1. PROBLEM DESCRIPTION

1.1 INTRODUCTION

The problem addressed by the system is real-time facial emotion recognition using computer vision and deep learning techniques. Facial emotion recognition involves the identification and analysis of facial expressions to infer the underlying emotional states of individuals. This problem is crucial in various domains, including human-computer interaction, mental health monitoring, market research, and security surveillance.

Real-time Processing: Processing video frames from a web cam feed in real-time requires efficient algorithms and optimizations to ensure timely analysis and response.

Face Detection: Accurately detecting and localizing faces within video frames is essential for isolating facial regions for emotion analysis.

Facial Region Extraction: Extracting facial regions from detected faces while preserving relevant features is necessary for effective emotion recognition.

Preprocessing: Preprocessing facial images to conform to the input requirements of the deep learning model, such as resizing and normalization, ensures consistent and optimal performance.

Emotion Recognition: Utilizing a pre-trained convolutional neural network (CNN) model to predict the emotional states based on the extracted facial regions. The model must accurately classify emotions across a range of expressions, including neutral, happiness, surprise, sadness, anger, disgust, and fear.

Real-time Visualization: Overlaying detected emotions onto the video feed in real-time provides immediate visual feedback on the emotional states inferred by the system, enhancing user interaction and understanding.

Model Integration: Loading a pre-trained CNN model for facial emotion recognition and integrating it with the real-time video processing pipeline.

2. SYSTEM STUDY

2.1 EXISTING SYSTEM

Traditionally, facial emotion recognition systems rely on a combination of computer vision techniques and machine learning algorithms. These systems often utilize handcrafted features such as Gabor filters or histogram of oriented gradients (HOG), along with classifiers like support vector machines (SVM) or decision trees. Face detection is commonly performed using methods like Haar cascade classifiers. However, these traditional approaches may struggle to accurately capture subtle facial expressions and might require significant domain knowledge for feature engineering.

Existing System Advantages

Proven Technology: The existing system utilizes well-established techniques in computer vision and deep learning for facial emotion recognition, ensuring reliability and accuracy.

Real-time Performance: It offers real-time processing capabilities, enabling immediate feedback on detected emotions.

Accessibility: The code provides a readily available solution that can be easily implemented by developers without extensive knowledge of machine learning or computer vision.

Cost-effectiveness: Since it utilizes pre-trained models and open-source libraries, the implementation costs are relatively low.

Versatility: The system can be adapted to various applications, such as human-computer interaction, mental health monitoring, and market research.

Existing System Disadvantages

Limited Emotion Recognition: The system is limited to recognizing only seven basic emotions, potentially overlooking more nuanced emotional expressions.

Dependency on Lighting and Pose: Performance may degrade in challenging lighting conditions or when faces are occluded or posed at extreme angles.

Hardware Requirements: Real-time processing may require substantial computational resources, limiting its deployment on resource-constrained devices.

Model Generalization: The pre-trained model may not generalize well to diverse demographics, leading to biases in emotion recognition across different populations.

Privacy Concerns: The system raises privacy concerns as it involves processing live video feeds, necessitating careful consideration of data security and user consent.

2.2 PROPOSED SYSTEM

The proposed system leverages deep learning techniques, particularly convolutional neural networks (CNNs), to address the limitations of traditional methods. CNNs can automatically learn hierarchical features directly from raw pixel data, enabling them to capture complex patterns and nuances in facial expressions more effectively. In the proposed system, video frames are captured from a webcam feed, and faces are detected using a Haar cascade classifier.

The detected facial regions are then preprocessed and fed into a pre-trained CNN model trained on datasets such as FER-2013. This CNN model predicts the emotion expressed in the face in real-time, overlaying the detected emotions on the video feed. By utilizing CNNs, the proposed system offers improved accuracy and scalability, making it a promising approach for real-time facial emotion recognition.

Proposed System Advantages

Improved Emotion Recognition: The proposed system could potentially improve emotion recognition accuracy by leveraging more advanced deep learning architectures or incorporating multimodal data fusion techniques.

Robustness to Environmental Factors: Advanced preprocessing techniques and robust models could enhance performance under challenging environmental conditions, such as varying lighting or occlusions.

Enhanced User Experience: By providing more detailed and accurate emotion analysis, theproposed system could enhance user experiences in applications like virtual reality, gaming, and personalized advertising.

Adaptability: The proposed system could be designed to adapt and personalize emotion recognition models based on user feedback and evolving data trends

Ethical Considerations: Incorporating ethical principles and transparency into the design ofthe proposed system could address privacy concerns and promote responsible use of facial emotion recognition technology.

Proposed System Disadvantages

Development Complexity: Implementing advanced deep learning techniques and multimodal fusion methods could increase the complexity of system development and

Resource Intensiveness: More sophisticated models and algorithms may require higher computational resources, potentially limiting deployment on low-power devices.

Data Requirements: Training robust emotion recognition models may require large and diverse datasets, which could pose challenges in data collection and annotation.

Ethical and Regulatory Challenges: Addressing privacy, bias, and fairness concerns in advanced emotion recognition systems may require navigating complex ethical and regulatorylandscapes.

Integration Complexity: Integrating the proposed system into existing applications or frameworks may require substantial modifications and compatibility testing.

3. SYSTEM CONFIGURATION

3.1 HARDWARE CONFIGURATION

Processor	-	I5
Speed	-	3 GHz
RAM	-	8 GB (min)
Hard Disk	-	500 GB
Key Board	-	Standard Windows Keyboard
Mouse	-	Two or Three Button Mouse
Monitor	-	LCD, LED

3.2 SOFTWARE CONFIGURATION

Operating System	-	Windows/7/10
Server	-	Anaconda, Jupyter
Front End	-	tkinter | GUI toolkit
Server side Script	-	Python , AIML

4. SOFTWARE DESCRIPTION

PYTHON

Python is an interpreter, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding; make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without chargefor all major platforms, and can be freely distributed.

Often, programmers fall in love with Python because of the increased productivity it provides. Sincethere is no compilation step, the edit-test-debug cycle is incredibly fast. Debugging Python programs is easy: a bug or bad input will never cause a segmentation fault. Instead, when the interpreter discovers an error, it raises an exception. When the program doesn't catch the exception,the interpreter prints a stack trace.

A source level debugger allows inspection of local and global variables, evaluation of arbitrary expressions, setting breakpoints, stepping through the code a lineat a time, and so on. The debugger is written in Python itself, testifying to Python's introspective power. On the other hand, often the quickest way to debug a program is to add a few print statements to the source: the fast edit-test-debug cycle makes this simple approach very effective. It ranges from simple automation tasks to gaming, web development, and even complex enterprise systems.

These are the areas where this technology is still the king with no or little competence: Machine learning as it has a plethora of libraries implementing machine learning algorithms. Python is a one-stop shop and relatively easy to learn, thus quite popular now. What other reasons exist for such universal popularity of this programming language and what companies have leveraged its opportunities to the max? Let’s talk about that.

Python technology is quite popular among programmers, but the practice shows that business owners are also Python development believers and for good reason. Software developers love it for its straight forward syntax and reputation as one of the easiest programming languages to learn. Business owners or CTOs appreciate the fact that there's a framework for pretty much anything – from web apps to machine learning.

Moreover,it is not just a language but more a technology platform that has come together through a gigantic collaboration from thousands of individual professional developers forming a huge and peculiar community of aficionados. So what is python used for and what are the tangible benefits the language brings to those who decided to use it? Below we're going to discover that.

Productivity and Speed It is a widespread theory within development circles that developing Python applicationsis approximately up to 10 times faster than developing the same application in Java or C/C++. The impressive benefit in terms of time saving can be explained by the clean object-oriented design, enhanced process control capabilities, and strong integration and text processing capacities.

Moreover, its own unit testing framework contributes substantially to its speed and productivity.

- Machine Learning
- GUI Applications (like Kivy, Tkinter, PyQt etc.)
- Web frameworks like Django (used by YouTube, Instagram, Dropbox)
- Image processing (like Opencv, Pillow)
- Web scraping (like Scrapy, BeautifulSoup, Selenium)
- Test frameworks
- Multimedia

Advantages of Python

Extensive Libraries

Python downloads with an extensive library and it contain code for various purposes like regular expressions, documentation-generation, unit-testing, web browsers, threading, databases, CGI, email, image manipulation, and more. So, we don't have to write the complete code for that manually.

Extensible

As we have seen earlier, Python can be extended to other languages. You can write some of your code in languages like C++ or C. This comes in handy, especially in projects.

Embeddable

Complimentary to extensibility, Python is embeddable as well. You can put your Python code in your source code of a different language, like C++. This lets us add scripting capabilities to our code in the other language.

Improved Productivity

The language's simplicity and extensive libraries render programmers more productive than languages like Java and C++ do. Also, the fact that you need to write less and get more things done.

IOT Opportunities

Since Python forms the basis of new platforms like Raspberry Pi, it finds the future bright for the Internet Of Things. This is a way to connect the language with the real world.

Simple and Easy

When working with Java, you may have to create a class to print 'Hello World'. But in Python, just a print statement will do. It is also quite easy to learn, understand, and code. This is why when people pick up Python, they have a hard time adjusting to other more verbose languages like Java.

Readable

Because it is not such a verbose language, reading Python is much like reading English. This is the reason why it is so easy to learn, understand, and code. It also does not need curly braces to define blocks, and indentation is mandatory. This further aids the readability of the code.

Object-Oriented

This language supports both the procedural and object-oriented programming paradigms. While functions help us with code reusability, classes and objects let us model the real world. A class allows the encapsulation of data and functions into one.

Free and Open-Source

Like we said earlier, Python is freely available. But not only can you download Python for free, but you can also download its source code, make changes to it, and even distribute it. It downloads with an extensive collection of libraries to help you with your tasks.

Portable

When you code your project in a language like C++, you may need to make some changes to it if you want to run it on another platform. But it isn’t the same with Python. Here, you need to code only once, and you can run it anywhere. This is called Write Once Run Anywhere (WORA). However, you need to be careful enough not to include any system-dependent features.

Interpreted

Lastly, we will say that it is an interpreted language. Since statements are executed one by one, debugging is easier than in compiled languages.

Advantages of Python over Other Languages

Less Coding

Almost all of the tasks done in Python requires less coding when the same task is done in other languages. Python also has an awesome standard library support, so you don’t have to search for any third-party libraries to get your job done. This is the reason that many people suggest learning Python to beginners.

Affordable

Python is free therefore individuals, small companies or big organizations can leverage the free available resources to build applications. Python is popular and widely used so it gives you better community support.

The 2019 Github annual survey showed us that Python has overtaken Java in the most popular programming language category.

Python is for Everyone

Python code can run on any machine whether it is Linux, Mac or Windows. Programmers need to learn different languages for different jobs but with Python, you can professionally build web apps, perform data analysis and machine learning, automate things, do web scraping and also build games and powerful visualizations. It is an all-rounder programming language.

Disadvantages of Python

So far, we’ve seen why Python is a great choice for your project. But if you choose it, you should be aware of its consequences as well. Let’s now see the downsides of choosing Python over another language.

Speed Limitations

We have seen that Python code is executed line by line. But since Python is interpreted, it often results in slow execution. This, however, isn’t a problem unless speed is a focal point for the project. In other words, unless high speed is a requirement, the benefits offered by Python are enough to distract us from its speed limitations.

Weak in Mobile Computing and Browsers

While it serves as an excellent server-side language, Python is much rarely seen on the client-side. Besides that, it is rarely ever used to implement smartphone-based applications. One such application is called Carbonnelle.

Design Restrictions

As you know, Python is dynamically-typed. This means that you don’t need to declare the type of variable while writing the code. It uses duck-typing. But wait, what’s that? Well, it just means that if it looks like a duck, it must be a duck. While this is easy on the programmers during coding, it can raise run-time errors.

Underdeveloped Database Access Layers

Compared to more widely used technologies like JDBC (Java DataBase Connectivity) and ODBC (Open DataBase Connectivity), Python’s database access layers are a bit underdeveloped. Consequently, it is less often applied in huge enterprises.

Simple

No, we’re not kidding. Python’s simplicity can indeed be a problem. Take my example. I don’t do Java, I’m more of a Python person. To me, its syntax is so simple that the verbosity of Java code seems unnecessary.

History of Python

What do the alphabet and the programming language Python have in common? Right, both start with ABC. If we are talking about ABC in the Python context, it's clear that the programming language ABC is meant. ABC is a general-purpose programming language and programming environment, which had been developed in the Netherlands, Amsterdam, at the CWI (Centrum Wiskunde &Informatica).

The greatest achievement of ABC was to influence the design of Python. Python was conceptualized in the late 1980s. Guido van Rossum worked that time in a project at the CWI, called Amoeba, a distributed operating system. In an interview with Bill Venners1, Guido van Rossum said: "In the early 1980s, I worked as an implementer on a team building a language called ABC at Centrum voor Wiskunde en Informatica (CWI). I don't know how well people know ABC's influence on Python. I try to mention ABC's influence because I'm indebted to everything I learned during that project and to the people who worked on it. "Later on in the same Interview, Guido van Rossum continued: "I remembered all my experience and some of my frustration with ABC.

I decided to try to design a simple scripting language that possessed some of ABC's better properties, but without its problems. So I started typing. I created a simple virtual machine, a simple parser, and a simple runtime. I made my own version of the various ABC parts that I liked. I created a basic syntax, used indentation for statement grouping instead of curly braces or begin-end blocks, and developed a small number of powerful data types: a hash table (or dictionary, as we call it), a list, strings, and numbers."

Machine Learning

Before we take a look at the details of various machine learning methods, let's start by looking at what machine learning is, and what it isn't. Machine learning is often categorized as a subfield of artificial intelligence, but I find that categorization can often be misleading at first brush. The study of machine learning certainly arose from research in this context, but in the data science application of machine learning methods, it's more helpful to think of machine learning as a means of building models of data.

Fundamentally, machine learning involves building mathematical models to help understand data. "Learning" enters the fray when we give these models tunable parameters that can be adapted to observed data; in this way the program can be considered to be "learning" from the data.

Once these models have been fit to previously seen data, they can be used to predict and understand aspects of newly observed data. I'll leave to the reader the more philosophical digression regarding the extent to which this type of mathematical, model-based "learning" is similar to the "learning" exhibited by the human brain.

Understanding the problem setting in machine learning is essential to using these tools effectively, and so we will start with some broad categorizations of the types of approaches we'll discuss here.

Categories of Machine Learning

At the most fundamental level, machine learning can be categorized into two main types: supervised learning and unsupervised learning.

Supervised learning involves some how modeling the relationship between measured features of data and some label associated with the data; once this model is determined, it can be used to apply labels to new, unknown data. This is further subdivided into classification tasks and regression tasks: in classification, the labels are discrete categories, while in regression, the labels are continuous quantities. We will see examples of both types of supervised learning in the following section.

Unsupervised learning involves modeling the features of a dataset without reference to any label, and is often described as "letting the dataset speak for itself." These models include tasks such as clustering and dimensionality reduction. Clustering algorithms identify distinct groups of data, while dimensionality reduction algorithms search for more succinct representations of the data. We will see examples of both types of unsupervised learning in the following section.

Need for Machine Learning

Human beings, at this moment, are the most intelligent and advanced species on earth because they can think, evaluate and solve complex problems. On the other side, AI is still in its initial stage and haven't surpassed human intelligence in many aspects. Then the question is that what is the need to make machine learn? The most suitable reason for doing this is, "to make decisions, based on data, with efficiency and scale".

Lately, organizations are investing heavily in newer technologies like Artificial Intelligence, Machine Learning and Deep Learning to get the key information from data to perform several real-world tasks and solve problems. We can call it data-driven decisions taken by machines, particularly to automate the process. These data-driven decisions can be used, instead of using programing logic, in the problems that cannot be programmed inherently.

The fact is that we can't do without human intelligence, but other aspect is that we all need to solve real-world problems with efficiency at a huge scale. That is why the need for machine learning arises.

Challenges in Machine Learning

While Machine Learning is rapidly evolving, making significant strides with cybersecurity and autonomous cars, this segment of AI as whole still has a long way to go. The reason behind is that ML has not been able to overcome number of challenges.

The challenges that ML is facing currently are − Quality of data − Having good-quality data for ML algorithms is one of the biggest challenges. Use of low-quality data leads to the problems related to data preprocessing and feature extraction.

Time-Consuming task − Another challenge faced by ML models is the consumption of time especially for data acquisition, feature extraction and retrieval.

Lack of specialist persons − As ML technology is still in its infancy stage, availability of expert resources is a tough job.

No clear objective for formulating business problems − Having no clear objective and well-defined goal for business problems is another key challenge for ML because this technology is not that mature yet.

Issue of overfitting & underfitting − If the model is overfitting or underfitting, it cannot be represented well for the problem.

Curse of dimensionality − Another challenge ML model faces is too many features of data points. This can be a real hindrance.

Difficulty in deployment − Complexity of the ML model makes it quite difficult to be deployed in real life.

Applications of Machine Learning

Machine Learning is the most rapidly growing technology and according to researchers we are in the golden year of AI and ML. It is used to solve many real-world complex problems which cannot be solved with traditional approach. Following are some real-world applications of ML −

- Emotion analysis
- Sentiment analysis
- Error detection and prevention
- Weather forecasting and prediction
- Stock market analysis and forecasting
- Speech synthesis
- Speech recognition
- Customer segmentation
- Object recognition
- Fraud detection
- Fraud prevention
- Recommendation of products to customer in online shopping

How to Start Learning Machine Learning?

Arthur Samuel coined the term "Machine Learning" in 1959 and defined it as a "Field of study that gives computers the capability to learn without being explicitly programmed".

And that was the beginning of Machine Learning! In modern times, Machine Learning is one of the most popular (if not the most!) career choices. According to Indeed, Machine Learning Engineer Is The Best Job of 2019 with a 344% growth and an average base salary of $146,085 per year.

But there is still a lot of doubt about what exactly is Machine Learning and how to start learning it? So this article deals with the Basics of Machine Learning and also the path you can follow to eventually become a full-fledged Machine Learning Engineer. Now let’s get started!!!

This is a rough road map you can follow on your way to becoming an insanely talented Machine Learning Engineer. Of course, you can always modify the steps according to your needs to reach your desired end-goal!

Step 1 – Understand the Prerequisites

In case you are a genius, you could start ML directly but normally, there are some prerequisites that you need to know which include Linear Algebra, Multivariate Calculus, Statistics, and Python. And if you don’t know these, never fear! You don’t need a Ph.D. degree in these topics to get started but you do need a basic understanding.

Learn Linear Algebra and Multivariate Calculus

Both Linear Algebra and Multivariate Calculus are important in Machine Learning. However, the extent to which you need them depends on your role as a data scientist.

If you are more focused on application heavy machine learning, then you will not be that heavily focused on maths as there are many common libraries available.

But if you want to focus on R&D in Machine Learning, then mastery of Linear Algebra and Multivariate Calculus is very important as you will have to implement many ML algorithms from scratch.

Learn Statistics

Data plays a huge role in Machine Learning. In fact, around 80% of your time as an ML expert will be spent collecting and cleaning data. And statistics is a field that handles the collection, analysis, and presentation of data. So it is no surprise that you need to learn it!!!

Some of the key concepts in statistics that are important are Statistical Significance, Probability Distributions, Hypothesis Testing, Regression, etc. Also, Bayesian Thinking is also a very important part of ML which deals with various concepts like Conditional Probability, Priors, and Posteriors, Maximum Likelihood, etc.

Learn Python

Some people prefer to skip Linear Algebra, Multivariate Calculus and Statistics and learn them as they go along with trial and error. But the one thing that you absolutely cannot skip is Python! While there are other languages you can use for Machine Learning like R, Scala, etc. Python is currently the most popular language for ML. In fact, there are many Python libraries that are specifically useful for Artificial Intelligence and Machine Learning such as Keras, TensorFlow, Scikit-learn, etc.

Step 2 – Learn Various ML Concepts

Now that you are done with the prerequisites, you can move on to actually learning ML (Which is the fun part!!!) It's best to start with the basics and then move on to the more complicated stuff. Some of the basic concepts in ML are:

Terminologies of Machine Learning

- Model – A model is a specific representation learned from data by applying some machine learning algorithm. A model is also called a hypothesis.
- Feature – A feature is an individual measurable property of the data. A set of numeric features can be conveniently described by a feature vector. Feature vectors are fed as input to the model. For example, in order to predict a fruit, there may be features like color, smell, taste, etc.
- Target (Label) – A target variable or label is the value to be predicted by our model. For the fruit example discussed in the feature section, the label with each set of input would be the name of the fruit like apple, orange, banana, etc.
- Training – The idea is to give a set of inputs (features) and its expected outputs (labels), so after training, we will have a model (hypothesis) that will then map new data to one of the categories trained on.
- Prediction – Once our model is ready, it can be fed a set of inputs to which it will provide a predicted output (label).

Types of Machine Learning

- Supervised Learning – This involves learning from a training dataset with labeled data using classification and regression models. This learning process continues until the required level of performance is achieved.

- Unsupervised Learning – This involves using unlabelled data and then finding the underlying structure in the data in order to learn more and more about the data itself using factor and cluster analysis models.
- Semi-supervised Learning – This involves using unlabelled data like Unsupervised Learning with a small amount of labeled data. Using labeled data vastly increases the learning accuracy and is also more cost-effective than Supervised Learning.
- Reinforcement Learning – This involves learning optimal actions through trial and error. So the next action is decided by learning behaviors that are based on the current state and that will maximize the reward in the future.

Advantages of Machine learning

Easily identifies trends and patterns

Machine Learning can review large volumes of data and discover specific trends and patterns that would not be apparent to humans. For instance, for an e-commerce website like Amazon, it serves to understand the browsing behaviors and purchase histories of its users to help cater to the right products, deals, and reminders relevant to them. It uses the results to reveal relevant advertisements to them.

No human intervention needed (automation)

With ML, you don’t need to babysit your project every step of the way. Since it means giving machines the ability to learn, it lets them make predictions and also improve the algorithms on their own.

A common example of this is anti-virus softwares; they learn to filter new threats as they are recognized. ML is also good at recognizing spam.

Continuous Improvement

As ML algorithms gain experience, they keep improving in accuracy and efficiency. This lets them make better decisions. Say you need to make a weather forecast model. As the amount of data you have keeps growing, your algorithms learn to make more accurate predictions faster.

Handling multi-dimensional and multi-variety data

Machine Learning algorithms are good at handling data that are multi-dimensional and multi-variety, and they can do this in dynamic or uncertain environments.

Wide Applications

You could be an e-tailer or a healthcare provider and make ML work for you. Where it does apply, it holds the capability to help deliver a much more personal experience to customers while also targeting the right customers.

Disadvantages of Machine Learning

Data Acquisition

Machine Learning requires massive data sets to train on, and these should be inclusive/unbiased, and of good quality. There can also be times where they must wait for new data to be generated.

Time and Resources

ML needs enough time to let the algorithms learn and develop enough to fulfill their purpose with a considerable amount of accuracy and relevancy. It also needs massive resources to function. This can mean additional requirements of computer power for you.

Interpretation of Results

Another major challenge is the ability to accurately interpret results generated by the algorithms. You must also carefully choose the algorithms for your purpose.

High error-susceptibility

Machine Learning is autonomous but highly susceptible to errors. Suppose you train an algorithm with data sets small enough to not be inclusive. You end up with biased predictions coming from a biased training set. This leads to irrelevant advertisements being displayed to customers. In the case of ML, such blunders can set off a chain of errors that can go undetected for long periods of time. And when they do get noticed, it takes quite some time to recognize the source of the issue, and even longer to correct it.

Python Development Steps

Guido Van Rossum published the first version of Python code (version 0.9.0) at alt.sources in February 1991. This release included already exception handling, functions, and the core data types of list, dict, str and others. It was also object oriented and had a module system.

Python version 1.0 was released in January 1994. The major new features included in this release were the functional programming tools lambda, map, filter and reduce, which Guido Van Rossum never liked.Six and a half years later in October 2000, Python 2.0 was introduced. This release included list comprehensions, a full garbage collector and it was supporting unicode.Python flourished for another 8 years in the versions 2.x before the next major release as Python 3.0 (also known as "Python 3000" and "Py3K") was released. Python 3 is not backwards compatible with Python 2.x. The emphasis in Python 3 had been on the removal of duplicate programming constructs and modules, thus fulfilling or coming close to fulfilling the 13th law of the Zen of Python: "There should be one -- and preferably only one -- obvious way to do it."Some changes in Python 7.3:

- Print is now a function
- Views and iterators instead of lists
- The rules for ordering comparisons have been simplified. E.g. a heterogeneous list cannot be sorted, because all the elements of a list must be comparable to each other.
- There is only one integer type left, i.e. int. long is int as well.
- The division of two integers returns a float instead of an integer. "//" can be used to have the "old" behaviour.
- Text Vs. Data Instead Of Unicode Vs. 8-bit

Purpose

We demonstrated that our approach enables successful segmentation of intra-retinal layers—even with low-quality images containing speckle noise, low contrast, and different intensity ranges throughout—with the assistance of the ANIS feature.

Python

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace.

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 is Interpreted − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- Python is Interactive − you can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

Python also acknowledges that speed of development is important. Readable and terse code is part of this, and so is access to powerful constructs that avoid tedious repetition of code. Maintainability also ties into this may be an all but useless metric, but it does say something about how much code you have to scan, read and/or understand to troubleshoot problems or tweak behaviors.

This speed of development, the ease with which a programmer of other languages can pick up basic Python skills and the huge standard library is key to another area where Python excels. All its tools have been quick to implement, saved a lot of time, and several of them have later been patched and updated by people with no Python background - without breaking.

Modules Used in Project

Tensorflow

TensorFlow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. It is used for both research and production at Google.

TensorFlow was developed by the Google Brain team for internal Google use. It was released under the Apache 2.0 open-source license on November 9, 2015.

Numpy

Numpy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays.

It is the fundamental package for scientific computing with Python. It contains various features including these important ones:

- A powerful N-dimensional array object
- Sophisticated (broadcasting) functions
- Tools for integrating C/C++ and Fortran code
- Useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, Numpy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined using Numpy which allows Numpy to seamlessly and speedily integrate with a wide variety of databases.

Pandas

Pandas is an open-source Python Library providing high-performance data manipulation and analysis tool using its powerful data structures. Python was majorly used for data munging and preparation. It had very little contribution towards data analysis. Pandas solved this problem. Using Pandas, we can accomplish five typical steps in the processing and analysis of data, regardless of the origin of data load, prepare, manipulate, model, and analyze. Python with Pandas is used in a wide range of fields including academic and commercial domains including finance, economics, Statistics, analytics, etc.

Matplotlib

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter Notebook, web application servers, and four graphical user interface toolkits. Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, error charts, scatter plots, etc., with just a few lines of code. For examples, see the sample plots and thumbnail gallery.

For simple plotting the pyplot module provides a MATLAB-like interface, particularly when combined with IPython. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users.

Scikit – learn

Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python. It is licensed under a permissive simplified BSD license and is distributed under many Linux distributions, encouraging academic and commercial use.

Python

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace.

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 is Interpreted − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- Python is Interactive − you can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

Python also acknowledges that speed of development is important. Readable and terse code is part of this, and so is access to powerful constructs that avoid tedious repetition of code. Maintainability also ties into this may be an all but useless metric, but it does say something about how much code you have to scan, read and/or understand to troubleshoot problems or tweak behaviors. This speed of development, the ease with which a programmer of other languages can pick up basic Python skills and the huge standard library is key to another area where Python excels. All its tools have been quick to implement, saved a lot of time, and several of them have later been patched and updated by people with no Python background - without breaking.

Python a versatile programming language doesn’t come pre-installed on your computer devices. Python was first released in the year 1991 and until today it is a very popular high-level programming language. Its style philosophy emphasizes code readability with its notable use of great whitespace.

The object-oriented approach and language construct provided by Python enables programmers to write both clear and logical code for projects. This software does not come pre-packaged with Windows.

There have been several updates in the Python version over the years. The question is how to install Python? It might be confusing for the beginner who is willing to start learning Python but this tutorial will solve your query. The latest or the newest version of Python is version 3.7.4 or in other words, it is Python 3.

Note: The python version 3.7.4 cannot be used on Windows XP or earlier devices.

Before you start with the installation process of Python. First, you need to know about your System Requirements. Based on your system type i.e. operating system and based processor, you must download the python version. My system type is a Windows 64-bit operating system. So the steps below are to install python version 3.7.4 on Windows 7 device or to install Python 3. Download the Python Cheat sheet here. The steps on how to install Python on Windows 10, 8 and 7 are divided into 4 parts to help understand better.

OpenCV

OpenCV was started at Intel in 1999 by Gary Bradsky, and the first release came out in 2000. Vadim Pisarevsky joined Gary Bradsky to manage Intel's Russian software OpenCV team. In 2005, OpenCV was used on Stanley, the vehicle that won the 2005 DARPA Grand Challenge. Later, its active development continued under the support of Willow Garage with Gary Bradsky and Vadim Pisarevsky leading the project. OpenCV now supports a multitude of algorithms relatedto Computer Vision and Machine Learning and is expanding day by day.

OpenCV supports a wide variety of programming languages such as C++, Python, Java, etc., and is available on different platforms including Windows, Linux, OS X, Android, and iOS. Interfacesfor high-speed GPU operations based on CUDA and OpenCL are also under active development.

OpenCV-Python is the Python API for OpenCV, combining the best qualities of the OpenCV C++API and the Python language.

OpenCV-Python

OpenCV-Python is a library of Python bindings designed to solve computer vision problems.

Python is a general purpose programming language started by Guido van Rossum that became verypopular very quickly, mainly because of its simplicity and code readability. It enables the programmer to express ideas in fewer lines of code without reducing readability.

Compared to languages like C/C++, Python is slower. That said, Python can be easily extended with C/C++, which allows us to write computationally intensive code in C/C++ and create Python wrappers that can be used as Python modules. This gives us two advantages: first, the code is as fast as the original C/C++ code (since it is the actual C++ code working in background) and second,it is easier to code in Python than C/C++. OpenCV-Python is a Python wrapper for the original OpenCV C++ implementation.

OpenCV-Python makes use of Numpy, which is a highly optimized library for numerical operationswith a MATLAB-style syntax. All the OpenCV array structures are converted to and from Numpyarrays. This also makes it easier to integrate with other libraries that use Numpy such as SciPy andMatplotlib.

NumPy (np):

NumPy is a fundamental package for scientific computing with Python, providing support for large multidimensional arrays and matrices, along with a collection of mathematical functions to operateon these arrays efficiently.

It is the backbone of many scientific and numerical computing libraries in Python, serving as the foundation for tasks such as linear algebra, Fourier analysis, and random number generation.

In this code, NumPy is essential for representing images as arrays, performing array operations such as resizing and normalization, and preparing input data for deep learning models.

Keras:

Keras is a high-level neural networks API, written in Python and capable of running on top of deeplearning frameworks such as TensorFlow, Theano, or Microsoft Cognitive Toolkit (CNTK).

It enables fast experimentation with deep neural networks, providing a user-friendly interface for building, training, and deploying models.

Keras abstracts away the complexities of low-level operations, allowing developers to focus on model architecture and experimentation.

In this code, Keras is used for loading a pre-trained convolutional neural network (CNN) model architecture from a JSON file and preprocessing images before feeding them into the model.

keras.models.model_from_json:

The model_from_json function in Keras is used to instantiate a Keras model from a JSON string representing the model architecture.

It allows models to be serialized and deserialized easily, facilitating model sharing, deployment, and transfer learning.

This function is particularly useful when the model architecture needs to be stored separately fromthe model weights or when models are trained and deployed in different environments.

keras.preprocessing.image:

The preprocessing module in Keras provides utilities and tools for preprocessing image data beforefeeding it into a neural network model.

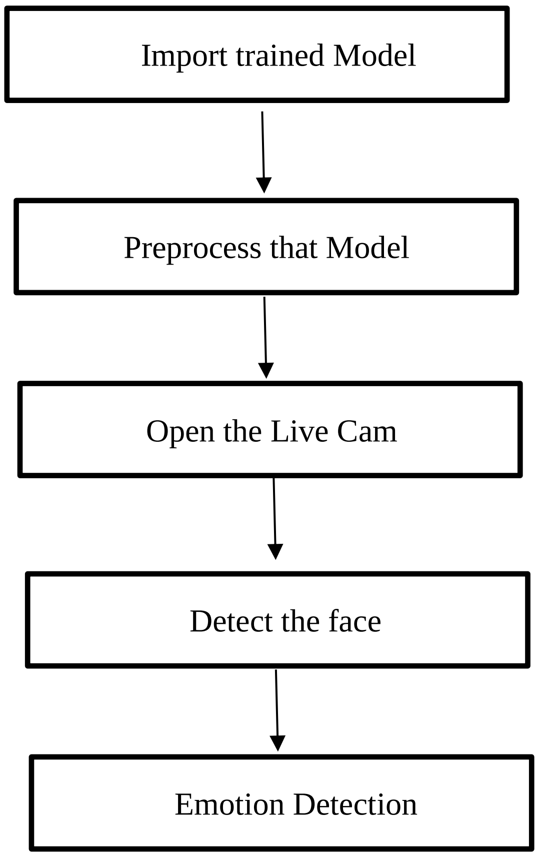
It includes functions for loading images from disk, resizing, normalization, data augmentation, andmore.

In this code, img_to_array converts images to NumPy arrays, which can be directly fed into the neural network model, while expand_dims adds an extra dimension to the array to match the expected input shape of the model.

5. SYSTEM DESIGN

In the real time of facial emotion recognition, the machine learning pipeline typically begins with data acquisition, where labeled facial image datasets are obtained. Following this, data preprocessing steps, including image resizing, data augmentation, and normalization, are performed to prepare the data for training. Model selection involves choosing an appropriate architecture, such as Convolutional Neural Networks (CNNs), and designing the model based on factors like dataset size and complexity. Subsequently, the model is trained on the training dataset, with hyper parameters adjusted based on performance on a separate validation set to prevent overfitting. Evaluation on a test set assesses the model's performance using metrics like accuracy and F1-score, ensuring its efficacy in recognizing emotions.

DATA FLOW DIAGRAM



6. SYSTEM IMPLEMENTATION

For system implementation, the first step involves obtaining a dataset for facial emotion recognition and preprocessing it by resizing, augmenting, and normalizing the images. Next, a suitable pre-trained convolutional neural network (CNN) architecture is selected and fine-tuned on the preprocessed dataset to predict the seven basic emotions. The model is then trained using techniques like mini-batch gradient descent and backpropagation. Finally, the trained model is integrated into a real-time system capable of capturing video frames from a webcam feed, detecting faces, and predicting emotions, with the ability to overlay the detected emotions on the video feed.

Import Required Packages

In this step, the necessary Python packages are imported to facilitate the implementation of real- time facial emotion recognition. These packages include:

OpenCV (cv2): A powerful library for computer vision tasks such as image and video processing.It provides functions for reading video frames, performing face detection, and displaying images.

NumPy (numpy): A fundamental package for numerical computing in Python. It offers support formulti-dimensional arrays and mathematical operations, which are essential for representing and manipulating image data.

Keras (keras): A high-level deep learning library that simplifies the development of neuralnetworks. Keras provides utilities for building, training, and evaluating deep learning models efficiently.

Load Pre-trained Model

This step involves loading a pre-trained convolutional neural network (CNN) model architecture from a JSON file. The model architecture describes the network's structure, including the arrangement of layers and connections. By loading this architecture, we can instantiate the neural network model for facial emotion recognition.

Load Model Weights

Once the model architecture is loaded, the next step is to load the pre-trained weights of the CNN model from an H5 file. These weights represent the learned parameters of the model obtained during the training phase. By loading these weights, the model gains the ability to make accurate predictions on new, unseen data.

Initialize Face Cascade Classifier

A Haar cascade classifier is initialized to detect faces within video frames. Haar cascades are machine learning-based classifiers that are trained to identify specific objects or features within images. In this case, the Haar cascade classifier is trained for frontal face detection, making it suitable for detecting faces in the webcam feed.

Start Video Capture

The video capture process is initiated to capture frames from the webcam in real-time. This involvescreating a video capture object that interfaces with the webcam device. By capturing video frames continuously, we can process each frame to detect faces and recognize emotions in real-time.

Face Detection and Emotion Recognition Loop

Within a loop, the system continuously captures video frames from the webcam feed. For each frame, the following steps are performed:

Convert the frame to grayscale: Grayscale conversion simplifies image processing and enhances the efficiency of face detection algorithms.

Detect faces in the grayscale frame: Utilize the initialized Haar cascade classifier to detect faces within the frame.

Extract facial regions of interest (ROIs): For each detected face, extract the region of the grayscaleframe corresponding to the detected face.

Preprocess the facial ROI: Resize and normalize the extracted facial region to match the input requirements of the pre-trained CNN model.

Feed the preprocessed facial ROI into the CNN model: Utilize the pre-trained model to predict theemotion expressed in the facial region.

Overlay the predicted emotion on the video frame: Display the predicted emotion label on the videoframe to provide visual feedback.

7. SOURCE CODE

img.py

```
import cv2 import argparse
import numpy as np
from keras.models import model_from_json
from keras.preprocessing import image

# Parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument('images/angry.jpg',help='images/angry.jpg')args = vars(ap.parse_args())

# Load model from JSON file
json_file =
open('top_models\\fer.json', 'r')loaded_model_json = json_file.read() json_file.close()
model = model_from_json(loaded_model_json)

# Load weights and them to model model.load_weights('top_models\\fer.h5')
classifier = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

Numpy.py
# start delvewheel patch
def _delvewheel_patch_1_5_2():
    import os
    libs_dir = os.path.abspath(os.path.join(os.path.dirname(__file__), os.pardir,
'numpy.libs'))
    if os.path.isdir(libs_dir):
        os.add_dll_directory(libs_dir)
_delvewheel_patch_1_5_2()
del _delvewheel_patch_1_5_2

# end delvewheel patch
```

```
import sys
import warnings
from ._globals import _NoValue, _CopyMode

# These exceptions were moved in 1.25 and are hidden from __dir__()
from .exceptions import (ComplexWarning, ModuleDeprecationWarning,
VisibleDeprecationWarning,TooHardError, AxisError)
# If a version with git hash was stored, use that instead
from . import version
from .version import __version__

# We first need to detect if we're being called as part of the numpy setup
# procedure itself in a reliable manner.
try:
__NUMPY_SETUP__
except NameError:
__NUMPY_SETUP__ = False

if __NUMPY_SETUP__:
sys.stderr.write('Running from numpy source directory.\n')
else:
# Allow distributors to run custom init code before importing numpy.core
from . import _distributor_init

try:
from numpy.__config__ import show as show_config
except ImportError as e:
msg = """Error importing numpy: you should not try to import numpy from
its source directory; please exit the numpy source tree, and relaunch
img = cv2.imread(args['images/angry.jpg'])

gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) faces_detected =
classifier.detectMultiScale(gray_img, 1.18, 5)
```

```
for (x, y, w, h) in faces_detected:
    cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)roi_gray = gray_img[y:y + w, x:x
    +h]
     roi_gray = cv2.resize(roi_gray, (48, 48)) img_pixels = image.img_to_array(roi_gray)
    img_pixels = np.expand_dims(img_pixels, axis=0)img_pixels /= 255.0
    predictions = model.predict(img_pixels) max_index = int(np.argmax(predictions))

    emotions = ['neutral', 'happiness', 'surprise', 'sadness', 'anger', 'disgust', 'fear']
    predicted_emotion = emotions[max_index]
    cv2.putText(img, predicted_emotion, (int(x), int(y)),
  cv2.FONT_HERSHEY_SIMPLEX, 0.75,(255, 255, 255), 2)
resized_img = cv2.resize(img, (1024, 768)) cv2.imshow('Facial Emotion
Recognition',resized_img)
  if cv2.waitKey(0) & 0xFF == ord('q'):cv2.destroyAllWindows()
```

livcam.py

```
import cv2
import numpy as np
from keras.models import model_from_jsonfrom keras.preprocessing import image

# Load model from JSON file
json_file = open('top_models\\fer.json', 'r')loaded_model_json = json_file.read()
json_file.close()
model = model_from_json(loaded_model_json)

# Load weights and them to model model.load_weights('top_models\\f fer.h5')
face_haar_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
cap = cv2.VideoCapture(0)
while True:
    ret, img = cap.read()if not ret:
argparse.py
import os as _os
import re as _re
import sys as _sys
import warnings
from gettext import gettext as _, ngettext
SUPPRESS = '==SUPPRESS=='
OPTIONAL = '?'
ZERO_OR_MORE = '*'
ONE_OR_MORE = '+'
PARSER = 'A...'
REMAINDER = '...'
_UNRECOGNIZED_ARGS_ATTR = '_unrecognized_args'

# ============================
# Utility functions and classes
# ============================
class _AttributeHolder(object):
    """Abstract base class that provides __repr__.

    The __repr__ method returns a string in the format::
        ClassName(attr=name, attr=name, ...)
    The attributes are determined either by a class-level attribute,
    '_kwarg_names', or by inspecting the instance __dict__.
    """

    def __repr__(self):
        type_name = type(self).__name__
        arg_strings = []
        star_args = {}
        for arg in self._get_args():
            arg_strings.append(repr(arg))
        for name, value in self._get_kwargs():
            if name.isidentifier():
                arg_strings.append('%s=%r' % (name, value))
            else:
                star_args[name] = value
        if star_args:
            arg_strings.append('**%s' % repr(star_args))
        return '%s(%s)' % (type_name, ', '.join(arg_strings))

    def _get_kwargs(self):
        return list(self.__dict__.items())

    def _get_args(self):
        return []


def _copy_items(items):
    if items is None:
        return []
    # The copy module is used only in the 'append' and 'append_const'
    # actions, and it is needed only when the default value isn't a list.
    # Delay its import for speeding up the common case.
    if type(items) is list:
        return items[:]
    import copy
    return copy.copy(items)



# ===============
# Formatting Help
# ===============



class HelpFormatter(object):
    """Formatter for generating usage messages and argument help strings.

    Only the name of this class is considered a public API. All the methods
    provided by the class are considered an implementation detail.
    """

    def __init__(self,
                 prog,
                 indent_increment=2,
                 max_help_position=24,
                 width=None):

        # default setting for width
        if width is None:
            import shutil
            width = shutil.get_terminal_size().columns
            width -= 2

        self._prog = prog
        self._indent_increment = indent_increment
        self._max_help_position = min(max_help_position,
                                      max(width - 20, indent_increment * 2))


         self._width = width
        self._current_indent = 0
        self._level = 0
        self._action_max_length = 0
        self._root_section = self._Section(self, None)
        self._current_section = self._root_section


        self._whitespace_matcher = _re.compile(r'\s+', _re.ASCII)
        self._long_break_matcher = _re.compile(r'\n\n\n+'
        break
        gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        faces_detected = face_haar_cascade.detectMultiScale(gray_img, 1.1, 6, minSize=(150, 150))
        for (x, y, w, h) in faces_detected:

                cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), thickness=2)
                roi_gray = gray_img[y:y + w, x:x + h]

                roi_gray = cv2.resize(roi_gray, (48, 48)) img_pixels
                = image.img_to_array(roi_gray) img_pixels =
                np.expand_dims(img_pixels, axis=0)img_pixels /=
                255.0
                predictions = model.predict(img_pixels) max_index = int(np.argmax(predictions))
                emotions = ['neutral', 'happiness', 'surprise', 'sadness', 'anger', 'disgust', 'fear']
                predicted_emotion = emotions[max_index]
                cv2.putText(img, predicted_emotion, (int(x), int(y)), cv2.FONT_HERSHEY_SIMPLEX,0.75,
         (255, 255, 255), 2)
                resized_img = cv2.resize(img, (1000, 700)) cv2.imshow('Facial Emotion
                Recognition',resized_img
               cap.release()          cv2.destroyAllWindows()
```

vid.py

```
import argparse
import cv2
import numpy as np
from keras.models import model_from_jsonfrom keras.preprocessing import image
# Parse the video file path argumentap = argparse.ArgumentParser()
ap.add_argument('video', help='path to input video file')args = vars(ap.parse_args())
# Loading JSON model
json_file = open('top_models\\fer.json', 'r')loaded_model_json = json_file.read() json_file.close()
model = model_from_json(loaded_model_json)
cap = cv2.VideoCapture(args['video'])
while True:
    ret, img = cap.read()if not ret:
        break
     gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
     faces_detected = face_haar_cascade.detectMultiScale(gray_img, 1.2, 6)
     for (x, y, w, h) in faces_detected:
        cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), thickness=2)roi_gray = gray_img[y:y +
        w, x:x + h]
        roi_gray = cv2.resize(roi_gray, (48, 48)) img_pixels = image.img_to_array(roi_gray)
        img_pixels = np.expand_dims(img_pixels, axis=0)img_pixels /= 255.0
        predictions = model.predict(img_pixels) max_index = int(np.argmax(predictions[0]))
        emotions = ['neutral', 'happiness', 'surprise', 'sadness', 'anger', 'disgust', 'fear', 'contempt']
        predicted_emotion = emotions[max_index]
         cv2.putText(img, predicted_emotion, (int(x), int(y)), cv2.FONT_HERSHEY_SIMPLEX,
0.75, (255, 255, 255), 2)


     resized_img = cv2.resize(img, (1024, 768)) cv2.imshow('Facial Emotion Recognition',
     resized_img)if cv2.waitKey(1) & 0xFF ==ord('q'):


        break cap.release()
```

```
cv2.destroyAllWindows()


cv2.putText(img, predicted_emotion, (int(x), int(y)), cv2.FONT_HERSHEY_SIMPLEX,0.75,
(255, 255, 255), 2)
    resized_img = cv2.resize(img, (1024, 768)) cv2.imshow('Facial Emotion Recognition',
    resized_img)if cv2.waitKey(1) & 0xFF ==ord('q'):

        break cap.release()
cv2.destroyAllWindows()
# start delvewheel patch
def _delvewheel_patch_1_5_2():
    import os
    libs_dir = os.path.abspath(os.path.join(os.path.dirname(__file__), os.pardir, 'numpy.libs'))
    if os.path.isdir(libs_dir):
        os.add_dll_directory(libs_dir)


_delvewheel_patch_1_5_2()
del _delvewheel_patch_1_5_2
# end delvewheel patch

import sys
import warnings

from ._globals import _NoValue, _CopyMode
# These exceptions were moved in 1.25 and are hidden from __dir__()
from .exceptions import (
    ComplexWarning, ModuleDeprecationWarning, VisibleDeprecationWarning,
    TooHardError, AxisError)


# If a version with git hash was stored, use that instead
from . import version
from .version import __version__

# We first need to detect if we're being called as part of the numpy setup
# procedure itself in a reliable manner.
try:
    __NUMPY_SETUP__
except NameError:
    __NUMPY_SETUP__ = False

if __NUMPY_SETUP__:
    sys.stderr.write('Running from numpy source directory.\n')
else:
    # Allow distributors to run custom init code before importing numpy.core
    from . import _distributor_init

    try:
        from numpy.__config__ import show as show_config
    except ImportError as e:
        msg = """Error importing numpy: you should not try to import numpy from
        its source directory; please exit the numpy source tree, and relaunch
        your python interpreter from there."""
        raise ImportError(msg) from e

    __all__ = [
        'exceptions', 'ModuleDeprecationWarning', 'VisibleDeprecationWarning',
        'ComplexWarning', 'TooHardError', 'AxisError']

    # mapping of {name: (value, deprecation_msg)}
    __deprecated_attrs__ = {}

    from . import core
    from .core import *
    from . import compat
    from . import exceptions
    from . import dtypes
    from . import lib
    # NOTE: to be revisited following future namespace cleanup.
    # See gh-14454 and gh-15672 for discussion.
    from .lib import *

    from . import linalg
    from . import fft
    from . import polynomial
    from . import random
    from . import ctypeslib
    from . import ma
    from . import matrixlib as _mat
    from .matrixlib import *

    # Deprecations introduced in NumPy 1.20.0, 2020-06-06
    import builtins as _builtins

    _msg = (
        "module 'numpy' has no attribute '{n}'.\n"
        "`np.{n}` was a deprecated alias for the builtin `{n}`. "
        "To avoid this error in existing code, use `{n}` by itself. "
        "Doing this will not modify any behavior and is safe. {extended_msg}\n"
        "The aliases was originally deprecated in NumPy 1.20; for more "
        "details and guidance see the original release note at:\n"
        "    https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations")

    _specific_msg = (
        "If you specifically wanted the numpy scalar type, use `np.{}` here.")

    _int_extended_msg = (
        "When replacing `np.{}`, you may wish to use e.g. `np.int64` "
        "or `np.int32` to specify the precision. If you wish to review "
        "your current use, check the release note link for "
        "additional information.")

    _type_info = [
        ("object", ""),  # The NumPy scalar only exists by name.
        ("bool", _specific_msg.format("bool_")),
        ("float", _specific_msg.format("float64")),
        ("complex", _specific_msg.format("complex128")),
        ("str", _specific_msg.format("str_")),
        ("int", _int_extended_msg.format("int"))]

    __former_attrs__ = {
         n: _msg.format(n=n, extended_msg=extended_msg)
         for n, extended_msg in _type_info
     }

    # Future warning introduced in NumPy 1.24.0, 2022-11-17
    _msg = (
        "`np.{n}` is a deprecated alias for `{an}`.  (Deprecated NumPy 1.24)")

    # Some of these are awkward (since `np.str` may be preferable in the long
    # term), but overall the names ending in 0 seem undesirable
    _type_info = [
        ("bool8", bool_, "np.bool_"),
        ("int0", intp, "np.intp"),
        ("uint0", uintp, "np.uintp"),
        ("str0", str_, "np.str_"),
        ("bytes0", bytes_, "np.bytes_"),
        ("void0", void, "np.void"),
        ("object0", object_,
            "`np.object0` is a deprecated alias for `np.object_`. "
            "`object` can be used instead.  (Deprecated NumPy 1.24)")]
```

```
# Some of these could be defined right away, but most were aliases to
# the Python objects and only removed in NumPy 1.24.  Defining them should
# probably wait for NumPy 1.26 or 2.0.
# When defined, these should possibly not be added to `__all__` to avoid
# import with `from numpy import *`.
__future_scalars__ = {"bool", "long", "ulong", "str", "bytes", "object"}

__deprecated_attrs__.update({
    n: (alias, _msg.format(n=n, an=an)) for n, alias, an in _type_info})

import math

__deprecated_attrs__['math'] = (math,
    "`np.math` is a deprecated alias for the standard library `math` "
    "module (Deprecated Numpy 1.25). Replace usages of `np.math` with "
    "`math`")

del math, _msg, _type_info

from .core import abs
# now that numpy modules are imported, can initialize limits
core.getlimits._register_known_types()

__all__.extend(['__version__', 'show_config'])
__all__.extend(core.__all__)
__all__.extend(_mat.__all__)
__all__.extend(lib.__all__)
__all__.extend(['linalg', 'fft', 'random', 'ctypeslib', 'ma'])

# Remove min and max from __all__ to avoid `from numpy import *` override
# the builtins min/max. Temporary fix for 1.25.x/1.26.x, see gh-24229.
__all__.remove('min')
__all__.remove('max')
__all__.remove('round')

# Remove one of the two occurrences of `issubdtype`, which is exposed as
# both `numpy.core.issubdtype` and `numpy.lib.issubdtype`.
__all__.remove('issubdtype')

# These are exported by np.core, but are replaced by the builtins below
# remove them to ensure that we don't end up with `np.long == np.int_`,
# which would be a breaking change.
del long, unicode
__all__.remove('long')
__all__.remove('unicode')

# Remove things that are in the numpy.lib but not in the numpy namespace
# Note that there is a test (numpy/tests/test_public_api.py:test_numpy_namespace)
# that prevents adding more things to the main namespace by accident.
# The list below will grow until the `from .lib import *` fixme above is
# taken care of
__all__.remove('Arrayterator')
del Arrayterator

# These names were removed in NumPy 1.20.  For at least one release,
# attempts to access these names in the numpy namespace will trigger
# a warning, and calling the function will raise an exception.
_financial_names = ['fv', 'ipmt', 'irr', 'mirr', 'nper', 'npv', 'pmt',
                    'ppmt', 'pv', 'rate']
__expired_functions__ = {
    name: (f'In accordance with NEP 32, the function {name} was removed '
           'from NumPy version 1.20.  A replacement for this function '
           'is available in the numpy_financial library: '
           'https://pypi.org/project/numpy-financial')
    for name in _financial_names}
```

```
# Filter out Cython harmless warnings
warnings.filterwarnings("ignore", message="numpy.dtype size changed")
warnings.filterwarnings("ignore", message="numpy.ufunc size changed")
warnings.filterwarnings("ignore", message="numpy.ndarray size changed")

# oldnumeric and numarray were removed in 1.9. In case some packages import
# but do not use them, we define them here for backward compatibility.
oldnumeric = 'removed'
numarray = 'removed'

def __getattr__(attr):
    # Warn for expired attributes, and return a dummy function
    # that always raises an exception.
    import warnings
    import math
    try:
        msg = __expired_functions__[attr]
    except KeyError:
        pass
    else:
        warnings.warn(msg, DeprecationWarning, stacklevel=2)

        def _expired(*args, **kwds):
            raise RuntimeError(msg)

        return _expired

    # Emit warnings for deprecated attributes
    try:
        val, msg = __deprecated_attrs__[attr]
    except KeyError:
        pass
    else:
        warnings.warn(msg, DeprecationWarning, stacklevel=2)
        return val

    if attr in __future_scalars__:
        # And future warnings for those that will change, but also give
        # the AttributeError
        warnings.warn(
            f"In the future `np.{attr}` will be defined as the "
            "corresponding NumPy scalar.", FutureWarning, stacklevel=2)

    if attr in __former_attrs__:
        raise AttributeError(__former_attrs__[attr])

    if attr == 'testing':
        import numpy.testing as testing
        return testing
    elif attr == 'Tester':
        "Removed in NumPy 1.25.0"
        raise RuntimeError("Tester was removed in NumPy 1.25.")

    raise AttributeError("module {!r} has no attribute "
                         "{!r}".format(__name__, attr))

def __dir__():
    public_symbols = globals().keys() | {'testing'}
    public_symbols -= {
        "core", "matrixlib",
        # These were moved in 1.25 and may be deprecated eventually:
        "ModuleDeprecationWarning", "VisibleDeprecationWarning",
        "ComplexWarning", "TooHardError", "AxisError"
    }
    return list(public_symbols)
```

```
# Pytest testing
from numpy._pytesttester import PytestTester
test = PytestTester(__name__)
del PytestTester

def _sanity_check():
    """
    Quick sanity checks for common bugs caused by environment.
    There are some cases e.g. with wrong BLAS ABI that cause wrong
    results under specific runtime conditions that are not necessarily
    achieved during test suite runs, and it is useful to catch those early.

    See https://github.com/numpy/numpy/issues/8577 and other
    similar bug reports.

    """
    try:
        x = ones(2, dtype=float32)
        if not abs(x.dot(x) - float32(2.0)) < 1e-5:
            raise AssertionError()
    except AssertionError:
        msg = ("The current Numpy installation ({!r}) fails to "
               "pass simple sanity checks. This can be caused for example "
               "by incorrect BLAS library being linked in, or by mixing "
               "package managers (pip, conda, apt, ...). Search closed "
               "numpy issues for similar problems.")
        raise RuntimeError(msg.format(__file__)) from None

_sanity_check()
del _sanity_check

def _mac_os_check():
    """
    Quick Sanity check for Mac OS look for accelerate build bugs.
    Testing numpy polyfit calls init_dgelsd(LAPACK)
    """
    try:
        c = array([3., 2., 1.])
        x = linspace(0, 2, 5)
        y = polyval(c, x)
        _ = polyfit(x, y, 2, cov=True)
    except ValueError:
        pass

if sys.platform == "darwin":
    from . import exceptions
    with warnings.catch_warnings(record=True) as w:
        _mac_os_check()
        # Throw runtime error, if the test failed Check for warning and error_message
        if len(w) > 0:
            for _wn in w:
                if _wn.category is exceptions.RankWarning:
                    # Ignore other warnings, they may not be relevant (see gh-25433).
                    error_message = f"{_wn.category.__name__}: {str(_wn.message)}"
                    msg = (
                        "Polyfit sanity test emitted a warning, most likely due "
                        "to using a buggy Accelerate backend."
                        "\nIf you compiled yourself, more information is available at:"
                        "\nhttps://numpy.org/devdocs/building/index.html"
                        "\nOtherwise report this to the vendor "
                        "that provided NumPy.\n\n{}\n".format(error_message))
                    raise RuntimeError(msg)
            del _wn
        del w
del _mac_os_check
```

```
# We usually use madvise hugepages support, but on some old kernels it
# is slow and thus better avoided.
# Specifically kernel version 4.6 had a bug fix which probably fixed this:
# https://github.com/torvalds/linux/commit/7cf91a98e607c2f935dbcc177d70011e95b8faff
import os
use_hugepage = os.environ.get("NUMPY_MADVISE_HUGEPAGE", None)
if sys.platform == "linux" and use_hugepage is None:
    # If there is an issue with parsing the kernel version,
    # set use_hugepages to 0. Usage of LooseVersion will handle
    # the kernel version parsing better, but avoided since it
    # will increase the import time. See: #16679 for related discussion.
    try:
        use_hugepage = 1
        kernel_version = os.uname().release.split(".")[:2]
        kernel_version = tuple(int(v) for v in kernel_version)
        if kernel_version < (4, 6):
            use_hugepage = 0
    except ValueError:
        use_hugepages = 0
elif use_hugepage is None:
    # This is not Linux, so it should not matter, just enable anyway
    use_hugepage = 1
else:
    use_hugepage = int(use_hugepage)

# Note that this will currently only make a difference on Linux
core.multiarray._set_madvise_hugepage(use_hugepage)
del use_hugepage

# Give a warning if NumPy is reloaded or imported on a sub-interpreter
# We do this from python, since the C-module may not be reloaded and
# it is tidier organized.
```

```
    core.multiarray._multiarray_umath._reload_guard()

    # default to "weak" promotion for "NumPy 2".
    core._set_promotion_state(
        os.environ.get("NPY_PROMOTION_STATE",
                       "weak" if _using_numpy2_behavior() else "legacy"))

    # Tell PyInstaller where to find hook-numpy.py
    def _pyinstaller_hooks_dir():
        from pathlib import Path
        return [str(Path(__file__).with_name("_pyinstaller").resolve())]

    # Remove symbols imported for internal use
    del os


# Remove symbols imported for internal use
del sys, warnings
```

8. SYSTEM TESTING

If all parts of the system are correct, the goal will be successfully achieved. In the testing process we test the actual system in an organization and gather errors from the new system operatesin full efficiency as stated. System testing is the stage of implementation, which is aimed to ensuringthat the system works accurately and efficiently.

In the testing process we test the actual system in an organization and gather errors from the new system Testing is vital to the success of the system. System testing makes a logical assumption andtake initiatives to correct the same. All the front-end and back-end connectivity are tested to be surethat the new system operates in full efficiency as stated. System testing is the stage of implementation, which is aimed at ensuring that the system works accurately and efficiently.

The main objective of testing is to uncover errors from the system. For the uncovering process wehave to give proper input data to the system. So we should have more conscious to give input data.It is important to give correct inputs to efficient testing.

Testing is done for each module. After testing all the modules, the modules are integrated and testing of the final system is done with the test data, specially designed to show that the system willoperate successfully in all its aspects conditions. Thus the system testing is a confirmation that all is correct and an opportunity to show the user that the system works. Inadequate testing or non- testing leads to errors that may appear few months later.

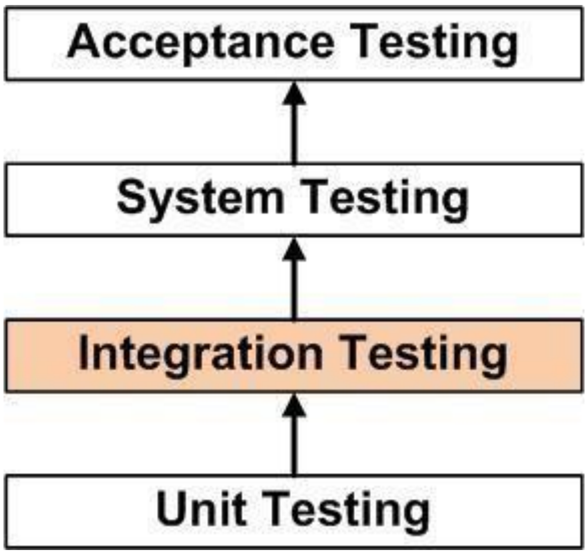
This will create two problems, Time delay between the cause and appearance of the problem.The effect of the system errors on files and records within the system. The purpose of the system testing is to consider all the likely variations to which it will be suggested and push the system to its limits.

The testing process focuses on logical intervals of the software ensuring that all the statements havebeen tested and on the function intervals (i.e.,) conducting tests to uncover errors and ensure that defined inputs will produce actual results that agree with the required results. Testing has to be doneusing the two common steps Unit testing and Integration testing. In the project system testing is made as follows:

The procedure level testing is made first. By giving improper inputs, the errors occurred are noted and eliminated. This is the final step in system life cycle. Here we implement the tested error-free system into real-life environment and make necessary changes, which runs in an online fashion.

Here system maintenance is done every months or year based on company policies, and is checkedfor errors like runtime errors, long run errors and other maintenances like table verification and reports.

Integration Testing is a level of software testing where individual units are combined and tested as agroup.



METHOD

Any of Black Box Testing, White Box Testing, and Grey Box Testing methods can be used.Normally, the method depends on your definition of 'unit'.

TASKS

- Integration Test Plan
  - Prepare
  - Review
  - Rework
  - Baseline
- Integration Test Cases/Scripts
  - Prepare
  - Review
  - Rework
  - Baseline
  - Integration
  - Perform

UNIT TESTING

Unit testing verification efforts on the smallest unit of software design, module. This is knownas “Module Testing”. The modules are tested separately. This testing is carried out during programming stage itself. In these testing steps, each module is found to be working satisfactorily as regard to the expected output from the module.

BLACK BOX TESTING

Black box testing, also known as Behavioral Testing, is a software testing method in which the internal structure/ design/ implementation of the item being tested is not known to the tester. These tests can be functional or non-functional, though usually functional.

WHITE-BOX TESTING

White-box testing (also known as clear box testing, glass box testing, transparent box testing, and structural testing) is a method of testing software that tests internal structures or workings of an application, as opposed to its functionality (i.e. black-box testing).

GREY BOX TESTING

Grey box testing is a technique to test the application with having a limited knowledge of the internal workings of an application. To test the Web Services application usually the Grey box testing is used. Grey box testing is performed by end-users and also by testers and developers.

INTEGRATION TESTING

Integration testing is a systematic technique for constructing tests to uncover error associated within the interface. In the project, all the modules are combined and then the entire programmer is tested as a whole.

In the integration-testing step, all the error uncovered is corrected for the next testing steps.

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.

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 for Data Synchronization:

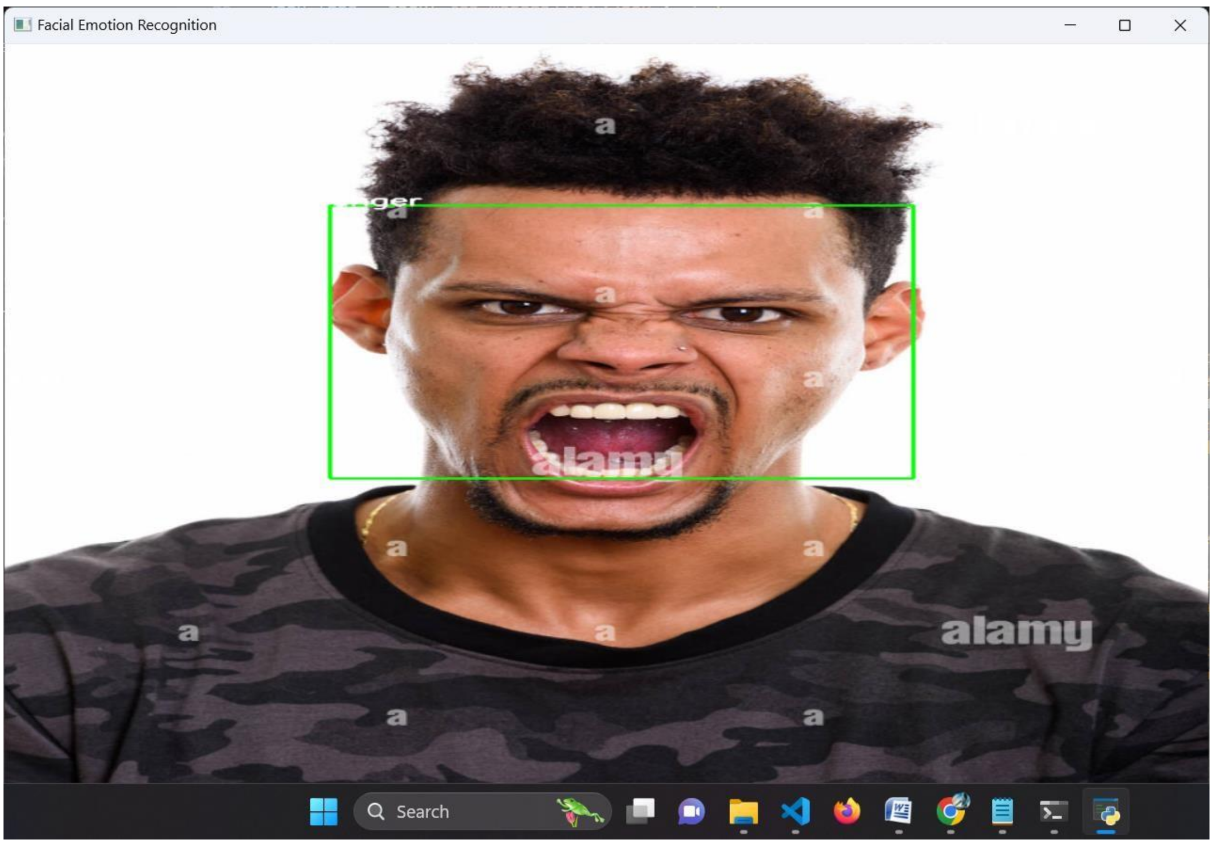
- ➢ The Acknowledgements will be received by the Sender Node after the Packet sare received by the Destination Node
- ➢ The Route add operation is done only when there is a Route request in need
- ➢ The Status of Nodes information is done automatically in the Cache Updating process

BUILD THE TEST PLAN

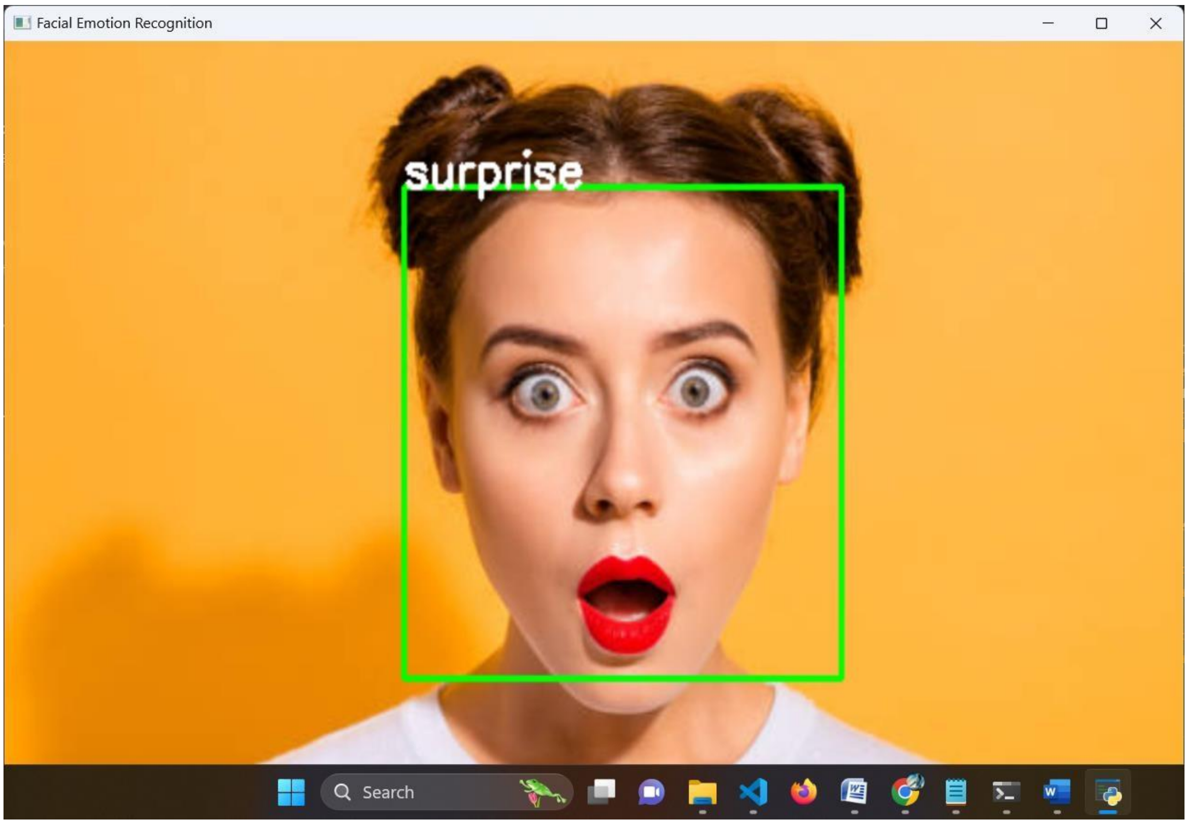
Any project can be divided into units that can be further performed for detailed processing. Then a testing strategy for each of this unit is carried out. Unit testing helps to identify the possiblebugs in the individual component, so the component that has bugs are identified and are rectified from errors.

9. SCREEN LAYOUT

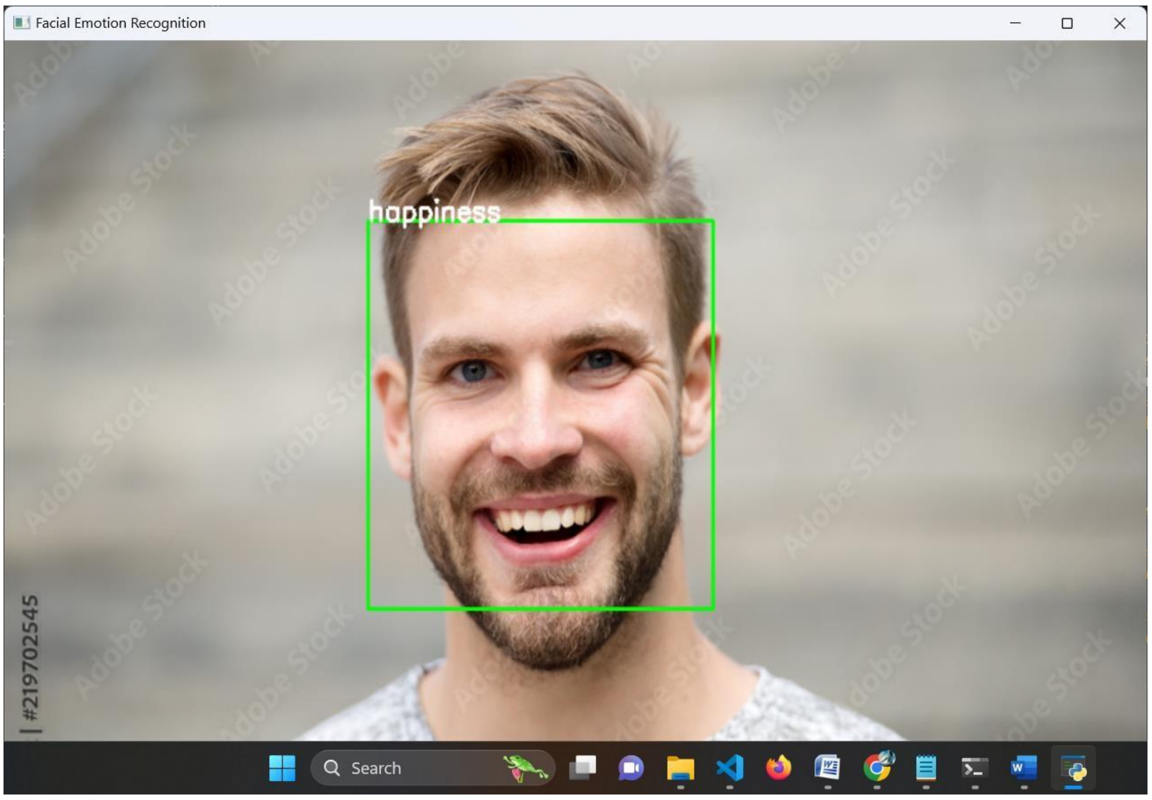
ANGRY



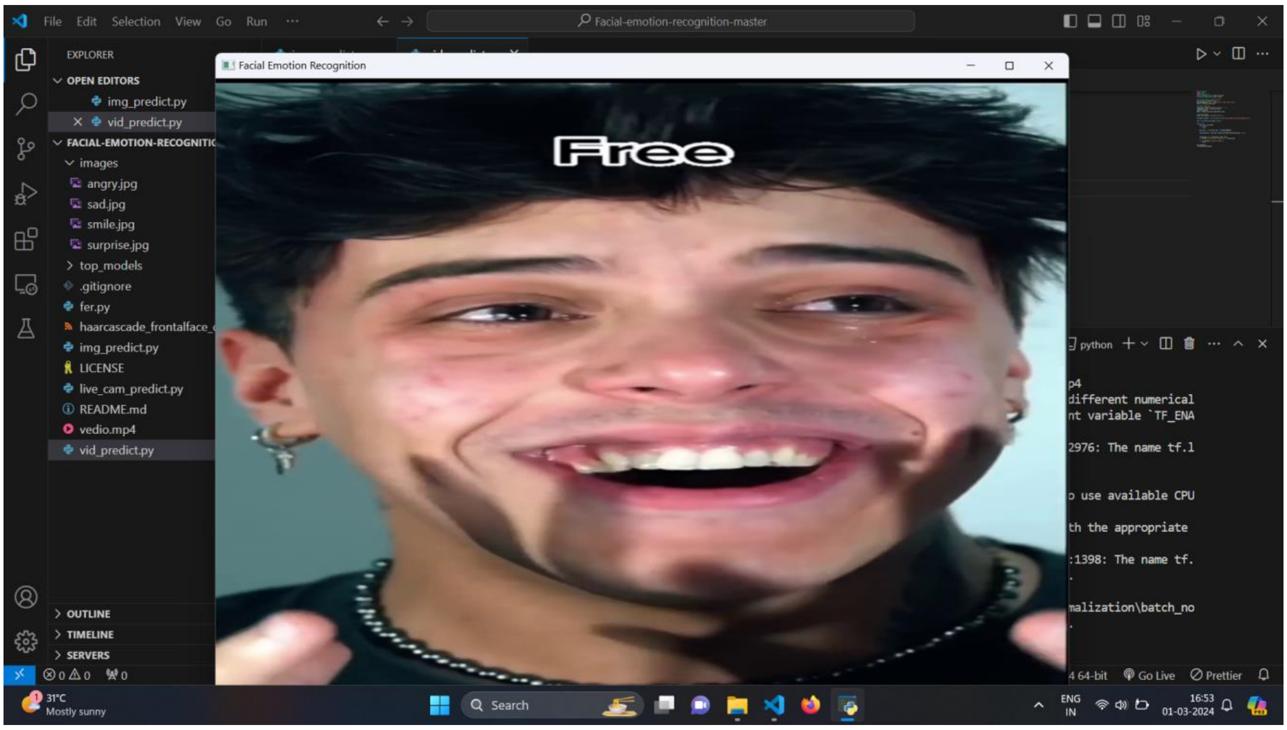
SURPRISE



HAPPINESS

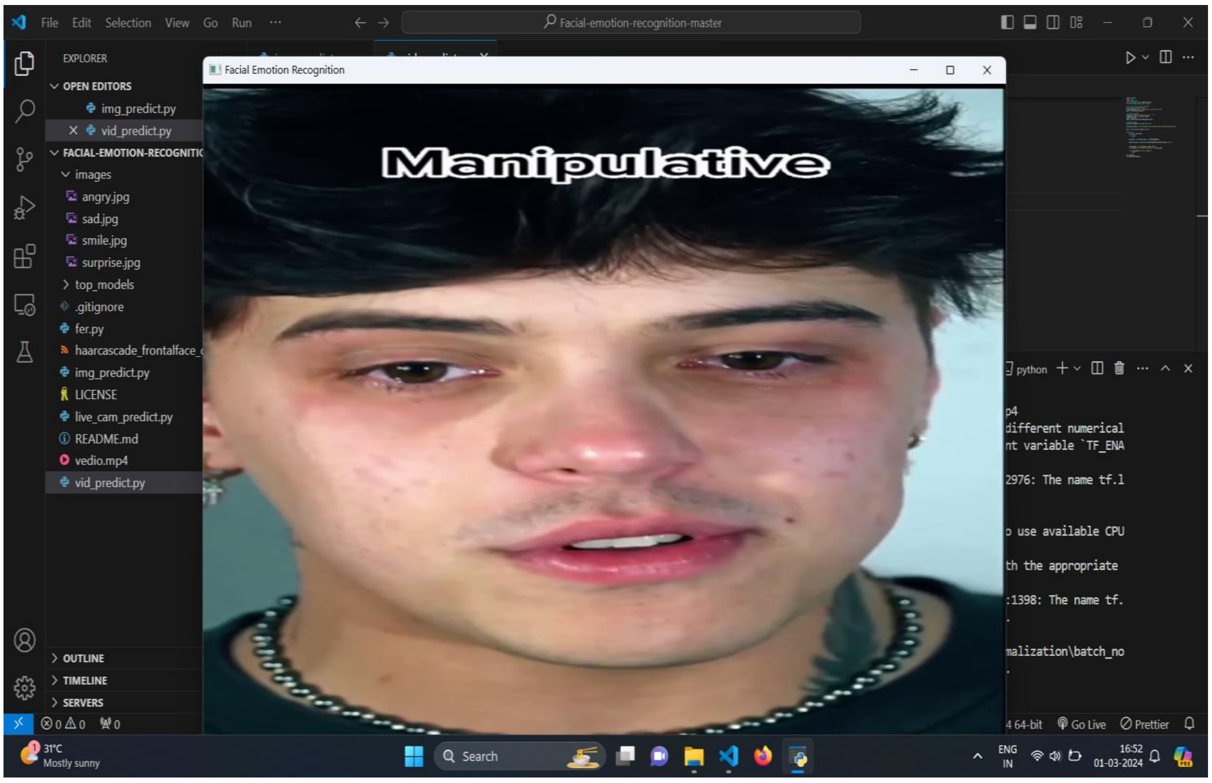


FREE



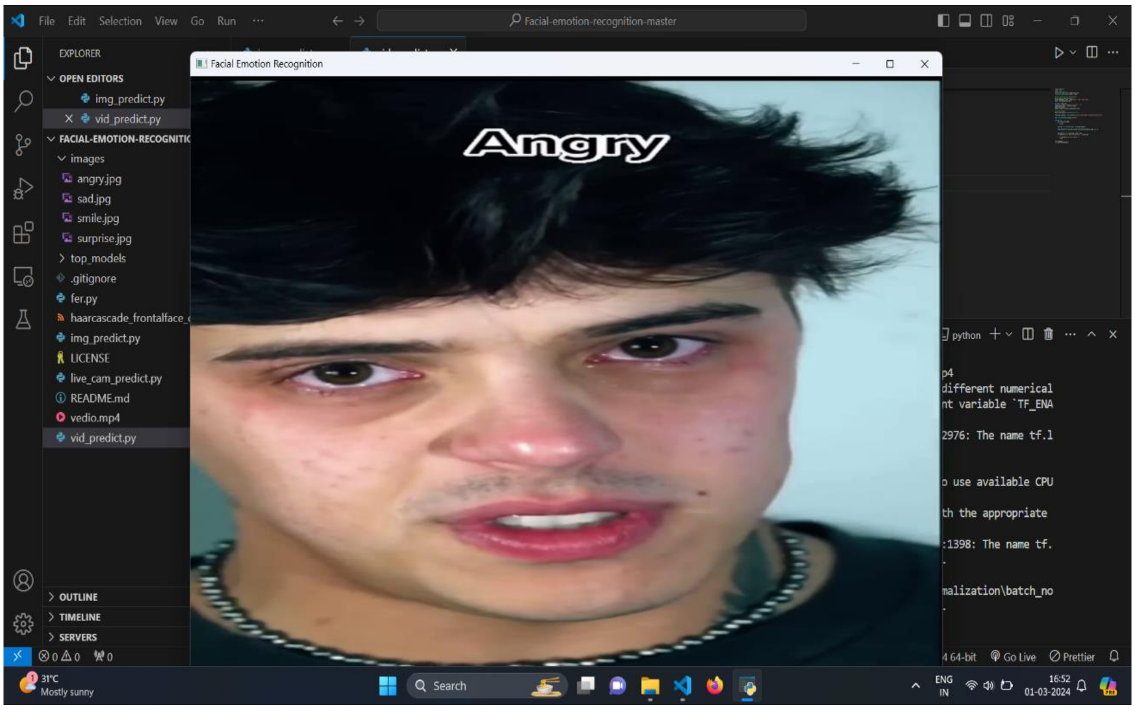
```
p4
different numerical
nt variable `TF_ENA

2976: The name tf.1

o use available CPU

th the appropriate

:1398: The name tf.

malization\batch_no
```

MANIPULATIVE



```
p4
different numerical
nt variable `TF_ENA

2976: The name tf.l

o use available CPU

th the appropriate

:1398: The name tf.
.

malization\batch_no
.
```

ANGRY



```
p4
different numerical
nt variable `TF_ENA

2976: The name tf.l


o use available CPU

th the appropriate

:1398: The name tf.
.

malization\batch_no
.
```

10. CONCLUSION & FUTURE ENHANCEMENT

10.1 CONCLUSION

In conclusion, the implemented system showcases the integration of computer vision and deeplearning techniques for real-time facial emotion recognition. By leveraging pre-trained convolutional neural network (CNN) models and Haar cascade classifiers, the system can accurately detect faces in live webcam video feeds and predict the corresponding emotions expressed by individuals.

The system's ability to recognize emotions in real-time has significant implications across various domains, including human-computer interaction, mental health monitoring, marketingresearch, and security surveillance. By providing immediate feedback on detected emotions, the system enhances user experiences, facilitates personalized interactions, and contributes to the development of emotion-aware technologies.

10.2 FUTURE ENHANCEMENT

In future the system can be enhanced with the following feature.

Improved Face Detection: Utilize more advanced face detection algorithms or deep learning-based face detectors such as SSD (Single Shot MultiBox Detector) or MTCNN (Multi-Task Cascaded Convolutional Networks) for better accuracy and robustness in detecting faces, especially under varying lighting conditions, occlusions, or pose variations.

Facial Landmark Detection: Incorporate facial landmark detection to accurately localize key facial landmarks such as eyes, nose, and mouth. This can help in better alignment of facial regions and improve emotion recognition accuracy.

Dynamic Region of Interest (ROI) Selection: Implement dynamic ROI selection to adaptively focus on the most expressive regions of the face for emotion analysis. This can involve tracking facial movements or changes in expression over time to refine the regions used for emotion recognition.

Model Architecture Enhancement: Experiment with more advanced CNN architectures or explore other deep learning models such as recurrent neural networks (RNNs) or attention mechanisms to capture temporal dependencies in facial expressions and improve emotion recognition performance.

Data Augmentation and Synthesis: Augment the training data with various transformations such as rotation, scaling, and brightness adjustments to improve model generalization and robustness to variations in facial expressions and environmental conditions.

Transfer Learning and Fine-Tuning: Investigate the use of transfer learning techniques by fine-tuning pre-trained models on larger and more diverse datasets to further improve emotion recognition accuracy, especially for less frequently occurring emotions or subtle expressions.

BIBLIOGRAPHY

BOOK REFERENCE

- Wesley J. Chun, “Core Python Programming”, Second Edition, Prentice Hall Publication, 2006.
- Timothy A Budd, “Exploring Python”, Tata McGraw Hill, New Delhi, ISBN:780071321228
- “Digital Image Processing”, S. Sridhar, Oxford University Press; Second edition, 2016.

WEB REFERENCE

- www://.w3schools.com/
- www://python.org
- www://docs.python.org
- www://programiz.com/python