A Project Report

on

Real Time Blind Assistance System using Machine Learning

Submitted in partial fulfillment of the requirements for the award of the Degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

by

D.GAYATHRI (19FE1A0532)

D.LAKSHMAN (19FE1A0534)

A.JAYA PRAKASH (19FE1A0507)

D.DINESH (19FE1A0531)

Under the guidance of

Dr.M.Vanitha , Ph.D

Professor

Department of Computer Science and Engineering

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VIGNAN'S LARA INSTITUTE OF TECHNOLOGY & SCIENCE

(Affiliated to Jawaharlal Nehru Technological University Kakinada, Kakinada)

(An ISO 9001:2015 Certified Institution, Approved by AICTE)

Vadlamudi, Guntur Dist., Andhra Pradesh-522213

March-2023.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VIGNAN'S LARA INSTITUTE OF TECHNOLOGY & SCIENCE

(Affiliated to Jawaharlal Nehru Technological University Kakinada, Kakinada)

(An ISO 9001:2015 Certified Institution, Approved by AICTE)

Vadlamudi, Guntur Dist, Andhra Pradesh-522213.

CERTIFICATE

This is to certify that the project report entitled "**Real Time Blind Assistance System using Machine Learning**" is a bona- fide work done by **D.GAYATHRI (19FE1A0532), D.LAKSHMAN (19FE1A0534), A. JAYA PRAKASH (19FE1A0507), D.DINESH (19FE1A0531)** under my guidance and submitted in fulfillment of the requirements for the award of the degree of Bachelor of Technology in **COMPUTER SCIENCE AND ENGINEERING** from **JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY KAKINADA, KAKINADA**. The work embodied in this project report is not submitted to any University or Institution for the award of any Degree or diploma.

Project Guide
Dr.M.VANITHA , Ph,D
Professor

Head of the Department
Dr. K. Venkateswara Rao, Ph.D.
Professor

External Examiner

DECLARATION

We hereby declare that the project report entitled "**Real Time Blind Assistance System using Machine Learning**" is a record of an original work done by us under the guidance of **Dr.M.Vanitha**, Professor of Computer Science and Engineering and this project report is submitted in the fulfilment of the requirements for the award of the Degree of Bachelor of Technology in Computer Science and Engineering. The results embodied in this project report are not submitted to any other University or Institute for the award of any Degree or Diploma.

	Project Members		**Signature**
Place: Vadlamudi	**D.GAYATHRI**	(19FE1A0532)	_____
Date:	**D.LAKSHMAN**	(19FE1A0534)	_____
	A.JAYA PRAKASH	(19FE1A0507)	_____
	D.DINESH	(19FE1A0531)	_____

ACKNOWLEDGMENT

The satisfaction that accompanies with the successful completion of any task would be incomplete without the mention of people whose ceaseless cooperation made it possible, whose constant guidance and encouragement crown all efforts with success.

We are grateful to **Dr. M.VANITHA**, Professor, Department of Computer Science and Engineering for guiding through this project and for encouraging right from the beginning of the project till successful completion of the project. Every interaction with him was an inspiration.

We thank **Dr. K. VENKATESWARA RAO**, Professor & HOD, Department of Computer Science and Engineering for support and Valuable suggestions.

We also express our thanks to **Dr. K. PHANEENDRA KUMAR**, Principal, Vignan's Lara Institute of Technology & Science for providing the resources to carry out the project.

We also express our sincere thanks to our beloved **Chairman Dr. LAVU RATHAIAH** for providing support and stimulating environment for developing the project.

We also place our floral gratitude to all other teaching and lab technicians for their constant support and advice throughout the project.

Project Members

D.GAYATHRI (19FE1A0532)

D.LAKSHMAN (19FE1A0534)

A.JAYA PRAKASH (19FE1A0507)

D.DINESH (19FE1A0531)

LIST OF CONTENTS

ABSTRACT

Visually impaired people face a lot of problems in their daily-life. Eye diseases usually cause blindness and visual impairment. According to the World Health Organization, around 40 million people are blind, while another 250 million have some form of visual impairment. They come across many troubles in their daily life, especially while navigating from one place to another on their own. They often depend on others for help to satisfy their day-to-day needs. But sometimes they will not be available as the time because they will be busy with their works. At that time living becomes difficult for them. So, it is quite a challenging task to implement a technological solution to assist them. So, in order to overcome these challenges several technologies have been developed for the assistance of visually impaired people. Among the various technologies being utilized to assist the blind, computer vision-based solutions are emerging as one of the most promising options due to their affordability and accessibility. One such attempt is that we would wish to make an Integrated Machine Learning System that allows the blind victims to identify and classify real-time objects generating voice feedback and distance. Which also produces warnings whether they are very close or far away from the thing. In the proposed system it will capture the image. After capturing it will classify what type of object it is. Then, it calculates the distance between object and user. After that it will acknowledge user about the object and distance. Keywords: Blindness, Visual impairment, Machine Learning, Real-time objects.

LIST OF FIGURES

LIST OF ABBREVATIONS

Sl. No	ABBREVATION	FULL FORM
1	SSD	Single Shot Detection
2	CNN	Convolutional Neural Network
3	K-NN	K-Nearest Neighbors
4	ML	Machine Learning
5	SVM	Support Vector Machine
6	ANN	Artificial Neural Network
7	LSTM	Long Short-Term Memory
8	RNN	Recurrent Neural Network
9	YOLO	You Only Look Once
10	R-CNN	Region-based Convolutional Neural Network

CHAPTER 1

INTRODUCTION

Visual impairment is a significant challenge for the blind, and it affects their daily life activities. According to the World Health Organization (WHO), there are approximately 285 million people with visual impairment worldwide, and 39 million of them are blind. Blindness can severely impact a person's quality of life and independence, making them dependent on others for assistance in carrying out their daily activities.

In the past, the visually impaired had to rely on canes, guide dogs, or the assistance of a sighted person to navigate their environment. However, with the recent advancement in technology, real-time blind assistance systems have been developed to aid the visually impaired. These systems use various technologies such as sensors, machine learning algorithms, and audio feedback to enable the visually impaired to navigate their environment safely and independently.

Machine learning has played a crucial role in the development of these systems. Machine learning algorithms enable the system to recognize and classify objects, obstacles, and hazards in real-time. These algorithms analyze the images captured by the sensors and identify the objects present in the environment. Once the objects in the environment have been identified, the system generates an audio feedback that describes the objects to the user. The feedback also provides the user with information on the distance between the object and the user.

One of the critical components of the real-time blind assistance system using machine learning is object detection and recognition. Object detection and recognition algorithms are essential for detecting and recognizing objects in the environment. These algorithms use various techniques such as image segmentation, feature extraction, and Machine Learning to analyze images and identify objects in real-time.

Another essential component of the real-time blind assistance system using machine learning is audio feedback. The audio feedback provides users with information about the objects in their environment. The feedback is generated in real-time and is based on the objects detected by the system. The audio feedback can also provide users with information on the distance between the object and the user, which helps them navigate their environment safely.

The real-time blind assistance system using machine learning provides several benefits to the visually impaired. One of the significant benefits is increased independence. The system allows the visually impaired to navigate their environment independently and with confidence. This can significantly improve their quality of life and enhance their sense of independence.

Another benefit of the real-time blind assistance system using machine learning is improved safety. The system provides real-time feedback on objects, obstacles, and hazards in the environment, reducing the risk of accidents and injuries. This can help the visually impaired to navigate their environment safely and avoid potential hazards.

Finally, the real-time blind assistance system using machine learning can also enhance the quality of life of the visually impaired. The system enables the visually impaired to perform their daily activities, such as shopping and traveling, with ease. This can significantly improve their sense of well-being and increase their overall quality of life.

CLASSIFICATION

Classification is a technique where we categorize data into a given number of classes. The main goal of a classification problem is to identify the category/class to which a new data will fall under. Image Classification is the task of assigning an input image one label from a fixed set of categories. This is one of the core problems in Computer Vision that, despite its simplicity, has a large variety of practical applications. Many other seemingly distinct Computer Vision tasks (such as object detection, segmentation) can be reduced to image classification.

TERMINOLOGIES IN CLASSIFICATION

- **Classifier:** An algorithm that maps the input data to a specific category.
- **Classification model:** A Classification model tries to draw some conclusion from the input values given for training. It will predict the class labels/categories for the new data.
- **Feature:** A feature is an individual measurable property of a phenomenon being observed.
- **Binary Classification:** Classification task with two possible outcomes. E.g., Gender Classification (Male/Female).
- **Multi class classification**: Classification with more than two classes. In multi class classification each sample is assigned to one and only one target label. E.g., An animal can be cat or dog but not both at the same time.
- **Multi-label classification:** Classification task where each sample is mapped to a set of target labels (more than one class). E.g., A news article can be about sports, a person, and location at the same time.

The following are the steps involved in building a classification model:

- Initialize: To initialize the classifier to be used.
- Train the classifier: All classifiers in scikit-learn uses a fit (X, y) method to fit the model (training) for the given train data X and train label y.
- Predict the target: Given an unlabelled observation X, the predict (X) returns the predicted label y.
- Evaluate the classifier model.

CLASSIFICATION ALGORITHMS

The most used classification algorithms along with the python code: Logistic Regression, Naïve Bayes, Convolution Neural Network, K-Nearest Neighbours, Decision Tree, Random Forest, and Support Vector Machine.

LOGISTIC REGRESSION

Logistic regression is a popular statistical method used to analyze the relationship between a binary outcome variable (i.e., a variable with two possible values) and one or more predictor variables. It is commonly used in many fields, including healthcare, marketing, and social sciences.

The goal of logistic regression is to determine the relationship between the predictor variables and the probability of the binary outcome variable. Logistic regression produces a logistic function, also known as a sigmoid function, which is used to estimate the probability of the outcome variable.

The logistic function has an S-shape, where the output ranges from 0 to 1. The logistic regression model estimates the coefficients for each predictor variable to determine its effect on the outcome variable. The coefficients represent the increase or decrease in the log odds of the outcome variable for a one-unit increase in the predictor variable.

The logistic regression model uses maximum likelihood estimation to estimate the coefficients of the predictor variables. The model estimates the probability of the outcome variable based on the predictor variables and then compares it to the actual observed outcome. The model adjusts the coefficients to improve the fit between the estimated and actual probabilities.

NAÏVE BAYES

Naive Bayes is a probabilistic machine learning algorithm that is commonly used for classification tasks, such as text classification and spam filtering. The algorithm is based on Bayes' theorem, which is a statistical theorem that describes the probability of an event, based on prior knowledge of conditions that might be related to the event.

The "Naive" in Naive Bayes refers to the assumption that the predictors are independent of each other. This assumption simplifies the calculation of the posterior probabilities, which are the probabilities of the outcomes given the input features.

The Naive Bayes algorithm works by first calculating the prior probabilities of each class based on the training data. Then, for a given set of input features, it calculates the posterior probabilities of each class using Bayes' theorem. The class with the highest probability is then assigned as the predicted class for the input features.

The algorithm is called "Naive" because it assumes that the predictors are independent, which is often not true in real-world data. Despite this simplifying assumption, Naive Bayes is known to be a powerful and efficient algorithm that can perform well even with limited training data.

K-NEAREST NEIGHBOURS

K-Nearest Neighbors (KNN) is a machine learning algorithm used for both classification and regression tasks. The algorithm is based on the principle that similar data points are likely to have similar outcomes.

In the KNN algorithm, the k nearest data points to the input data point are identified from the training data based on their similarity to the input data point. The similarity measure used can vary based on the type of data and the problem being solved. For example, the Euclidean distance metric is commonly used for continuous data, while the Hamming distance metric is used for categorical data.

Once the k nearest data points have been identified, the predicted outcome for the input data point is determined by taking the majority vote (in classification) or the average value (in regression) of the outcomes of the k nearest data points.

CONVOLUTION NEURAL NETWORKS

Convolutional Neural Networks (CNNs) are a type of deep learning model commonly used for image and video processing. CNNs are designed to process input data with grid-like topology, such as a 2D image, and extract features from it through the use of convolutional layers.

Convolutional layers are the core building blocks of CNNs. They consist of a set of learnable filters that are applied to the input image to extract specific features. Each filter is convolved across the entire image, creating a feature map that represents the response of that filter at each location. The filters are learned during the training process, allowing the network to learn relevant features from raw input data without manual feature engineering.

Pooling layers are another important component of CNNs. They are used to downsample the output of the convolutional layers, reducing the spatial resolution of the feature maps while retaining the most salient information. Common types of pooling layers include max pooling, average pooling, and global pooling.

The output of the convolutional and pooling layers is then fed into a fully connected layer, which is a traditional neural network layer that connects every input to every output. The fully connected layer is used to classify the input image or perform some other task based on the extracted features.

SUPPORT VECTOR MACHINE

SVM, or Support Vector Machine, is a supervised learning algorithm used for classification and regression analysis. SVMs are based on the idea of finding the hyperplane that best separates the input data into different classes. In the case of binary classification, an SVM attempts to find the hyperplane that maximizes the margin between the two classes, where the margin is the distance between the hyperplane and the closest data points of each class. The hyperplane is defined by a linear combination of the input features, where the weights are learned during the training process.

SVMs can also handle non-linearly separable data by using a kernel trick. The kernel function maps the input data to a higher-dimensional feature space where it is more likely to be linearly separable. This allows SVMs to find non-linear decision boundaries between classes. SVMs have several advantages over other classification algorithms, including high accuracy, robustness to outliers, and the ability to handle high-dimensional data. They have been successfully applied in a wide range of domains, including image recognition, text classification, and bioinformatics. One potential drawback of SVMs is their computational complexity, especially when dealing with large datasets. However, this can be mitigated by using efficient optimization techniques and parallel computing.

REGRESSION

Regression is a supervised learning technique used to predict continuous numerical values based on input features. It is commonly used for forecasting, prediction, and modeling in a variety of fields, such as finance, economics, engineering, and science.

In regression, the goal is to find a function that maps the input features to a numerical output value. The function is typically represented by a mathematical model, such as linear regression, polynomial regression, or non-linear regression. The model is trained using a set of labeled examples, where each example consists of input features and a corresponding output value.

During the training process, the model is optimized to minimize the difference between the predicted output value and the true output value. This is typically done by minimizing a cost function, such as mean squared error or mean absolute error. Once the model is trained, it can be used to make predictions on new, unseen data.Regression can be used for both simple and complex modeling tasks, from fitting a straight line to a set of data points to modeling complex, non-linear relationships between input features and output values. Some common applications of regression include predicting stock prices, estimating housing prices, and forecasting sales.

TERMINOLOGIES IN REGRESSION

1. Input features or predictors: These are the variables that are used to predict the output value in a regression model. Input features can be continuous, discrete, or categorical variables. For example, in a housing price prediction model, the input features could include the number of bedrooms, the size of the house, and the location.

2. Output or response variable: This is the variable that is being predicted by the regression model. It is also known as the dependent variable. In a housing price prediction model, the output variable would be the predicted price of the house.

3. Training data: This is the labeled dataset that is used to train the regression model. The training data consists of input features and corresponding output values. The model learns from this data and adjusts its parameters to minimize the difference between the predicted output values and the true output values.

4. Test data: This is the dataset that is used to evaluate the performance of the trained regression model. The test data consists of input features but does not include the output values. The model makes predictions on the test data, and the performance is evaluated by comparing the predicted values with the true values.

5. Model parameters: These are the coefficients or weights assigned to the input features in the regression model. The model parameters are learned during the training process and are used to make predictions on new data.

6. Residuals: These are the differences between the predicted output values and the true output values in the training data. The goal of regression is to minimize the residuals and achieve the best possible fit. Residual analysis can be used to check the assumptions of the regression model.

7. Overfitting: This occurs when the regression model is too complex and fits the training data too closely, resulting in poor generalization to new data. Overfitting can be avoided by using regularization techniques, such as L1 or L2 regularization, or by using a simpler model.

8. Underfitting: This occurs when the regression model is too simple and fails to capture the underlying relationship between the input features and output values. Underfitting can be avoided by using a more complex model or by adding more input features to the model.

9. Dependent Variable: The main factor in Regression analysis that we want to predict or understand is called the dependent variable. It is also called the target variable.

10. Independent Variable: The factors which affect the dependent variables, or which are used to predict the values of the dependent variables are called the independent variable, also called a predictor.

11. Outliers: Outlier is an observation that contains either a very low value or a very high value in comparison to other observed values. An outlier may hamper the result, so it should be avoided.

12. Multicollinearity: If the independent variables are more highly correlated with each other than other variables, then a such condition is called Multicollinearity. It should not be present in the dataset, because it creates problems while ranking the most affecting variable.

13. Underfitting and Overfitting: If our algorithm works well with the training dataset but not well with the test dataset, then such a problem is called Overfitting. And if our algorithm does not perform well even with the training dataset, then such a problem iscalled underfitting.

REGRESSION TECHNIQUES

LINEAR REGRESSION

It is used for predictive analysis. Linear regression is a linear approach for modelling the relationship between the criterion or the scalar response and the multiple predictors or explanatory variables. Linear regression focuses on the conditional probability distribution of the response given the values of the predictors. For linear regression, there is a danger of overfitting. The formula for linear regression is: $Y' = bX + A$.

POLYNOMIAL REGRESSION

It is used for curvilinear data. Polynomial regression is fits with the method of least squares. The goal of regression analysis is to model the expected value of a dependent variable y in regards to the independent variable x. The equation for polynomial regression also derived from linear regression equation that means Linear regression equation $Y= b_0+ b_1x$, is transformed into Polynomial regression equation $Y= b_0+b_1x+ b_2x^2+ b_3x^3+ + b_nx^n$.

Here Y is the predicted/target output, b_0, b_1,... b_n are the regression coefficients. x is our independent/input variable. The model is still linear as the coefficients are still linear with quadratic.

NEURAL NETWORK REGRESSION

You all must be aware of the power of neural networks in making predictions/assumptions. Each node in a neural network has a respective activation function that defines the output of the node based on a set of inputs. The last activation function can be manipulated to change a neural network into a regression model. One can use 'Keras' that is the appropriate python library for building neural networks in ML.The output of a neuron is mapped to a variety of values in neural network regression,thus ensuring non-linearity. You can choose a single parameter or a range of parameters for predicting output using neural network regression. The neurons (outputs of a neural network are well-connected with each other, along with a weight associated with each neuron. The well-connected neurons help in

predicting future values along with mapping a relationship between dependent and independent variables.

DECISION TREE REGRESSION

Non-linear regression in Machine Learning can be done with the help of decision tree regression. The main function of the decision tree regression algorithm is to split the dataset into smaller sets. The subsets of the dataset are created to plot the value of any data point that connects to the problem statement. The splitting of the data set by this algorithm results in a decision tree that has decision and leaf nodes. ML experts prefer this model in cases where there is not enough change in the data set. One should know that even a slight change in the data can cause a major change in the structure of the subsequent decision tree. One should also not prune the decision tree regressors too much as there will not be enough end nodes left to make the prediction. To have multiple end nodes (regression output values), one should not prune the decision tree regressors excessively.

SUPPORT VECTOR MACHINE (SVM)

SVM can be placed under both linear and non-linear types of regression in ML. The use cases of SVM can range from image processing and segmentation, predicting stock market patterns, text categorization, etc. When you have to identify the output in a multidimensional space, the SVM algorithm is used. In a multidimensional space, the data points are not represented as a point in a 2D plot. The data points are represented as a vector in a multidimensional space. A max-margin hyperplane is created under this model that separates the classes and assigns a value to each class. Freshers should know that an SVM model does not perform to its fullest extent when the dataset has more noise.

MACHINE LEARNING

Machine Learning technology is based on artificial neural networks (ANNs). These ANNs constantly receive learning algorithms and continuously growing amounts of data to increase the efficiency of training processes. The larger data volumes are, the more efficient this process is. The training process is called deep, because, with the time passing, a neural network covers growing number of levels. The deeper this network penetrates, the higher its productivity is. DL algorithms can create new tasks to solve current ones.

Advantages of Machine Learning

Creating New Features One of the main benefits of Machine Learning over various machine learning algorithms is its ability to generate new features from a limited series of features located in the training dataset. Therefore, Machine Learning algorithms can create

new tasks to solve current ones. What does it mean for data scientists working in technological startups? Since Machine Learning can create features without a human intervention, data scientistscan save much time on working with big data and relying on this technology. It allows them to use more complex sets of features in comparison with traditional machine learning software.

Advanced Analysis

Due to its improved data processing models, Machine Learning generates actionable results when solving data science tasks. While machine learning works only with labelled data, Machine Learning supports unsupervised learning techniques that allow the system to become smarter on its own. The capacity to determine the most important features allows Machine Learning to efficiently provide data scientists with concise and reliable analysis results.

MACHINE LEARNING CHALLENGES

Machine Learning is an approach that models human abstract thinking (or at least represents an attempt to approach it) rather than using it. However, this technology has a set of significant disadvantages despite all its benefits.

Continuous Input Data Management

In Machine Learning, a training process is based on analyzing large amounts of data. Although, fast- moving and streaming input data provide little time for ensuring an efficient training process. That is why data scientists have to adapt their Machine Learning algorithms in the way neural networks can handle large amounts of continuous input data.

Ensuring Conclusion Transparency

Another important disadvantage of Machine Learning software is that it is incapable of providing arguments why it has reached a certain conclusion. Unlike in case of traditional machine learning, you cannot follow an algorithm to find out why your system has decided that it is a cat on a picture, not a dog. To correct errors in DL algorithms, you have to revise the whole algorithm.

Resource-Demanding Technology

Machine Learning is a quite resource-demanding technology. It requires more powerful GPUs, high-performance graphics processing units, large amounts of storage to train the models, etc. Furthermore, this technology needs more time to train in comparison withtraditional machine learning.

Lack of flexibility

Despite the occasional warnings of AI taking over the world, Machine Learning algorithms are pretty simple in their nature. In order to solve a given problem, a Machine Learning network needs to be provided with data describing that specific problem, thus rendering the algorithm ineffective to solve any other problems. This is true no matter how similar they are to the original problem.

APPLICATIONS OF MACHINE LEARNING

Automatic speech recognition

Large-scale automatic speech recognition is the first and most convincing successful case of Machine Learning. LSTM RNNs can learn "Very Machine Learning" tasks that involve multi- second intervals containing speech events separated by thousands of discrete time steps, where one- time step corresponds to about 10 milliseconds. LSTM with forget gates is competitive with traditional speech recognizers on certain tasks. The initial success in speech recognition was based on small-scale recognition tasks base on TIMIT. The data set contains 630 speakers from eight major dialects of American English, where each speaker reads 10 sentences. Its small size lets many configurations be tried. More importantly, the TIMIT task concerns phone-sequence recognition, which, unlike word-sequence recognition, allows weakphone bigram language models. This lets the strength of the acoustic modelling aspects of speech recognition be more easily analysed.

Image recognition

A common evaluation set for image classification is the MNIST database data set. MNIST is composed of handwritten digits and includes 60,000 training examples and 10,000 test examples. As with TIMIT, its small size lets users test multiple configurations. A comprehensive list of results on this set is available. Machine Learning- based image recognition has become "superhuman", producing more accurate results than human contestants. Thisfirst occurred in 2011.

Visual art processing

Closely related to the progress that has been made in image recognition is the increasing application of Machine Learning techniques to various visual art tasks. DNNs have proven themselves capable, for example, of

- Identifying the style period of a given painting.
- Neural Style Transfer - capturing the style of a given artwork and applying it in a

visually pleasing manner to an arbitrary photograph or video, and

- Generating striking imagery based on random visual input fields.

Military

The United States Department of Defence applied Machine Learning to train robots in new tasksthrough observation.

Bioinformatics

An auto encoder ANN was used in bioinformatics, to predict gene ontology annotations and gene-function relationships. In medical informatics, Machine Learning was used to predict sleep quality based on data from wearables and predictions of health complications from electronic health record data. Machine Learning has also showed efficacy in healthcare. DL is used mostly for the prediction of the sleep quality based on data from the wearables of the people. It can also show the various health issues of the humans based on the electronic health record data.

Self-Driving cars

Machine Learning is the force that is bringing autonomous driving to life. A million sets of data are fed to a system to build a model, to train the machines to learn, and then test the results in a safe environment. Data from cameras, sensors, geo-mapping is helping create succinct and sophisticated models to navigate through traffic, identify paths, signage, pedestrian-only routes, and real-time elements like traffic volume and road blockages.

Fraud-Detection

Another domain benefitting from Machine Learning is the banking and financial sector that is plagued with the task of fraud detection with money transactions going digital. Auto encoders in Keras and TensorFlow are being developed to detect credit card frauds saving billions of dollars of cost in recovery and insurance for financial institutions. Fraud

SSD Algorithm:

SSD (Single Shot Detector) is an object detection algorithm used in computer vision that can quickly detect objects in images or videos in real-time. The SSD algorithm was developed by Wei Liu, et al. in 2016, and it is based on a deep neural network architecture.

The SSD algorithm works by using a single convolutional neural network (CNN) to simultaneously predict object locations and classify them in the input image. Unlike some other object detection algorithms, the SSD algorithm does not rely on region proposals or sliding windows. Instead, it divides the input image into a grid of smaller cells, and for each cell, the algorithm predicts a set of bounding boxes that contain objects and their corresponding class scores.

The SSD algorithm uses a base network (usually a pre-trained network like VGG16 or ResNet) to

extract features from the input image. The feature maps obtained from the base network are then processed by a series of convolution layers, each responsible for predicting bounding boxes and class scores at a specific scale.

At each scale, the SSD algorithm generates a fixed set of bounding boxes with different aspect ratios and sizes, called anchor boxes. These anchor boxes are used to detect objects of different sizes and shapes in the image.

The SSD algorithm uses two loss functions during training: a localization loss and a classification loss. The localization loss measures the difference between the predicted and ground truth bounding boxes, while the classification loss measures the difference between the predicted and ground truth class scores. These two loss functions are combined to form a single loss function that is used to train the SSD algorithm.

The SSD algorithm is a deep neural network-based object detection algorithm that can detect objects in real-time without relying on region proposals or sliding windows. It divides the input image into a grid of smaller cells and uses a set of fixed anchor boxes to detect objects of different sizes and shapes. The algorithm uses two loss functions during training to learn to predict accurate bounding boxes and class scores.

The SSD (Single Shot Detector) algorithm is composed of several key components that work together to enable fast and accurate object detection in images. The main components of the SSD algorithm are:

Base Network: The base network is typically a pre-trained deep convolution neural network (CNN) such as VGG16 or ResNet. The role of the base network is to extract features from the input image, which are then used by the rest of the network to make predictions.

Feature Pyramid: The feature pyramid is a set of feature maps that are generated by the base network at different scales. These feature maps are used to detect objects of different sizes and aspect ratios in the input image.

Convolution Layers: The convolution layers are responsible for predicting the locations and class scores of objects in the input image. These layers take the feature maps from the feature pyramid as input and produce a set of bounding boxes and corresponding class scores for each location in the feature maps.

Anchor Boxes: The anchor boxes are a set of pre-defined bounding boxes that are used as templates for detecting objects of different sizes and aspect ratios in the input image. The SSD algorithm generates a fixed set of anchor boxes at each location in the feature maps.

The SSD (Single Shot Detector) algorithm is composed of several layers that work together to detect objects in images. The layers of the SSD algorithm can be divided into two main categories:

1. The base network layers
2. The detection layers.

Base Network Layers: The base network layers are usually pre-trained convolution neural network (CNN) layers, such as VGG16 or ResNet, that are used to extract feature maps from the input image. These layers are responsible for learning hierarchical features that are useful for object detection. The

base network layers typically include convolution layers, pooling layers, and activation functions such as ReLU.

Detection Layers: The detection layers of the SSD algorithm are responsible for predicting the locations and class scores of objects in the input image. The detection layers consist of a set of convolution layers, each of which predicts a fixed number of bounding boxes and class scores for objects in the input image. The detection layers also include anchor boxes, which are used to generate the predicted bounding boxes. The detection layers use the feature maps generated by the base network layers as input.

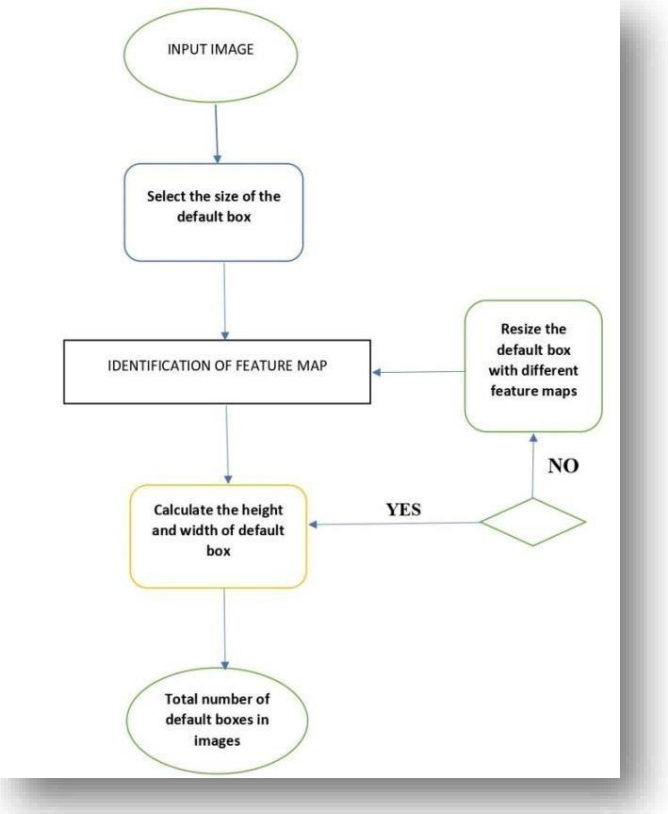

Fig 1.6.1 : Process of counting all default boxes in a system

The SSD (Single Shot Detector) algorithm can be implemented using popular deep learning frameworks such as TensorFlow, PyTorch, and Caffe. Here is a general overview of the steps involved in implementing the SSD algorithm:

Dataset Preparation: The first step in implementing the SSD algorithm is to prepare the dataset. The dataset should contain annotated images that include bounding box annotations for the objects of interest. The annotations should include the class label, the coordinates of the bounding box, and the confidence score.

Base Network: The second step is to create the base network using a pre-trained CNN such as VGG16 or ResNet. The base network is responsible for extracting feature maps from the input image. The pre-trained CNN can be loaded into the framework using pre-trained weights.

Feature Pyramid: The third step is to create the feature pyramid by adding convolutional layers on top

of the base network. The feature pyramid is used to detect objects of different sizes and aspect ratios in the input image.

Anchor Boxes: The fourth step is to define the anchor boxes. The anchor boxes are a set of pre-defined bounding boxes that are used as templates for detecting objects of different sizes and aspect ratios in the input image. The SSD algorithm generates a fixed set of anchor boxes at each location in the feature maps.

Detection Layers: The fifth step is to create the detection layers. The detection layers are a set of convolution layers that are attached to the top of each layer in the feature pyramid. Each detection layer predicts a fixed number of bounding boxes and class scores for objects in the input image.

Loss Function: The sixth step is to define the loss function. The loss function is used to train the SSD algorithm. The loss function is composed of two components: a localization loss and a classification loss.

Training: The seventh step is to train the SSD algorithm using the prepared dataset. During training, the weights of the network are adjusted to minimize the loss function.

Inference: The final step is to use the trained SSD algorithm to detect objects in new images. The SSD algorithm takes an input image, generates feature maps using the base network and the feature pyramid, and predicts the locations and class scores of objects using the detection layers.

OBJECT RECOGNIZATION

Object detection is a challenging problem in computer vision, as it requires identifying objects of various shapes, sizes, and orientations, often within cluttered and complex scenes. There are different approaches to object detection, including classical methods based on hand-crafted features and more recent deep learning-based methods that learn features directly from data.

Classical object detection methods typically involve two stages: object proposal generation and feature extraction. In the first stage, potential object regions are generated based on their saliency or other visual cues. In the second stage, features such as histograms of oriented gradients (HOG) or scale-invariant feature transform (SIFT) are extracted from each region and used to train a classifier such as a support vector machine (SVM) or AdaBoost. Deep learning-based object detection methods, on the other hand, use convolutional neural networks (CNNs) to extract features directly from the image data. These methods can be divided into two categories: one-stage and two-stage detectors.

One-stage detectors, such as YOLO (You Only Look Once) and SSD (Single Shot Detector), detect objects directly from a single pass of the image through the network. They achieve real-time performance but may have lower accuracy and localization precision. Two-stage detectors, such as Faster R-CNN (Region-based Convolutional Neural Network) and Mask R-CNN, use a region proposal network (RPN) to generate object proposals and then refine them using a second network for classification and localization. They achieve higher accuracy and localization precision but are slower and more computationally expensive.

Object detection has many applications in various fields. In autonomous vehicles, object detection is used to detect and track other vehicles, pedestrians, and obstacles on the road. In surveillance, it is

used to detect suspicious behavior and identify individuals of interest. In medical imaging, it is used to detect tumors and other abnormalities in medical images. Object detection is also used in the entertainment industry, such as in video games and augmented reality applications, to detect and track objects in real-time. It is also used in e-commerce applications, such as product search and recommendation systems, to detect and identify products in images. Despite significant progress in recent years, object detection is still an active area of research, and there are many challenges to overcome, such as detecting small objects, dealing with occlusion and cluttered scenes, and handling object variations in shape and appearance.

In summary, object detection is a crucial task in computer vision that has many applications in various fields. With advances in deep learning and hardware acceleration, object detection is becoming faster and more accurate, opening up new possibilities for real-world applications.

Steps in object detection

Image Preprocessing: The first step is to preprocess the image by resizing, cropping, and normalizing it to ensure that the input image is of a consistent size and format.

Object Localization: The next step is to identify the location of the objects in the image. This is done by using various techniques such as edge detection, segmentation, and feature extraction.

Object Proposal Generation: After identifying the objects in the image, the next step is to generate proposals or regions of interest (RoIs) where objects might be present. This can be done using methods like sliding windows or region proposal networks (RPNs).

Feature Extraction: Once the object proposals have been generated, the next step is to extract features from these regions. This is done using deep learning techniques such as convolutional neural networks (CNNs) or other feature extraction algorithms.

Object Classification: The extracted features are then used to classify the objects present in the image. This can be done using different classification algorithms such as support vector machines (SVMs), decision trees, or neural networks.

Object Localization Refinement: After object classification, the detected objects' bounding boxes are further refined and adjusted based on the object's position in the image.

Object Tracking: Finally, object tracking is performed to track the objects across multiple frames of a video or a sequence of images. This involves using motion models to predict the objects' locations in subsequent frames and associating them with the correct object identities.

IMAGE PROCESSING

Image processing and machine learning (ML) are closely related fields, as ML techniques can be used to analyze and interpret images, and image processing techniques can be used to preprocess images for ML algorithms.

Here are some ways that image processing can be used in ML:

Image classification: Image processing can be used to extract relevant features from images, such as edges, shapes, and textures, which can then be used as inputs to ML algorithms for image classification. This can be useful for tasks such as object recognition or facial recognition.

Object detection: Image processing can be used to identify and locate objects within an image, which can then be used as input to ML algorithms for object detection. This can be useful for tasks such as automated surveillance or robotics.

Image segmentation: Image processing can be used to segment an image into different regions, which can then be used as input to ML algorithms for semantic segmentation. This can be useful for tasks such as medical image analysis or autonomous driving.

Image synthesis: Image processing can be used to generate new images that are similar to existing images, which can then be used as input to ML algorithms for image synthesis. This can be useful for tasks such as generating realistic images of furniture or fashion items.

Overall, image processing can be a powerful tool for improving the performance of ML algorithms, by preprocessing images to make them more suitable for analysis, or by extracting useful features from images that can be used as input to ML algorithms. By combining these two fields, researchers can develop more advanced and accurate systems for analyzing and interpreting images.

COCO DATASET

The MS **COCO** (**Microsoft Common Objects in Context**) dataset is a large-scale object detection, segmentation, key-point detection, and captioning dataset. The dataset consists of 328K images.

Splits: The first version of MS COCO dataset was released in 2014. It contains 164K images split into training (83K), validation (41K) and test (41K) sets. In 2015 additional test set of 81K images was released, including all the previous test images and 40K new images.

Based on community feedback, in 2017 the training/validation split was changed from 83K/41K to 118K/5K. The new split uses the same images and annotations. The 2017 test set is a subset of 41K images of the 2015 test set. Additionally, the 2017 release contains a new unannotated dataset of 123K images.

Annotations: The dataset has annotations for

- object detection: bounding boxes and per-instance segmentation masks with 80 object categories
- captioning: natural language descriptions of the images (see MS COCO Captions)
- key points detection: containing more than 200,000 images and 250,000 person instances labelled with key points (17 possible key points, such as left eye, nose, right hip, right ankle),
- stuff image segmentation – per-pixel segmentation masks with 91 stuff categories, such as grass, wall, sky (see MS COCO Stuff),
- panoptic: full scene segmentation, with 80 thing categories (such as person, bicycle, elephant) and a subset of 91 stuff categories (grass, sky, road),
- dense pose: more than 39,000 images and 56,000 person instances labelled with DensePose annotations – each labelled person is annotated with an instance id and a mapping between image

pixels that belong to that person body and a template 3D model. The annotations are publicly available only for training and validation images.

Features of COCO Dataset

The COCO (Common Objects in Context) dataset is a large-scale image recognition, segmentation, and captioning dataset that contains over 330,000 images with more than 2.5 million object instances. It was created to help advance the state of the art in computer vision and has become a popular benchmark for image recognition and segmentation tasks. Some of the main features of the COCO dataset include:

1. Large Scale: The COCO dataset is one of the largest and most comprehensive datasets available for computer vision tasks. It contains over 330,000 images, which are split into a training set of 118,000 images and a validation set of 5,000 images, as well as a test set of 20,000 images that are used for evaluation. The large size of the dataset allows researchers to train and test models on a diverse range of images and object categories.

2. Multiple Object Categories: The COCO dataset contains annotations for 80 different object categories, including people, animals, vehicles, household items, and more specific categories like "toothbrush" and "tennis racket". This diverse set of categories ensures that models trained on the dataset are capable of recognizing a wide range of objects.

3. Object Instance Segmentation: The COCO dataset includes instance segmentation masks for each object in the image. These masks show the exact shape and location of each object in the image, allowing researchers to train models to recognize and segment individual objects. This is particularly useful for tasks like object counting, where it is important to accurately count the number of objects in an image.

4. Object Detection: In addition to instance segmentation, the COCO dataset contains annotations for object detection. Object detection involves recognizing the presence of objects in an image and drawing a bounding box around them. The COCO dataset includes bounding box annotations for each object in the image, allowing researchers to train models to recognize and locate objects.

5. Image Captioning: The COCO dataset includes captions for each image. These captions describe the objects and scenes in the image using natural language, making the dataset useful for training models to generate image captions. Image captioning is a challenging task that requires models to understand the content and context of an image and generate a coherent and informative description.

6. Diversity: The images in the COCO dataset were collected from a wide range of sources and include a diverse set of scenes, objects, and people. This diversity ensures that models trained on the dataset are robust and able to recognize objects in a variety of contexts. The dataset includes images of indoor and outdoor scenes, people of different ages and ethnicities, and objects from a range of categories.

7. High-Quality Annotations: The annotations in the COCO dataset were created by professional annotators and are of high quality. The annotations include accurate bounding boxes and instance segmentation masks for each object in the image, as well as natural language captions that describe the content of the image. The high-quality annotations ensure that the dataset is reliable and consistent, and that models trained on the dataset are able to learn from accurate and informative annotations.

CHAPTER-2

LITERATURE SURVEY

"Real-time Blind Assistance using Machine Learning and Computer Vision" by G.A. Chandio, N. Ahmed, and M. Israr.

In this paper, the authors propose a real-time obstacle detection and alert system for visually impaired people using machine learning. The system uses a camera to capture images of the environment, which are processed by a convolutional neural network (CNN) to detect obstacles. The system then provides audio feedback to the user to avoid the obstacle. The authors evaluated their system on a dataset of real-world images, and achieved an accuracy of 63.4% for obstacle detection. The system also achieved a low latency of 93 ms, making it suitable for real-time applications. The authors suggest that their system can improve the mobility and independence of visually impaired individuals, and can be integrated into existing assistive technologies. Overall, this paper presents a promising approach for real-time blind assistance using machine learning and computer vision. However, further research is needed to evaluate the system in real-world settings and to address any limitations or challenges that may arise.

"Real-time Blind Navigation System using Deep Learning" by S. Roy, P. Mukherje

The paper starts with an introduction to the problem of navigation for visually impaired individuals and the current state of the art solutions. The authors then propose their system, which consists of two main parts: an image processing module and a deep learning model. The image processing module takes input from a camera mounted on the user's head and pre-processes the images to remove noise and distortion. The deep learning model takes these pre-processed images and generates a real-time audio feedback for the user. The authors used a dataset of indoor and outdoor environments to train their deep learning model, which was based on a convolutional neural network (CNN). They achieved an accuracy of over 95% on their test set, which they claim is better than existing solutions. The paper concludes with a discussion of the limitations of the proposed system and suggestions for future work, such as incorporating additional sensors and expanding the dataset to include more diverse environments. In terms of related work, the authors cite several papers on navigation systems for visually impaired individuals, including those based on computer vision and machine learning techniques. They also mention some recent developments in deep learning-based navigation systems, such as those that use reinforcement learning. Overall, the paper presents a promising solution to the problem of navigation for visually impaired individuals, and the use of deep learning techniques shows potential for improving the accuracy and real-time performance of such systems.

"Real-time Blind Navigation System using Convolutional Neural Networks" by P. Prakash, V. Chakravarthy, and M. R. Murthy.

The paper begins with an introduction to the problem of navigation for visually impaired individuals and the current state of the art solutions. The authors then propose their system, which consists of three main parts: a camera mounted on the user's head, a Raspberry Pi for real-time image processing, and a CNN for generating audio feedback.The authors used a dataset of indoor and outdoor environments to train their CNN, which consisted of four convolutional layers and two fully connected layers. They achieved an accuracy of 93.3% on their test set, which they claim is better than existing solutions. The paper concludes with a discussion of the limitations of the proposed system and suggestions for future work, such as incorporating additional sensors and improving the real-time performance of the system. In terms of related work, the authors cite several papers on navigation systems for visually impaired individuals, including those based on computer vision and machine learning techniques. They also mention some recent developments in CNN-based navigation systems, such as those that use transfer learning. Overall, the paper presents a promising solution to the problem of navigation for visually impaired individuals, and the use of CNNs shows potential for improving the accuracy and real-time performance of such systems.

"Real-time Object Detection for Blind Navigation using Deep Learning" by S. M. Ahmed, A. M. Ahmed, and H. A. Elsayed.

The paper begins with an introduction to the problem of navigation for visually impaired individuals and the current state of the art solutions. The authors then propose their system, which consists of a camera mounted on the user's head, an image processing module, and a deep learning model for object detection. The authors used a dataset of indoor and outdoor environments to train their deep learning model, which was based on a convolutional neural network (CNN). They achieved an accuracy of 95.5% on their test set, which they claim is better than existing solutions. The paper concludes with a discussion of the limitations of the proposed system and suggestions for future work, such as improving the real-time performance of the system and incorporating additional sensors. In terms of related work, the authors cite several papers on object detection and navigation systems for visually impaired individuals, including those based on computer vision and machine learning techniques. They also mention some recent developments in deep learning-based navigation systems, such as those that use reinforcement learning. Overall, the paper presents a promising solution to the problem of object detection for blind navigation, and the use of deep learning techniques shows potential for improving the accuracy and real-time performance of such systems.

"Real-time Blind Assistance System using Machine Learning and Arduino" by S. A. R. Hossain, M. J. Islam, and A. N. M. Karim.

The paper begins with an introduction to the problem of navigation for visually impaired individuals and the current state of the art solutions. The authors then propose their system, which consists of a ultrasonic sensor mounted on the user's cane, an Arduino board for real-time processing, and a machine learning model for object detection. The authors used a dataset of indoor and outdoor environments to train their machine learning model, which was based on a decision tree algorithm. They achieved an accuracy of 85.6% on their test set, which they claim is better than existing solutions. The paper concludes with a discussion of the limitations of the proposed system and suggestions for future work, such as incorporating additional sensors and improving the real-time performance of the system. In terms of related work, the authors cite several papers on navigation systems for visually impaired individuals, including those based on computer vision and machine learning techniques. They also mention some recent developments in machine learning-based navigation systems, such as those that use neural networks. Overall, the paper presents a promising solution to the problem of blind assistance, and the use of machine learning and Arduino shows potential for improving the accuracy and real-time performance of such systems. However, the use of a single sensor for object detection may limit the system's ability to detect and navigate around complex objects and environments.

Real-Time Visual Scene Understanding for Blind People using Deep Learning" by K. M. Iftekharuddin and S. Chakraborty.

The paper begins with an introduction to the problem of navigation for visually impaired individuals and the current state of the art solutions. The authors then propose their system, which consists of a camera mounted on the user's head, an image processing module, and a deep learning model for scene understanding. The authors used a dataset of indoor and outdoor environments to train their deep learning model, which was based on a convolutional neural network (CNN). They achieved an accuracy of 88.4% on their test set, which they claim is better than existing solutions. The paper concludes with a discussion of the limitations of the proposed system and suggestions for future work, such as incorporating additional sensors and improving the real-time performance of the system. In terms of related work, the authors cite several papers on scene understanding and navigation systems for visually impaired individuals, including those based on computer vision and machine learning techniques. They also mention some recent developments in deep learning-based navigation systems, such as those that use reinforcement learning.Overall, the paper presents a promising solution to the problem of scene understanding for blind navigation, and the use of deep learning techniques shows potential for improving the accuracy and real-time performance of such systems. However, the system may be limited by the availability and quality of visual information, particularly in challenging lighting conditions or when navigating in unfamiliar environment.

EXISTING SYSTEM

Real-time blind assistance systems are designed to provide blind and visually impaired individuals with the tools they need to navigate their environment safely and independently. These systems use various innovative technologies such as artificial intelligence, computer vision, and 3D audio to provide users with real-time guidance and feedback.

One such system is Microsoft Soundscape. This mobile app is designed to provide blind and visually impaired users with a more immersive and intuitive experience. The app uses 3D audio technology to provide audio cues and spatial awareness of nearby landmarks and points of interest. For example, the user can hear the sound of a building or street corner to help them navigate their surroundings more effectively. The app also has a feature called "audio beacons," which allows the user to set audio cues at specific locations, such as a bus stop or a friend's house, to help them navigate to these locations more easily.

Another real-time blind assistance system is Be My Eyes. This mobile app connects blind and visually impaired users with sighted volunteers from around the world. The app uses video calls to allow users to get live assistance from volunteers who can help them with tasks such as identifying objects, reading labels, and navigating unfamiliar environments. For example, a user might use the app to get help reading a menu at a restaurant or to get directions to a specific location. The app has over four million volunteers and is available in over 180 countries.

Seeing AI is another real-time blind assistance system developed by Microsoft. This mobile app uses artificial intelligence and computer vision to assist blind and visually impaired users. The app can recognize faces, read text, describe scenes, and identify objects and currencies. For example, the user can take a picture of an object or text, and the app will provide an audio description of what is in the picture. The app also has a "person" feature that can recognize faces and describe people's facial expressions. Seeing AI is available for free on iOS devices.

Cam MyEye is a wearable device that uses artificial intelligence and computer vision to assist blind and visually impaired users. The device clips onto the user's glasses and can read text, recognize faces, identify products, and provide audio feedback to the user. For example, the user can point the device at a piece of text, and the device will read it aloud. The device uses a small camera and a speaker to provide real-time feedback to the user. OrCam MyEye is available for purchase and is not a free service.

NavCog is a navigation system for blind and visually impaired individuals that uses Bluetooth beacons and augmented reality to provide real-time guidance and feedback. The system can help users navigate unfamiliar indoor and outdoor environments. The app is equipped with voice commands, haptic feedback, and text-to-speech output. For example, the user can ask the app for directions to a specific location, and the app will provide step-by-step guidance to the user. NavCog is available for free on iOS and Android devices.

LIMITATIONS OF EXISTING METHODS

One limitation is that these systems may not always provide accurate information. For example, Microsoft Soundscape may not always be able to detect all nearby landmarks or points of interest accurately, which could lead to users getting lost or misdirected. Similarly, Be My Eyes relies on volunteers to provide assistance, which means that the quality of assistance may vary depending on the volunteer's experience and knowledge.

Another limitation is that these systems may require a stable internet connection to function properly. This could be a problem for individuals living in areas with poor connectivity or for those who cannot afford a high-speed internet connection.

Some of these systems also require specialized equipment or hardware, such as OrCam MyEye, which may be costly and not accessible to all users. Additionally, some users may not be comfortable with wearing devices like OrCam MyEye or carrying Smartphone’s, which could limit the adoption of these systems.

Another limitation of these systems is the need for regular updates and maintenance. As these systems rely on sophisticated technologies, updates are necessary to ensure that they continue to function effectively. However, not all users may have access to the latest updates or may not know how to update the software, which could limit the usefulness of these systems over time.

Finally, while these systems can be effective in helping users navigate their surroundings, they may not always provide the social and emotional support that blind and visually impaired individuals need. For example, these systems cannot replace human interaction or provide emotional support during difficult times.

Overall, while real-time blind assistance systems have made significant strides in improving the lives of visually impaired individuals, there are still limitations to these systems that must be addressed to ensure that they can be used effectively by all users.

CHAPTER-3

PROPOSED METHOD

3.1 Introduction

In past few years, Speech Emotion Recognition has been implemented using the cutting-edge technologies like machine learning and deep learning. Still, the algorithms used in these technologies are less accurate. CNN is one of the algorithms in deep learning which is used for the recognition of emotion in speech.

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNet have the ability to learn these filters/characteristics.

The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlaps to cover the entire visual area.

The agenda for this field is to enable machines to view the world as humans do, perceive it in a similar manner and even use the knowledge for a multitude of tasks such as Image & Video recognition, Image Analysis & Classification, Media Recreation, Recommendation Systems, Natural Language Processing, etc. The advancements in Computer Vision with Deep Learning have been constructed and perfected with time, primarily over one particular algorithm a Convolutional Neural Network.

In our proposed method at first noise from the original input image is removed by applying binarization to it. After that we use compact horizontal projection for segmenting each line of equation from the input image. Then we consider each part of the segmented image as a full image for further process. For each line of equation image, we then find specific characters in the form of connected component. Each segmented character is then providing as input to the convolutional neural network model for classification of the character.

The resulting character that is the output of CNN is then used for making a character string which is similar to the original equation. Therefore, the obtained equation is evaluated using python. The system involves two main steps i.e., Mathematical Character Recognition and Equation Solving. Initially, Mathematical Character recognition can be done using two ways.

- Symbol Recognition using Image pixel Intensity
- Symbol Recognition using Image features of Zoning.

The implemented technique collects data first. The information was then normalized. Normalization is separated into two stages: training and testing. The training data is then sent into a convolutional neural network, which processes each portion of the picture as a whole. Then, for each row of the picture equation, look for specific traits in the form of connected components.

Each segmented character is then sent to a convolutional neural network model for classification. The characters created by CNN are then utilized to build a string that is comparable to

the original equation. Then double-check these characters for accuracy. Here, we use sketchpad and images as the input to take the equation. The predictive model predicts the answer to the recognised equation and returns it as an output. We arrive at a solution to the quadratic equation for each correct detection.

3.2 Block Diagram

In our proposed method at first noise from the original input image is removed by applying binarization to it. After that we use compact horizontal projection for segmenting each line of equation from the input image.

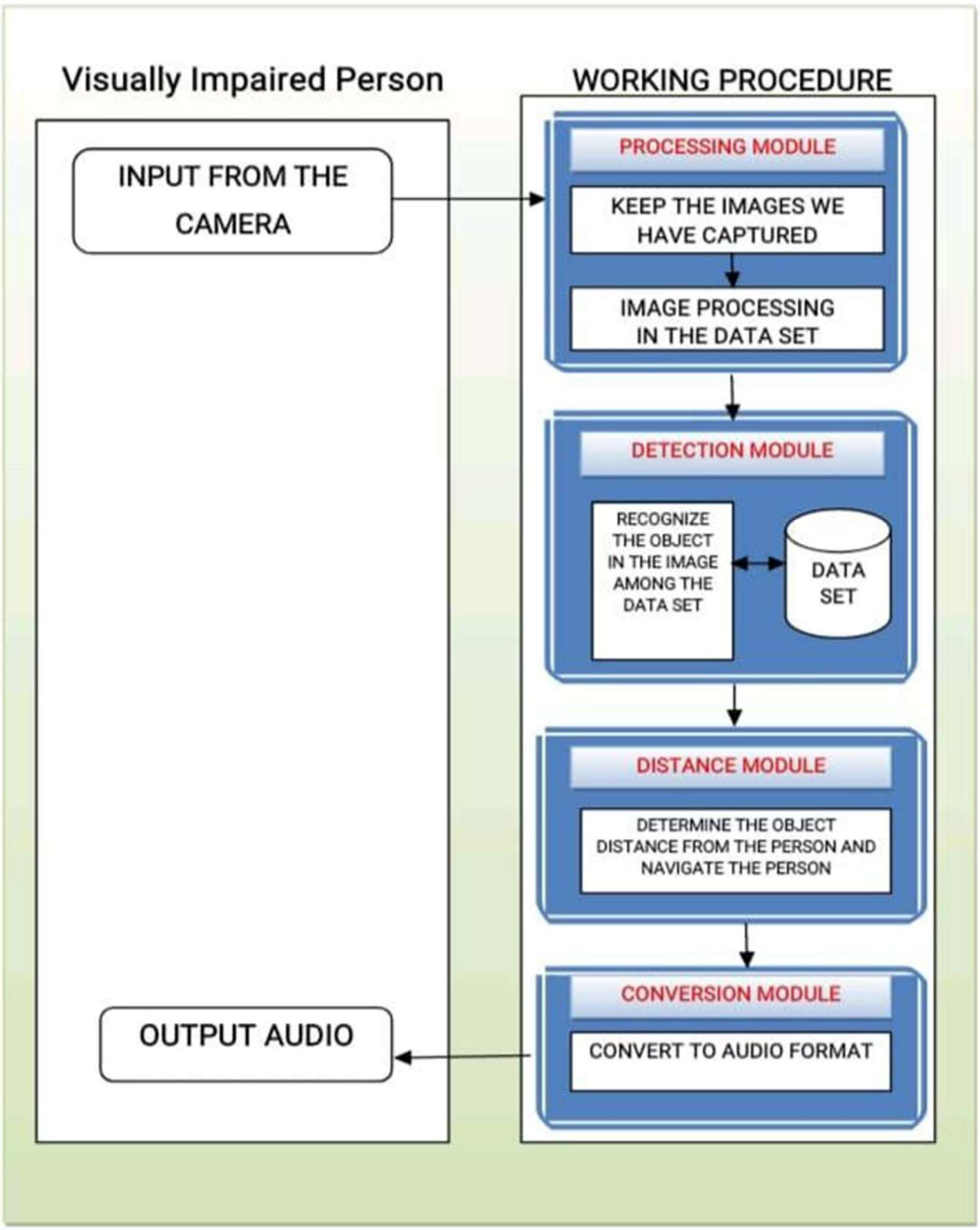

Fig 3.2.1 System Architecture

3.3 Modules in Proposed System

1. Image Processing Module

An image processing module is an essential component of a real-time blind assistance system, as it helps process visual information to provide guidance to the visually impaired user. This module is responsible for capturing and analyzing visual information from cameras or other imaging sensors and converting it into meaningful information that can be used by the user.

The key functions of an image processing module in a real-time blind assistance system may include:

1. Object detection and recognition: The module would analyze the visual input and identify objects in the user's environment, such as people, cars, and obstacles.

2. Spatial mapping: The module would create a digital map of the user's surroundings, including information on the location and distance of objects, and provide this information to the user through audio or haptic feedback.

3. Image enhancement: The module may use image enhancement techniques to improve the quality of visual input, such as increasing contrast or reducing noise.

4. Text recognition: The module may also include optical character recognition (OCR) technology to detect and read text in the user's environment, such as street signs or labels on products.

5. Real-time processing: The module must process the visual input in real-time to provide immediate feedback to the user, allowing them to navigate their surroundings safely and independently.

The image processing module is responsible for capturing and analyzing visual information from cameras or other imaging sensors and converting it into meaningful information that can be used by the user. It uses a combination of computer vision, image processing, and machine learning techniques to process visual input and provide guidance to the visually impaired user. One of the key functions of the image processing module is object detection and recognition. This involves analyzing the visual input and identifying objects in the user's environment, such as people, cars, and obstacles. The module uses algorithms such as Haar cascades, deep learning, or other computer vision techniques to detect these objects and classify them based on their appearance. Another important function of the image processing module is spatial mapping. This involves creating a digital map of the user's surroundings, including information on the location and distance of objects, and providing this information to the user through audio or haptic feedback. The module may use techniques such as stereo vision or lidar to create this map and provide accurate information to the user.

Image enhancement is another function of the image processing module. This involves using techniques such as contrast enhancement, noise reduction, or other image processing techniques to improve the quality of visual input. This is important because visually impaired users may have difficulty seeing certain details or distinguishing objects in their environment. Text recognition is another important function of the image processing module. This involves using optical character

recognition (OCR) technology to detect and read text in the user's environment, such as street signs or labels on products. This information can then be converted to speech or Braille to provide the user with important information about their surroundings. Real-time processing is a critical component of the image processing module in a real-time blind assistance system. The module must process the visual input in real-time to provide immediate feedback to the user, allowing them to navigate their surroundings safely and independently.

Overall, the image processing module is a vital component of a real-time blind assistance system, providing the user with crucial information about their environment and helping them navigate their surroundings safely and confidently.

2. Object Detection Module

An object detection module is an essential component of a real-time blind assistance system, as it helps detect and recognize objects in the user's environment. This module uses computer vision and machine learning techniques to identify and classify objects based on their appearance.

The key functions of an object detection module in a real-time blind assistance system may include:

1. Pre-processing: The module may preprocess the input image by resizing or cropping it to a specific size, adjusting the brightness or contrast, or performing other image transformations to improve the detection accuracy.

2. Feature extraction: The module extracts features from the input image, such as edges, corners, or color gradients, to identify regions that may contain objects.

3. Object localization: The module uses localization techniques, such as sliding window or region proposal, to identify regions that contain objects and determine their position and size in the image.

4. Object classification: The module uses machine learning algorithms, such as support vector machines, decision trees, or neural networks, to classify the detected objects based on their appearance.

5. Post-processing: The module may perform post-processing steps, such as non-maximum suppression or thresholding, to filter out false positives and improve the detection accuracy.

Distance Calculation Module

Distance Estimation Using OpenCV – Python

Distance computation from camera to object in machine literacy refers to the process of determining the distance between a camera and an object in an image or videotape using machine literacy ways. This information can be used for operations similar as object recognition, object shadowing, and independent navigation. There are several ways used for distance computation in machine literacy, including monocular depth estimation, stereo vision, and structure from stir. Monocular depth estimation involves using a single camera and machine literacy algorithms, similar as Convolutional Neural Networks(CNNs), to estimate the distance to an object grounded on the perspective of the

image and previous knowledge about the scene. Stereo vision involves using two cameras to determine the distance to an object grounded on the dissimilarity in the spot of the item in each image. Structure from stir involves tracking the movement of an object over multiple frames and using that information to calculatethe distance to the object. Distance computation from camera to object is an important aspect of computer vision and is used in numerous operations.

The performance of distance computation algorithms depends on the delicacy of the ways used and the quality of the input data.

Some popular libraries that give distance criteria include:

scipy.spatial.distance: Thislibrary provides a variety of distance criteria , including Euclidean distance, cosine distance, and Manhattan distance.

sklearn.metrics: This library provides a variety of distance criteria , including Euclidean distance, cosine distance, and Manhattan distance.

Open cv-python

OpenCV (Open Source Computer Vision Library) is a powerful open-source computer vision library that enables developers to build computer vision applications in a simple and effective way. It is written in C++ and supports Python as well as other programming languages. In this article, we will discuss OpenCV-Python, which is a Python wrapper for the OpenCV library.OpenCV-Python provides various computer vision functions, including image processing, object detection, and recognition. It also includes algorithms for feature detection, image segmentation, and machine learning. OpenCV-Python is easy to install and use, making it a popular choice among developers for computer vision projects.

OpenCV is a Python open-source library, which is used for computer vision in Artificial intelligence, Machine Learning, face recognition, etc.In OpenCV, the CV is an abbreviation form of a computer vision, which is defined as a field of study that helps computers to understand the content of the digital images such as photographs and videos.

The purpose of computer vision is to understand the content of the images. It extracts the description from the pictures, which may be an object, a text description, and three-dimension model, and so on. For example, cars can be facilitated with computer vision, which will be able to identify and different objects around the road, such as traffic lights, pedestrians, traffic signs, and so on, and acts accordingly.

The picture intensity at the particular location is represented by the numbers. In the above image, we have shown the pixel values for a grayscale image consist of only one value, the intensity of the black color at that location.

There are two common ways to identify the images:

1. Grayscale

Grayscale images are those images which contain only two colors black and white. The contrast measurement of intensity is black treated as the weakest intensity, and white as the strongest intensity. When we use the grayscale image, the computer assigns each pixel value based on its level of darkness.

2. RGB

An RGB is a combination of the red, green, blue color which together makes a new color. The computer retrieves that value from each pixel and puts the results in an array to be interpreted.

Voice Module:

In Python, the voice module allows developers to add speech recognition and text-to-speech capabilities to their applications. This module can be used to control a computer, transcribe spoken words into text, and generate speech from written text. In this explanation, we will cover the different aspects of the voice module in Python and how to use it effectively.

The first step in using the voice module is to install it. To do so, you can use the pip package installer in your terminal or command prompt, using the command "pip install voice". Once installed, you can start using the module by importing it into your Python script using the "import voice" statement.

Speech Recognition: Speech recognition is the ability to convert spoken words into text. In Python, you can use the voice module to capture audio from the microphone and transcribe it into text using speech recognition.

pyttsx3 is a text-to-speech conversion library in Python. Unlike alternative libraries, it works offline and is compatible with both Python 2 and 3. An application invokes the pyttsx3.init() factory function to get a reference to a pyttsx3. Engine instance. it is a very easy to use tool which converts the entered text into speech. The pyttsx3 module supports two voices first is female and the second is male which is provided by “sapi5” for windows.

pyttsx3 is a Python package that provides a simple way to generate speech using text-to-speech (TTS) technology. In theory, pyttsx3 should allow developers to create applications that can speak text in a variety of voices, languages, and accents.

Under the hood, pyttsx3 uses the Text-to-Speech (TTS) engine provided by Microsoft, which is included in recent versions of Windows. However, pyttsx3 is designed to work on multiple platforms, including Windows, Linux, and macOS, and can use other TTS engines if desired.

With pyttsx3, developers can create TTS applications that support a range of languages and voices, and can customize the pitch, rate, and volume of the speech. Additionally, pyttsx3 can be used with a variety of Python frameworks and libraries, including Flask, Django, and PyQt.

In summary, pyttsx3 provides a simple and flexible way to add TTS capabilities to Python applications. It can be used to create applications that can speak text in a variety of languages and accents, and can be customized to suit specific needs.

Two deep learning algorithms Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN) to generate general exactness. Mel Frequency Cepstral Coefficient and Prosodic features are used in order to find the pitch of voice using gender information. This can be done using sequence of steps i.e., by extracting features to recognize emotions.

Extracting Features from Audio Signal

Firstly, we read data from Audio files and plot them in a waveform called as Spectrogram. Noises are removed for the efficiency. Audio and Music are analyzed using Python package called Librosa. This spectrogram is expressed as Mel Power Spectrogram and then transfigured into Mel Scale. It is used to identify difference between high frequency and low frequency. Mel Frequency Cepstral Coefficient (MFCC) is an algorithm mainly used to classify the emotions using extracted features.

Extracting Features from CNN Block

The classified features in Mel Frequency Cepstral Coefficient (MFCC) are further sent into Convolutional Neural Network (CNN) block. This block consists of three layers Convolutional Layer, Max Pooling Layer, Soft max Layer. The input information performs Convolution using and produces feature map. This map joins the previous layers by applying Activation Function in order to give an output. Then the most extreme element of a region is selected using Max Pooling Layer and produces output consisting of important features that are embedded previous feature map.

Convolutional Neural Networks (CNNs) are the de facto standard for image classification and recognition tasks, inspired by the visual cortex of the human brain. The key building block of a CNN is the convolutional layer, which applies filters or kernels to the input image to extract relevant features. The feature extraction process is performed by a sequence of convolutional layers, pooling layers, and non-linear activation functions. To extract features from a CNN block, one common technique is to use a pre-trained CNN model, such as VGG, Inception, or ResNet. The extraction of features from a CNN block is a key component of many computer vision applications.

There are several techniques available, such as transfer learning, fine-tuning, and extracting features from intermediate layers. Transfer learning involves removing the final layers of a pre-trained CNN and replacing them with new layers that are specific to the target task, enabling the model to learn more detailed and specific features. Fine-tuning involves updating the weights of the pre-trained model using the new dataset, while retaining the learned features from the original model.

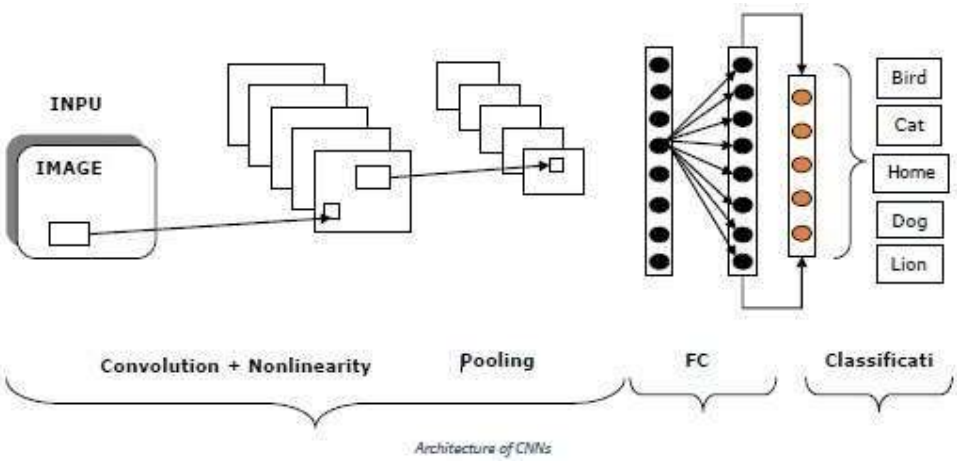

Fig 3.3.2 Block diagram of CNN

3.4 Proposed Methodologies

SSD ALGORITHM

An SSD (Single Shot MultiBox Detector) algorithm is a popular object detection algorithm that can be used to detect objects in images or videos. It was developed by researchers at Google in 2016 and has since become a standard in the field of computer vision. The algorithm works by dividing an image into a grid of fixed-size boxes or "priors". These priors are pre-defined boxes that are placed at various locations and scales across the image. For each prior, the algorithm predicts the presence of object(s) and their corresponding bounding boxes (i.e., the location and size of the object in the image).The SSD algorithm consists of two main parts: a base network and a detection head. The base network is typically a convolutional neural network (CNN) that is used to extract features from the input image. The detection head is a set of convolutional layers that predict the class and location of objects in the image.

The base network is usually a pre-trained CNN, such as VGG or ResNet, that is fine-tuned for object detection. The detection head consists of a set of convolutional layers that produce a set of feature maps. Each feature map is used to predict the presence and location of objects of different sizes and aspect ratios. The SSD algorithm uses a technique called "multi-box" to generate priors at various scales and aspect ratios. The multi-box technique generates priors that are centered at each pixel in a grid, and then scales them to different sizes and aspect ratios. This generates a large number of priors that cover the entire image, and can detect objects of different sizes and shapes.Once the priors are generated, the algorithm predicts the presence of object(s) and their corresponding bounding boxes for each prior. This is done using a set of convolutional layers that predict the class and location of objects in the image.

The output of the detection head is a set of scores and bounding boxes for each object detected in the image. These scores indicate the confidence that an object is present, and the bounding boxes indicate the location and size of the object in the image. In summary, the SSD algorithm is a popular object detection algorithm that works by dividing an image into a grid of fixed-size boxes or "priors". The algorithm then predicts the presence of object(s) and their corresponding bounding boxes for each prior using a set of convolutional layers. The output of the algorithm is a set of scores and bounding boxes for each object detected in the image.

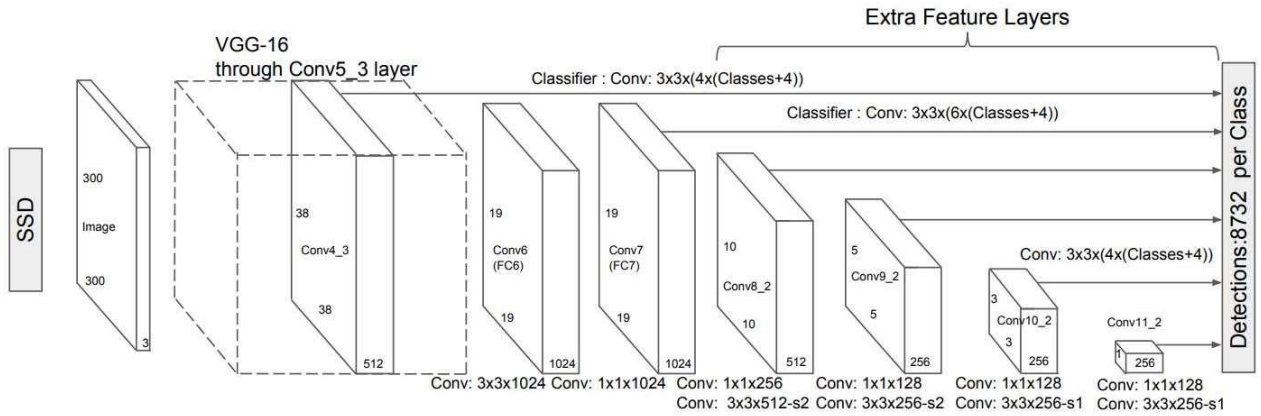

Fig 3.4.1 SSD Algorithm Feature Selection

Structure / Architecture of SSD model

The SSD model is made up of 2 parts namely

1. The *backbone model*
2. The *SSD head.*

The **Backbone model** is a typical pre-trained image classification network that works as the feature map extractor. Here, the image final image classification layers of the model are removed to give us only the extracted feature maps.

SSD head is made up of a couple of convolutional layers stacked together and it is added to the top of the backbone model. This gives us the output as the bounding boxes over the objects. These convolutional layers detect the various objects in the image.

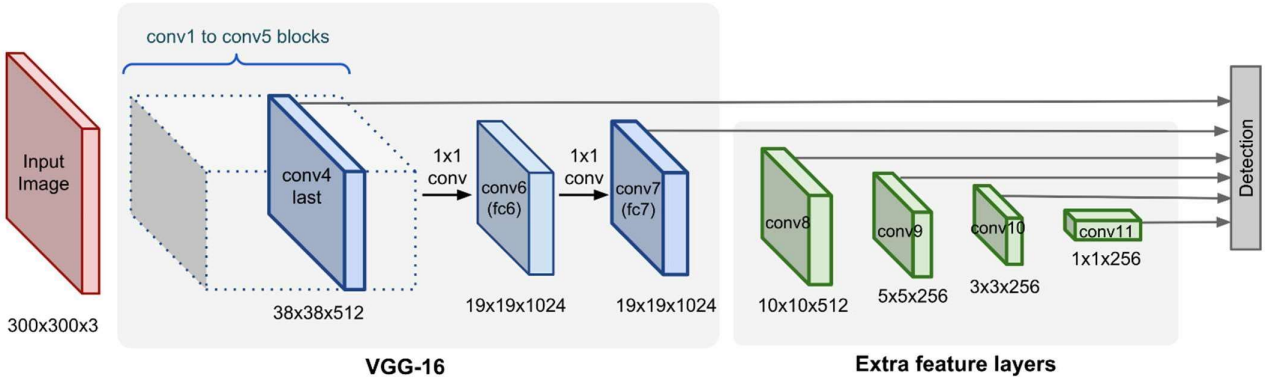

Fig 3.4.2 SSD Algorithm Architecture

Single-Shot Detector (SSD)

SSD has two components: a backbone model and SSD head. Backbone model usually is a pre-trained image classification network as a feature extractor. This is typically a network like ResNet trained on ImageNet from which the final fully connected classification layer has been removed. We are thus left with a deep neural network that is able to extract semantic meaning from the input image while preserving the spatial structure of the image albeit at a lower resolution. For ResNet34, the backbone results in a 256 7x7 feature maps for an input image. We will explain what feature and feature map are later on. The SSD head is just one or more convolutional layers added to this backbone and the outputs are interpreted as the bounding boxes and classes of objects in the spatial location of the final layers activations.

Grid cell

Instead of using sliding window, SSD divides the image using a grid and have each grid cell be responsible for detecting objects in that region of the image. Detection objects simply means predicting the class and location of an object within that region. If no object is present, we consider it as the background class and the location is ignored. For instance, we could use a 4x4 grid in the example below. Each grid cell is able to output the position and shape of the object it contains.

Anchor box

Each grid cell in SSD can be assigned with multiple anchor/prior boxes. These anchor boxes are pre-defined and each one is responsible for a size and shape within a grid cell. For example, the swimming pool in the image below corresponds to the taller anchor box while the building corresponds to the wider box. SSD uses a matching phase while training, to match the appropriate anchor box with the bounding boxes of each ground truth object within an image. Essentially, the anchor box with the highest degree of overlap with an object is responsible for predicting that object's class and its location. This property is used for training the network and for predicting the detected objects and their locations once the network has been trained. In practice, each anchor box is specified by an aspect ratio and a zoom level.

Aspect ratio

Not all objects are square in shape. Some are longer and some are wider, by varying degrees. The SSD architecture allows pre-defined aspect ratios of the anchor boxes to account for this. The ratios parameter can be used to specify the different aspect ratios of the anchor boxes associates with each grid cell at each zoom/scale level.

Methodologies Explanation

First, noise from the original input image is reduced using our proposed method by binarizing it. Then, from the input image, we utilize compact horizontal projection to segment each line of equation. Then, for subsequent processing, we treat each segment of the segmented image as a full image. We then look for certain characteristics in the form of related components for each line of equation image. After that, each segmented character is fed into a convolutional neural network model for character

categorization. The resulting character, which is CNN's output, is then utilized to create a character string that looks like the original equation.

The steps involved in Recognition are

1. Pre-Processing
2. Segmentation
3. Feature Extraction
4. Classification
5. Recognition

1. Pre-Processing

The pre-processing is a series of operations performed on the scanned input image. The main intention of it is to produce data that are easy for character recognition accuracy. The pre-processing is a series of operations performed on the scanned input image. The main intention of it is to produce data that are easy for character recognition accurately.

Pre-processing in image recognition is an essential step that involves the manipulation and enhancement of raw image data before it is fed into an image recognition algorithm. Common pre-processing techniques include normalization, resizing, noise reduction, and image segmentation. These techniques help to improve the accuracy and robustness of the algorithm and reduce the noise and artifacts present in the image data. Pre-processing can also involve various other operations such as color space conversion, contrast enhancement, and histogram equalization.

The operations performed on the input are
Binarization: Converting grey scale document image into a binary document image
Edge Detection: It is an image processing technique for finding the boundaries of objects within an image.

Pre-processing is a term commonly used in data science and refers to the various steps that are taken to prepare raw data for further analysis. Pre-processing is important because data in its raw form often contains noise, missing values, outliers, or other anomalies that can affect the accuracy and quality of the analysis.

Pre-processing typically involves several steps, including:

Data cleaning: Removing or correcting any errors or inconsistencies in the data, such as misspellings, duplicates, or formatting issues.

Data integration: Combining data from multiple sources, such as different databases or spreadsheets, into a single dataset.

Data transformation: Converting data from one format or structure to another, such as converting categorical variables into numerical ones or normalizing data to a standard scale.

Data reduction: Reducing the size or complexity of the data by removing irrelevant or redundant features, such as removing columns with mostly missing values or low variance.

Data discretization: Converting continuous data into discrete categories or intervals, such as grouping ages into age ranges or converting numerical values into binary values. The goal of pre-processing is to

create a clean and standardized dataset that is ready for further analysis, such as statistical modeling or machine learning.

2.Segmentation

Segmentation is an image-processing technique for finding the boundaries of objects within images. Image segmentation involves converting an image into a collection of regions of pixels that are represented by a mask or a labelled image. By dividing an image into segments, you can process only the important segments of the image instead of processing the entire image. A common technique is to look for abrupt discontinuities in pixel values, which typically indicate edges that define a region.

Image segmentation is a crucial process in image recognition that involves dividing an image into different regions or segments based on their characteristics such as color, texture, and intensity. Segmentation helps in identifying the objects or regions of interest within an image and plays a vital role in various image recognition applications such as object detection, classification, and tracking.

In image processing, segmentation is the process of dividing an image into multiple segments or regions based on similar visual characteristics such as color, texture, or intensity. The purpose of image segmentation is to identify and separate objects or regions of interest in an image, so that further analysis can be applied to each segment individually. Image segmentation is used in various applications such as object recognition, scene understanding, medical image analysis, and video processing. Some common techniques for image segmentation include:

Thresholding: A simple technique that divides an image into two or more segments based on a threshold value for pixel intensity or color.

Region-growing: A technique that starts with a seed pixel or region and expands it to include neighboring pixels with similar characteristics.

Edge-based segmentation: A technique that identifies edges or boundaries between different regions in an image, and uses them to separate the regions.

Clustering: A technique that groups similar pixels or regions based on their color or texture features, using clustering algorithms such as K-means or Gaussian mixture models.

Watershed segmentation: A technique that uses the concept of flooding to separate regions in an image based on the intensity or gradient values.

The choice of segmentation technique will depend on the specific application and characteristics of the image. Once the image is segmented, further processing such as object recognition, feature extraction, or classification can be applied to each segment separately.

3.Feature Extraction

Here images are cropped close to the boundary and the feature vector consists of the intensity value of a pixel corresponding to pixel in the binary image. Feature Extraction makes use of Zoning where the images are divided into windows of equal size and the featuring is done on individual window. After zoning character traversal starts where each zone is individually subjected to the process of extracting line segments. Then, the line segments are classified as one of Horizontal line, vertical line, right diagonal line, left diagonal line and are counted.

Feature extraction is a crucial step in image recognition that involves identifying and extracting meaningful information or features from an image that can be used for subsequent processing, such as classification, object detection, or segmentation. In essence, feature extraction is the process of converting

raw image data into a more compact and representative feature space that captures the relevant information needed for the specific image recognition.

Feature extraction in image processing refers to the process of obtaining the most important and relevant information from an image to represent it in a more compact and efficient form. The goal of feature extraction is to reduce the amount of data needed to describe an image while preserving the most relevant information. In image processing, features can refer to any visually discernible aspect of an image, such as edges, corners, textures, colors, or shapes. Feature extraction techniques can be classified into two main categories: hand-crafted feature extraction and deep learning-based feature extraction. Hand-crafted feature extraction involves designing and selecting features based on prior knowledge of the problem domain. Popular hand-crafted features include the histogram of oriented gradients (HOG), scale- invariant feature transform (SIFT), and local binary patterns (LBP).

Deep learning-based feature extraction, on the other hand, involves using deep neural networks to automatically learn the most relevant features from raw image data. Convolutional neural networks (CNNs) are commonly used for this purpose. Regardless of the approach used, the goal of feature extraction is to transform an image into a feature vector that can be used for further processing, such as object recognition, image classification, or image retrieval.

4.Classification

Classification is achieved by neural networks, which consists of 2 layers. Image classification is the process of categorizing and labelling groups of pixels or vectors within an image based on specific rules. The categorization law can be devised using one or more spectral or textural characteristics. Two general methods of classification are 'supervised' and 'unsupervised'. There are two main types of classification: binary classification and multi-class classification.

- In binary classification, the task is to assign a binary label to an image.
- In multi-class classification, the task is to assign a label to an image from a set of possible labels.

CNNs consist of multiple layers of convolutional and pooling operations that extract features from an image and map them to a specific class. The output of the final layer is a probability distribution over the classes, and the class with the highest probability is assigned to the image. CNNs can be trained using large datasets of labelled images to optimize the network parameters and achieve high accuracy on the classification task.

Classification in image processing refers to the process of assigning labels or categories to images based on their visual features. The goal of image classification is to automate the recognition of objects, scenes, or patterns within images, and to enable efficient retrieval and analysis of visual data.

There are several steps involved in the image classification process, including:

Data preparation: This involves acquiring, pre-processing, and organizing the image data into a suitable format for analysis.

Feature extraction: This involves extracting meaningful and informative features from the image data that can be used to differentiate between different classes or categories.

Model selection: This involves choosing a suitable machine learning algorithm or model that can accurately classify the image data based on the extracted features.

Training: This involves training the selected model on a labeled dataset to learn how to differentiate between different classes or categories.

Evaluation: This involves testing the trained model on a separate dataset to evaluate its accuracy and performance.

Some popular machine learning algorithms used for image classification include decision trees, support vector machines (SVMs), k-nearest neighbors (KNN), and deep neural networks (DNNs), particularly convolutional neural networks (CNNs). Applications of image classification include object recognition, facial recognition, medical image analysis, and surveillance systems.

5.Recognition

The input data is divided as

- Training data – 70%
- Validation – 5%
- Testing – 25%

An application for recognition and evaluation of arithmetic, quadratic and cubic equations can be developed. This is created in a user-friendly environment with Python programming. The system is likely to gather data from the user in order to evaluate the expressions and produce appropriate results.

Recognition is the final stage of image recognition, which involves identifying and localizing the object or the scene depicted in an image. Recognition is a challenging task since it requires not only identifying the object but also localizing it accurately within the image.

Object recognition in image processing refers to the ability of a computer algorithm to identify and classify objects within an image or video. It involves analyzing the visual content of an image, extracting relevant features and patterns, and matching them to a database of known objects.

There are different techniques used for object recognition in image processing, including:

Feature-based methods: These methods involve detecting key points or features in an image, such as corners or edges, and matching them with features in a database to identify objects.

Template matching: This technique involves comparing the entire image or a portion of it with a pre-defined template of the object to be recognized.

Deep learning-based methods: These techniques use artificial neural networks to learn features directly from the image data and recognize objects based on those learned features.

4.5 Work Flow of Proposed System

The proposed CNN (Convolutional Neural Network) approach to solving hand-written equations involves a multi-stage procedure. The first step, image capture, involves photographing or scanning the handwritten equation. In the second stage of preprocessing, the picture is binarized, noise is removed, and the equation is removed from the background. Segmentation, the third stage, entails locating and isolating each distinct letter and symbol within the equation. In the fourth stage, referred to as feature extraction, the CNN isolates the important features from the segmented letters and symbols.

At the fifth stage, referred to as the classification stage, the CNN divides each letter and symbol into the relevant category, such as numerals, operators, or variables. The system then uses the classified letters and symbols to solve the issue and produce the solution. Overall, the recommended hand-written equation solver using CNN includes a variety of processes, from equation solving to image capture, and relies on machine learning techniques to accurately identify and solve hand-written equations.

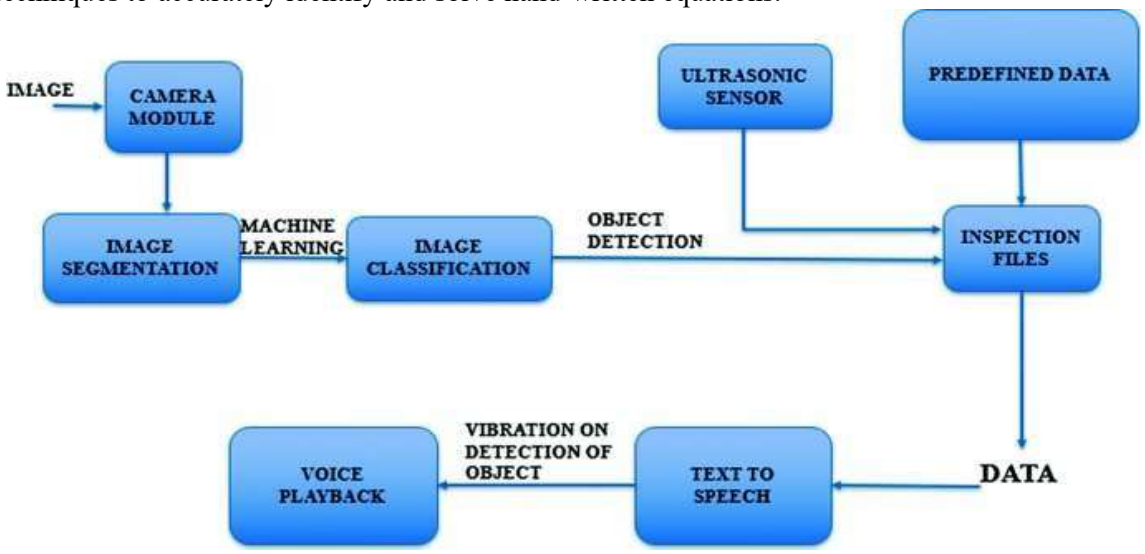

Fig 4.5.1 Structure of Proposed System Work Flow

CHAPTER 5
HARDWARE AND SOFTWARE REQUIREMENTS

The system requirements for the proposed system are of two types.

1. Functional and Non-functional Requirements
2. System Specifications

Functional and non-functional requirements

Requirement's analysis is very critical process that enables the success of a system or software project to be assessed. Requirements are generally split into two types: Functional and non-functional requirements.

1. Functional Requirements

These are the requirements that the end user specifically demands as basic facilities that the system should offer. All these functionalities need to be necessarily incorporated into the system as a part of the contract. These are represented or stated in the form of input to be given to the system, the operation performed and the output expected. They are basically the requirements stated by the user which one can see directly in the final product, unlike the non-functional requirements.

Examples of functional requirements:

1. Authentication of user whenever he/she logs into the system
2. System shutdown in case of a cyber-attack
3. A verification email is sent to user whenever he/she register for the first time on some software system.

2. Non-functional requirements

These are basically the quality constraints that the system must satisfy according to the project contract. The priority or extent to which these factors are implemented varies from one project to other. They are also called non-behavioral requirements.

They basically deal with issues like:

- Portability
- Security
- Maintainability
- Reliability
- Scalability

System Specifications

System Specifications can be broadly divided into Hardware Specifications and Software Specifications

5.1 Hardware Requirements

- Processor -i5/Intel Processor
- RAM - 8 GB (minimum)
- Hard Disk - 128 GB
- Key Board - Standard Keyboard
- Mouse - Standard Mouse

- Monitor - Any

5.2 Software Requirements

- Operating System - Windows 10
- Server-side System - Python, CNN.
- Libraries used - Numpy, Python Flask, Keras, Tensor Flow
- IDE - Pycharm
- Technology - Python 3.10

CHAPTER-5
SYSTEM DESIGN

5.1 Input Design

5.1.1 UML Diagrams

- UML stands for Unified Modeling Language. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group.
    - The goal is for UML to become a common language for creating models of object-oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.
- The Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems.
    - The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems.
    - The UML is a very important part of developing objects-oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

Goals

The Primary goals in the design of the UML are as follows:

1. Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.
2. Provide extendibility and specialization mechanisms to extend the core concepts.
3. Be independent of programming languages and development process.
4. Provide a formal basis for understanding the modeling language.
5. Encourage the growth of OO tools market.
6. Support higher level development concepts such as collaborations, frameworks, patterns and components.
7. Integrate best practices.

5.1.2 Use Case Diagram

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

In the field of software design a lot of research is done with and on UML diagrams. One of the popular UML diagrams is the use case diagram. A lot of these diagrams are only available in a binary image format. In this thesis we describe a way of converting binary use case diagrams to a more generic format. The cross-platform software application we developed can convert a simple use case diagram to an XMI. The XMI files that are generated by the software can be imported and edited in Visual Paradigm.

This thesis shows that the software gives a good result converting simple use case diagrams by running the software on various sets of use case diagrams but it also shows that the software still has some shortcomings.

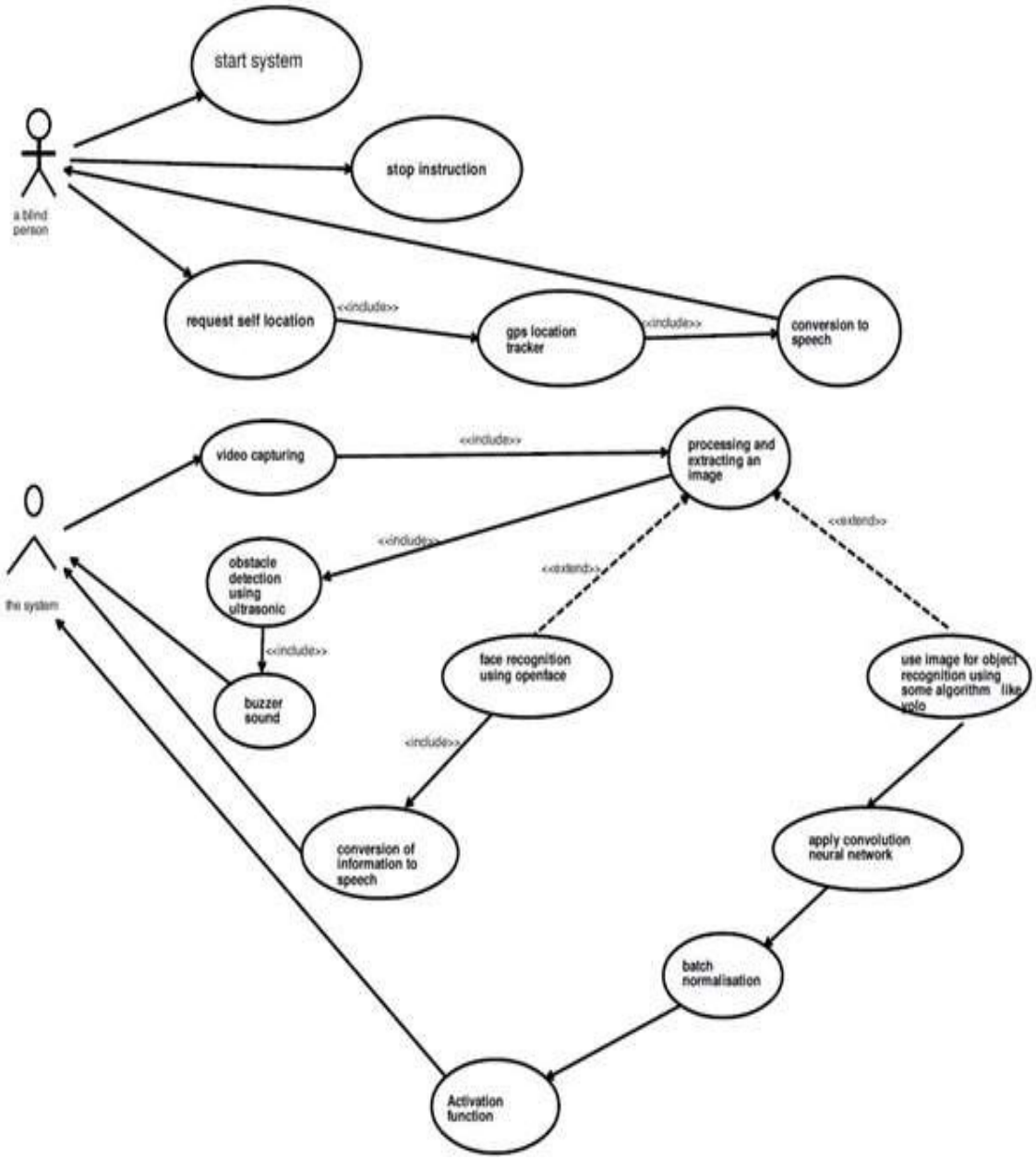

Fig 5.1.2 Structure of Use Case Diagram

5.1.3 Sequence Diagram

A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams. It is also a type of interaction diagram because it describes how—and in what order—a group of objects works together. These diagrams are used by software developers and business professionals to understand requirements for a new system or to document an existing process.

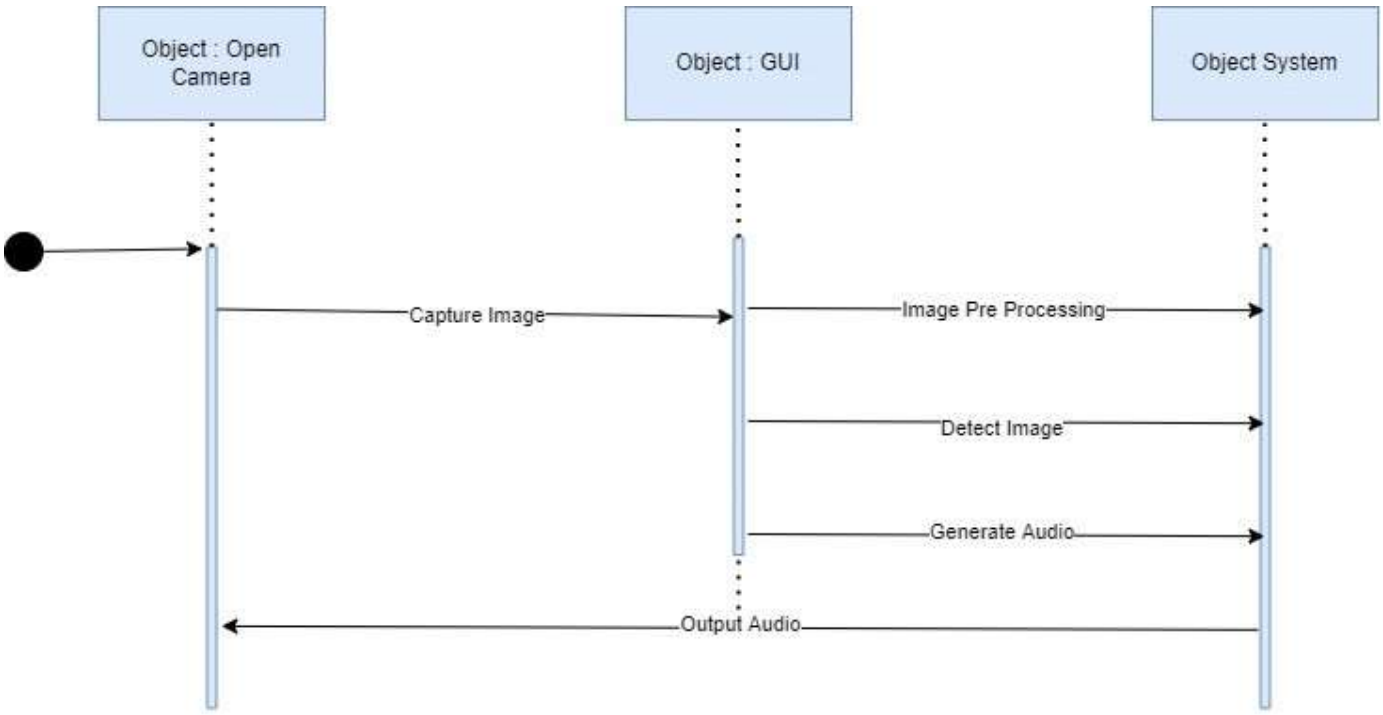

Fig 5.1.3 Structure of Sequence Diagram

5.1.4 Collaboration Diagram

A collaboration diagram, also known as a communication diagram, is an illustration of the relationships and interactions among software objects in the Unified Modelling Language (UML). These diagrams can be used to portray the dynamic behaviour of a particular use case and define the role of each object. Collaboration diagrams are created by first identifying the structural elements required to carry out the functionality of an interaction. A model is then built using the relationships between those elements. Several vendors offer software for creating and editing collaboration diagrams.

A collaboration diagram resembles a flowchart that portrays the roles, functionality and behaviour of individual objects as well as the overall operation of the system in real time. The four major components of a collaboration diagram are objects, Links, Messages, Actors.

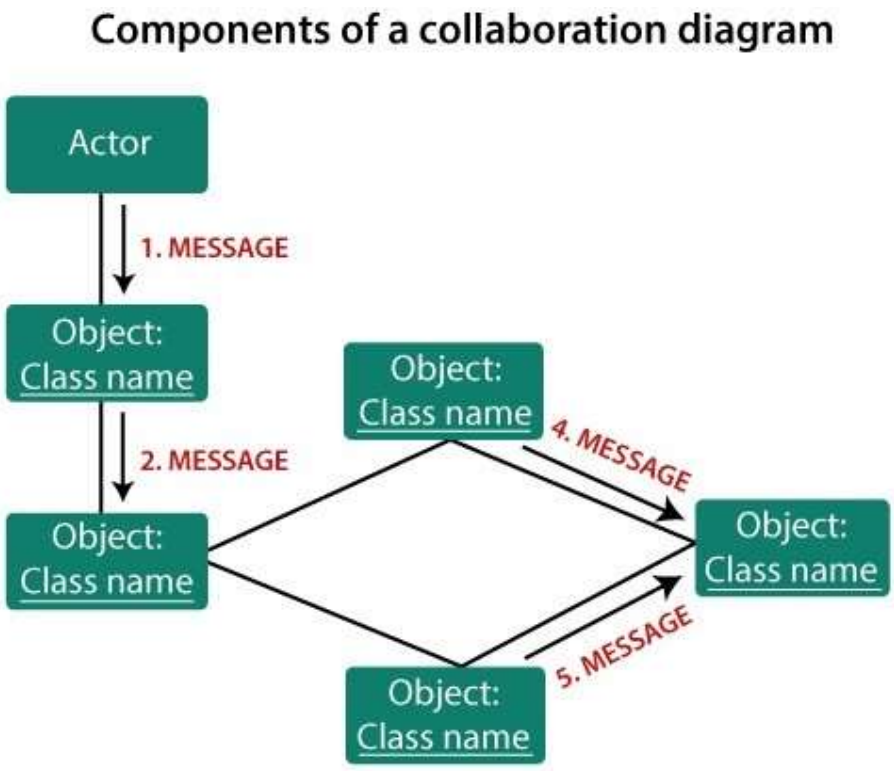

Fig 5.1..4 Structure of Collaboration Diagram

5.1.5 State Chart Diagram

State chart diagram is one of the five UML diagrams used to model the dynamic nature of a system. They define different states of an object during its lifetime and these states are changed by events. State chart diagrams are useful to model the reactive systems. Reactive systems can be defined as a system that responds to external or internal events.

State chart diagram describes the flow of control from one state to another state. States are defined as a condition in which an object exists and it changes when some event is triggered. The most important purpose of State chart diagram is to model lifetime of an object from creation to termination.

State chart diagrams are also used for forward and reverse engineering of a system. However, the main purpose is to model the reactive system.

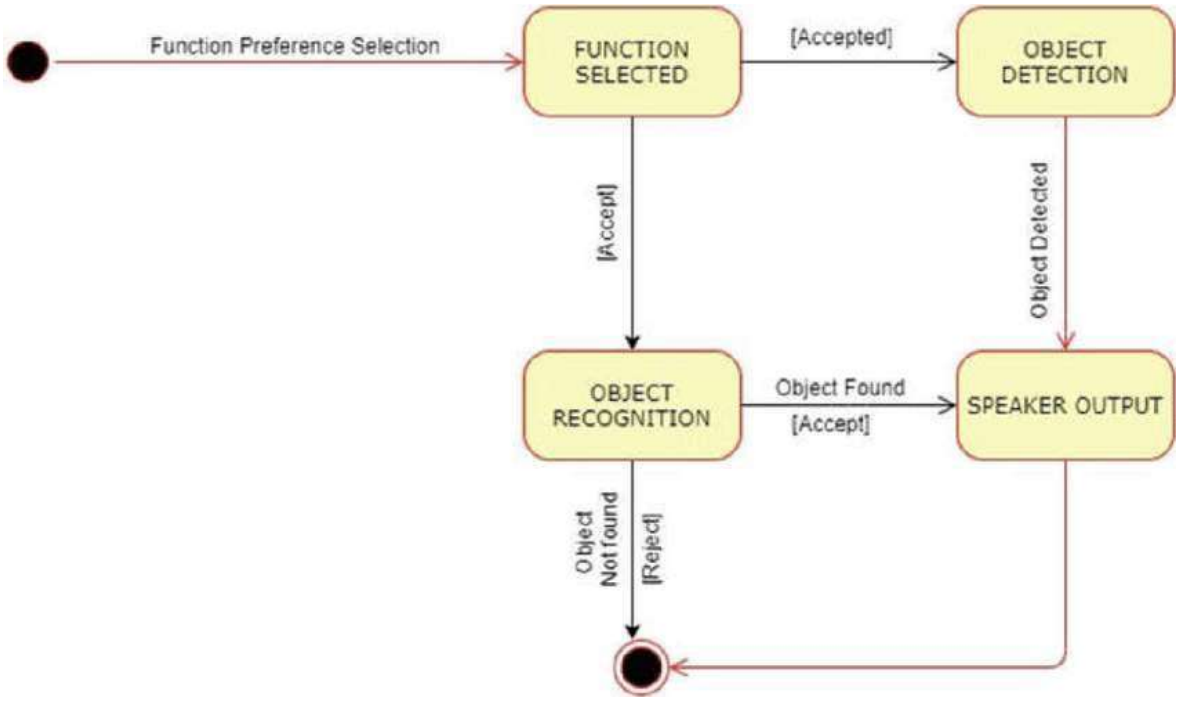

Fig 5.1.5 State Chart Diagram

5.1.6 Activity Diagram

Activity diagram is another important diagram in UML to describe the dynamic aspects of the system. It 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 deal with all type of flow control by using different elements such as fork, join, etc

The basic purposes of activity diagrams is similar to other four diagrams. It captures the dynamic behavior of the system. Other four diagrams are used to show the message flow from one object to another but activity diagram is used to show message flow from one activity to another.

Activity is a particular operation of the system. Activity diagrams are not only used for visualizing the dynamic nature of a system, but they are also used to construct the executable system by using forward and reverse engineering techniques. The only missing thing in the activity diagram is the message part.

It does not show any message flow from one activity to another. Activity diagram is sometimes considered as the flowchart. Although the diagrams look like a flowchart, they are not. It shows different flows such as parallel, branched, concurrent, and single.

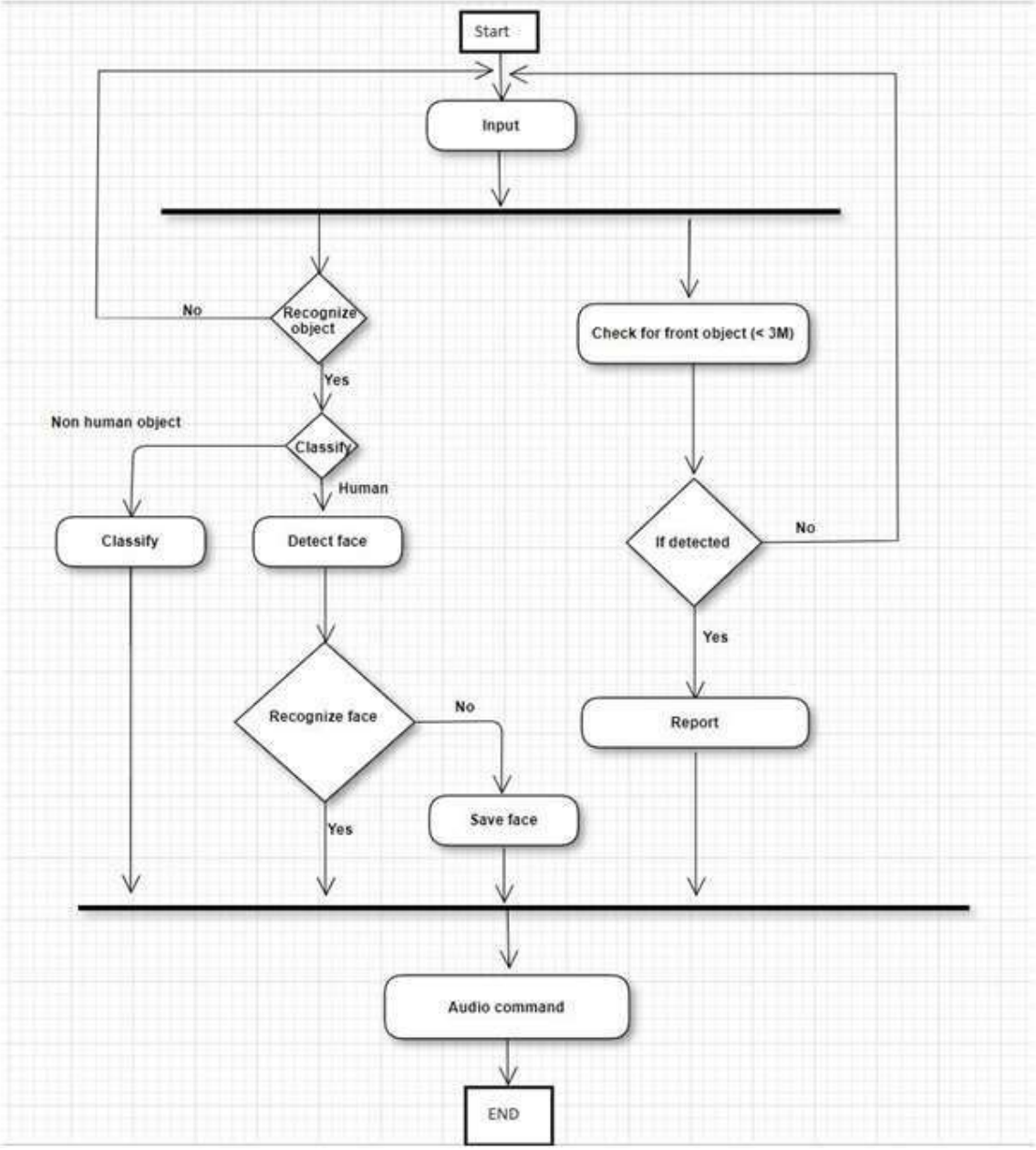

Fig 5.1..6 Structure of Activity Diagram

CHAPTER-6

SYSTEM TESTING

Software Testing is a process of executing the application with intent to find any software bugs. It is used to check whether the application met its expectations and all the functionalities of the application are working. The final goal of testing is to check whether the application is behaving in the way it is supposed to under specified conditions. All aspects of the code are examined to check the quality of the application. The primary purpose of testing is to detect software failures so that defects may be uncovered and corrected. The test cases are designed in such way that scope of finding the bugs is maximum.

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

6.1 Testing Levels

There are various testing levels based on the specificity of the test

6.1.1 Unit testing

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

6.2.2 Integration testing

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

6.2.3 Functional testing

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input	: identified classes of valid input must be accepted.
Invalid Input	: identified classes of invalid input must be rejected.
Functions	: identified functions must be exercised.
Output	: identified classes of application outputs must be exercised.
Systems/Procedures	: interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

6.2.4 System Testing

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration-oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

6.2.5 White Box Testing

White box testing is a type of testing where the software tester is familiar with the inner workings, structure, and language of the software, or at the at least, knows what it is intended to do. It has a goal. It is employed to test regions that are inaccessible from a black box level. Structural testing and code-based testing are other names for this kind of testing. By understanding the application's code structure, including statements, branching, loops, and conditions, test cases may be created for white box testing. Testing professionals can spot possible flaws, logical mistakes, and areas for code optimization by looking at the code's internal workings. White box testing contributes to the quality and dependability of the application's code, making it a crucial step in the software testing process.

6.2.6 Black Box Testing

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box you cannot see into it. The test provides inputs and responds to outputs without considering how the software works.

Unit Testing

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases. Unit testing is a software testing technique that involves testing individual units or components of an application in isolation from the rest of the application. The purpose of unit testing is to validate that each unit of code is functioning correctly and meets its requirements. Unit testing typically involves writing automated tests that simulate input/output behavior for each unit and comparing the actual output to the expected output. By performing unit testing, developers can catch defects early in the development process, reduce the likelihood of regressions, and improve the overall quality of the code. Unit testing is an essential practice in agile and DevOps methodologies and is a key part of modern software development practices.

Test strategy and approach

Field testing will be performed manually and functional tests will be written in detail.

Test objectives

- All field entries must work properly.
- Pages must be activated from the identified link.
- The entry screen, messages and responses must not be delayed.

Features to be tested

- Verify that the entries are of the correct format
- No duplicate entries should be allowed
- All links should take the user to the correct page.

Integration Testing

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects. The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

Acceptance Testing

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements. Acceptance testing is a software testing technique that verifies whether a software application meets the business requirements and is acceptable for delivery to the end-users. This type of testing is usually conducted by the customers or end-users to ensure that the application meets their expectations and requirements.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

6.3 System Test Cases

A test case is a set of test data, preconditions, expected results and post conditions, developed for a test scenario to verify compliance with a specific requirement. I have designed and executed a few test cases to check if the project meets the functional requirements.

The testers need to focus on for the following: -

- Test with new data, rather than the original training data. If necessary, split your training set into two groups: one that does training, and one that does test. Better, obtain and use fresh data if you are able.
- Understand the architecture of the network as a part of the testing process. Testers won't necessarily understand how the neural network was constructed, but need to understand whether it meets requirements. And based on the measurements that they are testing, they may have to recommend a radically different approach, or admit the software is just not capable of doing what it was asked to do with confidence.
- Lines Segmentation is one the most important process in image preprocessing. Line segmentation divides images into lines. The main goal of line segmentation is to separate out the line of text from the image.
- Separating out the individual characters from the image which is already divided into lines of text in line segmentation. Here CNN distinguishes between character and non-character regions creating fixed size bounding boxes around characters to improve the accuracy.
- Binarization is the method of converting any grayscale image (multi-tone image) into black-white image (two-tone image). To perform the binarization process, first, find the threshold value of the grayscale and check whether a pixel has a particular gray value or not. If the gray value of the pixels is greater than the threshold, then those pixels are converted into white. Similarly, if the gray value of the pixels is lesser than the threshold, then those pixels are converted into black.

CHAPTER 7
RESULTS

SNAPSHOTS

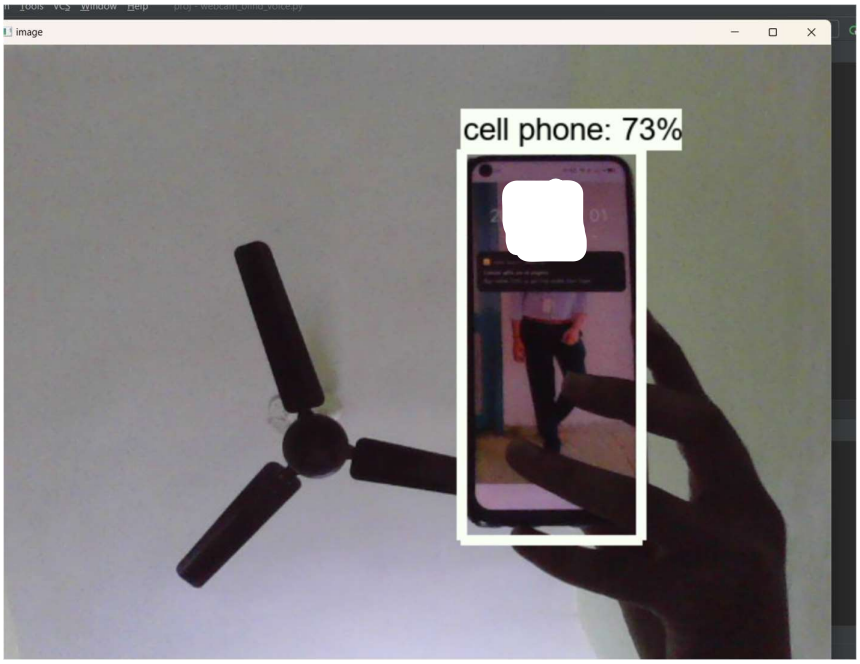

Fig 7.1: Detecting the cell phone and it's accuracy

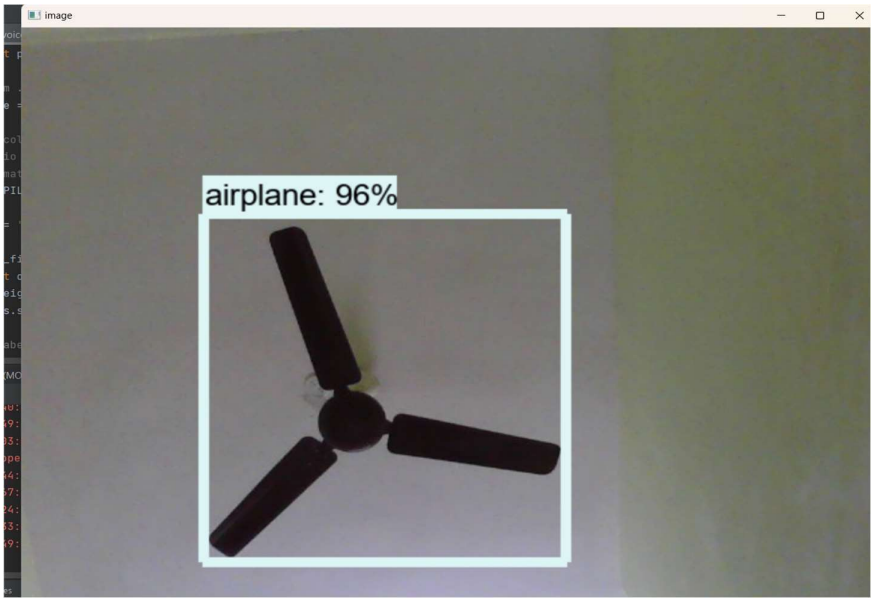

Fig 7.2 : Detecting the Fan and its accuracy

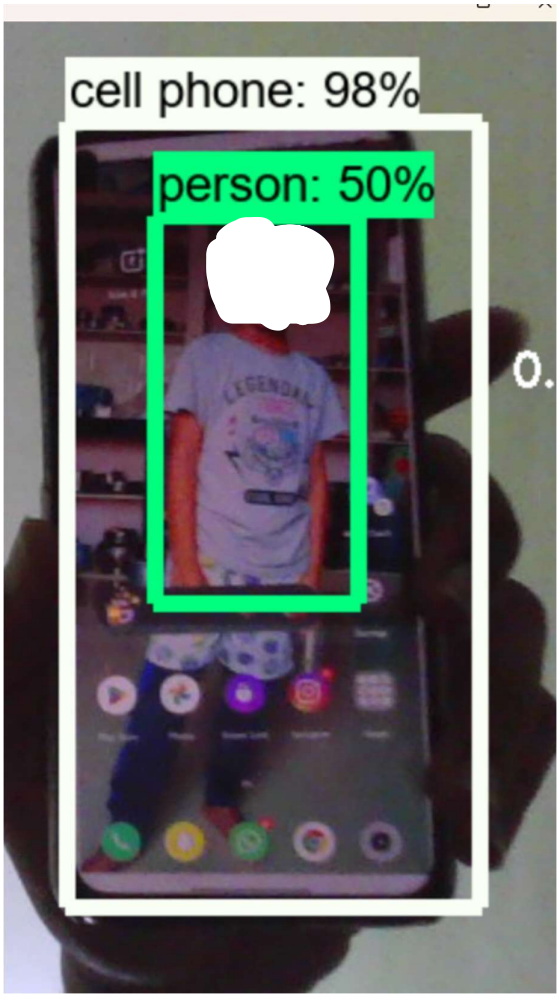

Fig 7.3: Detecting both the person and cell phone

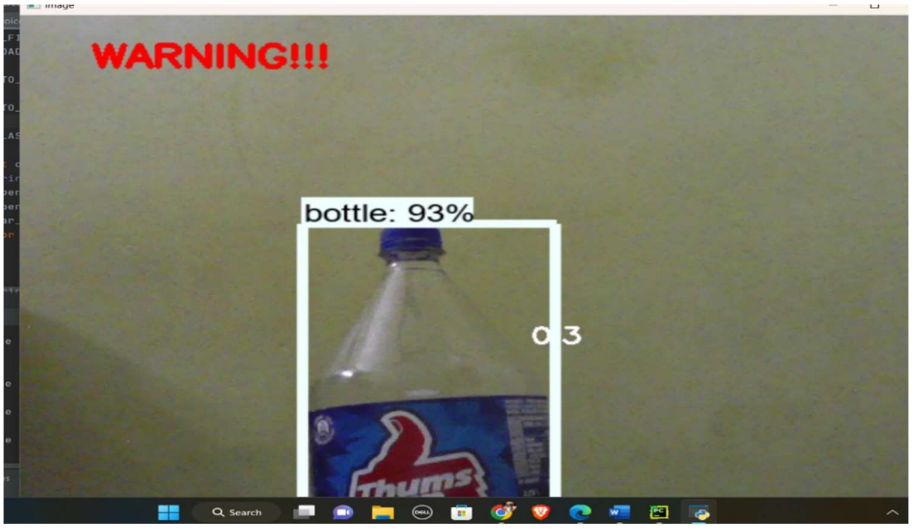

Fig 7.4: Detecting water bottle and showing warning as it is closer to camera

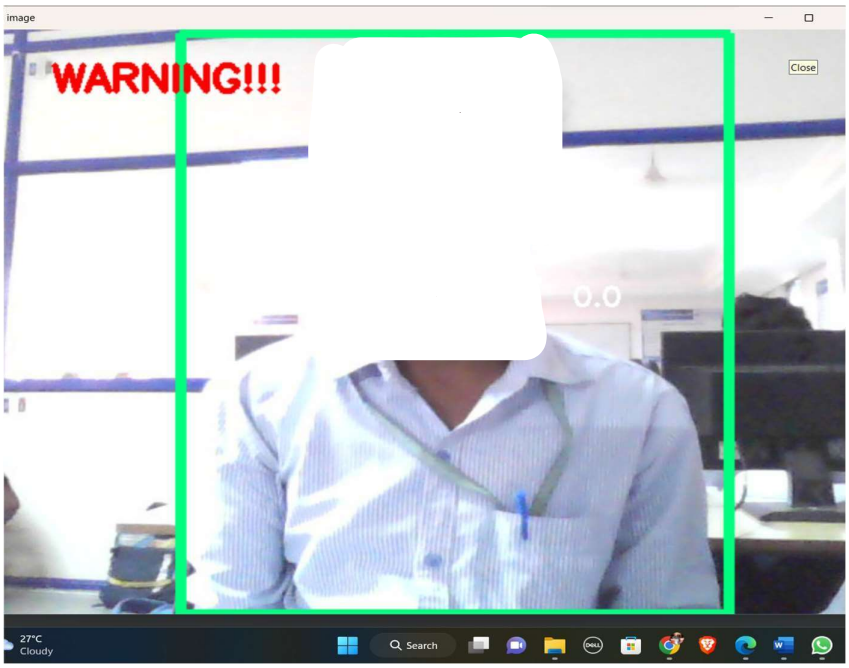

Fig 7.5: Showing warning When the person is very close to camera

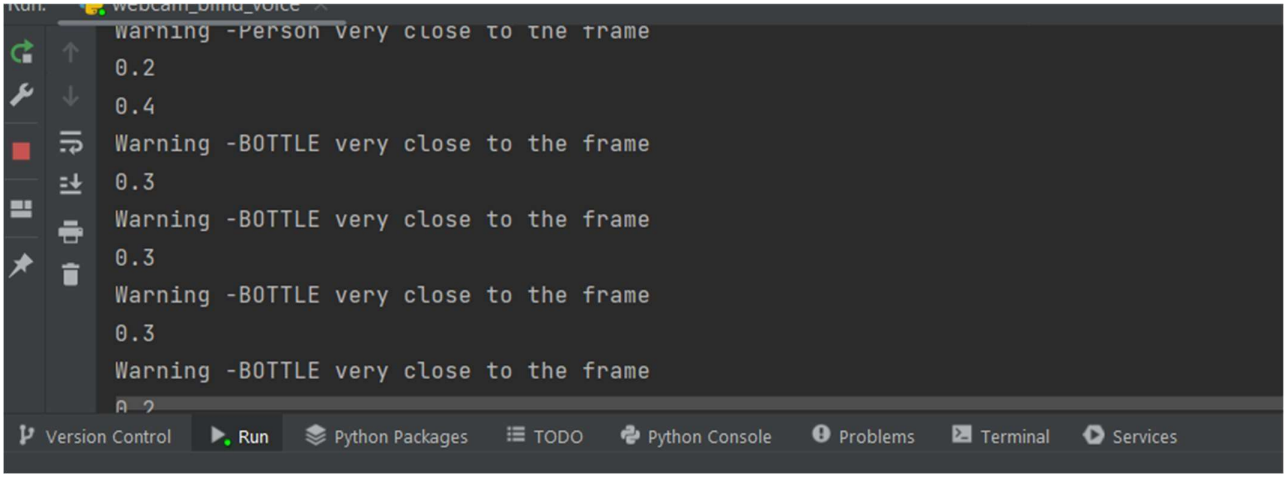

Fig 7.6: Console Output

CHAPTER-8

CONCLUSION AND FUTURE WORK

In this study, we tried to identify an object that was displayed in front of a webcam. The developed model was tested and trained using frameworks from the Tensor Flow object identification API. A good frames per second solution is necessary to solve the input and output issues that arise when reading a frame from a web camera. We focused on threading technology as a result, which significantly increases frames per second and subsequently reduces processing time for each item. The object detection box takes about 3-5 seconds to move over the next object in the video, even though the program correctly identifies each object in front of the image captured through the frame of the webcam of the system. Another potential area for future research is the development of more personalized and adaptive assistance systems. Machine learning algorithms can be used to learn the individual preferences and needs of the user, and adapt the system accordingly. This can include learning the user's preferred walking speed, or the types of objects they encounter most frequently, and adjusting the feedback accordingly to the things.

CHAPTER - 9
REFERENCES

[1] S. Antol, A.Agrawal, J.Lu, M.Mitchell, D.Batra, C. Lawrence Zitnick, and D. Parikh, vqa: Visual Question Answering", arXiv:1505.00468[cs.CL], pp. 2425-2433, 2015. K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual at

[2] J. Mao, W. Xu, Y. Yang, J. Wang, Z. Huang, and A. Yuille, "Deep captioning with multimodal recurrent neural networks (m-rnn)," arXiv preprint arXiv:1412.6632, 2014.

[3] A. Karpathy and L. Fei-Fei, "Deep visual-semantic alignments for generating image descriptions," arXiv:1412.2306 [cs.CV], pp. 3128– 3137, 2015.

[4] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell, "Long-term recurrent convolutional networks for visual recognition and description," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, no. 4, pp. 677– 691, 1 April 2017

[5] A. Farhadi, M. Hejrati, M. A. Sadeghi, P. Young, C. Rashtchian, J. Hockenmaier, and D. Forsyth, "Every picture tells a story: Generating sentences from images," Computer Vision – ECCV 2010, pp. 15–29, 2010.

[6] R. Socher, A. Karpathy, Q. V. Le, C. D. Manning, and A. Y. Ng, "Grounded compositional semantics for finding and describing images with sentences," Transactions of the Association for Computational Linguistics, vol. 2, pp. 207– 218, 2014

[7] A. Frome, G. S. Corrado, J. Shlens, S. Bengio, J. Dean, M. Ranzato, and T. Mikolov, "Devise: A deep visualsemantic embedding model," NIPS'13: Proceedings of the 26th International Conference on Neural Information Processing System, pp. 2121–2129, 2013.

[8] A. Frome, G. S. Corrado, J. Shlens, S. Bengio, J. Dean, M. Ranzato, and T. Mikolov, "Devise: A deep visualsemantic embedding model," NIPS'13: Proceedings of the 26th International Conference on Neural Information Processing System, pp. 2121–2129, 2013

[9] J. Lin and M. Sun, "A YOLO-Based Traffic Counting System," 2018 Conference on Technologies and Applications of Artificial Intelligence (TAAI), Taichung, 2018, pp. 82-85

[10] K. Zhang, C. Gao, L. Guo, et al., "Age group and gender estimation in the wild with deep RoR architecture," IEEE Access, vol. 5, pp. 22492–22503, 2017.

[11] Liu, X., et al.: AgeNet: deeply learned regressor and classifier for robust apparent age estimation. In: Proceedings of the IEEE International Conference on Computer Vision Workshops, pp. 16–24 (2015)

[12] Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. Adv. Neural Inf. Process. Syst. 25, 1097–1105 (2012)

[13] Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)

[14] Mandolandola, F.N., Han, S., Moskowitz, M.W., Ashraf, K., Dally, W.J., Keutzer, K.: SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5 MB model size. arXiv preprint arXiv:1602.07360 (2016)

[15] Chollet, F.: Xception: Machine Learning with depthwise separable convolutions. In: Proceedingsof the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1251–1258 (2017)

[16] Szegedy, C., et al.: Going deeper with convolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1–9 (2015)

[17] He, K., Gkioxari, G., Dollar, P., Girshick, R.: Mask R-CNN. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 2961–2969 (2017)

[18] Smith, P., Chen, C.: Transfer learning with deep CNNs for gender recognition and age estimation. In: 2018 IEEE International Conference on Big Data (Big Data), pp. 2564–2571. IEEE, December 2018

[19] W.-L. Chao, J.-Z. Liu, and J.-J. Ding, "Facial age estimation based on label-sensitive learning and age-oriented regression," Pattern Recognit., vol. 46, no. 3, pp. 628–641, Mar. 2013.

[20] T. Ahonen, A. Hadid, and M. Pietikainen, "Face description with local binary patterns: Application to face recognition," IEEE Trans. Pattern Anal. Mach. Intell., vol. 28, no. 12, pp. 2037–2041, Dec. 2006.

APPENDIX

WEB CAM BLIND .py

```
import os
import tarfile
import numpy as np
import pytesseract
import pyttsx3
import six.moves.urllib as urllib
import tensorflow as tf
import torch
from torch.autograd import Variable as V
from torch.nn import functional as F
from torchvision import transforms as trn

# from .engine import Engine
engine = pyttsx3.init()
from PIL import Image
arch = 'resnet18'
model_file = 'whole_%s_places365_python36.pth.tar' % arch
if not os.access(model_file, os.W_OK):
    weight_url = 'http://places2.csail.mit.edu/models_places365/' + model_file
    os.system('wget ' + weight_url)

#  =  label_map_util.create_category_index(categories)
pytesseract.pytesseract.tesseract_cmd = 'C:\\Program Files (x86)\\Tesseract-OCR\\tesseract'
from object_detection.utils import label_map_util
# /object_detection/'  m2
from object_detection.utils import visualization_utils as vis_util
MODEL_NAME = 'ssd_inception_v2_coco_2017_11_17'
```

```
MODEL_FILE = MODEL_NAME + '.tar.gz'
DOWNLOAD_BASE = 'http://download.tensorflow.org/models/object_detection/'

PATH_TO_CKPT = MODEL_NAME + '/frozen_inference_graph.pb'
PATH_TO_LABELS = os.path.join('data', 'mscoco_label_map.pbtxt')
NUM_CLASSES = 90
if not os.path.exists(MODEL_NAME + '/frozen_inference_graph.pb'):
    print('Downloading the model')
    opener = urllib.request.URLopener()
    opener.retrieve(DOWNLOAD_BASE + MODEL_FILE, MODEL_FILE)
    tar_file = tarfile.open(MODEL_FILE)
    for file in tar_file.getmembers():
        file_name = os.path.basename(file.name)
        if 'frozen_inference_graph.pb' in file_name:
            tar_file.extract(file, os.getcwd())
    print('Download complete')
else:
    print('Model already exists')

detection_graph = tf.Graph()
with detection_graph.as_default():
    od_graph_def = tf.compat.v1.GraphDef()
    with tf.io.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
        serialized_graph = fid.read()
        od_graph_def.ParseFromString(serialized_graph)
        tf.import_graph_def(od_graph_def, name='')

label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
categories = label_map_util.convert_label_map_to_categories(label_map,
max_num_classes=NUM_CLASSES , use_display_name=True)
category_index = label_map_util.create_category_index(categories)
url = 'http://10.67.208.240:8080//shot.jpg'
```

```
import cv2

cap = cv2.VideoCapture(0)

with detection_graph.as_default():
    with tf.compat.v1.Session(graph=detection_graph) as sess:
        ret = True
        while (ret):
            ret, image_np = cap.read()
            if cv2.waitKey(20) & 0xFF == ord('b'):
                cv2.imwrite('opencv' + '.jpg', image_np)

                model_file = 'whole_%s_places365_python36.pth.tar' % arch
                if not os.access(model_file, os.W_OK):
                    weight_url = 'http://places2.csail.mit.edu/models_places365/' + model_file
                    os.system('wget ' + weight_url)

                useGPU = 1
                if useGPU == 1:
                    model = torch.load(model_file)
                else:
                    model = torch.load(model_file, map_location=lambda storage,
                                                                loc: storage) # model trained in GPU could be deployed in
CPU machine like this!
                model.eval()
                centre_crop = trn.Compose([
                    trn.Resize((256, 256)),
                    trn.CenterCrop(224),
                    trn.ToTensor(),
                    trn.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
                ])
```

```
file_name =  'categories_places365.txt'
if not os.access(file_name, os.W_OK):
    synset_url =
'https://raw.githubusercontent.com/csailvision/places365/master/categories_places365.txt'
    os.system('wget ' + synset_url)
classes = list()
with open(file_name) as class_file:
    for line in class_file:
        classes.append(line.strip().split(' ')[0][3:])
classes = tuple(classes)

img_name = 'opencv.jpg'
if not os.access(img_name, os.W_OK):
    img_url = 'http://places.csail.mit.edu/demo/' + img_name
    os.system('wget ' + img_url)

img = Image.open(img_name)
input_img = V(centre_crop(img).unsqueeze(0), volatile=True)

logit = model.forward(input_img)
h_x = F.softmax(logit, 1).data.squeeze()
probs, idx = h_x.sort(0, True)

print('POSSIBLE SCENES ARE: ' + img_name)
engine.say("Possible Scene may be")
engine.say(img_name)

for i in range(0, 5):
    engine.say(classes[idx[i]])
    print('{}'.format(classes[idx[i]]))
```

```
# Expand dimensions since the model expects images to have shape: [1, None, None, 3]
image_np_expanded = np.expand_dims(image_np, axis=0)
image_tensor = detection_graph.get_tensor_by_name('image_tensor:0')
# Each box represents a part of the image where a particular object was detected.
boxes = detection_graph.get_tensor_by_name('detection_boxes:0')
# Each score represent how level of confidence for each of the objects.
# Score is shown on the result image, together with the class label.
scores = detection_graph.get_tensor_by_name('detection_scores:0')
classes = detection_graph.get_tensor_by_name('detection_classes:0')
num_detections = detection_graph.get_tensor_by_name('num_detections:0')
# Actual detection.
(boxes, scores, classes, num_detections) = sess.run(
    [boxes, scores, classes, num_detections],
    feed_dict={image_tensor: image_np_expanded})

# Visualization of the results of a detection.
if cv2.waitKey(2) & 0xFF == ord('a'):
    vis_util.vislize_boxes_and_labels_on_image_array(
        image_np,
        np.squeeze(boxes),
        np.squeeze(classes).astype(np.int32),
        np.squeeze(scores),
        category_index,
        use_normalized_coordinates=True,
        line_thickness=8)
else:
    vis_util.visualize_boxes_and_labels_on_image_array(
        image_np,
        np.squeeze(boxes),
        np.squeeze(classes).astype(np.int32),
        np.squeeze(scores),
        category_index,
```

```
            use_normalized_coordinates=True,
            line_thickness=8)
        if cv2.waitKey(2) & 0xFF == ord('r'):
            text = pytesseract.image_to_string(image_np)
            print(text)
            engine.say(text)
            engine.runAndWait()

        for i, b in enumerate(boxes[0]):

            # car  bus  truck
            if classes[0][i] == 3 or classes[0][i] == 6 or classes[0][i] == 8:
                if scores[0][i] >= 0.5:
                    mid_x = (boxes[0][i][1] + boxes[0][i][3]) / 2
                    mid_y = (boxes[0][i][0] + boxes[0][i][2]) / 2
                    apx_distance = round(((1 - (boxes[0][i][3] - boxes[0][i][1])) ** 4), 1)
                    cv2.putText(image_np, '{}'.format(apx_distance), (int(mid_x * 800), int(mid_y * 450)),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255), 2)

                    if apx_distance <= 0.5:
                        if mid_x > 0.3 and mid_x < 0.7:
                            cv2.putText(image_np, 'WARNING!!!', (50, 50), cv2.FONT_HERSHEY_SIMPLEX,
1.0,
                                        (0, 0, 255), 3)
                            print("Warning -Vehicles Approaching")
                            engine.say("Warning -Vehicles Approaching")
                            engine.runAndWait()

            if classes[0][i] == 44:
                if scores[0][i] >= 0.5:
                    mid_x = (boxes[0][i][1] + boxes[0][i][3]) / 2
                    mid_y = (boxes[0][i][0] + boxes[0][i][2]) / 2
```

```
            apx_distance = round(((1 - (boxes[0][i][3] - boxes[0][i][1])) ** 4), 1)
            cv2.putText(image_np, '{}'.format(apx_distance), (int(mid_x * 800), int(mid_y * 450)),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255), 2)
            print(apx_distance)
            engine.say(apx_distance)
            engine.say("units")
            engine.say("BOTTLE IS AT A SAFER DISTANCE")

            if apx_distance <= 0.5:
                if mid_x > 0.3 and mid_x < 0.7:
                    cv2.putText(image_np, 'WARNING!!!', (50, 50), cv2.FONT_HERSHEY_SIMPLEX,
1.0,
                                (0, 0, 255), 3)
                    print("Warning -BOTTLE very close to the frame")
                    engine.say("Warning -BOTTLE very close to the frame")
                    engine.runAndWait()
        if classes[0][i] == 1:
            if scores[0][i] >= 0.5:
                mid_x = (boxes[0][i][1] + boxes[0][i][3]) / 2
                mid_y = (boxes[0][i][0] + boxes[0][i][2]) / 2
                apx_distance = round(((1 - (boxes[0][i][3] - boxes[0][i][1])) ** 4), 1)
                cv2.putText(image_np, '{}'.format(apx_distance), (int(mid_x * 800), int(mid_y * 450)),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255), 2)
                print(apx_distance)
                engine.say(apx_distance)
                engine.say("units")
                engine.say("Person is AT A SAFER DISTANCE")

                if apx_distance <= 0.5:
                    if mid_x > 0.3 and mid_x < 0.7:
                        cv2.putText(image_np, 'WARNING!!!', (50, 50), cv2.FONT_HERSHEY_SIMPLEX,
1.0,
```

```
                    (0, 0, 255), 3)
                print("Warning -Person very close to the frame")
                engine.say("Warning -Person very close to the frame")
                engine.runAndWait()

    #     plt.figure(figsize=IMAGE_SIZE)
    #     plt.imshow(image_np)
    # cv2.imshow('IPWebcam',image_np)
    cv2.imshow('image', cv2.resize(image_np, (1024, 768)))
    if cv2.waitKey(2) & 0xFF == ord('t'):
        cv2.destroyAllWindows()
        cap.release()
        break


# {1: {'id': 1, 'name': 'person'}, 2: {'id': 2, 'name': 'bicycle'}, 3: {'id': 3, 'name': 'car'}, 4: {'id': 4, 'name': 'motorcycle'}, 5: {'id': 5, 'name': 'airplane'}, 6: {'id': 6, 'name': 'bus'}, 7: {'id': 7, 'name': 'train'}, 8: {'id': 8, 'name': 'truck'}, 9: {'id': 9, 'name': 'boat'}, 10: {'id': 10, 'name': 'traffic light'}, 11: {'id': 11, 'name': 'fire hydrant'}, 13: {'id': 13, 'name': 'stop sign'}, 14: {'id': 14, 'name': 'parking meter'}, 15: {'id': 15, 'name': 'bench'}, 16: {'id': 16, 'name': 'bird'}, 17: {'id': 17, 'name': 'cat'}, 18: {'id': 18, 'name': 'dog'}, 19: {'id': 19, 'name': 'horse'}, 20: {'id': 20, 'name': 'sheep'}, 21: {'id': 21, 'name': 'cow'}, 22: {'id': 22, 'name': 'elephant'}, 23: {'id': 23, 'name': 'bear'}, 24: {'id': 24, 'name': 'zebra'}, 25: {'id': 25, 'name': 'giraffe'}, 27: {'id': 27, 'name': 'backpack'}, 28: {'id': 28, 'name': 'umbrella'}, 31: {'id': 31, 'name': 'handbag'}, 32: {'id': 32, 'name': 'tie'}, 33: {'id': 33, 'name': 'suitcase'}, 34: {'id': 34, 'name': 'frisbee'}, 35: {'id': 35, 'name': 'skis'}, 36: {'id': 36, 'name': 'snowboard'}, 37: {'id': 37, 'name': 'sports ball'}, 38: {'id': 38, 'name': 'kite'}, 39: {'id': 39, 'name': 'baseball bat'}, 40: {'id': 40, 'name': 'baseball glove'}, 41: {'id': 41, 'name': 'skateboard'}, 42: {'id': 42, 'name': 'surfboard'}, 43: {'id': 43, 'name': 'tennis racket'}, 44: {'id': 44, 'name': 'bottle'}, 46: {'id': 46, 'name': 'wine glass'}, 47: {'id': 47, 'name': 'cup'}, 48: {'id': 48, 'name': 'fork'}, 49: {'id': 49, 'name': 'knife'}, 50: {'id': 50, 'name': 'spoon'}, 51: {'id': 51, 'name': 'bowl'}, 52: {'id': 52, 'name': 'banana'}, 53: {'id': 53, 'name': 'apple'}, 54: {'id': 54, 'name': 'sandwich'}, 55: {'id': 55, 'name': 'orange'}, 56: {'id': 56, 'name': 'broccoli'}, 57: {'id': 57, 'name': 'carrot'}, 58: {'id': 58, 'name': 'hot dog'}, 59: {'id': 59, 'name': 'pizza'}, 60: {'id': 60, 'name': 'donut'}, 61: {'id': 61, 'name': 'cake'}, 62: {'id': 62, 'name': 'chair'}, 63: {'id': 63, 'name': 'couch'}, 64: {'id': 64, 'name': 'potted plant'}, 65: {'id': 65, 'name': 'bed'}, 67: {'id': 67, 'name': 'dining table'}, 70: {'id': 70, 'name': 'toilet'}, 72: {'id': 72, 'name': 'tv'}, 73: {'id': 73, 'name': 'laptop'}, 74: {'id': 74, 'name': 'mouse'}, 75: {'id': 75, 'name': 'remote'}, 76: {'id': 76, 'name': 'keyboard'}, 77: {'id': 77, 'name': 'cell phone'}, 78: {'id': 78, 'name': 'microwave'}, 79: {'id': 79, 'name': 'oven'}, 80: {'id': 80, 'name': 'toaster'}, 81: {'id': 81, 'name': 'sink'}, 82: {'id': 82, 'name': 'refrigerator'}, 84: {'id': 84, 'name': 'book'}, 85: {'id': 85, 'name': 'clock'}, 86: {'id': 86, 'name': 'vase'}, 87: {'id': 87, 'name': 'scissors'}, 88: {'id': 88, 'name': 'teddy bear'}, 89: {'id': 89, 'name': 'hair drier'}, 90: {'id': 90, 'name': 'toothbrush'}}
```

open("yolo-coco/coco.names").read().strip().split("\n")

Object Detection Webcam .py

```
import numpy as np
import os
import six.moves.urllib as urllib
import sys
import tarfile
import tensorflow as tf
import zipfile
import cv2
from collections import defaultdict
from io import StringIO
from matplotlib import pyplot as plt
from PIL import Image
from utils import label_map_util
from utils import visualization_utils as vis_util

# Define the video stream
cap = cv2.VideoCapture(0) # Change only if you have more than one webcams
#object_detection/'
# What model to download.
# Models can bee found here: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md
MODEL_NAME = 'mask_rcnn_inception_resnet_v2_atrous_coco_2018_01_28'
MODEL_FILE = MODEL_NAME + '.tar.gz'
#DOWNLOAD_BASE = 'http://download.tensorflow.org/models/   m1

# Path to frozen detection graph. This is the actual model that is used for the object detection.
```

```
PATH_TO_CKPT = MODEL_NAME + '/frozen_inference_graph.pb'

# List of the strings that is used to add correct label for each box.
PATH_TO_LABELS = os.path.join('data', 'mscoco_map.pbtxt')

# Number of classes to detect
NUM_CLASSES = 90
# Download Model
if not os.path.exists(os.path.join(os.getcwd(), MODEL_FILE)):
    print("Downloading model")
    opener =  urllib.request.URLopener()
    opener.retrieve(DOWNLOAD_BASE + MODEL_FILE,  MODEL_FILE)
    tar_file = tarfile.open(MODEL_FILE)
    for file in tar_file.getmembers():
        file_name = os.path.basename(file.name)
        if 'frozen_inference_graph.pb' in file_name:
            tar_file.extract(file, os.getcwd())


# Load a (frozen) Tensorflow model into memory.
detection_graph = tf.Graph()
with detection_graph.as_default():
    od_graph_def = tf.compat.v1.GraphDef()
    with tf.io.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
        serialized_graph = fid.read()
        od_graph_def.ParseFromString(serialized_graph)
        tf.import_graph_def(od_graph_def, name='')


# Loading label map
```

```
# Label maps map indices to category names, so that when our convolution network predicts `5`, we know that this corresponds to `airplane`. Here we use internal utility functions, but anything that returns a dictionary mapping integers to appropriate string labels would be fine
label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
categories = label_map_util.convert_label_map_to_categories(
    label_map, max_num_classes=NUM_CLASSES, use_display_name=True)
category_index =

#'data', 'mscoco_label_map.pbtxt'
# Helper code
def load_image_into_numpy_array(image):
    (im_width, im_height) = image.size
    return np.array(image.getdata()).reshape(
        (im_height, im_width, 3)).astype(np.uint8)



# Detection
with detection_graph.as_default():
    with tf.compat.v1.Session(graph=detection_graph) as sess:
        while True:
            # Read frame from camera
            ret, image_np = cap.read()
            # Expand dimensions since the model expects images to have shape: [1, None, None, 3]
            image_np_expanded = np.expand_dims(image_np, axis=0)
            # Extract image tensor
            image_tensor = detection_graph.get_tensor_by_name('image_tensor:0')
            # Extract detection boxes
            boxes = detection_graph.get_tensor_by_name('detection_boxes:0')
            # Extract detection scores
            scores = detection_graph.get_tensor_by_name('detection_scores:0')
            # Extract detection classes
            classes = detection_graph.get_tensor_by_name('detection_classes:0')
```

```
        # Extract number of detectionsd
        num_detections = detection_graph.get_tensor_by_name(
            'num_detections:0')
        # Actual detection.
        (boxes, scores, classes, num_detections) = sess.run(
            [boxes, scores, classes, num_detections],
            feed_dict={image_tensor: image_np_expanded})
        # Visualization of the results of a detection.
        vis_util.visualize_boxes_and_labels_on_image_array(
            image_np,
            np.squeeze(boxes),
            np.squeeze(classes).astype(np.int32),
            np.squeeze(scores),
            category_index,
            use_normalized_coordinates=True,
            line_thickness=8)
        print(label_map)

        # Display output # label_map_util.create_category_index(categories)
        cv2.imshow('object detection', cv2.resize(image_np, (800, 600)))

        if cv2.waitKey(25) & 0xFF == ord('q'):
            cv2.destroyAllWindows()
            break
```

Label_map_util.py

```
# Copyright 2017 The TensorFlow Authors. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
```

```
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

"""Label map utility functions."""
import logging
import tensorflow as tf
from google.protobuf import text_format
from object_detection.protos import string_int_label_map_pb2

def _validate_label_map(label_map):
  """Checks if a label map is valid.

  Args:
    label_map: StringIntLabelMap to validate.

  Raises:
    ValueError: if label map is invalid.
  """
  for item in label_map.item:
    if item.id < 0:
      raise ValueError('Label map ids should be >= 0.')
    if (item.id == 0 and item.name != 'background' and
        item.display_name != 'background'):
      raise ValueError('Label map id 0 is reserved for the background label')
def create_category_index(categories):
```

```
  """Creates dictionary of COCO compatible categories keyed by category id.

  Args:
    categories: a list of dicts, each of which has the following keys:
      'id': (required) an integer id uniquely identifying this category.
      'name': (required) string representing category name
        e.g., 'cat', 'dog', 'pizza'.

  Returns:
    category_index: a dict containing the same entries as categories, but keyed
      by the 'id' field of each category.
  """
  category_index = {}
  for cat in categories:
    category_index[cat['id']] = cat
  return category_index


def get_max_label_map_index(label_map):
  """Get maximum index in label map.

  Args:
    label_map: a StringIntLabelMapProto

  Returns:
    an integer
  """
  return max([item.id for item in label_map.item])
def convert_label_map_to_categories(label_map,
                                    max_num_classes,
                                    use_display_name=True):
  """Given label map proto returns categories list compatible with eval.
```

```
This function converts label map proto and returns a list of dicts, each of
which has the following keys:
  'id': (required) an integer id uniquely identifying this category.
  'name': (required) string representing category name
    e.g., 'cat', 'dog', 'pizza'.
We only allow class into the list if its id-label_id_offset is
between 0 (inclusive) and max_num_classes (exclusive).
If there are several items mapping to the same id in the label map,
we will only keep the first one in the categories list.

Args:
  label_map: a StringIntLabelMapProto or None. If None, a default categories
    list is created with max_num_classes categories.
  max_num_classes: maximum number of (consecutive) label indices to include.
  use_display_name: (boolean) choose whether to load 'display_name' field as
  category name. If  False or if the display_name field  does  not  exist,  uses
  'name' field as category names instead.

Returns:
  categories: a list of dictionaries representing all possible categories.
"""
categories = []
list_of_ids_already_added = []
if not label_map:
  label_id_offset = 1
  for class_id in range(max_num_classes):
    categories.append({
        'id': class_id + label_id_offset,
        'name': 'category_{}'.format(class_id + label_id_offset)
    })
  return categories
for item in label_map.item:
```

```
    if not 0 < item.id <= max_num_classes:
      logging.info(
          'Ignore item %d since it falls outside of requested '
          'label range.', item.id)
      continue
    if use_display_name and item.HasField('display_name'):
      name = item.display_name
    else:
      name = item.name
    if item.id not in list_of_ids_already_added:
      list_of_ids_already_added.append(item.id)
      categories.append({'id': item.id, 'name': name})
  return categories


def load_labelmap(path):
  """Loads label map proto.

  Args:
    path: path to StringIntLabelMap proto text file.
  Returns:
    a StringIntLabelMapProto
  """
  with tf.io.gfile.GFile(path, 'r') as fid:
    label_map_string = fid.read()
    label_map = string_int_label_map_pb2.StringIntLabelMap()
    try:
      text_format.Merge(label_map_string, label_map)
    except text_format.ParseError:
    label_map.ParseFromString(label_map_string)
  _validate_label_map(label_map)
  return label_map
```

```
def get_label_map_dict(label_map_path,
                       use_display_name=False,
                       fill_in_gaps_and_background=False):
  """Reads a label map and returns a dictionary of label names to id.

  Args:
    label_map_path: path to StringIntLabelMap proto text file.
    use_display_name: whether to use the label map items' display names as keys.
    fill_in_gaps_and_background: whether to fill in gaps and background with
    respect to the id field in the proto. The id: 0 is reserved for the
    'background' class and will be added if it is missing. All other missing
    ids in range(1, max(id)) will be added with a dummy class name
    ("class_<id>") if they are missing.

  Returns:
    A dictionary mapping label names to id.

  Raises:
    ValueError: if fill_in_gaps_and_background and label_map has non-integer or
    negative values.
  """
  label_map = load_labelmap(label_map_path)
  label_map_dict = {}
  for item in label_map.item:
    if use_display_name:
    label_map_dict[item.display_name] = item.id
    else:
      label_map_dict[item.name] = item.id

  if fill_in_gaps_and_background:
    values = set(label_map_dict.values())
```

```
  if 0 not in values:
    label_map_dict['background'] = 0
  if not all(isinstance(value, int) for value in values):
    raise ValueError('The values in label map must be integers in order to'
                     'fill_in_gaps_and_background.')
  if not all(value >= 0 for value in values):
    raise ValueError('The values in the label map must be positive.')

  if len(values) != max(values) + 1:
    # there are gaps in the labels, fill in gaps.
    for value in range(1, max(values)):
      if value not in values:
        # TODO(rathodv): Add a prefix 'class_' here once the tool to generate
        # teacher annotation adds this prefix in the data.
        label_map_dict[str(value)] = value

  return label_map_dict


def create_categories_from_labelmap(label_map_path, use_display_name=True):
  """Reads a label map and returns categories list compatible with eval.

  This function converts label map proto and returns a list of dicts, each of
  which has the following keys:
    'id': an integer id uniquely identifying this category.
    'name': string representing category name e.g., 'cat', 'dog'.

  Args:
    label_map_path: Path to `StringIntLabelMap` proto text file.
    use_display_name: (boolean) choose whether to load 'display_name' field
    as category name. If False or if the display_name field does not exist,
```

```
    uses 'name' field as category names instead.

  Returns:
    categories: a list of dictionaries representing all possible categories.
  """
  label_map = load_labelmap(label_map_path)
  max_num_classes = max(item.id for item in label_map.item)
  return convert_label_map_to_categories(label_map, max_num_classes,
                                         use_display_name)


def create_category_index_from_labelmap(label_map_path, use_display_name=True):
  """Reads a label map and returns a category index.

  Args:
    label_map_path: Path to `StringIntLabelMap` proto text file.
    use_display_name: (boolean) choose whether to load 'display_name' field
    as category name.  If False or if the display_name field does not exist,
    uses 'name' field as category names instead.

  Returns:
    A category index, which is a dictionary that maps integer ids to dicts
    containing categories, e.g.
    {1: {'id': 1, 'name': 'dog'}, 2: {'id': 2, 'name': 'cat'}, ...}
  """
  categories = create_categories_from_labelmap(label_map_path, use_display_name)
  return create_category_index(categories)


def create_class_agnostic_category_index():
  """Creates a category index with a single `object` class."""
  return {1: {'id': 1, 'name': 'object'}}
```