**Sign Language Prediction Using CNN**

**Abstract**

Inability to speak is considered to be true disability. People with this disability use different modes to communicate with others, there are number of methods available for their communication one such common method of communication  is  sign  language. Developing sign language application for deaf people can be very important, as they’ll be able to communicate easily with even those who don’t understand sign language. Our project aims at taking the basic step in bridging the communication gap between normal people, deaf and dumb people using sign language. The main focus of this work is to create a vision based system to identify sign language gestures from the video sequences. The reason for choosing a system based on vision relates to the fact that it provides a simpler and more intuitive way of communication between a human and a computer. In this report, 5 different  gestures  have  been  considered. We used the following approach for the classification of sign language gestures:­ Video sequences contain both the temporal as well as the spatial features. So we have used CNN models to train both the temporal as well as the spatial features. To train the model on the spatial features of the video sequences we have used CNN model which is a deep CNN(convolutional neural net). Trained CNN model was used to make predictions for individual frames to obtain a sequence of predictions or pool layer outputs for each video. Now this sequence of prediction or pool layer outputs was given to RNN to train on the temporal features. We have created our own data set used consists of Indian sign Language(ISA) Gestures, with around 300 images belonging to 5 gestures categories.

**Problem statement**

Inability to speak is considered to be true disability. People with this disability use different modes to communicate with others, there are number of methods available for their communication one such common method of communication  is  sign  language. Developing sign language application for deaf people can be very important, as they’ll be able to communicate easily with even those who don’t understand sign language.

**Working Domain: Machine Learning:**

Machine learning (ML) is the scientific study of algorithms and statistical models that computer systems use to perform a specific task without using explicit instructions, relying on patterns and inference instead. It is seen as a subset of artificial intelligence. Machine learning algorithms build a mathematical model based on sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to perform the task. Machine learning algorithms are used in a wide variety of applications, such as email filtering and computer vision, where it is difficult or infeasible to develop a conventional algorithm for effectively performing the task.Machine learning is closely related to computational statistics, which focuses on making predictions using computers. The study of mathematical optimization delivers methods, theory and application domains to the field of machine learning. Data mining is a field of study within machine learning, and focuses on exploratory data analysis through unsupervised learning. In its application across business problems, machine learning is also referred to as predictive analytics.

**Overview**

The name machine learning was coined in 1959 by Arthur Samuel. Tom M. Mitchell provided a widely quoted, more formal definition of the algorithms studied in the machine learning field: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E." This definition of the tasks in which machine learning is concerned offers a fundamentally operational definition rather than defining the field in cognitive terms. This follows Alan Turing's proposal in his paper "Computing Machinery and Intelligence", in which the question "Can machines think?" is replaced with the question "Can machines do what we (as thinking entities) can do?". In Turing's proposal the various characteristics that could be possessed by a thinking machine and the various implications in constructing one are exposed.

**Machine learning tasks**

A support vector machine is a supervised learning model that divides the data into regions separated by a linear boundary. Here, the linear boundary divides the black circles from the white.

Machine learning tasks are classified into several broad categories. In supervised learning, the algorithm builds a mathematical model from a set of data that contains both the inputs and the desired outputs. For example, if the task were determining whether an image contained a certain object, the training data for a supervised learning algorithm would include images with and without that object (the input), and each image would have a label (the output) designating whether it contained the object. In special cases, the input may be only partially available, or restricted to special feedback.[clarification needed] Semi-supervised learning algorithms develop mathematical models from incomplete training data, where a portion of the sample input doesn't have labels.Classification algorithms and regression algorithms are types of supervised learning. Classification algorithms are used when the outputs are restricted to a limited set of values. For a classification algorithm that filters emails, the input would be an incoming email, and the output would be the name of the folder in which to file the email. For an algorithm that identifies spam emails, the output would be the prediction of either "spam" or "not spam", represented by the Boolean values true and false. Regression algorithms are named for their continuous outputs, meaning they may have any value within a range. Examples of a continuous value are the temperature, length, or price of an object.In unsupervised learning, the algorithm builds a mathematical model from a set of data which contains only inputs and no desired output labels. Unsupervised learning algorithms are used to find structure in the data, like grouping or clustering of data points. Unsupervised learning can discover patterns in the data, and can group the inputs into categories, as in feature learning. Dimensionality reduction is the process of reducing the number of "features", or inputs, in a set of data.

Active learning algorithms access the desired outputs (training labels) for a limited set of inputs based on a budget, and optimize the choice of inputs for which it will acquire training labels. When used interactively, these can be presented to a human user for labeling. Reinforcement learning algorithms are given feedback in the form of positive or negative reinforcement in a dynamic environment, and are used in autonomous vehicles or in learning to play a game against a human opponent. Other specialized algorithms in machine learning include topic modeling, where the computer program is given a set of natural language documents and finds other documents that cover similar topics. Machine learning algorithms can be used to find the unobservable probability density function in density estimation problems. Meta learning algorithms learn their own inductive bias based on previous experience. In developmental robotics, robot learning algorithms generate their own sequences of learning experiences, also known as a curriculum, to cumulatively acquire new skills through self-guided exploration and social interaction with humans. These robots use guidance mechanisms such as active learning, maturation, motor synergies, and imitation.[clarification needed]

**History and relationships to other fields**

Arthur Samuel, an American pioneer in the field of computer gaming and artificial intelligence, coined the term "Machine Learning" in 1959 while at IBM. A representative book of the machine learning research during 1960s was the Nilsson's book on Learning Machines, dealing mostly with machine learning for pattern classification. The interest of machine learning related to pattern recognition continued during 1970s, as described in the book of Duda and Hart in 1973. In 1981 a report was given on using teaching strategies so that a neural network learns to recognize 40 characters (26 letters, 10 digits, and 4 special symbols) from a computer terminal. As a scientific endeavor, machine learning grew out of the quest for artificial intelligence. Already in the early days of AI as an academic discipline, some researchers were interested in having machines learn from data. They attempted to approach the problem with various symbolic methods, as well as what were then termed "neural networks"; these were mostly perceptrons and other models that were later found to be reinventions of the generalized linear models of statistics. Probabilistic reasoning was also employed, especially in automated medical diagnosis.

However, an increasing emphasis on the logical, knowledge-based approach caused a rift between AI and machine learning. Probabilistic systems were plagued by theoretical and practical problems of data acquisition and representationBy 1980, expert systems had come to dominate AI, and statistics was out of favor. Work on symbolic/knowledge-based learning did continue within AI, leading to inductive logic programming, but the more statistical line of research was now outside the field of AI proper, in pattern recognition and information retrieval. 755 Neural networks research had been abandoned by AI and computer science around the same time. This line, too, was continued outside the AI/CS field, as "connectionism", by researchers from other disciplines including Hopfield, Rumelhart and Hinton. Their main success came in the mid-1980s with the reinvention of backpropagation.

Machine learning, reorganized as a separate field, started to flourish in the 1990s. The field changed its goal from achieving artificial intelligence to tackling solvable problems of a practical nature. It shifted focus away from the symbolic approaches it had inherited from AI, and toward methods and models borrowed from statistics and probability theory. It also benefited from the increasing availability of digitized information, and the ability to distribute it via the Internet.

**Relation to data mining**

Machine learning and data mining often employ the same methods and overlap significantly, but while machine learning focuses on prediction, based on known properties learned from the training data, data mining focuses on the discovery of (previously) unknown properties in the data (this is the analysis step of knowledge discovery in databases). Data mining uses many machine learning methods, but with different goals; on the other hand, machine learning also employs data mining methods as "unsupervised learning" or as a preprocessing step to improve learner accuracy. Much of the confusion between these two research communities (which do often have separate conferences and separate journals, ECML PKDD being a major exception) comes from the basic assumptions they work with: in machine learning, performance is usually evaluated with respect to the ability to reproduce known knowledge, while in knowledge discovery and data mining (KDD) the key task is the discovery of previously unknown knowledge. Evaluated with respect to known knowledge, an uninformed (unsupervised) method will easily be outperformed by other supervised methods, while in a typical KDD task, supervised methods cannot be used due to the unavailability of training data.

**Relation to optimization**

Machine learning also has intimate ties to optimization: many learning problems are formulated as minimization of some loss function on a training set of examples. Loss functions express the discrepancy between the predictions of the model being trained and the actual problem instances (for example, in classification, one wants to assign a label to instances, and models are trained to correctly predict the pre-assigned labels of a set of examples). The difference between the two fields arises from the goal of generalization: while optimization algorithms can minimize the loss on a training set, machine learning is concerned with minimizing the loss on unseen samples.

**Relation to statistics**

Machine learning and statistics are closely related fields in terms of methods, but distinct in their principal goal: statistics draws population inferences from a sample, while machine learning finds generalizable predictive patterns. According to Michael I. Jordan, the ideas of machine learning, from methodological principles to theoretical tools, have had a long pre-history in statistics. He also suggested the term data science as a placeholder to call the overall field.

**Introduction**

The aim of this system is to elevate people withhearing disability and help them socialize with commonpeople. It is a form of non-verbal communication. Signlanguage is the structured form as each gesture represents aunique element or a character. With the advent ofadvancement in science and engineering many researchersare working on different methodologies that could take theevent of human computer interaction to a much higher extent.The computer is trained in such a way that it could translatethe sign to text for static as well as dynamic frames. Thesystem is designed and implemented to recognize signlanguage gestures. The signs are captured using web cam andare pre-processed. In pre-processing stage, we usebackground subtraction to eliminate the background whichmakes this system to adapt to any dynamic background. Themain difficulty while implementing in software based is thatthe image must be properly captured and filtered.

**Literature Survey**

I n the recent years, there has been tremendous research on the hand sign language gesture recognition. The technology for gesture recognition is given below.

2.1  Vision­based In vision­based methods computer camera is the input device for observing the information of hands or fingers. The Vision Based methods require only a camera, thus realizing a natural interaction between humans and computers without the use of any extra devices. These systems tend to complement biological vision by describing artificial vision systems that are implemented in software and/or hardware. This poses a challenging problem as these systems need to be background invariant, lighting insensitive, person and camera independent to achieve real time performance. Moreover, such systems must be optimized to meet the requirements, including accuracy and robustness.

The  vision  based  hand  gesture  recognition  system  is  shown  in  fig.­­:

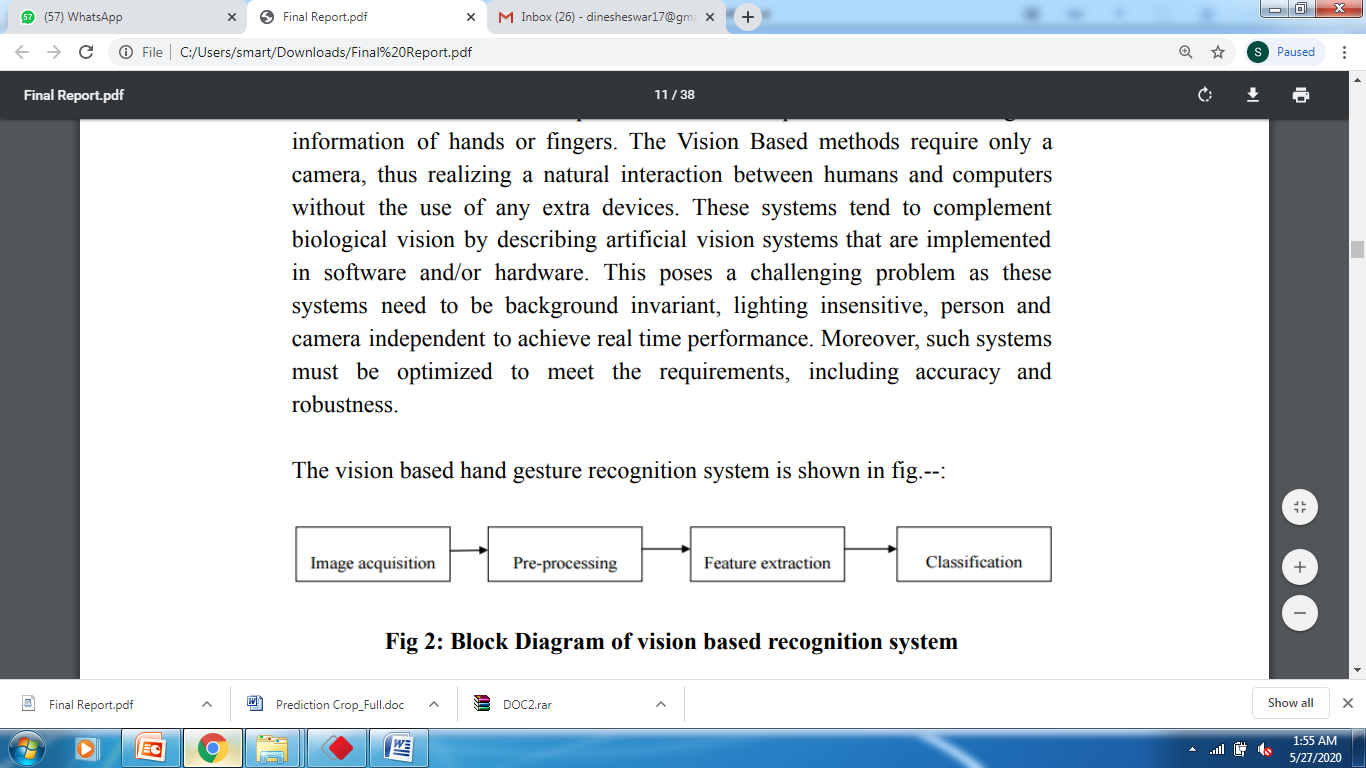


Fig  1:  Block  Diagram  of  vision  based  recognition  system

Vision based analysis, is based on the way human beings perceive information about their surroundings, yet it is probably the most difficult toimplementinasatisfactory way.Several different approaches havebeen testedso  far.

1. One is to build a three­dimensional model of the human hand. The model is matched to images of the hand by one or more cameras, and parameters corresponding to palm orientation and joint angles are estimated. These parameters  are  then  used  to  perform  gesture  classification.

2. Second one to capture the image using a camera then extract some feature and those features are used as input in a classification algorithm for classification.

2.1.1  Handshape   recognition   for  Argentinian   Sign  Language   using ProbSom  In this paper, a method for hand gesture recognition of Argentinian sign language (LSA) is proposed. This paper offers two main contributions: first, the creation of a database of handshapes for the Argentinian Sign Language (LSA). Secondly, a technique for image processing, descriptor extraction and subsequent handshape classification using a supervised adaptation of self­organizing maps that is called ProbSom. This technique is compared to others in the state of the art, such as Support Vector Machines (SVM), Random Forests, and Neural Networks. The ProbSom­based neural classifier, using the proposed  descriptor,  achieved  an  accuracy  rate  above  90%.

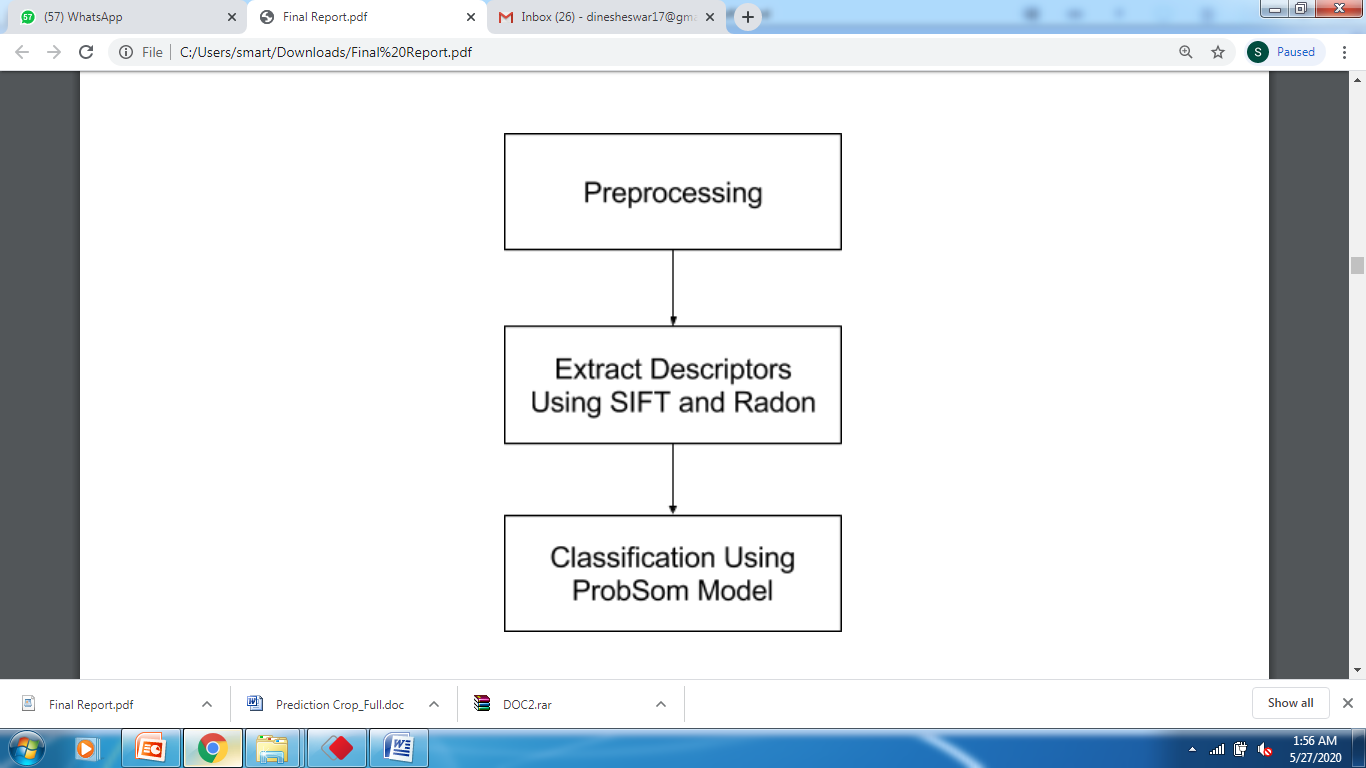


Fig  2:  Block  Diagram  of  Hand  Gesture  Recognition  System  for  LSA

2.1.2  Automatic   Indian  Sign  Language   Recognition   for  Continuous Video  Sequence  The proposed system comprises of four major modules: Data Acquisition, Pre­processing, Feature Extraction and Classification. Pre­processing stage involves Skin Filtering and histogram matching after which Eigen­vector based Feature Extraction and Eigen value weighted Euclidean distance based Classification Technique was used. 24 different alphabets were considered in this  paper  where  96%  recognition  rate  was  obtained.

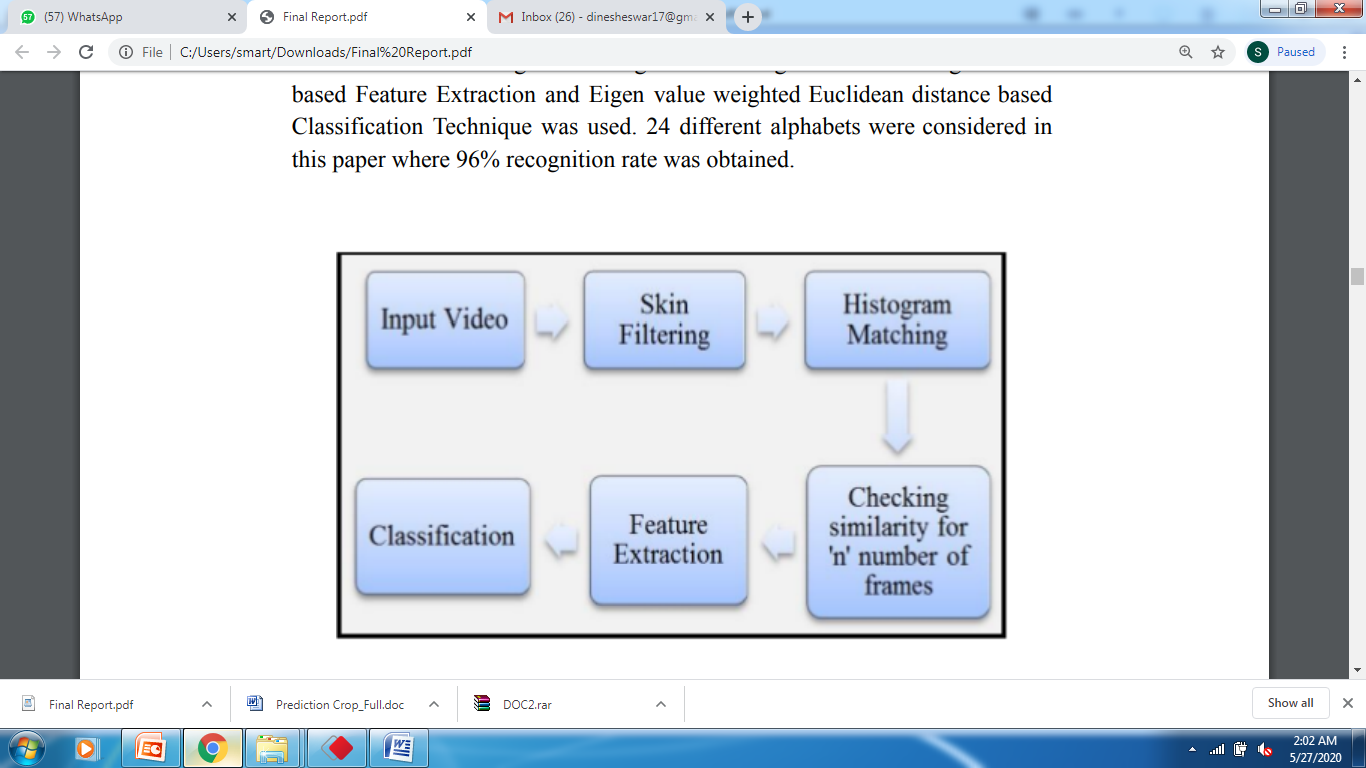
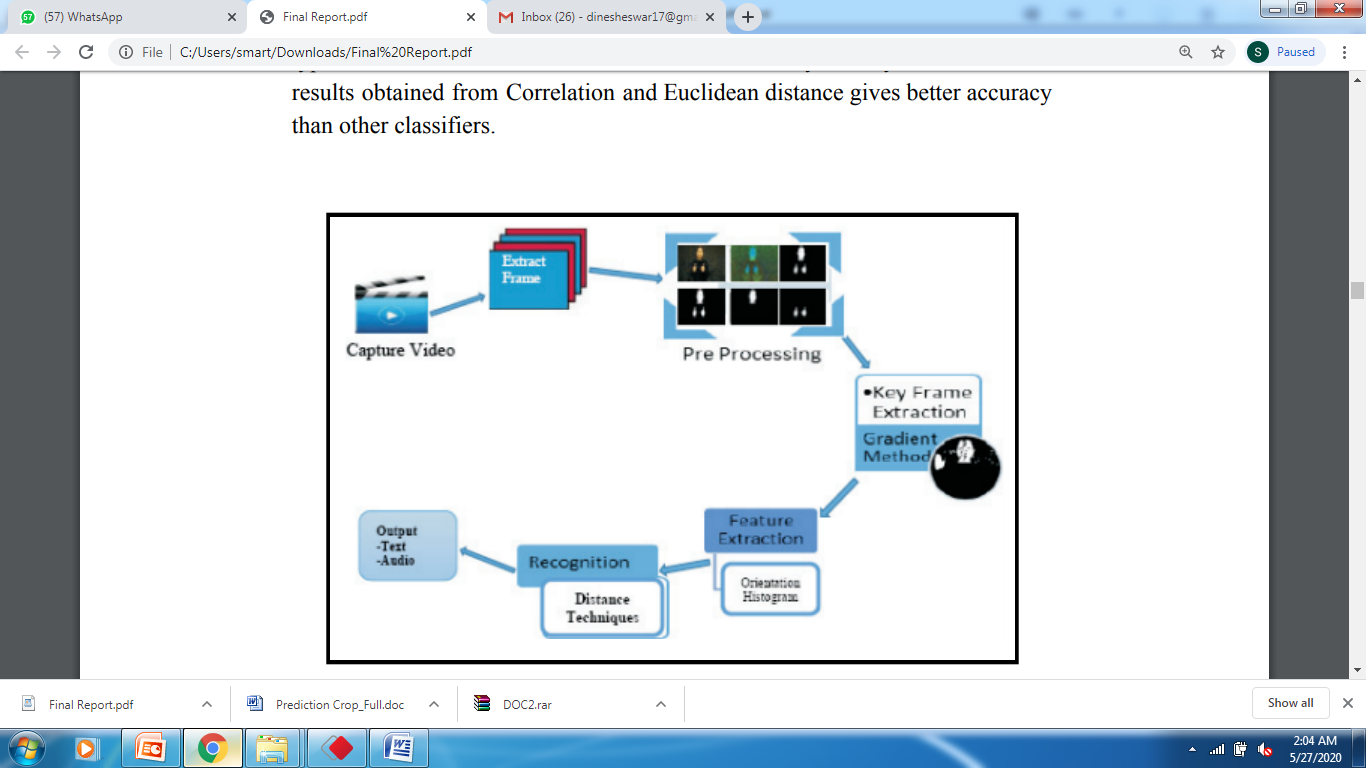


Fig  3:  System  Overview

2.1.3 Continuous   Indian Sign  Language   Gesture  Recognition   and Sentence Formation Recognizing a sign language gestures from continuous gestures is a very challenging research issue. The researchers solved this problem using gradient based  key  frame  extraction  method.  These  key  frames  were  helpful  in  splitting continuous sign language gestures into sequence of signs as well as for removing uninformative frames. After splitting of gestures each sign has been treated as an isolated gesture. Then features of pre­processed gestures were extracted using Orientation Histogram (OH) with Principal Component Analysis (PCA) is applied for reducing dimension of features obtained after OH. Experiments were performed on their own continuous ISL dataset which was created using canon EOS camera in Robotics and Artificial Intelligence laboratory (IIIT­A). Probes were tested using various types of classifiers like Euclidean distance, Correlation, Manhattan distance, city block distance etc. Comparative analysis of their proposed scheme was performed with various types of distance classifiers. From the above analysis they found that the results obtained from Correlation and Euclidean distance gives better accuracy than  other  classifiers.

Fig  4:  General  Diagram  of  the  Work

2.1.4  Recognition   of  isolated  Indian  Sign  Language   Gesture  in  Real Time

This paper demonstrates the statistical techniques for recognition of ISL gestures in real time which comprises both the hands. A video database was created by the authors and utilized which contained several videos for large number of signs. Direction histogram is the feature used for classification due to its appeal for illumination and orientation invariance. Two different approaches utilized for recognition were Euclidean distance and K­nearest neighbor  metrics.

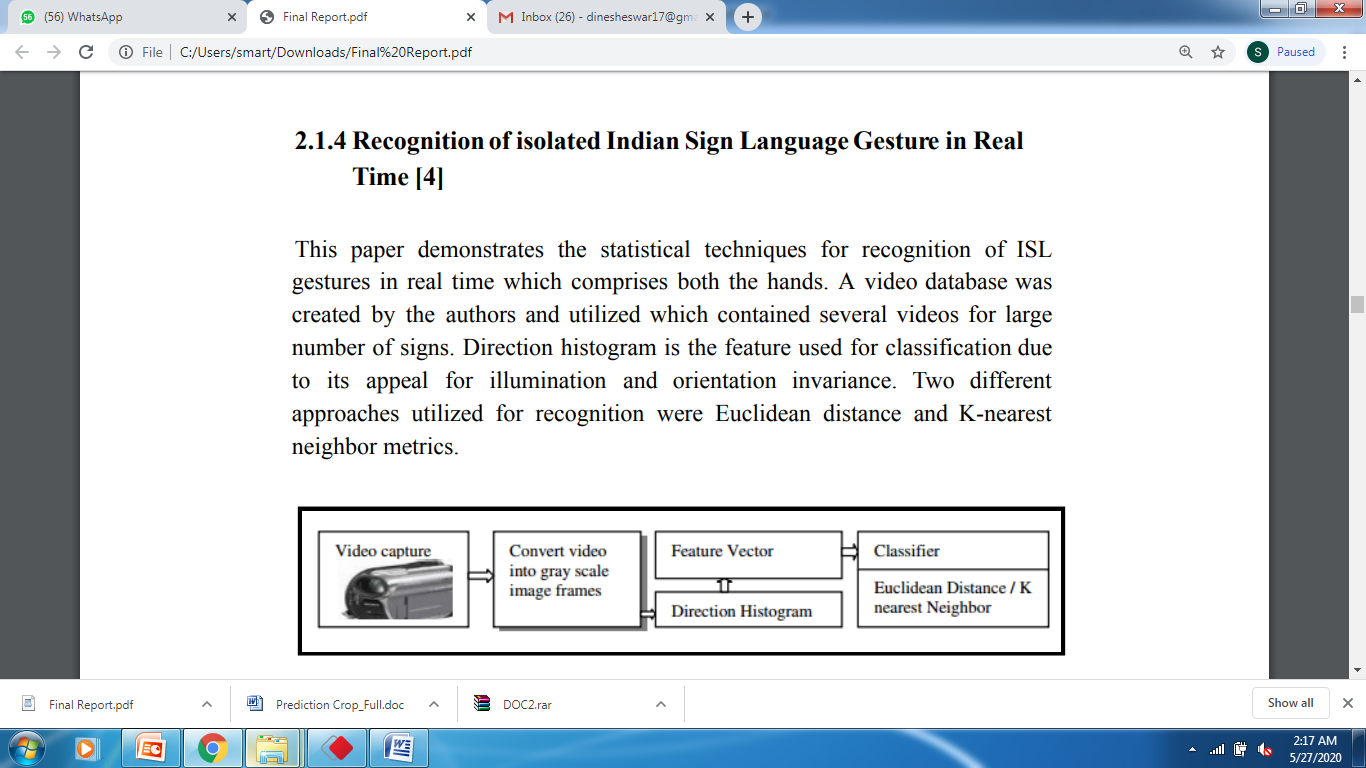


Fig 5:  Methodology for  real  time  ISL  classification

**Existing System:**

Roel Verschaeren proposes a CNN model that recognizes a set of 50 different signs in the Flemish Sign Language with an error of 2.5%, using the Microsoft Kinect. Unfortunately, this work is limited in the sense that it considers only a single person in a fixed environment. In an American Language recognition system is presented with a vocabulary of 30 words. They constructed appearance-based representations and a hand tracking system to be classified with a hidden Markov model (HMM). An error rate of 10.91% is achieved on the RWTH-BOSTON-50 database. The approach in uses the Microsoft Kinect to extract appearance-based hand features and track the position in 2D and 3D.

**Disadvantages of Existing System:**

Accuracy is less

Implementation cost is high and more complexity.

**PROPOSED SYSTEM** :

We proposed convolutional neural networks can be used to accurately recognize different signs of a sign language, with users and surroundings not included in the training set. This generalization capacity of CNNs in spatiotemporal data can contribute to the broader research field on automatic sign language recognition.

**Advantages:**

* + Design an automatic sign language recognition with low cost and less complexity.
  + To develop an system which helpful for deaf and dum people.

**Objectives:**

Our project aims at taking the basic step in bridging the communication gap between normal people, deaf and dumb people using sign language.

The main focus of this work is to create a vision based system to identify sign language gestures from the imags sequences. The reason for choosing a system based on vision relates to the fact that it provides a simpler and more intuitive way of communication between a human and a computer.

**Chapter3: SYSTEM REQUIREMENT SPECIFICATION:**

**3.1 FEASIBILITY STUDY**

The feasibility of the project is analyzed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations involved in the feasibility analysis are

* ECONOMICAL FEASIBILITY
* TECHNICAL FEASIBILITY
* SOCIAL FEASIBILITY

**ECONOMICAL FEASIBILITY**

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

**TECHNICAL FEASIBILITY**

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

**SOCIAL FEASIBILITY**

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

**Functionality Requirements**

This section describes the functional requirements of the system for those requirements which are expressed in the natural language style.

1. Create a Desktop application using Tkinter.
2. User should create the dataset for each characters.
3. System will train the characters by using the user images.
4. System will captures the hand symbols through the camera.
5. System will recognize the character from the sign.
6. Application should accurately recognize the characters from sign and automatically.

### **Non Functionality Requirements**

These are requirements that are not functional in nature, that is, these are constraints within which the system must work.

* The program must be self-contained so that it can easily be moved from one Computer to another. It is assumed that network connection will be available on the computer on which the program resides.
* Capacity, scalability and availability.

The system shall achieve 100 per cent availability at all times.

The system shall be scalable to support additional clients and volunteers.

* Maintainability.

The system should be optimized for supportability, or ease of maintenance as far as possible. This may be achieved through the use documentation of coding standards, naming conventions, class libraries and abstraction.

* Randomness, verifiability and load balancing.

The system should be optimized for supportability, or ease of maintenance as far as possible. This may be achieved through the use documentation of coding standards, naming conventions, class libraries and abstraction. It should have randomness to check the nodes and should be load balanced.

**HARDWARE REQUIREMENTS:**

* System : Intel I3.
* Hard Disk : 120 GB.
* Monitor : 15’’ LED
* Input Devices : Keyboard, Mouse
* Ram : 4 GB

**SOFTWARE REQUIREMENTS:**

* Operating system : Windows 7.
* Coding Language : PYTHON 3.6
* Tools : PythonIDLE

**Description about the Tools Used:**

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales. Van Rossum led the language community until stepping down as leader in July 2018.

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

Python interpreters are available for many operating systems. CPython, the reference implementation of Python, is open source software[30] and has a community-based development model, as do nearly all of Python's other implementations. Python and CPython are managed by the non-profit Python Software Foundation.

**History**

Python was conceived in the late 1980s by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to the ABC language (itself inspired by SETL, capable of exception handling and interfacing with the Amoeba operating system. Its implementation began in December 1989. Van Rossum's long influence on Python is reflected in the title given to him by the Python community: Benevolent Dictator For Life (BDFL) – a post from which he gave himself permanent vacation on July 12, 2018.

Python 2.0 was released on 16 October 2000 with many major new features, including a cycle-detecting garbage collector and support for Unicode.

Python 3.0 was released on 3 December 2008. It was a major revision of the language that is not completely backward-compatible. Many of its major features were backported to Python 2.6.x[37] and 2.7.x version series. Releases of Python 3 include the 2to3 utility, which automates (at least partially) the translation of Python 2 code to Python 3.

Python 2.7's end-of-life date was initially set at 2015 then postponed to 2020 out of concern that a large body of existing code could not easily be forward-ported to Python 3. In January 2017, Google announced work on a Python 2.7 to Go transcompiler to improve performance under concurrent workloads.

**Features and philosophy**

Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of its features support functional programming and aspect-oriented programming (including by meta programming and metaobjects (magic methods)). Many other paradigms are supported via extensions, including design by contract and logic programming.

Python uses dynamic typing, and a combination of reference counting and a cycle-detecting garbage collector for memory management. It also features dynamic name resolution (late binding), which binds method and variable names during program execution.

Python's design offers some support for functional programming in the Lisp tradition. It has filter(), map(), and reduce() functions; list comprehensions, dictionaries, and sets; and generator expressions.[47] The standard library has two modules (itertools and functools) that implement functional tools borrowed from Haskell and Standard ML.

The language's core philosophy is summarized in the document The Zen of Python (PEP 20), which includes aphorisms such as:

Beautiful is better than ugly

Explicit is better than implicit

Simple is better than complex

Complex is better than complicated

**Readability counts**

Rather than having all of its functionality built into its core, Python was designed to be highly extensible. This compact modularity has made it particularly popular as a means of adding programmable interfaces to existing applications. Van Rossum's vision of a small core language with a large standard library and easily extensible interpreter stemmed from his frustrations with ABC, which espoused the opposite approach.

While offering choice in coding methodology, the Python philosophy rejects exuberant syntax (such as that of Perl) in favor of a simpler, less-cluttered grammar. As Alex Martelli put it: "To describe something as 'clever' is not considered a compliment in the Python culture." Python's philosophy rejects the Perl "there is more than one way to do it" approach to language design in favor of "there should be one—and preferably only one—obvious way to do it".

Python's developers strive to avoid premature optimization, and reject patches to non-critical parts of the CPython reference implementation that would offer marginal increases in speed at the cost of clarity. When speed is important, a Python programmer can move time-critical functions to extension modules written in languages such as C, or use PyPy, a just-in-time compiler. Cython is also available, which translates a Python script into C and makes direct C-level API calls into the Python interpreter.

An important goal of Python's developers is keeping it fun to use. This is reflected in the language's name—a tribute to the British comedy group Monty Python—and in occasionally playful approaches to tutorials and reference materials, such as examples that refer to spam and eggs (from a famous Monty Python sketch) instead of the standard foo and bar.

A common neologism in the Python community is pythonic, which can have a wide range of meanings related to program style. To say that code is pythonic is to say that it uses Python idioms well, that it is natural or shows fluency in the language, that it conforms with Python's minimalist philosophy and emphasis on readability. In contrast, code that is difficult to understand or reads like a rough transcription from another programming language is called unpythonic.

Users and admirers of Python, especially those considered knowledgeable or experienced, are often referred to as Pythonists, Pythonistas, and Pythoneers.

**Syntax and semantics**

Python is meant to be an easily readable language. Its formatting is visually uncluttered, and it often uses English keywords where other languages use punctuation. Unlike many other languages, it does not use curly brackets to delimit blocks, and semicolons after statements are optional. It has fewer syntactic exceptions and special cases than C or Pascal.

**Indentation**

Python uses whitespace indentation, rather than curly brackets or keywords, to delimit blocks. An increase in indentation comes after certain statements; a decrease in indentation signifies the end of the current block. Thus, the program's visual structure accurately represents the program's semantic structure. This feature is also sometimes termed the off-side rule.

**Statements and control flow**

The assignment statement (token '=', the equals sign). This operates differently than in traditional imperative programming languages, and this fundamental mechanism (including the nature of Python's version of variables) illuminates many other features of the language. Assignment in C, e.g., x = 2, translates to "typed variable name x receives a copy of numeric value 2". The (right-hand) value is copied into an allocated storage location for which the (left-hand) variable name is the symbolic address. The memory allocated to the variable is large enough (potentially quite large) for the declared type. In the simplest case of Python assignment, using the same example, x = 2, translates to "(generic) name x receives a reference to a separate, dynamically allocated object of numeric (int) type of value 2." This is termed binding the name to the object. Since the name's storage location doesn't contain the indicated value, it is improper to call it a variable. Names may be subsequently rebound at any time to objects of greatly varying types, including strings, procedures, complex objects with data and methods, etc. Successive assignments of a common value to multiple names, e.g., x = 2; y = 2; z = 2 result in allocating storage to (at most) three names and one numeric object, to which all three names are bound. Since a name is a generic reference holder it is unreasonable to associate a fixed data type with it. However at a given time a name will be bound to some object, which will have a type; thus there is dynamic typing.

The if statement, which conditionally executes a block of code, along with else and elif (a contraction of else-if).

The for statement, which iterates over an iterable object, capturing each element to a local variable for use by the attached block.

The while statement, which executes a block of code as long as its condition is true.

The try statement, which allows exceptions raised in its attached code block to be caught and handled by except clauses; it also ensures that clean-up code in a finally block will always be run regardless of how the block exits.

The raise statement, used to raise a specified exception or re-raise a caught exception.

The class statement, which executes a block of code and attaches its local namespace to a class, for use in object-oriented programming.

The def statement, which defines a function or method.

The with statement, from Python 2.5 released on September 2006,[59] which encloses a code block within a context manager (for example, acquiring a lock before the block of code is run and releasing the lock afterwards, or opening a file and then closing it), allowing Resource Acquisition Is Initialization (RAII)-like behavior and replaces a common try/finally idiom.

The pass statement, which serves as a NOP. It is syntactically needed to create an empty code block.

The assert statement, used during debugging to check for conditions that ought to apply.

The yield statement, which returns a value from a generator function. From Python 2.5, yield is also an operator. This form is used to implement coroutines.

The import statement, which is used to import modules whose functions or variables can be used in the current program. There are three ways of using import: import <module name> [as <alias>] or from <module name> import \* or from <module name> import <definition 1> [as <alias 1>], <definition 2> [as <alias 2>], ....

The print statement was changed to the print() function in Python 3.

Python does not support tail call optimization or first-class continuations, and, according to Guido van Rossum, it never will. However, better support for coroutine-like functionality is provided in 2.5, by extending Python's generators. Before 2.5, generators were lazy iterators; information was passed unidirectionally out of the generator. From Python 2.5, it is possible to pass information back into a generator function, and from Python 3.3, the information can be passed through multiple stack levels.

**Expressions**

Some Python expressions are similar to languages such as C and Java, while some are not:

Addition, subtraction, and multiplication are the same, but the behavior of division differs. There are two types of divisions in Python. They are floor division and integer division. Python also added the \*\* operator for exponentiation.

From Python 3.5, the new @ infix operator was introduced. It is intended to be used by libraries such as NumPy for matrix multiplication.

In Python, == compares by value, versus Java, which compares numerics by value and objects by reference. (Value comparisons in Java on objects can be performed with the equals() method.) Python's is operator may be used to compare object identities (comparison by reference). In Python, comparisons may be chained, for example a <= b <= c.

Python uses the words and, or, not for its boolean operators rather than the symbolic &&, ||, ! used in Java and C.

Python has a type of expression termed a list comprehension. Python 2.4 extended list comprehensions into a more general expression termed a generator expression.

Anonymous functions are implemented using lambda expressions; however, these are limited in that the body can only be one expression.

Conditional expressions in Python are written as x if c else y (different in order of operands from the c ? x : y operator common to many other languages).

Python makes a distinction between lists and tuples. Lists are written as [1, 2, 3], are mutable, and cannot be used as the keys of dictionaries (dictionary keys must be immutable in Python). Tuples are written as (1, 2, 3), are immutable and thus can be used as the keys of dictionaries, provided all elements of the tuple are immutable. The + operator can be used to concatenate two tuples, which does not directly modify their contents, but rather produces a new tuple containing the elements of both provided tuples. Thus, given the variable t initially equal to (1, 2, 3), executing t = t + (4, 5) first evaluates t + (4, 5), which yields (1, 2, 3, 4, 5), which is then assigned back to t, thereby effectively "modifying the contents" of t, while conforming to the immutable nature of tuple objects. Parentheses are optional for tuples in unambiguous contexts.

Python features sequence unpacking where multiple expressions, each evaluating to anything that can be assigned to (a variable, a writable property, etc.), are associated in the identical manner to that forming tuple literals and, as a whole, are put on the left hand side of the equal sign in an assignment statement. The statement expects an iterable object on the right hand side of the equal sign that produces the same number of values as the provided writable expressions when iterated through, and will iterate through it, assigning each of the produced values to the corresponding expression on the left.

Python has a "string format" operator %. This functions analogous to printf format strings in C, e.g. "spam=%s eggs=%d" % ("blah", 2) evaluates to "spam=blah eggs=2". In Python 3 and 2.6+, this was supplemented by the format() method of the str class, e.g. "spam={0} eggs={1}".format("blah", 2). Python 3.6 added "f-strings": blah = "blah"; eggs = 2; f'spam={blah} eggs={eggs}'.

Python has various kinds of string literals:

Strings delimited by single or double quote marks. Unlike in Unix shells, Perl and Perl-influenced languages, single quote marks and double quote marks function identically. Both kinds of string use the backslash (\) as an escape character. String interpolation became available in Python 3.6 as "formatted string literals".

Triple-quoted strings, which begin and end with a series of three single or double quote marks. They may span multiple lines and function like here documents in shells, Perl and Ruby.

Raw string varieties, denoted by prefixing the string literal with an r. Escape sequences are not interpreted; hence raw strings are useful where literal backslashes are common, such as regular expressions and Windows-style paths. Compare "@-quoting" in C#.

Python has array index and array slicing expressions on lists, denoted as a[key], a[start:stop] or a[start:stop:step]. Indexes are zero-based, and negative indexes are relative to the end. Slices take elements from the start index up to, but not including, the stop index. The third slice parameter, called step or stride, allows elements to be skipped and reversed. Slice indexes may be omitted, for example a[:] returns a copy of the entire list. Each element of a slice is a shallow copy.

In Python, a distinction between expressions and statements is rigidly enforced, in contrast to languages such as Common Lisp, Scheme, or Ruby. This leads to duplicating some functionality. For example:

**List comprehensions vs. for-loops**

Conditional expressions vs. if blocks

The eval() vs. exec() built-in functions (in Python 2, exec is a statement); the former is for expressions, the latter is for statements.

Statements cannot be a part of an expression, so list and other comprehensions or lambda expressions, all being expressions, cannot contain statements. A particular case of this is that an assignment statement such as a = 1 cannot form part of the conditional expression of a conditional statement. This has the advantage of avoiding a classic C error of mistaking an assignment operator = for an equality operator == in conditions: if (c = 1) { ... } is syntactically valid (but probably unintended) C code but if c = 1: ... causes a syntax error in Python.

**Methods**

Methods on objects are functions attached to the object's class; the syntax instance.method(argument) is, for normal methods and functions, syntactic sugar for Class.method(instance, argument). Python methods have an explicit self parameter to access instance data, in contrast to the implicit self (or this) in some other object-oriented programming languages (e.g., C++, Java, Objective-C, or Ruby).

**Typing**

Python uses duck typing and has typed objects but untyped variable names. Type constraints are not checked at compile time; rather, operations on an object may fail, signifying that the given object is not of a suitable type. Despite being dynamically typed, Python is strongly typed, forbidding operations that are not well-defined (for example, adding a number to a string) rather than silently attempting to make sense of them.

Python allows programmers to define their own types using classes, which are most often used for object-oriented programming. New instances of classes are constructed by calling the class (for example, SpamClass() or EggsClass()), and the classes are instances of the metaclass type (itself an instance of itself), allowing metaprogramming and reflection.

Before version 3.0, Python had two kinds of classes: old-style and new-style. The syntax of both styles is the same, the difference being whether the class object is inherited from, directly or indirectly (all new-style classes inherit from object and are instances of type). In versions of Python 2 from Python 2.2 onwards, both kinds of classes can be used. Old-style classes were eliminated in Python 3.0.

The long term plan is to support gradual typing and from Python 3.5, the syntax of the language allows specifying static types but they are not checked in the default implementation, CPython. An experimental optional static type checker named mypy supports compile-time type checking.

**Mathematics**

Python has the usual C language arithmetic operators (+, -, \*, /, %). It also has \*\* for exponentiation, e.g. 5\*\*3 == 125 and 9\*\*0.5 == 3.0, and a new matrix multiply @ operator is included in version 3.5.[80] Additionally, it has a unary operator (~), which essentially inverts all the bits of its one argument. For integers, this means ~x=-x-1. Other operators include bitwise shift operators x << y, which shifts x to the left y places, the same as x\*(2\*\*y) , and x >> y, which shifts x to the right y places, the same as x//(2\*\*y) .

**The behavior of division has changed significantly over time:**

Python 2.1 and earlier use the C division behavior. The / operator is integer division if both operands are integers, and floating-point division otherwise. Integer division rounds towards 0, e.g. 7/3 == 2 and -7/3 == -2.

Python 2.2 changes integer division to round towards negative infinity, e.g. 7/3 == 2 and -7/3 == -3. The floor division // operator is introduced. So 7//3 == 2, -7//3 == -3, 7.5//3 == 2.0 and -7.5//3 == -3.0. Adding from \_\_future\_\_ import division causes a module to use Python 3.0 rules for division (see next).

Python 3.0 changes / to be always floating-point division. In Python terms, the pre-3.0 / is classic division, the version-3.0 / is real division, and // is floor division.

Rounding towards negative infinity, though different from most languages, adds consistency. For instance, it means that the equation (a + b)//b == a//b + 1 is always true. It also means that the equation b\*(a//b) + a%b == a is valid for both positive and negative values of a. However, maintaining the validity of this equation means that while the result of a%b is, as expected, in the half-open interval [0, b), where b is a positive integer, it has to lie in the interval (b, 0] when b is negative.

Python provides a round function for rounding a float to the nearest integer. For tie-breaking, versions before 3 use round-away-from-zero: round(0.5) is 1.0, round(-0.5) is −1.0. Python 3 uses round to even: round(1.5) is 2, round(2.5) is 2.

Python allows boolean expressions with multiple equality relations in a manner that is consistent with general use in mathematics. For example, the expression a < b < c tests whether a is less than b and b is less than c. C-derived languages interpret this expression differently: in C, the expression would first evaluate a < b, resulting in 0 or 1, and that result would then be compared with c.

Python has extensive built-in support for arbitrary precision arithmetic. Integers are transparently switched from the machine-supported maximum fixed-precision (usually 32 or 64 bits), belonging to the python type int, to arbitrary precision, belonging to the Python type long, where needed. The latter have an "L" suffix in their textual representation. (In Python 3, the distinction between the int and long types was eliminated; this behavior is now entirely contained by the int class.) The Decimal type/class in module decimal (since version 2.4) provides decimal floating point numbers to arbitrary precision and several rounding modes.[90] The Fraction type in module fractions (since version 2.6) provides arbitrary precision for rational numbers.

Due to Python's extensive mathematics library, and the third-party library NumPy that further extends the native capabilities, it is frequently used as a scientific scripting language to aid in problems such as numerical data processing and manipulation.

**Libraries**

Python's large standard library, commonly cited as one of its greatest strengths, provides tools suited to many tasks. For Internet-facing applications, many standard formats and protocols such as MIME and HTTP are supported. It includes modules for creating graphical user interfaces, connecting to relational databases, generating pseudorandom numbers, arithmetic with arbitrary precision decimals, manipulating regular expressions, and unit testing.

Some parts of the standard library are covered by specifications (for example, the Web Server Gateway Interface (WSGI) implementation wsgiref follows PEP 333), but most modules are not. They are specified by their code, internal documentation, and test suites (if supplied). However, because most of the standard library is cross-platform Python code, only a few modules need altering or rewriting for variant implementations.

As of March 2018, the Python Package Index (PyPI), the official repository for third-party Python software, contains over 130,000 packages with a wide range of functionality, including:

**Graphical user interfaces**

Web frameworks

Multimedia

Databases

Networking

Test frameworks

Automation

Web scraping

Documentation

System administration

Scientific computing

Text processing

Image processing

**Development environments**

Most Python implementations (including CPython) include a read–eval–print loop (REPL), permitting them to function as a command line interpreter for which the user enters statements sequentially and receives results immediately.

Other shells, including IDLE and IPython, add further abilities such as auto-completion, session state retention and syntax highlighting.

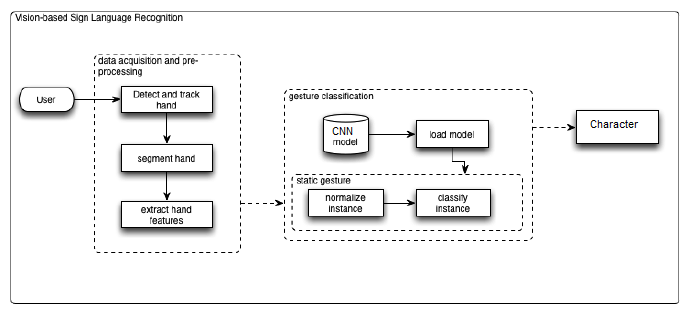
As well as standard desktop integrated development environments, there are Web browser-based IDEs; SageMath (intended for developing science and math-related Python programs); PythonAnywhere, a browser-based IDE and hosting environment; and Canopy IDE, a commercial Python IDE emphasizing scientific computing.

**Implementations**

CPython is the reference implementation of Python. It is written in C, meeting the C89 standard with several select C99 features. It compiles Python programs into an intermediate bytecode which is then executed by its virtual machine. CPython is distributed with a large standard library written in a mixture of C and native Python. It is available for many platforms, including Windows and most modern Unix-like systems. Platform portability was one of its earliest priorities.

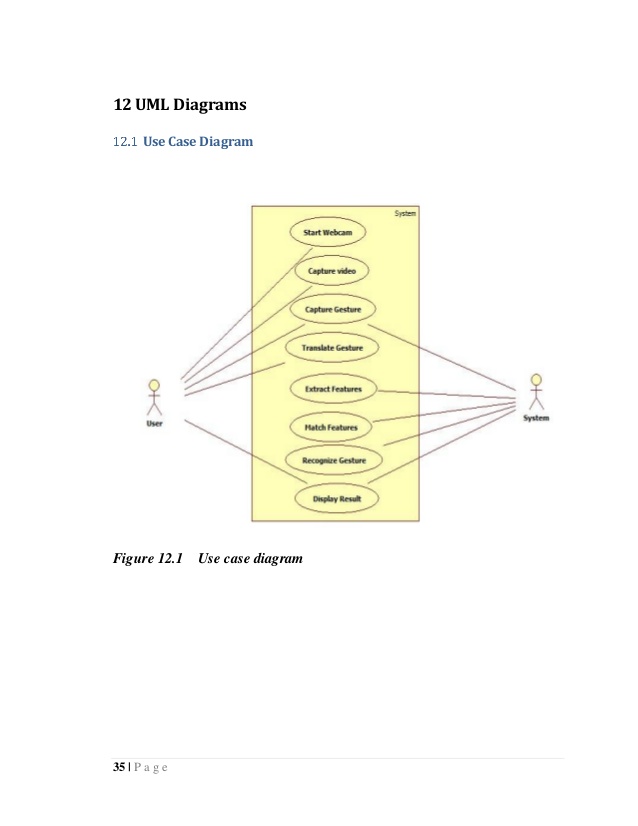
**System Design**

**System Architecture:**

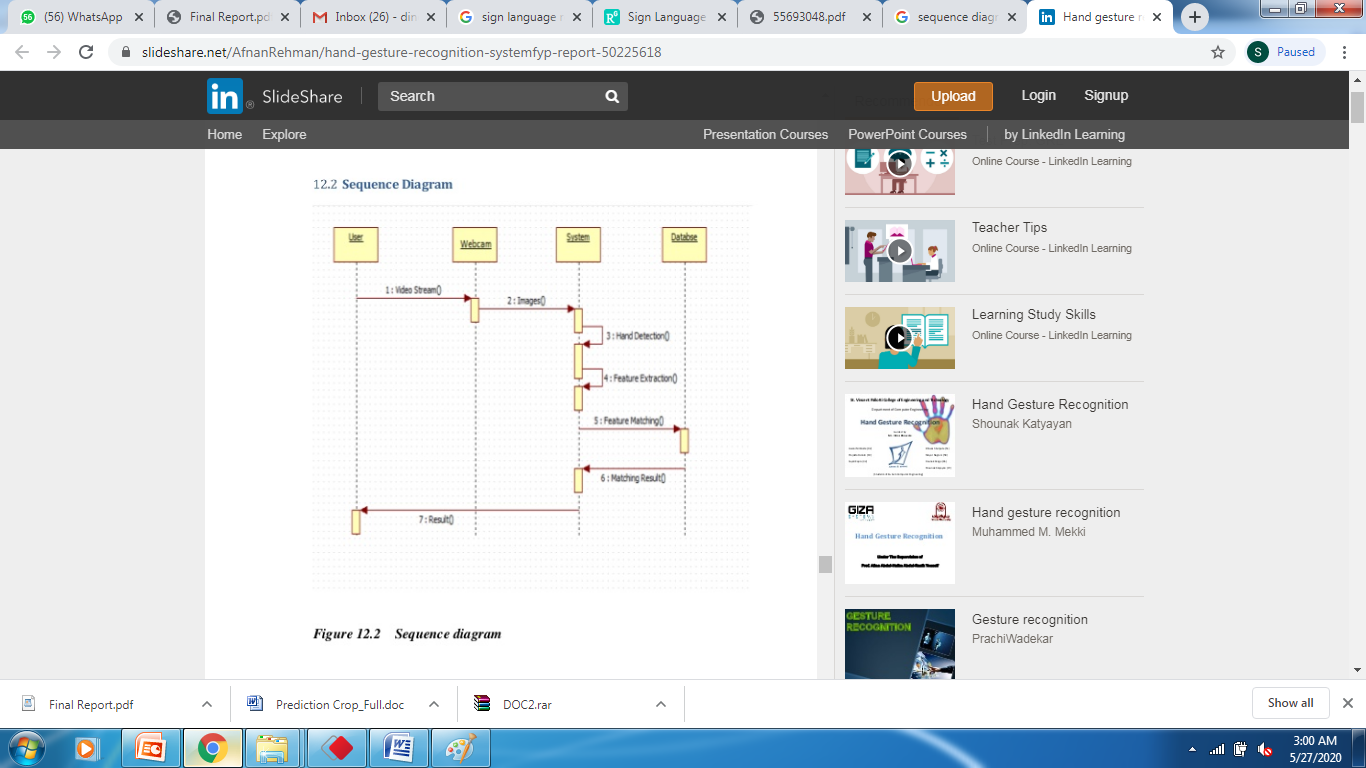
****

The system is currently developed by us. When user will open this application they have to show their hands on camera. System will detect hand and track the hand and extract the features from the captured image. Then system loads the pre-trained model and classify the character. The system will display the recognized character on the screen dynamically.

**Use Case Diagram:**



**Sequence Diagram:**

****

**Data Flow Diagram:**

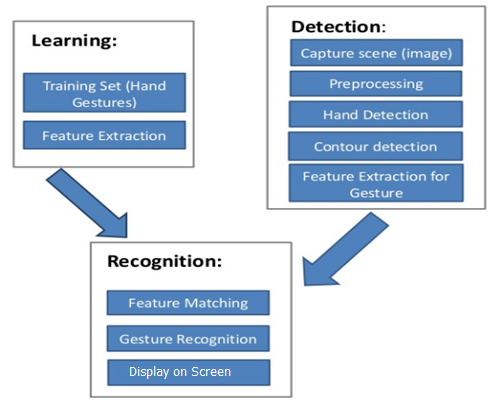
**Level:0**

Hand signs

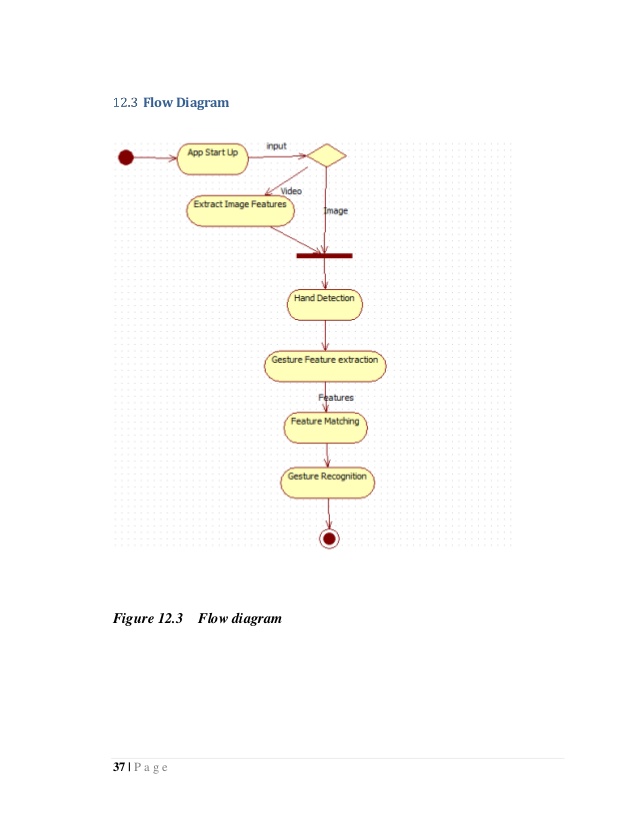
Character Recognize

Level: 0 describes the overall process of the project. We are using Hand signs as input. System will use CNN algorithm to recognize the characters from the user input.

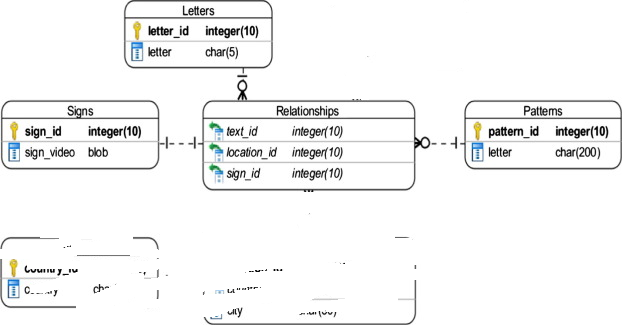
**Level: 1**

****

**Activity Diagram:**



**Class Diagram:**

****

**Implementation**

The system developed by us includes following modules as follows:

1. Data Set Creation
2. Preprocessing
3. Training Model
4. Recognition

**Module Descriptions:**

1. **Data Set Creation**

We use the data set which is created by ourselves challenge in this work. More specifically, Track 3: Gesture Spotting. This dataset consists of 5 different Indian gestures, performed by user with variations in gesture movement.

1. **Preprocessing**

This is the second module of system.Our first step in the preprocessing stage is cropping the highest hand and the upper body using the given joint information. We discovered that the highest hand is the most interesting. If both hands are used, they perform the same (mirrored) movement. If one hand is used, it is always the highest one. If the left hand is used, the videos are mirrored. This way, the model only needs to learn one side. The preprocessing results in four video samples (hand and body with depth and gray-scale) of resolution 64x64x32 (32 frames of size 64x64). Furthermore, the noise in the depth maps is reduced with thresholding, background removal using the user index, and median filtering.

1. **Training Model:**

For the pooling method, we use max-pooling: only the maximum value in a local neighborhood of the feature map remains. To accommodate video data, the max-pooling is performed in three dimensions. However, using 2D convolutions resulted in a better validation accuracy than 3D convolutions. The architecture of the model consists of two CNNs, one for extracting hand features and one for extracting upper body features. Each CNN is three layers deep. A classical ANN with one hidden layer provides classification after concatenating the outcomes of both CNNs. Also, local contrast normalization (LCN) is applied in the first two layers and all artificial neurons are rectified linear units (ReLUs ). By using CNN we are providing the training of the collected data.

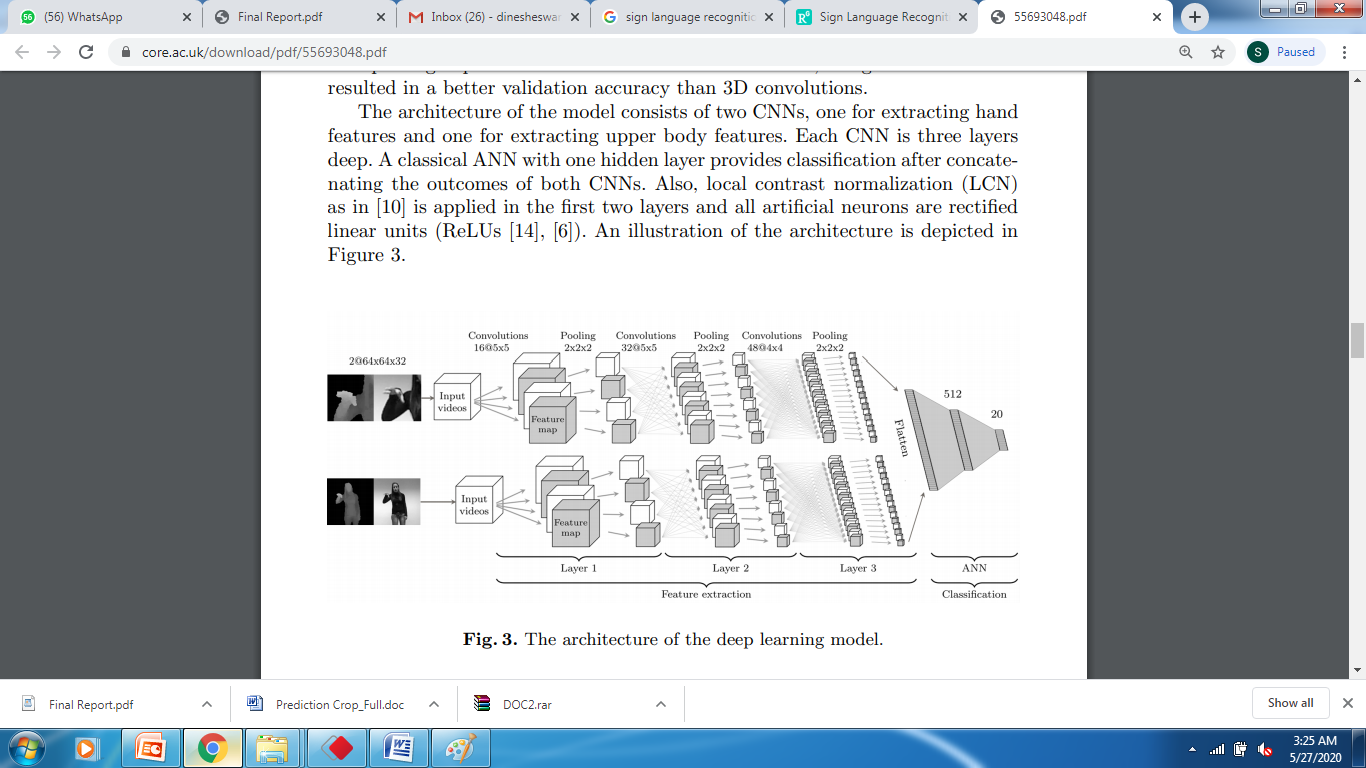
1. **Recognition:**

We use the sliding windows technique, where each possible interval of 32 frames is evaluated with the trained model (as previously described). Consecutive intervals with identical classes and sufficiently high classification probability (thresholding) are considered as a gesture segment.

**Methodologies:**

Convolutional Neural Network (CNN) CNNs are feature extraction models in deep learning that recently have proven to be to be very successful at image recognition. As of now, the models are in use by various industry leaders like Google, Facebook and Amazon. And recently, researchers at Google applied CNNs on video data. CNNs are inspired by the visual cortex of the human brain. The artificial neurons in a CNN will connect to a local region of the visual field, called a receptive field. This is accomplished by performing discrete convolutions on the image with filter values as trainable weights. Multiple filters are applied for eachchannel, and together with the activation functions of the neurons, they form feature maps. This is followed by a pooling scheme, where only the interesting information of the feature maps are pooled together.

For the pooling method, we use max-pooling: only the maximum value in a local neighborhood of the feature map remains. To accommodate video data, the max-pooling is performed in three dimensions. However, using 2D convolutions resulted in a better validation accuracy than 3D convolutions. The architecture of the model consists of two CNNs, one for extracting hand features and one for extracting upper body features. Each CNN is three layers deep. A classical ANN with one hidden layer provides classification after concatenating the outcomes of both CNNs. Also, local contrast normalization (LCN) is applied in the first two layers and all artificial neurons are rectified linear units (ReLUs ).



**text to speech api:**

**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. It supports three TTS engines :

* *sapi5* – SAPI5 on Windows
* *nsss* – NSSpeechSynthesizer on Mac OS X
* *espeak* – eSpeak on every other platform

**Installation** To install the pyttsx3 module, first of all, you have to open the terminal and write

pip install pyttsx3

In this project if user press the letter s in keyboard system will use the pyttxs3 libray and speak the predicted text.

If k==ord('s'):

            engine.say(sente)

          engine.runAndWait()

**natural lang processing:**

In this project we are using enchant library to get the suggessitions of the next words.

**Enchant** is a module in python which is used to check the spelling of a word, gives suggestions to correct words. Also, gives antonym and synonym of words. It checks whether a word exists in dictionary or not.

**enchant.Dict()**

**enchant.Dict()** is an inbuilt method of enchant module. It is used to create a Dict object, which is the most important object in the enchantt module. The Dict object represents the dictionary of a particular language.

***Syntax :****enchant.Dict(tag)****Parameter :****tag : the code of the language dictionary(optional)****Returns :****a Dict object*

*d=enchant.Dict("en\_US")*

      try:

            disp=""

            sent1=sente.split()

a=d.suggest(str(sent1[-1]))

for cc,tt in enumerate(a):

                if cc==10:

                    break

                disp+=str(cc)+str(".")+str(tt)+" "

**5.3 Coding:**

**Kindly add from your laptop**

### **6. SYSTEM TESTING**

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.

**6.1TYPES OF TESTS**

**6.1.1Unit 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 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.

**6.3Functional test**

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.

**6.4 System Test**

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.

**6.5 White Box Testing**

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

**6.6 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.

**6.7 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.

**6.7.1 Test strategy and approach**

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

**6.7.2 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.

**6.7.3 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.

# 6.8 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.

**6.9 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.

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

**Test Cases**

|  |  |
| --- | --- |
| **Test Case#** | UTC01 |
| **Test Name** | User input format |
| **Test Description** | To test user input as image |
| **Input** | Hand image |
| **Expected Output** | It should read and detect the hand region |
| **Actual Output** | Detected the hand region |
| **Test Result** | Success |
| **Test Case#** | UTC02 |
| **Test Name** | User input format |
| **Test Description** | To test user input as image |
| **Input** | Null |
| **Expected Output** | Show the error |
| **Actual Output** | Shown error |
| **Test Result** | Success |

|  |  |
| --- | --- |
| **Test Case#** | UTC03 |
| **Test Name** | Recognize sign |
| **Test Description** | To test whether Recognizing sign or not? |
| **Input** | Hand sign |
| **Expected Output** | It Should Recognize |
| **Actual Output** | Recognized and generated the output on screen |
| **Test Result** | Success |

|  |  |
| --- | --- |
| **Test Case#** | UTC04 |
| **Test Name** | Test case for importing valid python libraries |
| **Test Description** | To test whether an algorithm to implement congestion nodes works without sklearn and opencv, keras,librosa,playaudios models |
| **Input** | Import all valid libraries sklearn, flask and keras libraries |
| **Expected Output** | An error should be thrown specifying “error importing libraries sklearn, flask and keras libraries” |
| **Actual Output** | An error is thrown |
| **Test Result** | Success |

**8. Results Screen Shot:**

Kindly add from your system

**Conclusion:**

In this project, Hand gestures are a powerful way for human communication, with lots of potential applications in the area of human computer interaction. Vision­based hand gesture recognition techniques have many proven advantages compared with traditional devices. However, hand gesture recognition is a difficult problem and the current work is only a small contribution towards achieving the results needed in the field of sign language gesture recognition. This report presented a vision­based system able to interpret isolated hand gestures from the  Indian Sign  Language(ISA). Videos are difficult to classify because they contain both the temporal as well as the spatial features. We have used CNN model to classify on the spatial and temporal features. We obtained an accuracy of 95.217 %.

**Feature Enhancement:**

This system focuses more on user character detection from the sign in future we can implement with the words.

**Performance Evaluation:**

**Bibliography:**

[1] Ronchetti, Franco, Facundo Quiroga, César Armando Estrebou, and Laura Cristina Lanzarini. "Handshape recognition for argentinian sign language using probsom." Journal of Computer Science & Technology  16  (2016).

[2] Singha, Joyeeta, and Karen Das. "Automatic Indian Sign Language Recognition for Continuous Video Sequence." ADBU Journal of Engineering  Technology  2,  no.  1  (2015).

[3] Tripathi, Kumud, and Neha Baranwal GC Nandi. "Continuous Indian Sign Language Gesture Recognition and Sentence Formation."  Procedia  Computer  Science  54  (2015):  523­531.

[4] Nandy, Anup, Jay Shankar Prasad, Soumik Mondal, Pavan Chakraborty, and Gora Chand Nandi. "Recognition of isolated indian sign language gesture in real time." Information Processing and  Management  (2010):  102­107.

[5] Bengio, Yoshua, Patrice Simard, and Paolo Frasconi. "Learning long­term dependencies with gradient descent is difficult." IEEE transactions  on  neural  networks  5,  no.  2  (1994):  157­166.

[6] Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short­term memory."  Neural  computation  9,  no.  8  (1997):  1735­1780.

[7] Ronchetti, Franco, Facundo Quiroga, César Armando Estrebou, Laura Cristina Lanzarini, and Alejandro Rosete. "LSA64: An Argentinian Sign Language Dataset." In XXII Congreso Argentino de  Ciencias  de  la  Computación  (CACIC  2016).  2016.

[8] Kingma, Diederik, and Jimmy Ba. "Adam: A method for stochastic optimization."  arXiv  preprint  arXiv:1412.6980  (2014).

[9] Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back­propagating errors." Cognitive modeling  5,  no.  3  (1988):

[10] Hahnloser, Richard HR, Rahul Sarpeshkar, Misha A. Mahowald, Rodney J. Douglas, and H. Sebastian Seung. "Digital selection and analogue amplification coexist in a cortex­inspired silicon circuit." Nature 405, no. 6789 (2000): 947­951.12 Bottou, Léon. "Large­scale machine learning with stochastic gradient descent." In Proceedings of  COMPSTAT'2010,  pp.  177­186.  Physica­Verlag  HD,  2010.

[11] Copyright © William Vicars, Sign Language resources at LifePrint.com,  http://lifeprint.com/asl101/topics/wallpaper1.htm

[12] https://medium.com/technologymadeeasy/the­best­explanation­of­co nvolutional­neural­networks­on­the­internet­fbb8b1ad5df8

[13] https://www.quora.com/What­is­an­intuitive­explanation­of­Convol utional­Neural­Networks

[14] Abadi, Martín, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado et al. "Tensorflow: Large­scale machine learning on heterogeneous distributed systems."  arXiv  preprint  arXiv:1603.04467  (2016).

[15] Cooper, Helen, Brian Holt, and Richard Bowden. "Sign language recognition." In Visual Analysis of Humans, pp. 539­562. Springer London,  2011.

[16] Zhang, Chenyang, Xiaodong Yang, and YingLi Tian. "Histogram of 3D facets: A characteristic descriptor for hand gesture recognition." In Automatic Face and Gesture Recognition (FG), 2013 10th IEEE International  Conference  and  Workshops  on,  pp.  1­8.  IEEE,  2013.

[17] Cooper, Helen, Eng­Jon Ong, Nicolas Pugeault, and Richard Bowden. "Sign language recognition using sub­units." Journal of Machine  Learning  Research  13,  no.  Jul  (2012):  2205­2231.