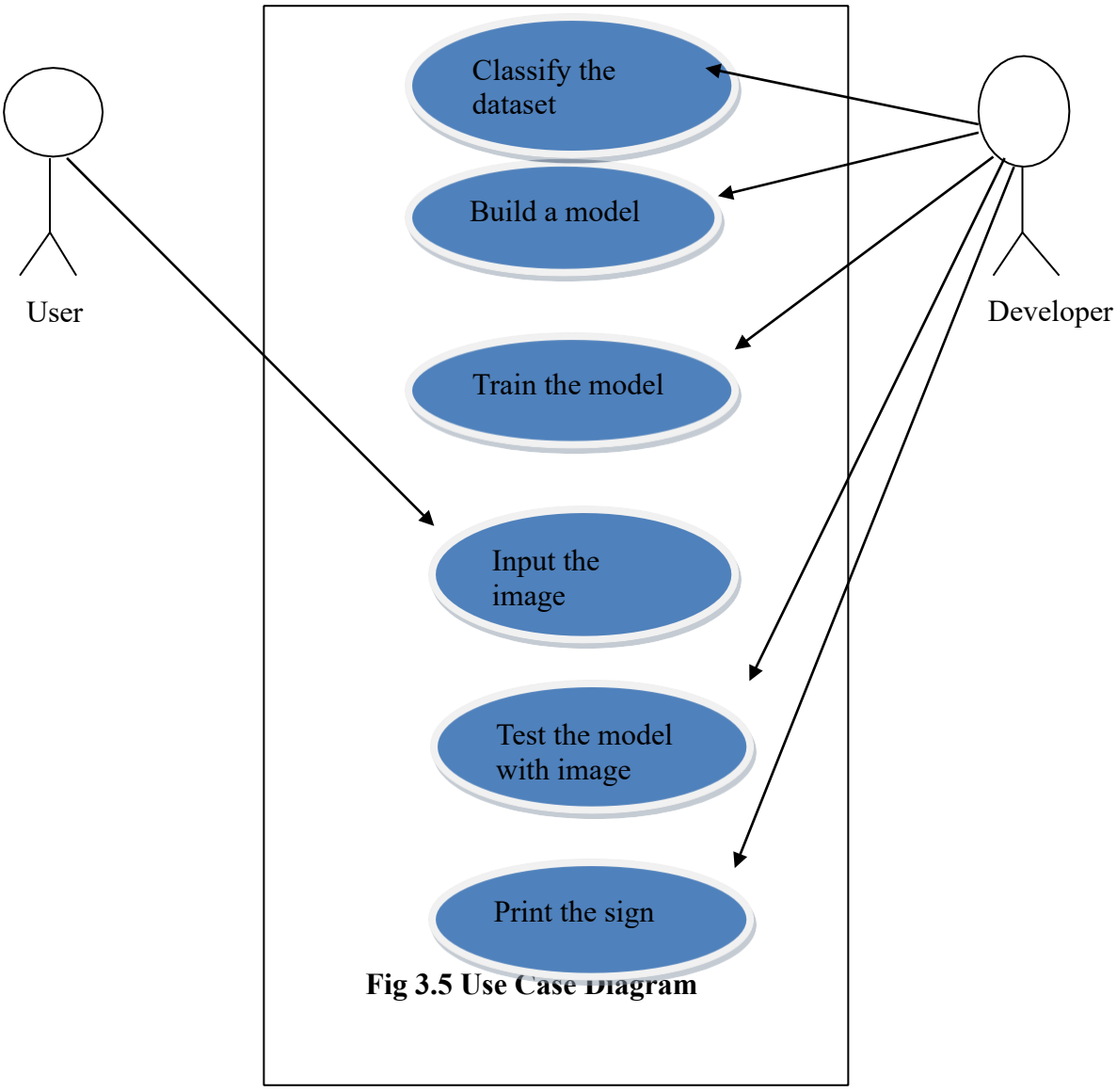


Fig 3.5 Use Case Diagram

Fig 3.5 Usecase Diagram

***Actor*:**

- User
- Developer

***Usecases for User*:**

- Input the image

***Usecases for Developer*:**

- Classify the dataset
- Build the model
- Train the model
- Test the model
- Print the output

3.3.2 Activity Diagram

Activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system. The control flow is drawn from one operation to another. This flow can be sequential, branched, or concurrent. Activity diagrams are used for modeling the dynamic aspects of systems.

Activity diagrams emphasize the flow of control from activity to activity. An activity is an ongoing non atomic execution within a state machine. Activities ultimately result in some action, which is made up of executable atomic computations that result in a change in state of the system or the return of a value. Actions encompass calling another operation, sending a signal, creating or destroying an object, or some pure computation, such as evaluating an expression. Graphically, an activity diagram is a collection of vertices and arcs.

An activity diagram is essentially a flowchart, showing flow of control from activity to activity. This involves modeling the sequential (and possibly concurrent) steps in a computational process. In an activity diagram model, the flow of an object moves from state to state at different points in the flow of control. Activity diagrams may stand alone to visualize, specify, construct, and document the dynamics of a society of objects, or they may be used to model the flow of control of an operation.

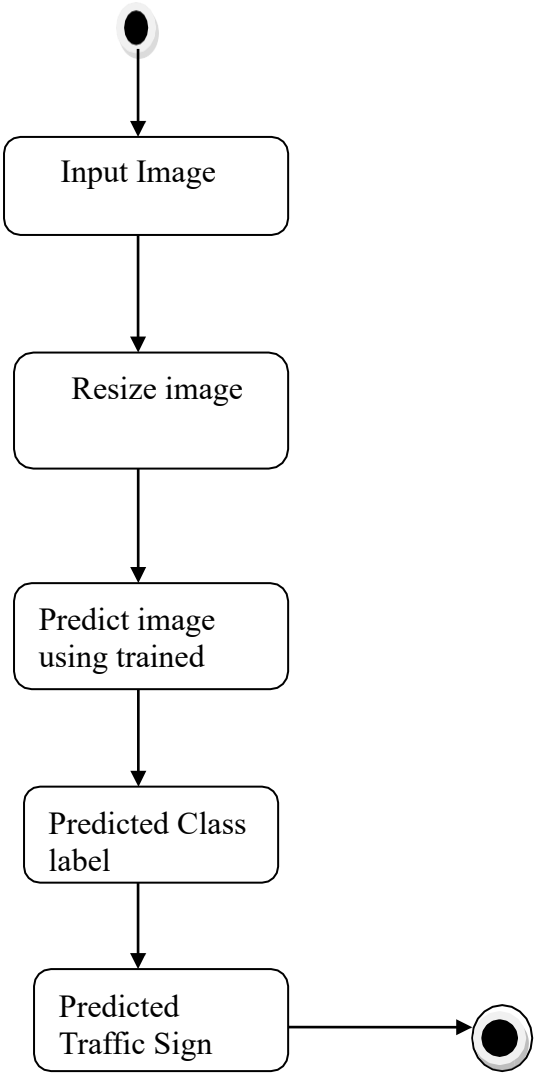


Fig3.6 Activity Diagram

3.4 SYSTEM REQUIREMENTS

System requirements are the configuration that a system must have in order for a hardware or software application to run smoothly and efficiently.

3.4.1 Software Requirements

1. ***Numpy*:**

   NumPy is a Python library used for working with arrays. NumPy aims to provide an array object that is up to 50x faster than traditional Python lists. The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy.

2. ***Keras*:** Keras follows best practices for reducing cognitive loads: it offers consistent and simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear and actionable error messages.

3. ***Tensorflow*:**

   TensorFlow is an end-to-end open source platform for machine learning. It has a flexible ecosystem of tools, libraries that lets developers easily build and deploy ML applications.

4. ***Pil*:**

   PIL is an imaging library for Python programming language. Script is a piece of code that's used to automate system oriented tasks.

5. ***OpenCV*:**

   OpenCV is an open-source Python library. It extracts the description from the real-time image or digital image, which may be an object, a text description.

6. ***Sklearn*:**

   Sklearn provides efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python. This library is built upon NumPy, SciPy and Matplotlib.

3.4.2 Hardware Requirements

1. For training end to end version of CNN 3g of gpu memory
2. RAM 2GB (64bit)
3. Hard disk space 16GB (32bit)

3.5 SYSTEM DESIGN

System Design is the process of designing the architecture, components, and interfaces for a system so that it meets the end-user requirements. In this phase, a logical system is built which fulfils the given requirements. Design phase of software development deals with transforming the client's requirements into a logically working system.

***Architectural design*:** The architectural design of a system emphasizes the design of the system architecture that describes the structure, behavior and more views of that system and analysis.

***Logical design*:** The logical design of a system pertains to an abstract representation of the data flows, inputs and outputs of the system. This is often conducted via modelling, using an over-abstract (and sometimes graphical) model of the actual system. In the context of systems, designs are included. Logical design includes entity-relationship diagrams (ER diagrams).

***Physical design*:** The physical design relates to the actual input and output processes of the system. This is explained in terms of how data is input into a system, how it is

verified/authenticated, how it is processed, and how it is displayed. In physical design, the following requirements about the system are decided.

1. Input requirement,
2. Output requirements,
3. Storage requirements,
4. Processing requirements,
5. System control and backup or recovery.

3.5.1 Class Diagram

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 represent the whole system. A class diagram is a diagram that shows a set of classes, interfaces, and collaborations and their relationships. Graphically, a class diagram is a collection of vertices and arcs. A class diagram shares the same common properties as do all other diagrams name and graphical content that are a projection into a model.

Class diagram describes the attributes and operations of a class and also the constraints imposed on the system. The class diagrams are widely used in the modeling of object oriented systems because they are the only UML diagrams, which can be mapped directly with object-oriented languages. Here, the classes we implemented are:

1. Pre-Process image
2. One hot encoding
3. Build model
4. Train model
5. Test model

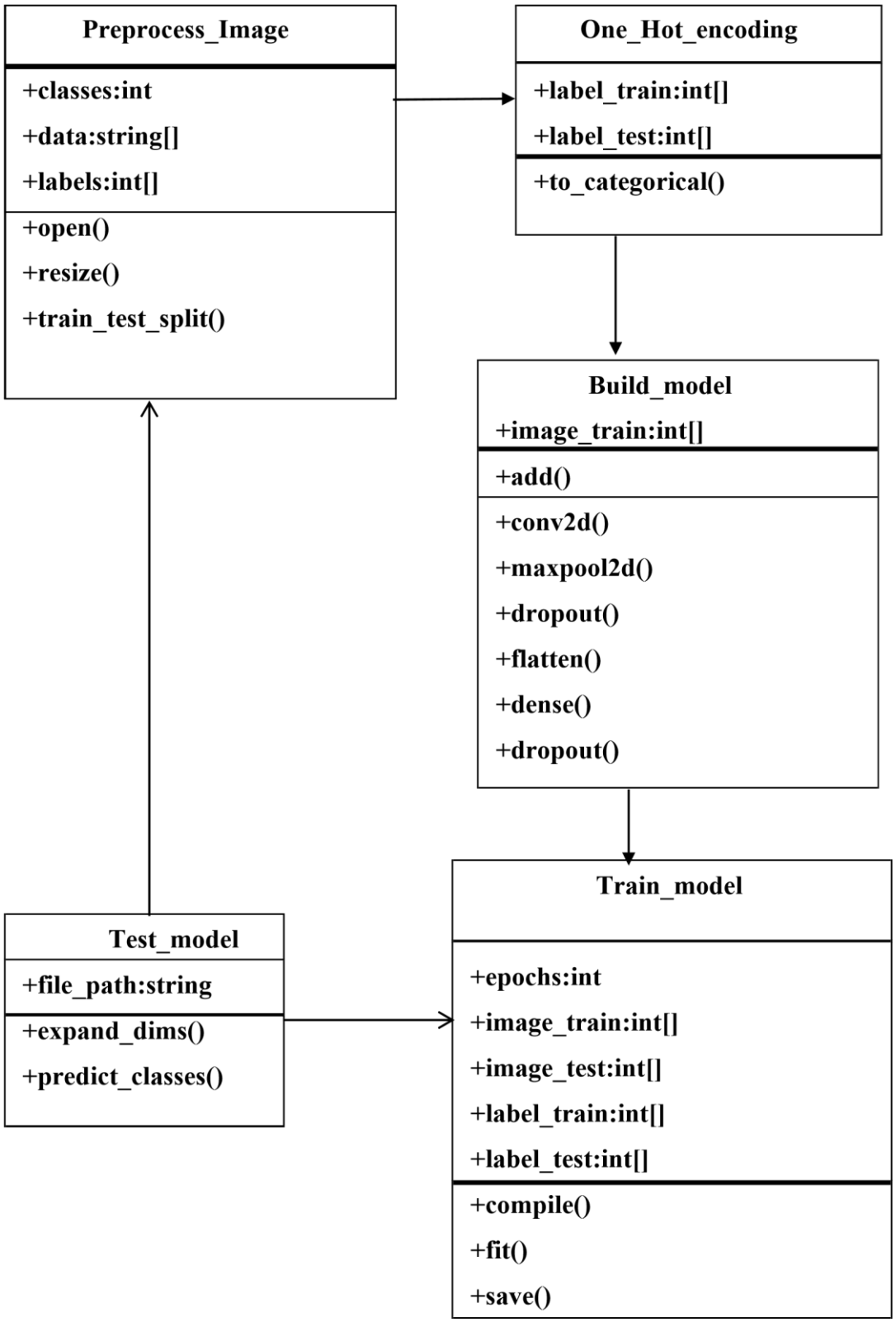


Fig3.7 Class Diagram

3.5.2 Sequence Diagram

UML Sequence Diagrams are interaction diagrams that detail how operations are carried out. They capture the interaction between objects in the context of a collaboration. Sequence Diagrams are time focus and they show the order of the interaction visually by

using the vertical axis of the diagram to represent time what messages are sent and when. A sequence diagram shows object interactions arranged in time sequence. It depicts the objects 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.

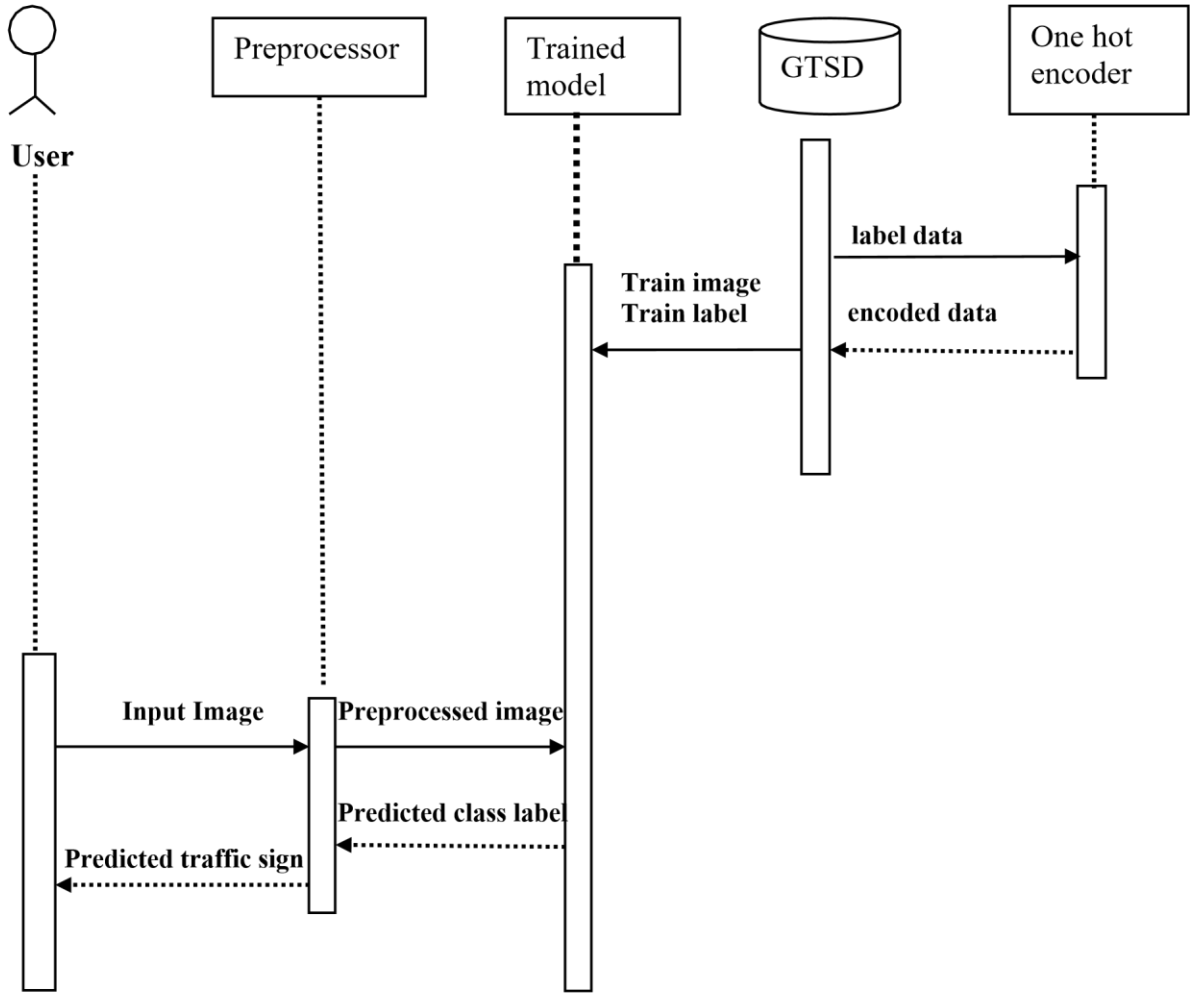


Fig 3.8 Sequence Diagrams

3.6 E-R DIAGRAMS

An entity relationship diagram (ERD) shows the relationships of entity sets. An entity in this context is an object, a component of data. An entity set is a collection of similar entities. These entities can have attributes that define its properties.

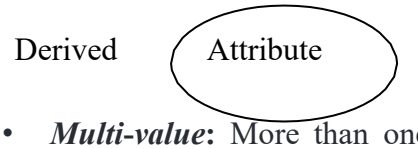
ER Diagrams are composed of entities, relationships and attributes. They also depict cardinality, which defines relationships in terms of numbers. Here's a glossary:

- ***Entity***
  A definable thing such as a person, object, concept or event that can have data stored about it. Think of entities as nouns. Examples: a customer, student, car or product. Typically shown as a rectangle.

- ***Entity type*****:** A group of definable things, such as students or athletes, whereas the entity would be the specific student or athlete. Other examples: customers, cars or products.
- ***Entity set*****:** Same as an entity type, but defined at a particular point in time, such as students enrolled in a class on the first day. Other examples: Customers who purchased last month, cars currently registered in Florida. A related term is instance, in which the specific person or car would be an instance of the entity set.
- ***Entity categories*****:** Entities are categorized as strong, weak or associative. A strong entity can be defined solely by its own attributes, while a weak entity cannot. An associative entity associates entities (or elements) within an entity set.
- ***Entity keys*****:** Refers to an attribute that uniquely defines an entity in an entity set. Entity keys can be super, candidate or primary.
- ***Super key*****:** A set of attributes (one or more) that together define an entity in an entity set.
- ***Candidate key*****:** A minimal super key, meaning it has the least possible number of attributes to still be a super key. An entity set may have more than one candidate key.
- ***Primary key*****:** A candidate key chosen by the database designer to uniquely identify the entity set.
- ***Foreign key*****:** Identifies the relationship between entities.
- ***Relationship*****:** How entities act upon each other or are associated with each other. Think of relationships as verbs. For example, the named student might register for a course. The two entities would be the student and the course, and the relationship depicted is the act of enrolling, connecting the two entities in that way. Relationships are typically shown as diamonds or labels directly on the connecting lines**.**
- ***Recursive relationship*****:** The same entity participates more than once in the relationship.
- ***Attribute*****:** A property or characteristic of an entity. Often shown as an oval or circle**.**

- ***Descriptive attribute*:** A property or characteristic of a relationship (versus of an entity).
- ***Attribute categories*:** Attributes are categorized as simple, composite, derived, as well as single-value or multi-value.
- ***Simple*:** Means the attribute value is atomic and can't be further divided, such as a phone number.
- ***Composite*:** Sub-attributes spring from an attribute.
- ***Derived*:** Attributed is calculated or otherwise derived from another attribute, such as age from a birth date.



- ***Multi-value*:** More than one attribute value is denoted, such as multiple phone numbers for a person.
- ***Single-value*:** Just one attribute value. The types can be combined, such as: simple single-value attributes or composite multi-value attributes.
- ***Cardinality*:** Defines the numerical attributes of the relationship between two entities or entity sets. The three main cardinal relationships are one-to-one, one-to- many, and many-many. A one-to-one example would be one student associated with one mailing address**.** A one-to-many example (or many-to-one, depending on the relationship direction): One student registers for multiple courses, but all those courses have a single line back to that one student.
- ***Many-to-many example*:** Students as a group are associated with multiple faculty members, and faculty members in turn are associated with multiple students.
- ***Cardinality views*:** Cardinality can be shown as look-across or same-side, depending on where the symbols are shown.
- ***Cardinality constraints*:** The minimum or maximum numbers that apply to a relationship.

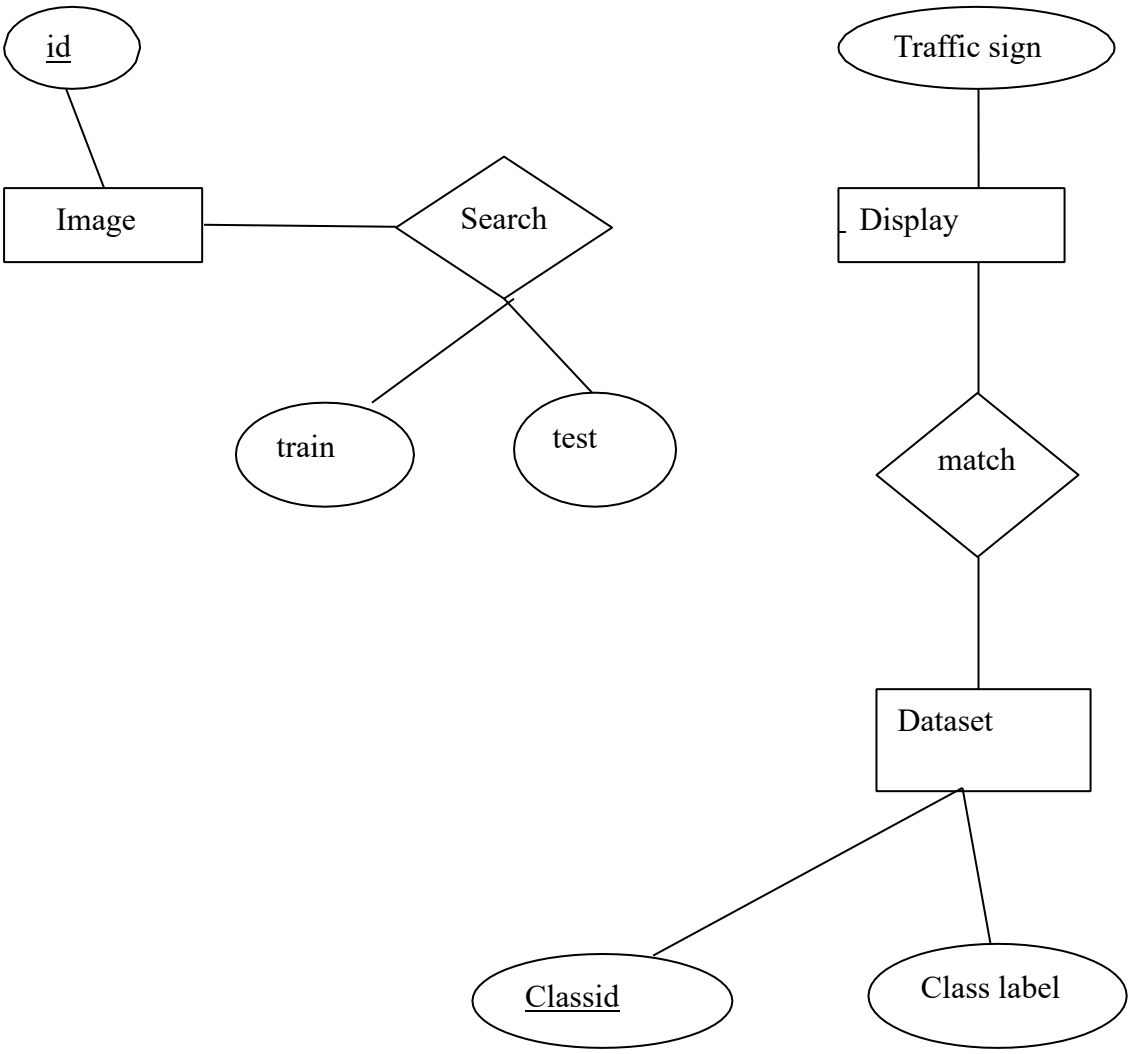


Fig 3.9 E-R diagram

CHAPTER – 4

IMPLEMENTATION

4.0 TECHNOLOGIES USED : PYTHON, DEEP LEARNING

4.0.1 Python:

Python is an interpreted high-level general-purpose programming language. Its language constructs as well as its object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects. A Python framework is an interface or tool that allows developers to build ML models easily.

4.0.2 Deep learning:

Deep learning is a subset of machine learning, which is essentially a neural network with three or more layers. These neural networks attempt to simulate the behavior of the human brain allowing it to "learn" from large amounts of data. While a neural network with a single layer can still make approximate predictions, additional hidden layers can help

to optimize and refine for accuracy. Deep learning eliminates some of data pre-processing that is typically involved with machine learning.

Deep learning algorithms can ingest and process unstructured data, like text and images, and it automates feature extraction, removing some of the dependency on human experts. Deep neural networks consist of multiple layers of interconnected nodes, each building upon the previous layer to refine and optimize the prediction or categorization. Convolutional neural networks (CNNs), used primarily in computer vision and image classification applications, can detect features and patterns within an image, enabling tasks, like object detection or recognition. In 2015, a CNN bested a human in an object recognition challenge for the first time.

Traffic sign recognition is implemented in four steps. They are:

1. Explore the dataset
2. Build a CNN model
3. Train the model
4. Test the model

4.0.2 Explore the dataset:

Our 'train' folder contains 43 folders each representing a different class. The range of the folder is from 0 to 42. With the help of the OS module, we iterate over all the classes and append images and their respective labels in the data and labels list. The PIL library is used to open image content into an array.

4.0.3 Build a CNN model:

The process of building a cnn model takes place in 5 layers

1. *Convolutional Layer:*

   This layer is the first layer that is used to extract the various features from the input images. In this layer, the mathematical operation of convolution is performed between the input image and a filter of a particular size MxM. By sliding the filter over the input image, the dot product is taken between the filter and the parts of the input image with respect to the size of the filter (MxM).

   The output is termed as the Feature map which gives us the information about the image such as the corners and the edges. Later, this feature map is fed to other layers to learn several other features of the input image.

2. *Pooling layer:*

Pooling layer takes Convolutional layer output as input and this input image is considered as feature map. In most cases, a Convolutional Layer is followed by a Pooling Layer. The primary aim of this layer is to decrease the size of the convolved feature map to reduce the computational costs. This is performed by decreasing the connections between layers and independently operates on each feature map. Depending upon method used, there are several types of Pooling operations. In pooling layer, the size of the feature map is reduced. In order to reduce the size, there are three functions: Maxpooling, Averagepooling, Sumpooling. In Max Pooling, the largest element is taken from feature map. Average Pooling calculates the average of the elements in a predefined sized Image section. The total sum of the elements in the predefined section is computed in Sum Pooling. The Pooling Layer usually serves as a bridge between the Convolutional Layer and the FC Layer.

3. *Fully connected layer:*

Fully connected layer consists of weights and biases along with the neurons. In this stage the classification process starts. The input for this layer are the flattened images from the previous layer outputs. Fully connected layer acts as a bridge to connect neurons between two different layers. These layers are usually placed before the output layer and form the last few layers of a CNN Architecture. In this, the input image from the previous layers are flattened and fed to the FC layer. The flattened vector then undergoes few more FC layers where the mathematical functions operations usually take place. In this stage, the classification process begins to take place.

4. *Dropout:*

Dropout layer implements the dropping of neurons. Usually, when all the features are connected to the FC layer, it can cause overfitting in the training dataset. Overfitting occurs when a particular model works so well on the training data causing a negative impact in the model's performance when used on a new data.

To overcome this problem, a dropout layer is utilised wherein a few neurons are dropped from the neural network during training process resulting in reduced size of the model. On passing a dropout of 0.3, 30% of the nodes are dropped out randomly from the neural network.

5. *Activation layer:*

Activation layer is the most important layer in the CNN model. The Activation functions add non-linearity to the network. They are used to learn and

approximate any kind of continuous and complex relationship between variables of the network. In simple words, it decides which information of the model should fire in the forward direction and which ones should not at the end of the network. There are several commonly used activation functions such as the ReLU, Softmax, tanH and the Sigmoid functions. Each of these functions have a specific usage. For a binary classification CNN model, sigmoid and softmax functions are preferred an for a multi-class classification, generally softmax us used.

4.0.4 The architecture of our cnn model:

Model: SEQUENTIAL

Layers:

- Conv2D layer (filter=32, kernel_size=(5,5), activation="relu")
- Conv2D layer (filter=32, kernel_size=(5,5), activation="relu")
- MaxPool2D layer (pool_size=(2,2))
- Dropout layer (rate=0.25)
- Conv2D layer (filter=64, kernel_size=(3,3), activation="relu")
- Conv2D layer (filter=64, kernel_size=(3,3), activation="relu")
- MaxPool2D layer (pool_size=(2,2))
- Dropout layer (rate=0.25)
- Flatten layer to squeeze the layers into 1 dimension
- Dense Fully connected layer (256 nodes, activation="relu")
- Dropout layer (rate=0.5)
- Dense layer (43 nodes, activation="softmax")

4.0.5 Train the model:

Training the model we built is our next step. In order to train the model we use compiler and fit functions. During the training process, we tweak and change the weights of our model to try and minimize that loss function, and make our predictions as correct and optimized as possible. optimizers shape and mould your model into its most accurate possible form by futzing with the weights. The loss function is the guide to the terrain, telling the optimizer when it's moving in the right or wrong direction. Optimizers are related to model accuracy. Here we have implemented ADAM optimizer as it requires less memory to optimize large datasets. The results after training the model are saved in a file. As the results consists of large data , that are stored in h5 file which is special file. An H5 file is a data file saved in the Hierarchical Data Format (HDF). It contains multidimensional arrays of scientific data.

4.0.6 Testing the model:

For testing process, first we have to load the model. The class labels and their respective signs are stored in a dictionary. When the image is uploaded, the image is resized and converted into a numpy array. Now we compare these arrays with the model we have loaded, the output we get is the class label respective to the image we have uploaded. As we already mentioned the class labels and their respective signs, we will get the traffic sign respective to that class label.

0	speed limit(20kmph)	17	veh>3.5 ton prohibited	34	turn right ahead
1	speed limit(30kmph)	18	no entry	35	turn left ahead
2	speed limit(50kmph)	19	general caution	36	ahead only
3	speed limit(60kmph)	20	dangerous curve left	37	go straight or right
4	Speed limit(70kmph)	21	dangerous curve right	38	go straight or left
5	Speed limit(80kmph)	22	double curve	39	keep right
6	Endofspeedlimit(80km	23	bumpy road	40	keep left
7	speed limit(100kmph)	24	slippery road	41	roundabout mandatory
8	speed limit(120kmph)	25	road narrows on the rig	42	end of no passing
9	speed limit(120kmph)	26	road work	43	end no passing veh>3.5 ton
10	no passing	27	traffic signals		
11	no passing veh>3.5 ton	28	pedestrains		
12	right of way	29	children crossing		
13	priority road	30	bicycles crossing		
14	yield	31	beware of ice/snow		
15	stop	32	wild animals crossing		
16	no vehicles	33	end speed+assing limits		

Table 4.1 class labels and traffic signs

4.1 TOOLS USED

The tools used to implement algorithm are:

4.1.1 Keras: Keras library is used to build the model. The functions that are used to insert the five layers in cnn model are:

- *Convolutional layer:* To insert a convolutional layer , we use conv2d function which is present in keras library.

- *Pooling layer:* Here we are implementing maxpooling technique. Maxpooling is a pooling technique in which the largest size elements are reduced. Hence, to insert pooling layer we use Maxpool2d function.
- *Fully connected layer:* Dense function is used to insert a fully connected layer which is present in keras library.
- *Dropout layer:* Dropout function is used to insert the dropout layer which specifies a rate at which the neurons should be removed. The rate at which the neurons should be dropped out is between 0 an 1.
- *Activation layer:* Activation functions relu and softmax are used to insert the activation layer. Softmax activation functions gives results more accurate. Hence softmax activation function is implemented at output layer and relu activation function is implemented in all hidden layers.

***4.1.2 Pil*:** PIL is an image processing library in python. PIL has an Image function which is used to open the image specified in the particular path and is also used to resize the image. As the CNN algorithm takes only specific size of images. Hence, we have to resize all the images present in our dataset as well as whenever we upload an image for testing purpose, the image has to be resized. This total process takes place by using PIL library.

***4.1.3 Tkinter*:** Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.

4.2 COMPONENT DIAGRAM:

A component diagram shows a set of components and their relationships. A component is a physical and replaceable part of a system that conforms to and provides the realization of a set of interfaces. Every component must have a name that distinguishes it from other components.

Components represent the physical packaging of otherwise logical components and are at a different level of abstraction. A component is logically and physically cohesive and thus denotes a meaningful structural and behavioral chunk of a larger system and it may also be reused across many systems.

Graphically, a component diagram is a collection of vertices and arcs. Component diagrams commonly contain components, interfaces, dependency, generalization, association, and realization relationships, notes, constraints. Component diagrams may also contain packages or subsystems, both of which are used to group elements of the model into larger chunks.

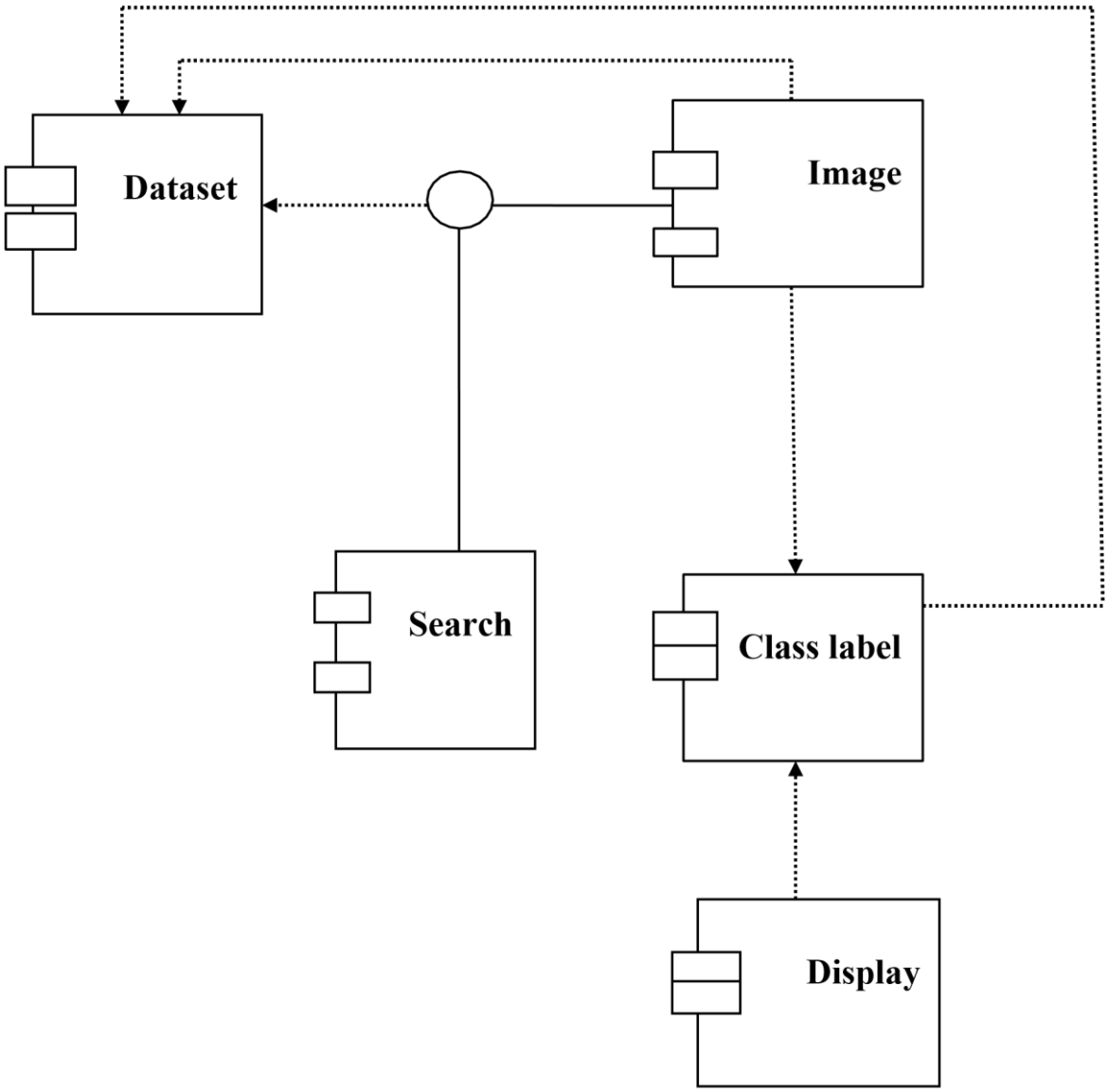


Fig 4.1 Component diagram

4.3 PSEUDOCODE/ALGORITHM

***Python code for exploring the dataset, building and training the cnn model*:**

```
import numpy as np import cv2 from PIL import Image import os from
sklearn.model_selection import train_test_split from tensorflow.keras.utils
import to_categorical from keras.models import Sequential, load_model from
keras.layers import Conv2D, MaxPool2D, Dense, Flatten, Dropout images =
[] labels = [] for i in range(43):
    path = os.path.join(os.getcwd(),'train',str(i))
    images_list = os.listdir(path) for a in
    images_list:
        try:
            image = Image.open(path + '\\'+ a)
            image = image.resize((30,30))
            image = np.array(image)
```

```
            images.append(image)
            labels.append(i)
        except:
            print("Error loading image")
images = np.array(images) labels = np.array(labels) image_train, image_test,
label_train, label_test = train_test_split(images, labels,
test_size=0.33,train_size=0.67,random_state=42) label_train =
to_categorical(label_train, 43) label_test = to_categorical(label_test, 43) model
= Sequential() model.add(Conv2D(filters=32, kernel_size=(5,5),
activation='relu', input_shape=image_train.shape[1:]))
model.add(Conv2D(filters=32, kernel_size=(5,5), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25)) model.add(Conv2D(filters=64, kernel_size=(3, 3),
activation='relu')) model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2))) model.add(Dropout(rate=0.25))
model.add(Flatten()) model.add(Dense(256, activation='relu'))
model.add(Dropout(rate=0.5)) model.add(Dense(43, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
epochs = 17 result =model.fit(image_train, label_train, batch_size=64, epochs=epochs,
validation_data=(image_test, label_test)) model.save("model.h5")
```

Python code for testing the model and implementing gui:

```
import tkinter as tk from tkinter
import filedialog from tkinter import
* from PIL import ImageTk, Image
import numpy from keras.models
import load_model model =
load_model('model.h5') classes = {
1:'Speed limit (20km/h)',
        2:'Speed limit (30km/h)',
        3:'Speed limit (50km/h)',
        4:'Speed limit (60km/h)',
        5:'Speed limit (70km/h)',
        6:'Speed limit (80km/h)',
        7:'End of speed limit (80km/h)',
```

8:'Speed limit (100km/h)', 9:'Speed
limit (120km/h)',
10:'No passing',
11:'No passing veh over 3.5 tons',
12:'Right-of-way at intersection',
13:'Priority road',
14:'Yield',
15:'Stop',
16:'No vehicles',
17:'Veh > 3.5 tons prohibited',
18:'No entry',
19:'General caution',
20:'Dangerous curve left',
21:'Dangerous curve right',
22:'Double curve',
23:'Bumpy road',
24:'Slippery road',
25:'Road narrows on the right',
26:'Road work',
27:'Traffic signals',
28:'Pedestrians',
29:'Children crossing',
30:'Bicycles crossing',
31:'Beware of ice/snow',
32:'Wild animals crossing',
33:'End speed + passing limits',
34:'Turn right ahead',
35:'Turn left ahead',
36:'Ahead only',
37:'Go straight or right',
38:'Go straight or left',
39:'Keep right',
40:'Keep left',
41:'Roundabout mandatory',
42:'End of no passing',

```
        43:'End no passing veh > 3.5 tons' }
top=tk.Tk() top.geometry('800x600')
top.title('Traffic sign Recognition')
label=Label(top, font=('arial',15,'bold'))
sign_image = Label(top) def
classify(file_path):
    image = Image.open(file_path) image =
    image.resize((30,30)) image =
    numpy.expand_dims(image, axis=0) image =
    numpy.array(image) pred =
    model.predict_classes([image])[0] sign =
    classes[pred+1] s="class label :
    "+str(pred+1) sig=" traffic sign : "+sign
    label.configure(text=s+sig)
def upload_image(): try:
        file_path=filedialog.askopenfilename()
        uploaded=Image.open(file_path)
        im=ImageTk.PhotoImage(uploaded)
        sign_image.configure(image=im)
        sign_image.image=im
        label.configure(text='') classify(file_path)
    except:
        pass
upload=Button(top,text="Upload sign",command=upload_image,padx=10,pady=0)
upload.configure(foreground='black',font=('arial',10,'bold'))
upload.pack(side=TOP,pady=5) sign_image.pack(side=TOP,expand=True)
label.pack(side=TOP,expand=True) top.mainloop()
```

4.4 DEPLOYMENT DIAGRAM

Deployment diagram is a diagram that shows the configuration of run time processing nodes and the components that live on them. Graphically, a deployment diagram is a collection of vertices and arcs. It uses nodes to model the topology of the hardware on which the system executes. A node typically represents a processor or a device on which

components may be deployed. Deployment diagrams commonly contain nodes, dependency and association relationships.

Deployment diagrams are used to visualize the topology of the physical components of a system, where the software components are deployed. Deployment diagrams are used for describing the hardware components, where software components are deployed.

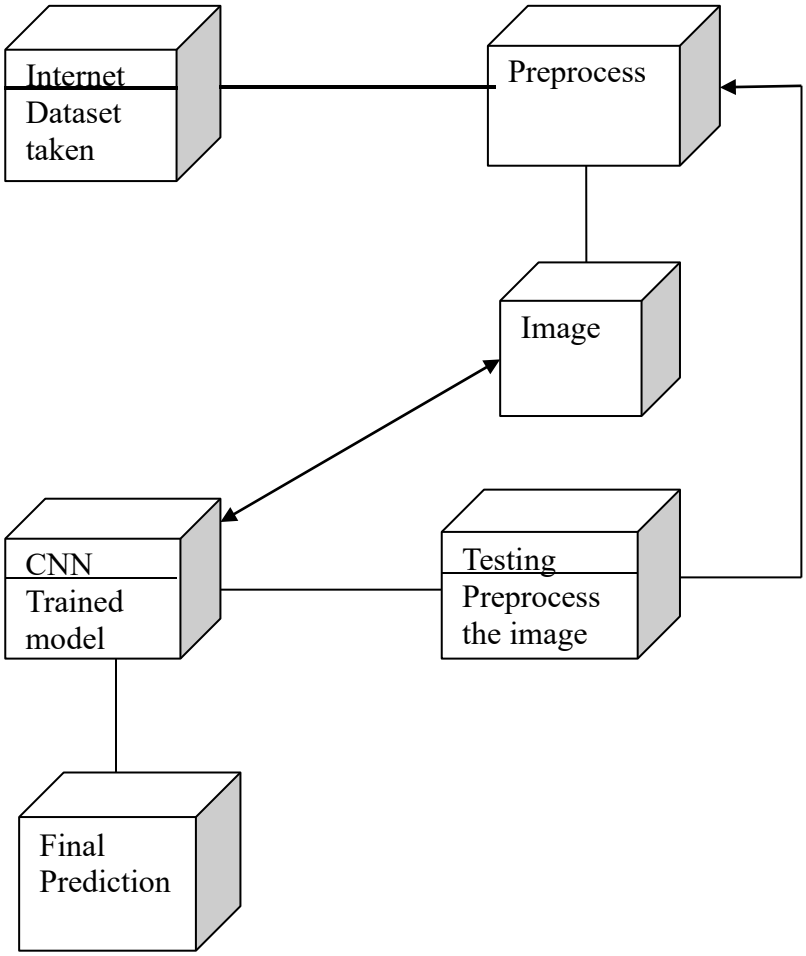


Fig 4.2 Deployment diagram

4.5 SCREENSHOTS

Epoch is equal to the number of times the algorithm sees the entire data set. So, each time the algorithm has seen all samples in the dataset, one epoch has completed.

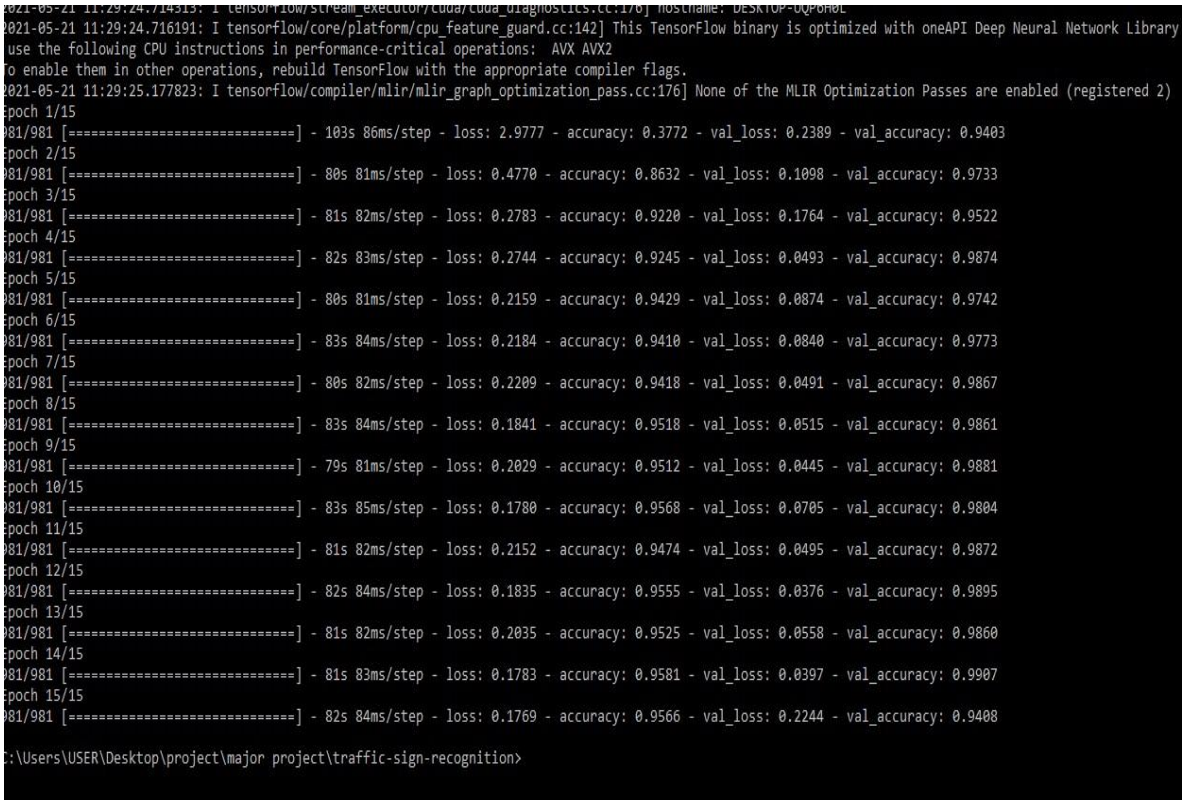


Fig4.3 Trained model for Epoch 15

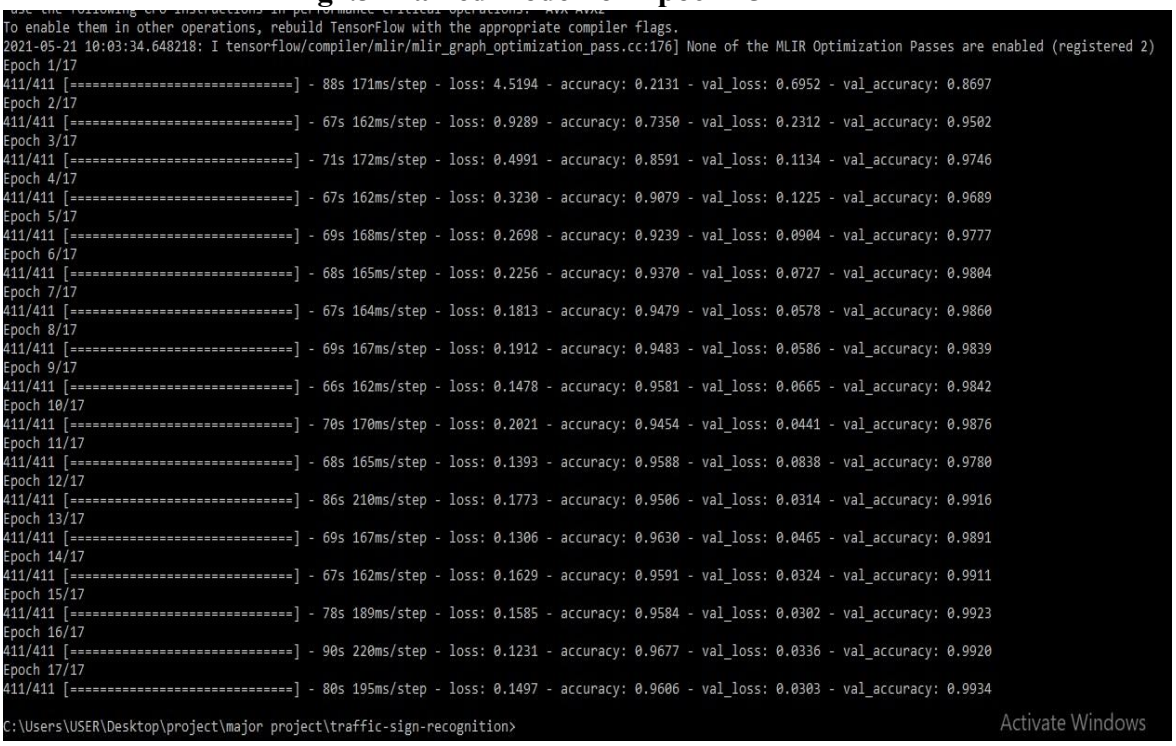
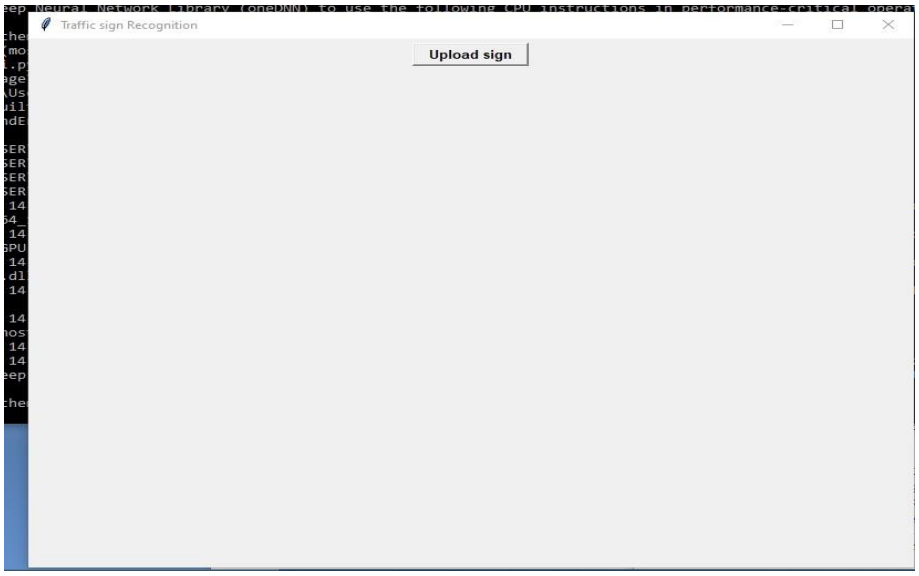


Fig4.4 Trained model for Epoch 17

CHAPTER – 5 TESTING

5.1 TEST CASES

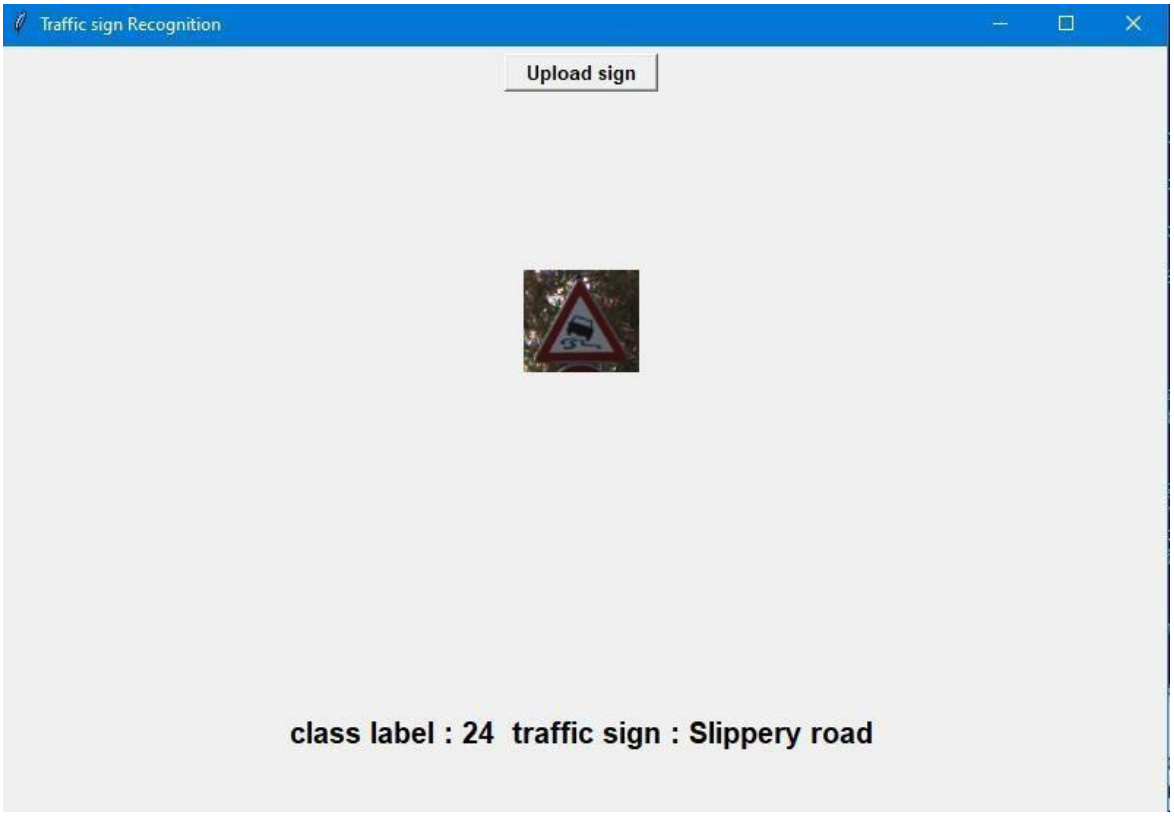
Test Case-1:


Traffic sign Recognition
Upload sign


Traffic sign Recognition
Upload sign
class label : 2 traffic sign : Speed limit (30km/h)

Test Case 2:


Traffic sign Recognition
Upload sign


Traffic sign Recognition
Upload sign
class label : 24 traffic sign : Slippery road

Test Case-3:

CHAPTER – 6 SUMMARY OR CONCLUSION

The main aim of this project is to implement an algorithm such that it can work on any system. Traffic sign recognition project is implemented by using Convolutional Neural Networks which is a deep learning algorithm. The CNN model is build by using keras library. The reason for using keras library is, it is portable i.e, it can work on any system rather than Pytorch. Pytorch can only be implemented on the system which support AMD gpu. Here, we

were taking German Traffic sign dataset and we are preprocessing all the images present in the dataset. We will split those images and class labels into two parts one part can be used for training the model and another part can be used for testing the model. And then we will build a cnn model using keras library functions and train the model with the data we have training dataset. For the testing purpose, we will upload an image and then the uploaded image is again preprocessed as the CNN algorithm accepts only specific size of inputs. The preprocessed input image is then predicted using the results of the trained model where we get the class label as the output. By using that class label we can predict the traffic sign.

BIBLIOGRAPHY

[1] Zhang, Z.J.; Li, W.Q.; Zhang, D.; Zhang, W. A review on recognition of traffic signs. In Proceedings of the 2014 International Conference on E-Commerce, E-Business and E-Service (EEE), Hong Kong, China, 1–2 May 2014; pp. 139–144.

[2] https://www.sciencedirect.com/topics/engineering/convolutional-neural-network

[3] Arunima Singh | Dr. Ashok Kumar Sahoo **"Traffic Sign Recognition**" Published in International Journal of Trend in Scientific Research and Development (ijtsrd), ISSN: 2456-6470, Volume-2 | Issue-4, June 2018, pp.122-126,
URL: https://www.ijtsrd.com/papers/ijtsrd12783.pdf

USER MANUAL

INTRODUCTION:

Traffic sign recognition plays a major role in the development of autonomous vehicles. As there is no human intervention in that technology, recognizing traffic signs is important in order to reduce the road accidents. Hence, we have build an algorithm using convolutional neural networks which is deep learning algorithm which recognizes the traffic sign once the detection stage is completed.

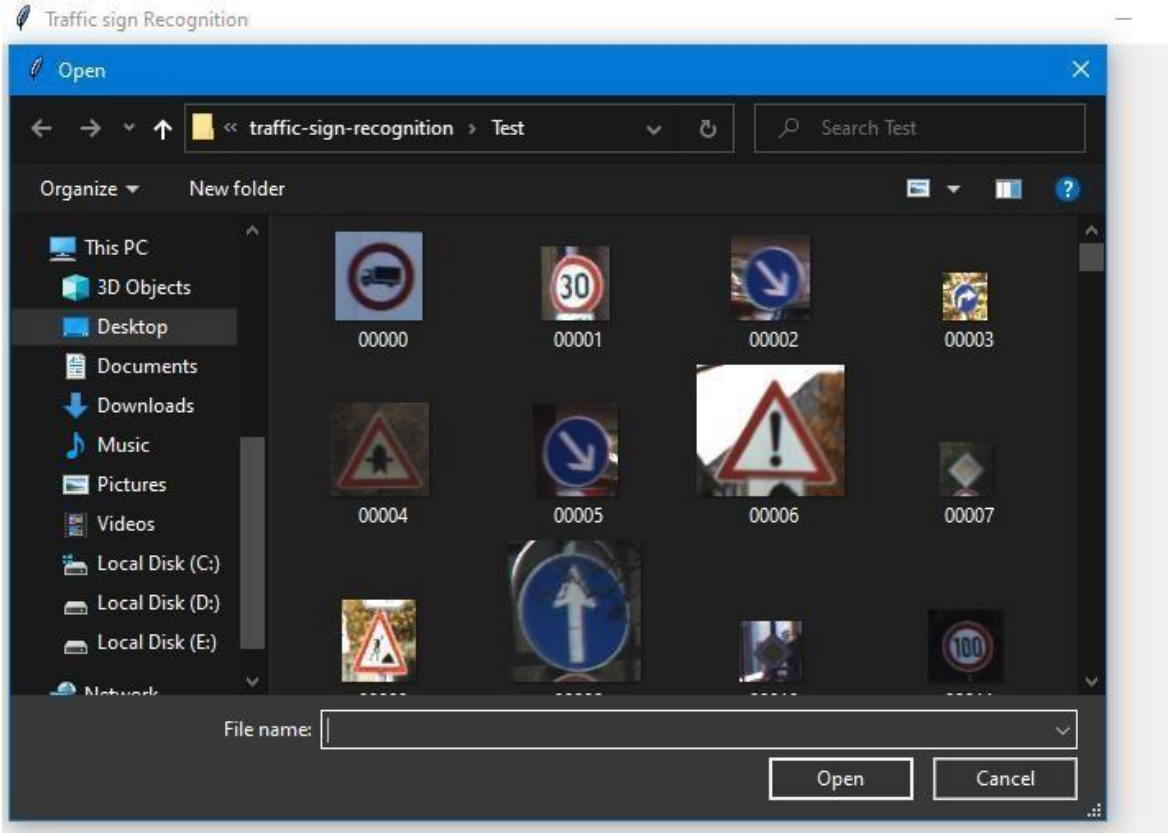
REQUIREMENTS:

A dataset is required which contains numerous traffic sign images. We were considering German Traffic Sign dataset as it contains almost all the traffic sign images. A image is required to recognize the traffic.

PROCEDURE:

Step1:In order to recognize the sign, first we have to upload an image.

Step2: After clicking the upload button, it will open a dialog box to select an image for recognition.


Traffic sign Recognition
Open
traffic-sign-recognition › Test
Search Test
Organize
New folder
This PC
3D Objects
Desktop
Documents
Downloads
Music
Pictures
Videos
Local Disk (C:)
Local Disk (D:)
Local Disk (E:)
00000
00001
00002
00003
00004
00005
00006
00007
File name:
Open
Cancel

Step3: After opening the image and after successful execution of algorithm in backend we get the result.