EMOTION DETECTION MODEL

A PROJECT REPORT

In partial fulfilment of the requirements for the award of the degree

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

Under the guidance of

DHRUBA RAY

BY

ARINDAM SAHOO

ARNAB GHORUI

ARANYA JANA

SUJATA MONDAL

NARULA INSTITUTE OF TECHNOLOGY

In association with

(ISO9001:2015)

SDF Building, Module #132, Ground Floor, Salt Lake City, GP Block, Sector V, Kolkata, West Bengal 700091

(Note: All entries of the proforma of approval should be filled up with appropriate and complete information. Incomplete proforma of approval in any respect will be summarily rejected.)

1. Title of the Project: **EMOTION DETECTION MODEL**
2. Project Members: **ARINDAM SAHOO**
   ARNAB GHORUI
   ARANYA JANA
   SUJATA MONDAL
3. Name of Guide: **MR. DHRUBA RAY**
4. Address: **Ardent Computech Pvt. Ltd**
   (An ISO 9001:2015 Certified)
   SDF Building, Module #132, Ground Floor, Salt Lake City, GP Block, Sector V, Kolkata, West Bengal, 700091

Project Version Control History

Version	**Primary Author**	**Description of Version**	**Date Completed**
Final	ARINDAM SAHOO ARNAB GHORUI ARANYA JANA SUJATA MONDAL	Project Report	10TH APRIL,2023

Signature of Team Member　　　　　　　　Signature of Approver

Date:　　　　　　　　　　　　　　　　　Date:

For Office Use Only

MR. DHRUBA RAY

DECLARATION

We hereby declare that the project work being presented in the project proposal entitled **"EMOTION DETECTION MODEL"** in partial fulfilment of the requirements for the award of the degree of **BACHELOR OF TECHNOLOGY at ARDENT COMPUTECH PVT. LTD, SALTLAKE, KOLKATA, WEST BENGAL,** is an authentic work carried out under the guidance of MR**. DHRUBA A RAY**. The matter embodied in this project work has not been submitted elsewhere for the award of any degree of our knowledge and belief.

Date:

Name of the Student: ARINDAM SAHOO
ARNAB GHORUI
ARANYA JANA
SUJATA MONDAL

Signature of the students:

Ardent Computech Pvt. Ltd (An ISO 9001:2015 Certified) SDF Building, Module #132, Ground Floor, Salt Lake City, GP Block, Sector V, Kolkata, West Bengal 700091

CERTIFICATE

This is to certify that this proposal of minor project entitled **“EMOTION DETECTION MODEL”** is a record of work, carried out by ***ARINDAM SAHOO, ARNAB GHORUI, ARANYA JANA, SUJATA MONDL*** under my guidance at **ARDENT COMPUTECH PVT. LTD.** In my opinion, the report in its present form is in partial fulfilment of the requirements for the award of the degree of **BACHELOR OF TECHNOLOGY** and as per regulations of the **ARDENT®.** To the best of my knowledge, the results embodied in this report, are original in nature and worthy of incorporation in the present version of the report.

Guide / Supervisor

MR. DHRUBA RAY

Ardent Computech Pvt. Ltd (An ISO 9001:2015 Certified) SDF Building,

Module #132, Ground Floor, Salt Lake City, GP Block, Sector V, Kolkata, West Bengal 700091

ACKNOWLEDGEMENT

Success of any project depends largely on the encouragement and guidelines of many others. I take this sincere opportunity to express my gratitude to the people who have been instrumental in the successful completion of this project work.

I would like to show our greatest appreciation to ***Mr. DHRUBA RAY***, Project Engineer at Ardent, Kolkata. I always feel motivated and encouraged every time by his valuable advice and constant inspiration; without his encouragement and guidance this project would not have materialized.

Words are inadequate in offering our thanks to the other trainees, project assistants and other members at Ardent Computech Pvt. Ltd. for their encouragement and cooperation in carrying out this project work. The guidance and support received from all the members and who are contributing to this project, was vital for the success of this project.

CONTENTS

OVERVIEW

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and has fewer syntactical constructions than other languages.

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 is Object-Oriented: Python supports Object-Oriented style or technique of programming that encapsulates code within objects.

Python is a Beginner's Language: Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

HISTORY OF PYTHON

Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands. Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, Small Talk, UNIX shell, and other scripting languages. Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL). Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

FEATURES OF PYTHON

Easy-to-learn: Python has few Keywords, simple structure and clearly defined syntax. This allows a student to pick up the language quickly.

Easy-to-Read: Python code is more clearly defined and visible to the

eyes. Easy -to-Maintain: Python's source code is fairly easy-to-maintain.

A broad standard library: Python's bulk of the library is very portable and cross platform compatible on UNIX, Windows, and Macintosh.

Interactive Mode: Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.

Portable: Python can run on the wide variety of hardware platforms and has the same interface on all platforms.

Extendable: You can add low level modules to the python interpreter. These modules enables programmers to add to or customize their tools to be more efficient.

Databases: Python provides interfaces to all major commercial databases.

GUI Programming: Python supports GUI applications that can be created and ported to many system calls, libraries, and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.

Scalable: Python provides a better structure and support for large programs than shell scripting.

Apart from the above-mentioned features, Python has a big list of good features, few are listed below:

• It support functional and structured programming methods as well as OOP. • It can be used as a scripting language or can be compiled to byte code for building large applications.
• It provides very high level dynamic datatypes and supports dynamic type checking.
• It supports automatic garbage collections.
• It can be easily integrated with C, C++, COM, ActiveX, CORBA and JAVA.

ENVIRONMENT SETUP

Open a terminal window and type "python" to find out if it is already installed and which version is installed.

• UNIX (Solaris, Linux, FreeBSD, AIX, HP/UX, SunOS, IRIX, etc.) •

Win 9x/NT/2000

• Macintosh (Intel, PPC, 68K)

• OS/2

• DOS (multiple versions)

• PalmOS

• Nokia mobile phones

• Windows CE

• Acorn/RISC OS

BASIC SYNTAX OF PYTHON PROGRAM

Type the following text at the Python prompt and press the Enter –

```
>>> print "Hello, Python!"
```

If you are running new version of Python, then you would need to use print statement with parenthesis as in **print("Hello, Python!").**
However in Python version 2.4.3, this produces the following result –

```
Hello, Python!
```

Python Identifiers

A Python identifier is a name used to identify a variable, function, class, module or other object. An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9).
Python does not allow punctuation characters such as @, $, and % within identifiers. Python is a case sensitive programming language.

Python Keywords

The following list shows the Python keywords. These are reserved words and you cannot use them as constant or variable or any other identifier names. All the Python keywords contain lowercase letters only.

And, exec, not
Assert, finally, or
Break, for, pass
Class, from, print
continue, global, raise
def, if, return
del, import, try
elif, in, while
else, is, with
except, lambda, yield

Lines & Indentation

Python provides no braces to indicate blocks of code for class and function definitions or flow control. Blocks of code are denoted by line indentation, which is rigidly enforced. The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount. For example –

```
if True:
        print "True"
else:
        print "False"
```

Command Line Arguments

Many programs can be run to provide you with some basic information about how they should be run. Python enables you to do this with -h −

```
$ python-h
usage: python [option]...[-c cmd|-m mod | file |-][arg]...
```

Options and arguments (and corresponding environment variables):

-c cmd: program passed in as string(terminates option list)

-d : debug output from parser (also PYTHONDEBUG=x)

-E : ignore environment variables (such as PYTHONPATH)

-h : print this help message and exit [etc.]

VARIABLE TYPES

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

Assigning Values to Variables

Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables.

```
counter=10 # An integer assignment
weight=10.60 # A floating point
name="Ardent" # A string
```

Multiple Assignment

Python allows you to assign a single value to several variables simultaneously. For example −

```
a = b = c = 1
a,b,c = 1,2,"hello"
```

Standard Data Types

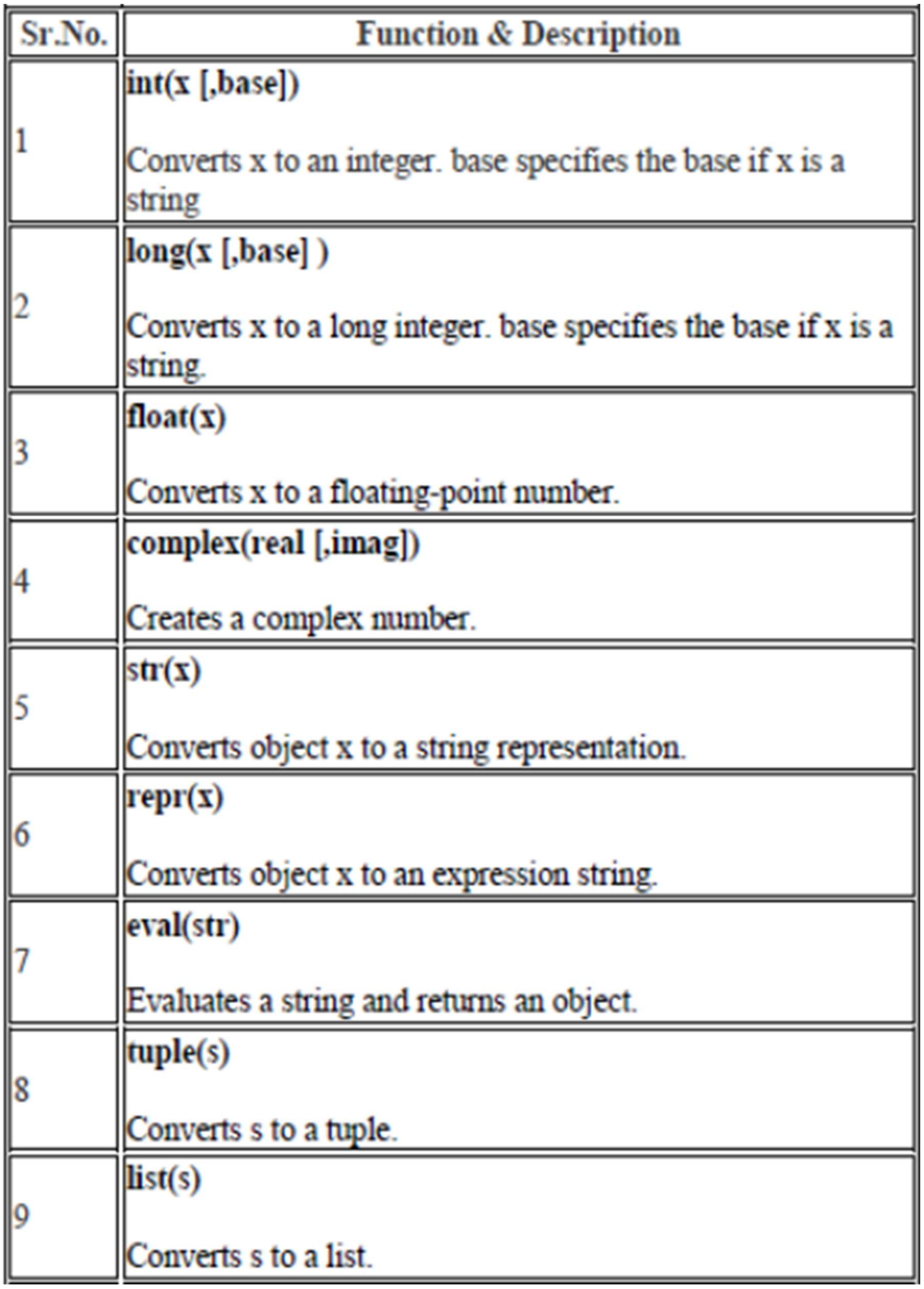
The data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters. Python has five standard data types −

- String
- List
- Tuple
- Dictionary
- Number

Data Type Conversion

Sometimes, you may need to perform conversions between the built-in types. To convert between types, you simply use the type name as a function.

There are several built-in functions to perform conversion from one data type to another.



Sr.No.	Function & Description
1	**int(x [,base])** Converts x to an integer. base specifies the base if x is a string
2	**long(x [,base])** Converts x to a long integer. base specifies the base if x is a string.
3	**float(x)** Converts x to a floating-point number.
4	**complex(real [,imag])** Creates a complex number.
5	**str(x)** Converts object x to a string representation.
6	**repr(x)** Converts object x to an expression string.
7	**eval(str)** Evaluates a string and returns an object.
8	**tuple(s)** Converts s to a tuple.
9	**list(s)** Converts s to a list.

FUNCTIONS

Defining a Function

```
• def function name( parameters ):
        "function_docstring"
        function suite
        return [expression]
```

Pass by reference vs Pass by value

All parameters (arguments) in the Python language are passed by reference. It means if you change what a parameter refers to within a function, the change also reflects back in the calling function. For example –

```
# Function definition is here

def change me(mylist):
        "This changes a passed list into this function"
        mylist.append([1,2,3,4]);
        print"Values inside the function: ",mylist
        return

# Now you can call changeme function

mylist=[10,20,30];
change me(mylist);
print” Values outside the function: ",mylist
```

Here, we are maintaining reference of the passed object and appending values in the same object. So, this would produce the following result −

```
Values inside the function: [10, 20, 30, [1, 2, 3, 4]]
Values outside the function: [10, 20, 30, [1, 2, 3, 4]]
```

Global vs. Local variables

Variables that are defined inside a function body have a local scope, and those defined outside have a global scope . For Example

```
total=0; # This is global variable.

# Function definition is here
def sum( arg1, arg2 ):
```

<u># Add both the parameters and return them."</u>

```
total= arg1 + arg2; # Here total is local variable.
print"Inside the function local total: ", total
return total;
```

<u># Now you can call sum function</u>

```
sum(10,20);
Print"Outside the function global total: ", total
```

When the above code is executed, it produces the following result −

```
Inside the function local total: 30
Outside the function global total: 0
```

MODULES

A module allows you to logically organize your Python code. Grouping related code into a module makes the code easier to understand and use. A module is a Python object with arbitrarily named attributes that you can bind and reference.

The Python code for a module named *aname* normally resides in a file named *aname.py*. Here's an example of a simple module, support.py

```
def print_func( par ):
        print"Hello : ", par
        return
```

The *import* Statement

You can use any Python source file as a module by executing an import statement in some other Python source file. The *import* has the following syntax –

```
Import module1 [, module2 [… moduleN]
```

PACKAGES

A package is a hierarchical file directory structure that defines a single Python application environment that consists of modules and sub packages and sub-subpackages, and so on.

Consider a file *Pots.py* available in *Phone* directory. This file has following line of source code −

```
def Pots ():
        print "I'm Pots Phone"
```

Similar way, we have another two files having different functions with the same name as above –

- *Phone/Isdn.py* file having function Isdn ()
- *Phone/G3.py* file having function G3 ()

Now, create one more file __init__.py in *Phone* directory −

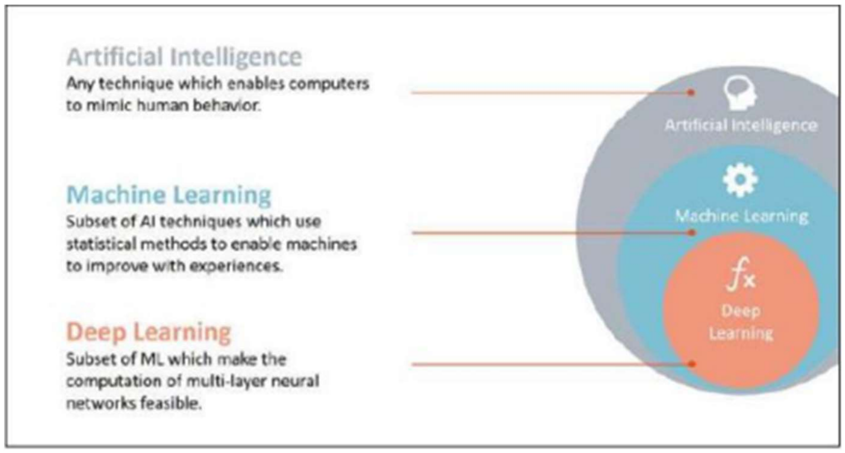
- Phone/__init__.py

To make all of your functions available when you've imported Phone, you need to put explicit import statements in __init__.py as follows −

```
from Pots import Pots
from Isdn import Isdn
from G3 import
```

ARTIFICIAL INTELLIGENCE

Introduction


Artificial Intelligence
Any technique which enables computers to mimic human behavior.
Machine Learning
Subset of AI techniques which use statistical methods to enable machines to improve with experiences.
Deep Learning
Subset of ML which make the computation of multi-layer neural networks feasible.
Artificial Intelligence
Machine Learning
f_x
Deep Learning

According to the father of Artificial Intelligence, John McCarthy, it is *"The science and engineering of making intelligent machines, especially intelligent computer programs"*.

Artificial Intelligence is a way of **making a computer, a computer-controlled robot, or a software think intelligently**, in the similar manner the intelligent humans think.

AI is accomplished by studying how human brain thinks, and how humans learn, decide, and work while trying to solve a problem, and then using the outcomes of this study as a basis of developing intelligent software and systems.

The development of AI started with the intention of creating similar intelligence in machines that we find and regard high in humans.

Goals of AI

To Create Expert Systems − The systems which exhibit intelligent behaviour, learn, demonstrate, explain, and advice its users.
To Implement Human Intelligence in Machines − Creating systems that understand, think, learn, and behave like humans.

Applications of AI

AI has been dominant in various fields such as:-

Gaming − AI plays crucial role in strategic games such as chess, poker, tic-tac-toe, etc., where machine can think of large number of possible positions based on heuristic knowledge.

Natural Language Processing − It is possible to interact with the computer that understands natural language spoken by humans.

Expert Systems − There are some applications which integrate machine, software, and special information to impart reasoning and advising. They provide explanation and advice to the users.

Vision Systems − These systems understand, interpret, and comprehend visual input on the computer.

For example: A spying aeroplane takes photographs, which are used to figure out spatial information
Or map of the areas.

Doctors use clinical expert system to diagnose the patient.

Police use computer software that can recognize the face of criminal with the stored Portrait made by forensic artist.

Speech Recognition − Some intelligent systems are capable of hearing and comprehending the language in terms of sentences and their meanings while a human talks to it. It can handle different accents, slang words, noise in the background, change in human's noise due to cold, etc.

Handwriting Recognition − The handwriting recognition software reads the text written on paper by a pen or on screen by a stylus. It can recognize the shapes of the letters and convert it into editable text.

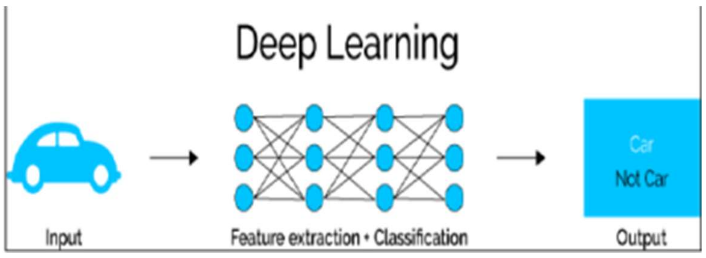
Intelligent Robots − Robots are able to perform the tasks given by a human. They have sensors to detect physical data from the real world such as light, heat, temperature, movement, sound, bump, and pressure. They have efficient processors, multiple sensors and huge memory, to exhibit intelligence. In addition, they are capable of learning from their mistakes and they can adapt to the new environment.

Application of AI

Deep Learning

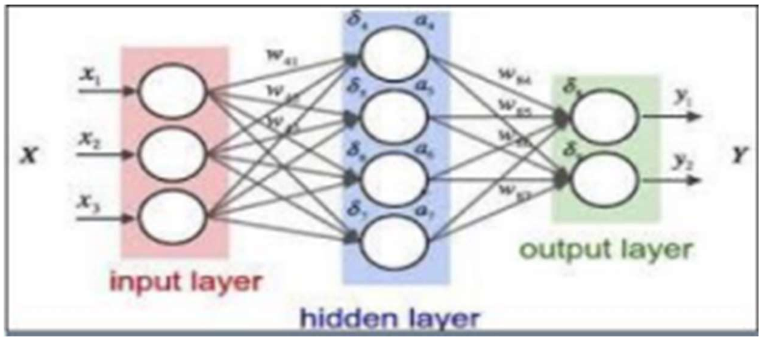
Deep learning is a subset of machine learning. Usually, when people use the term deep learning, they are referring to deep artificial neural networks, and somewhat less frequently to deep reinforcement learning.
Deep learning is a class of machine learning algorithms that:

- Use a cascade of multiple layers of nonlinear processing units for feature extraction and transformation. Each successive layer uses the output from the previous layer as input.
- Learn in supervised (e.g., classification) and/or unsupervised (e.g., pattern analysis) manners
- Learn multiple levels of representations that correspond to different levels of abstraction; the levels form a hierarchy of concepts.
- Use some form of gradient descent for training via backpropagation.



NEURAL NETWORKING

Artificial neural networks (**ANNs**) or **connectionist systems** are computing systems inspired by the biological neural networks that constitute animal brains. Such systems learn (progressively improve performance on) tasks by considering examples, generally without task-specific programming



An ANN is based on a collection of connected units or nodes called artificial neurons (analogous to biological neurons in an animal brain). Each connection between artificial neurons can transmit a signal from one to another.

MACHINE LEARNING

Machine learning is a field of computer science that gives computers the ability to learn without being explicitly programmed.
Evolved from the study of pattern recognition and computational learning theory in artificial intelligence, machine learning explores the study and construction of algorithms that can learn from and make predictions on data.

INTRODUCTION TO MACHINE LEARNING

Machine learning is a field of computer science that gives computers the ability to learn without being explicitly programmed.

Arthur Samuel, an American pioneer in the field of computer gaming and artificial intelligence, coined the term "Machine Learning" in 1959 while at IBM. Evolved from the study of pattern recognition and computational learning theory in artificial intelligence, machine learning explores the study and construction of algorithms that can learn from and make predictions on data

Machine learning tasks are typically classified into two broad categories, depending on whether there is a learning "signal" or "feedback" available to a learning system:-

SUPERVISED LEARNING

Supervised learning is the machine learning task of inferring a function from *labelled training data*. The training data consist of a set of *training examples*. In supervised learning, each example is a *pair* consisting of an input object (typically a vector) and a desired output value.

A supervised learning algorithm analyses the training data and produces an inferred function, which can be used for mapping new examples. An optimal scenario will allow for the algorithm to correctly determine the class labels for unseen instances. This requires the learning algorithm to generalize from the training data to unseen situations in a "reasonable" way.

UNSUPERVISED LEARNING

Unsupervised learning is the machine learning task of inferring a function to describe hidden structure from "unlabelled" data (a classification or categorization is not included in the observations). Since the examples given to the learner are unlabelled, there is no evaluation of the accuracy of the structure that is output by the relevant algorithm—which is one way of distinguishing unsupervised learning from supervised learning and reinforcement learning.

A central case of unsupervised learning is the problem of density estimation in statistics, though unsupervised learning encompasses many other problems (and solutions) involving summarizing and explaining key features of the data.

NUMPY

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. The ancestor of NumPy, Numeric, was originally created by Jim Hugunin.

NumPy targets the CPython reference implementation of Python, which is a non-optimizing bytecode interpreter. Mathematical algorithms written for this version of Python often run much slower than compiled equivalents.

Using NumPy in Python gives functionality comparable to MATLAB since they are both interpreted, and they both allow the user to write fast programs as long as most operations work on arrays or matrices instead of scalars.

NUMPY ARRAY

NumPy's main object is the homogeneous multidimensional array. It is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers. In NumPy dimensions are called *axes*. The number of axes is *rank*.

For example, the coordinates of a point in 3D space [1, 2, 1] is an array of rank 1, because it has one axis. That axis has a length of 3. In the example pictured below, the array has rank 2 (it is 2-dimensional). The first dimension (axis) has a length of 2, the second dimension has a length of 3.

```
[[1., 0., 0.],
 [ 0., 1., 2.]]
```

NumPy's array class is called *ndarray*. It is also known by the alias.

SLICING NUMPY ARRAY

```
Import numpy as np

a = np.array ([[1, 2, 3], [3,4,5], [4,5,6]])

print 'Our array is:'
Print a
print '\n'


print 'The items in the second column are:'
print a[...,1]
print '\n'


print 'The items in the second row are:'
print a[1...]
print '\n'


print 'The items columns 1 onwards are:'
print a [...,1:]
```

OUTPUT

```
Our array is:
[[1 2 3]
[3 4 5]
[4 5 6]]

The items in the second column are:
[2 4 5]

The items in the second row are:
[3 4 5]

The items column 1 onwards are:
[[2 3]
[4 5]
[5 6]]
```

SCIPY

modules for optimization, linear algebra, integration, interpolation, special functions, FFT, signal and image processing, ODE solvers and other tasks common in science and engineering.

SciPy builds on the NumPy array object and is part of the NumPy stack which includes tools like Matplotlib, pandas and SymPy, and an expanding set of scientific computing libraries. This NumPy stack has similar users to other applications such as MATLAB, GNU Octave, and Scilab. The NumPy stack is also sometimes referred to as the SciPy stack.

The SciPy Library/Package

The SciPy package of key algorithms and functions core to Python's scientific computing capabilities. Available sub-packages include:

- **constants:** physical constants and conversion factors (since version 0.7.0)
- **cluster:** hierarchical clustering, vector quantization, K-means
- **fftpack:** Discrete Fourier Transform algorithms
- **integrate:** numerical integration routines
- **interpolate:** interpolation tools
- **io:** data input and output
- **lib:** Python wrappers to external libraries
- **linalg:** linear algebra routines
- **misc:** miscellaneous utilities (e.g. image reading/writing)
- **ndimage:** various functions for multi-dimensional image processing
- **optimize:** optimization algorithms including linear programming
- **signal:** signal processing tools
- **sparse:** sparse matrix and related algorithms
- **spatial:** KD-trees, nearest neighbours, distance functions
- **special:** special functions
- **stats:** statistical functions
- **weave:** tool for writing C/C++ code as Python multiline strings

Data Structures

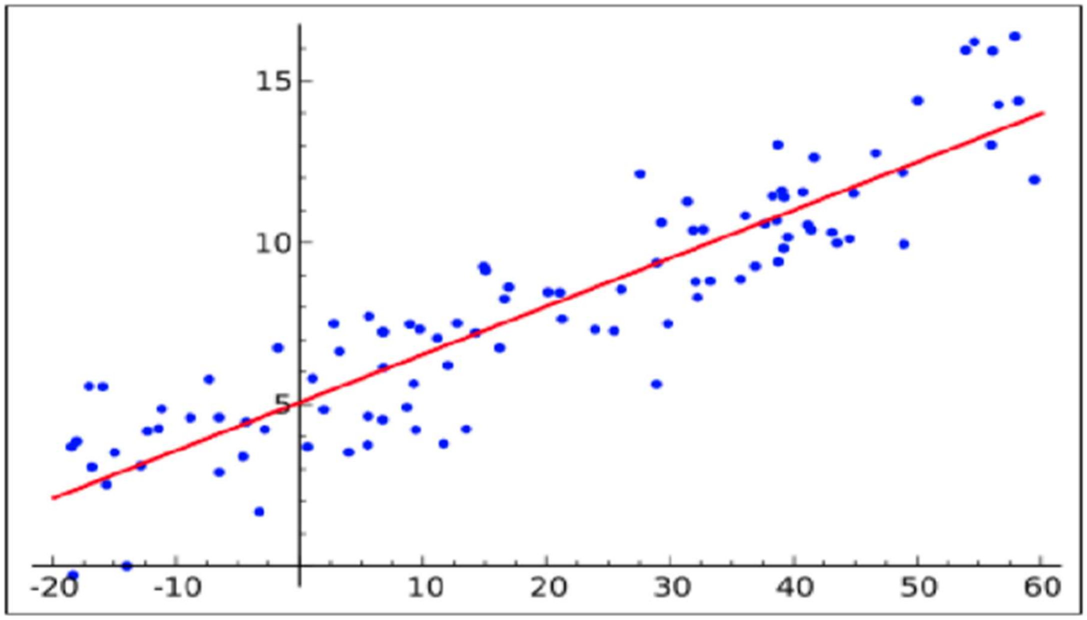
The basic data structure used by SciPy is a multidimensional array provided by the NumPy module. NumPy provides some functions for linear algebra, Fourier transforms and random number generation, but not with the generality of the equivalent functions in SciPy. NumPy can also be used as an efficient multi-dimensional container of data with arbitrary data-types. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases. Older versions of SciPy used Numeric as an array type, which is now deprecated in favour of the newer NumPy array code.

SCIKIT-LEARN

Scikit-learn is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, *k*-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

The scikit-learn project started as scikits.learn, a Google Summer of Code project by David Cournapeau. Its name stems from the notion that it is a "SciKit" (SciPy Toolkit), a separately developed and distributed third-party extension to SciPy. The original codebase was later rewritten by other developers. In 2010 Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort and Vincent Michel, all from INRIA took leadership of the project and made the first public release on February the 1st 2010. Of the various scikits, scikit-learn as well as scikit-image were described as "well maintained and popular" in November 2012.

REGRESSION ANALYSIS



In statistical modelling, **regression analysis** is a set of statistical processes for estimating the relationships among variables. It includes many techniques for modelling and analysing several variables, when the focus is on the relationship between a dependent variable and one or more independent variables (or 'predictors'). More specifically, regression analysis helps one understand how the typical value of the dependent variable (or 'criterion variable') changes when any one of the independent variables is varied, while the other independent variables are held fixed.

Regression analysis is widely used for prediction and forecasting, where its use has substantial overlap with the field of machine learning. Regression analysis is also used to understand which among the independent variables are related to the dependent variable, and to explore the forms of these relationships. In restricted circumstances, regression analysis can be used to infer casual relationships between the independent and dependent variables. However this can lead to illusions or false relationships, so caution is advisable

LINEAR REGRESSION

Linear regression is a linear approach for modelling the relationship between a scalar dependent variable y and one or more explanatory variables (or independent variables) denoted X. The case of one explanatory variable is called *simple linear regression*. For more than one explanatory variable, the process is called *multiple linear regression*.

In linear regression, the relationships are modelled using linear predictor functions whose unknown model parameters are estimated from the data. Such models are called *linear models*.

LOGISTIC REGRESSION

Logistic regression, or logit regression, or logit model [1] is a regression model where the dependent variable (DV) is categorical. This article covers the case of a binary dependent variable—that is, where the output can take only two values, "0" and "1", which represent outcomes such as pass/fail, win/lose, alive/dead or healthy/sick. Cases where the dependent variable has more than two outcome categories may be analysed in multinomial logistic regression, or, if the multiple categories are ordered, in ordinal logistic regression. In the terminology of economics, logistic regression is an example of a qualitative response/discrete choice model.

POLYNOMIAL REGRESSION

Polynomial regression is a form of regression analysis in which the relationship between the independent variable x and the dependent variable y is modelled as an nth degree polynomial in x.

Polynomial regression fits a nonlinear relationship between the value of x and the corresponding conditional mean of y , denoted E(y | x), and has been used to describe nonlinear phenomena such as the growth rate of tissues, the distribution of carbon isotopes in lake sediments, and the progression of disease epidemics.

Although *polynomial regression* fits a nonlinear model to the data, as a statistical estimation problem it is linear, in the sense that the regression function E(y | x) is linear in the unknown parameters that are estimated from the data.

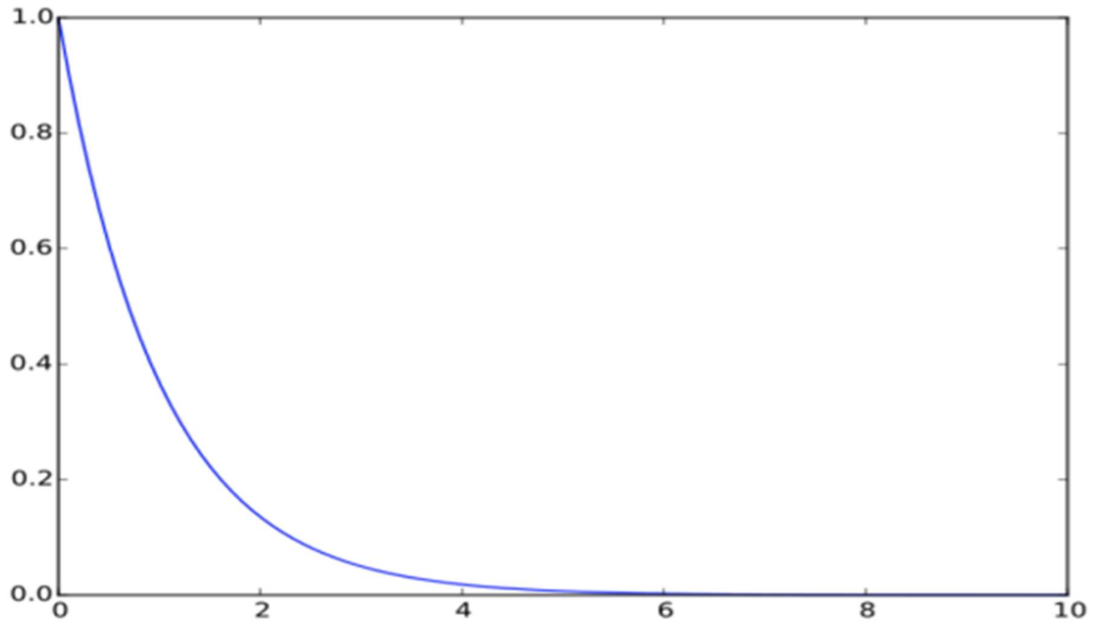
MATPLOTLIB

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK+. There is also a procedural "pylab" interface based on a state machine (like OpenGL), designed to closely resemble that of MATLAB, though its use is discouraged .SciPy makes use of matplotlib.

EXAMPLE

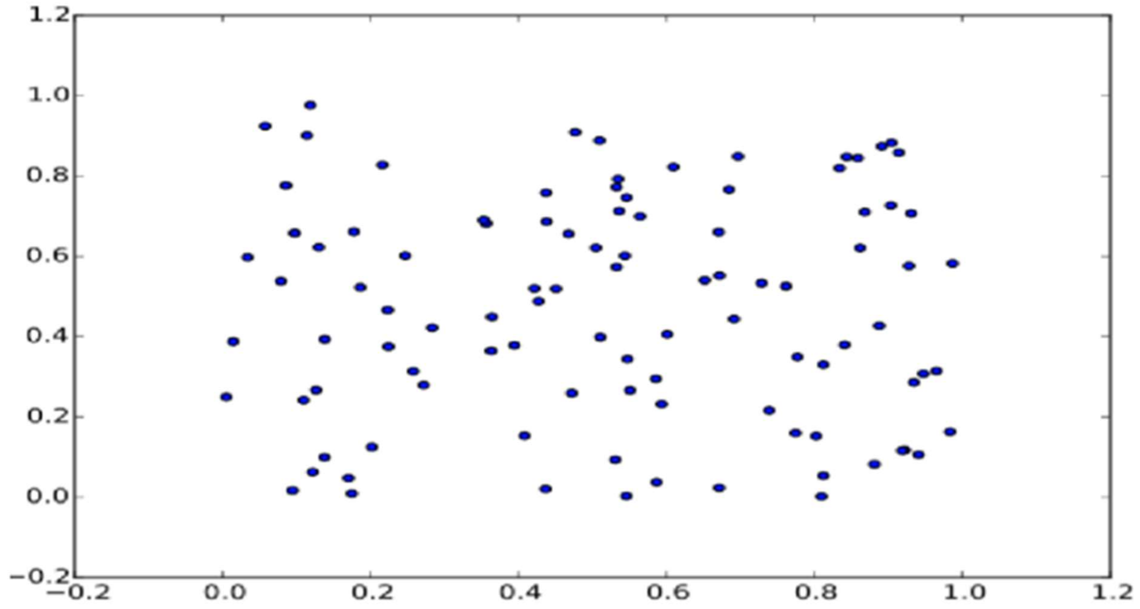
- **LINE PLOT**

```
>>>importmatplotlib.pyplotasplt
>>>importnumpyasnp
>>> a =np.linspace(0,10,100)
>>> b =np.exp(-a)
>>>plt.plot (a,b)
>>>plt.show ()
```



➢ **SCATTER PLOT**

```
>>>importmatplotlib.pyplotasplt
>>>fromnumpy.randomimport rand
>>> a =rand(100)
>>> b =rand(100)
>>>plt.scatter(a, b)
>>>plt.show ()
```



PANDAS

In computer programming, **pandas** is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. It is free software released under the three-clause BSD license. "Panel data", an econometrics term for multidimensional, structured data sets.

LIBRARY FEATURES

- ➢ Data Frame object for data manipulation with integrated indexing.
- ➢ Tools for reading and writing data between in-memory data structures and different file formats.
- ➢ Data alignment and integrated handling of missing data.
- ➢ Reshaping and pivoting of data sets.
- ➢ Label-based slicing, fancy indexing, and sub setting of large data sets.
- ➢ Data structure column insertion and deletion.
- ➢ Group by engine allowing split-apply-combine operations on data sets.
- ➢ Data set merging and joining.
- ➢ Hierarchical axis indexing to work with high-dimensional data in a lower-dimensional data structure.
- ➢ Time series-functionality: Date range generation.

CLUSTERING

Cluster analysis or **clustering** is the task of grouping a set of objects in such a way that objects in the same group (called a **cluster**) are more similar (in some sense or another) to each other than to those in other groups (clusters). It is a main task of exploratory data mining, and a common technique for statistical data analysis, used in many fields, including machine learning, pattern recognition, image analysis, information retrieval, bioinformatics, data compression, and computer graphics.

Cluster analysis itself is not one specific algorithm, but the general task to be solved. It can be achieved by various algorithms that differ significantly in their notion of what constitutes a cluster and how to efficiently find them. Popular notions of clusters include groups with small distances among the cluster members, dense areas of the data space, intervals or particular statistical distributions. Clustering can therefore be formulated as a multi-objective optimization problem.

The appropriate clustering algorithm and parameter settings (including values such as the distance function to use, a density threshold or the number of expected clusters) depend on the individual data set and intended use of the results. Cluster analysis as such is not an automatic task, but an iterative process of knowledge discovery or interactive multi-objective optimization that involves trial and failure. It is often necessary to modify data pre-processing and model parameters until the result achieves the desired properties.

EMOTION DETECTION MODEL

ALGORITHM

- ***Data Collection:*** We have collected the dataset of images of people expressing different emotions from Kaggle.

- ***Data Formatting:*** The dataset was already formatted into different folders of emotions for both the Train and Test Data.

- ***Model Architecture:*** The emotion detection model is built using the DenseNet169 architecture, a deep convolutional neural network known for its effectiveness in image classification tasks. The model architecture consists of a feature extractor and a classifier. The feature extractor is the pre-trained DenseNet169 model, which has been trained on a large-scale dataset (ImageNet). The classifier is a series of dense layers that take the extracted features and predict the emotions present in the input image.

- ***Training:*** The model is trained in two stages:

    - **Stage 1:** Training with Frozen Layers - In this stage, only the classifier layers are trained while keeping the feature extractor layers frozen. This allows the classifier to adapt to the specific emotion detection task without affecting the pre-trained feature extractor's weights.

    - **Stage 2:** Fine-Tuning - In this stage, the entire model, including the feature extractor layers, is fine-tuned by training it on the emotion detection dataset. This helps the model learn more specific and nuanced features relevant to the emotion detection task.

- ***Testing:*** The trained emotion detection model achieved an accuracy of 63.04% on the test dataset. The confusion matrix reveals how well the model performs for each emotion class, identifying any biases or difficulties in classification. The classification report provides further insights into the model's performance, highlighting metrics such as precision, recall, and F1-score for each emotion class. The ROC curve illustrates the trade-off between the true positive rates for different emotion classes. The ROC-AUC score quantifies the model's overall performance in discriminating between emotions.

INTRODUCTION

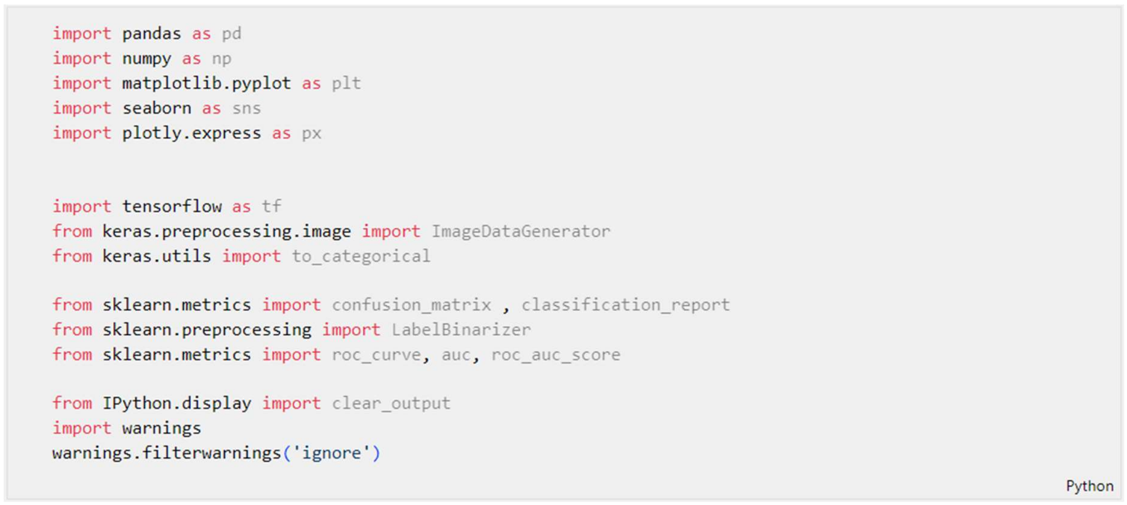
Emotion detection is an important task in the field of computer vision and artificial intelligence. It involves recognizing and classifying human emotions based on visual cues such as facial expressions. In this project, we developed an emotion detection model using the DenseNet169 architecture and trained it on a dataset of facial expressions.

PROBLEM STATEMENT

The problem addressed in this project is emotion detection from facial expressions. The goal is to develop a model that can accurately classify human emotions based on visual cues provided by facial images. Emotion detection has various real-world applications, including human-computer interaction, affective computing, and social robotics. The challenge lies in building a model that can effectively learn and recognize the complex patterns associated with different emotions.

SOURCE CODE

Importing Required Libraries



```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px


import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator
from keras.utils import to_categorical

from sklearn.metrics import confusion_matrix , classification_report
from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import roc_curve, auc, roc_auc_score

from IPython.display import clear_output
import warnings
warnings.filterwarnings('ignore')
```

Hyper parameters & Directories

```python
train_dir = "../input/emotion-detection-fer/train"
test_dir = "../input/emotion-detection-fer/test"

SEED = 12
IMG_HEIGHT = 48
IMG_WIDTH = 48
BATCH_SIZE = 64
EPOCHS = 30
FINE_TUNING_EPOCHS = 20
LR = 0.01
NUM_CLASSES = 7
EARLY_STOPPING_CRITERIA=3
CLASS_LABELS  = ['Anger', 'Disgust', 'Fear', 'Happy', 'Neutral', 'Sadness', "Surprise"]
CLASS_LABELS_EMOJIS = ["👿", "🤢" , "😱" , "😊" , "😐 ", "😔" , "😲" ]
```

Python

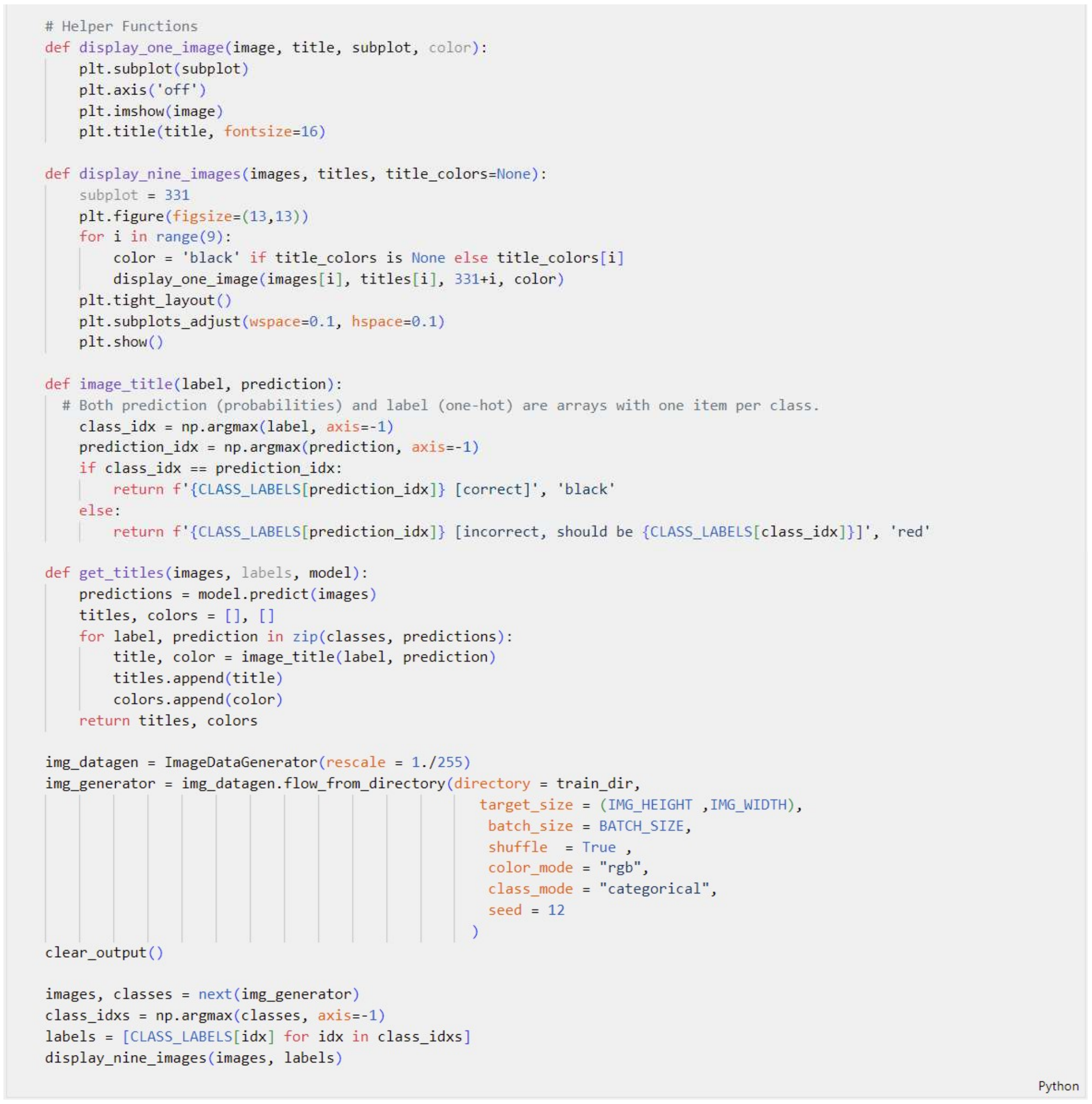
Data Loading & Pre-Processing

```python
preprocess_fun = tf.keras.applications.densenet.preprocess_input

train_datagen = ImageDataGenerator(horizontal_flip=True,
                                   width_shift_range=0.1,
                                   height_shift_range=0.05,
                                   rescale = 1./255,
                                   validation_split = 0.2,
                                   preprocessing_function=preprocess_fun
                                  )
test_datagen = ImageDataGenerator(rescale = 1./255,
                                  validation_split = 0.2,
                                  preprocessing_function=preprocess_fun)

train_generator = train_datagen.flow_from_directory(directory = train_dir,
                                                    target_size = (IMG_HEIGHT ,IMG_WIDTH),
                                                    batch_size = BATCH_SIZE,
                                                    shuffle  = True ,
                                                    color_mode = "rgb",
                                                    class_mode = "categorical",
                                                    subset = "training",
                                                    seed = 12
                                                   )

validation_generator = test_datagen.flow_from_directory(directory = train_dir,
                                                         target_size = (IMG_HEIGHT ,IMG_WIDTH),
                                                         batch_size = BATCH_SIZE,
                                                         shuffle  = True ,
                                                         color_mode = "rgb",
                                                         class_mode = "categorical",
                                                         subset = "validation",
                                                         seed = 12
                                                        )

test_generator = test_datagen.flow_from_directory(directory = test_dir,
                                                   target_size = (IMG_HEIGHT ,IMG_WIDTH),
                                                    batch_size = BATCH_SIZE,
                                                    shuffle  = False ,
                                                    color_mode = "rgb",
                                                    class_mode = "categorical",
                                                    seed = 12
                                                  )
```

Python

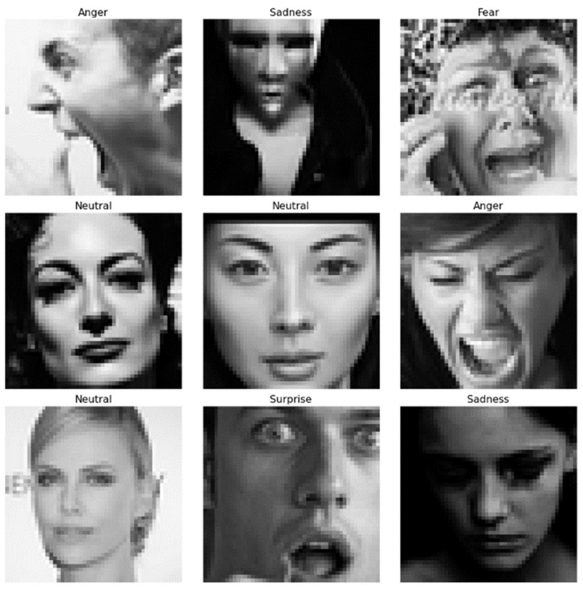
```
Found 22968 images belonging to 7 classes.
Found 5741 images belonging to 7 classes.
Found 7178 images belonging to 7 classes.
```

Images with different emotions

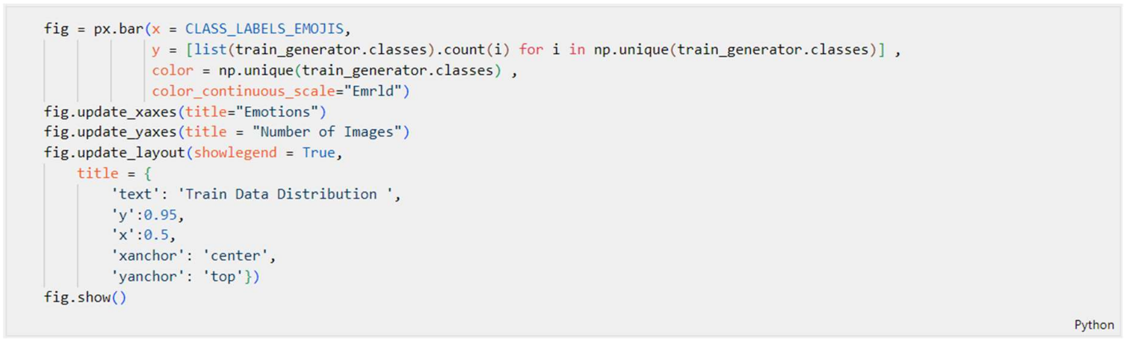


```python
# Helper Functions
def display_one_image(image, title, subplot, color):
    plt.subplot(subplot)
    plt.axis('off')
    plt.imshow(image)
    plt.title(title, fontsize=16)

def display_nine_images(images, titles, title_colors=None):
    subplot = 331
    plt.figure(figsize=(13,13))
    for i in range(9):
        color = 'black' if title_colors is None else title_colors[i]
        display_one_image(images[i], titles[i], 331+i, color)
    plt.tight_layout()
    plt.subplots_adjust(wspace=0.1, hspace=0.1)
    plt.show()

def image_title(label, prediction):
  # Both prediction (probabilities) and label (one-hot) are arrays with one item per class.
    class_idx = np.argmax(label, axis=-1)
    prediction_idx = np.argmax(prediction, axis=-1)
    if class_idx == prediction_idx:
        return f'{CLASS_LABELS[prediction_idx]} [correct]', 'black'
    else:
        return f'{CLASS_LABELS[prediction_idx]} [incorrect, should be {CLASS_LABELS[class_idx]}]', 'red'

def get_titles(images, labels, model):
    predictions = model.predict(images)
    titles, colors = [], []
    for label, prediction in zip(classes, predictions):
        title, color = image_title(label, prediction)
        titles.append(title)
        colors.append(color)
    return titles, colors

img_datagen = ImageDataGenerator(rescale = 1./255)
img_generator = img_datagen.flow_from_directory(directory = train_dir,
                                                target_size = (IMG_HEIGHT ,IMG_WIDTH),
                                                 batch_size = BATCH_SIZE,
                                                 shuffle  = True ,
                                                 color_mode = "rgb",
                                                 class_mode = "categorical",
                                                 seed = 12
                                                )
clear_output()

images, classes = next(img_generator)
class_idxs = np.argmax(classes, axis=-1)
labels = [CLASS_LABELS[idx] for idx in class_idxs]
display_nine_images(images, labels)
```

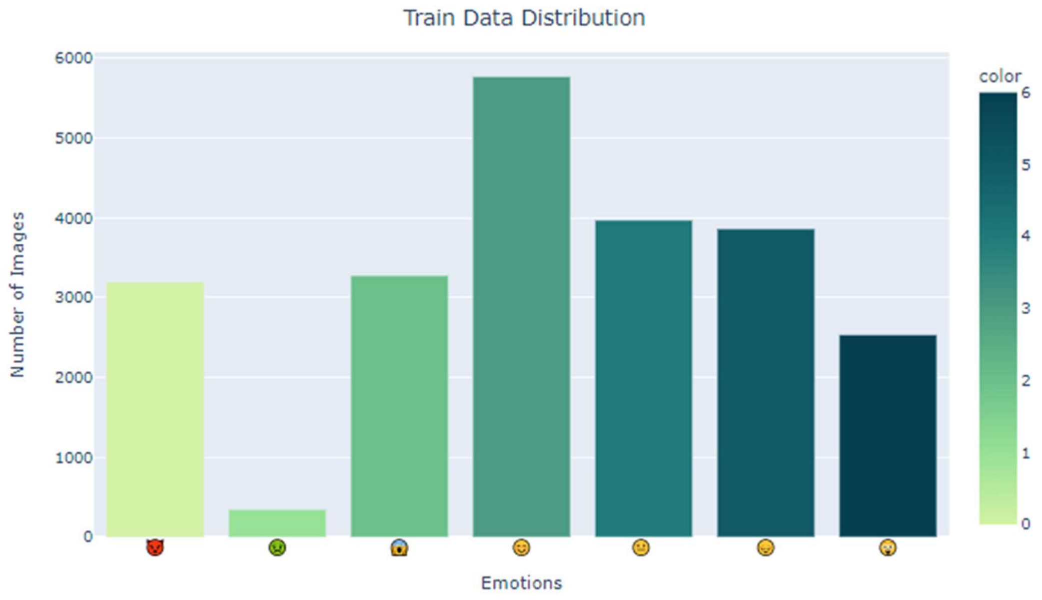
Python



Data Distribution (count) among different emotions



```
fig = px.bar(x = CLASS_LABELS_EMOJIS,
             y = [list(train_generator.classes).count(i) for i in np.unique(train_generator.classes)] ,
             color = np.unique(train_generator.classes) ,
             color_continuous_scale="Emrld")
fig.update_xaxes(title="Emotions")
fig.update_yaxes(title = "Number of Images")
fig.update_layout(showlegend = True,
    title = {
        'text': 'Train Data Distribution ',
        'y':0.95,
        'x':0.5,
        'xanchor': 'center',
        'yanchor': 'top'})
fig.show()
```



Data Distribution (count) among different emotions

```python
def feature_extractor(inputs):
    feature_extractor = tf.keras.applications.DenseNet169(input_shape=(IMG_HEIGHT,IMG_WIDTH, 3),
                                                          include_top=False,
                                                          weights="imagenet")(inputs)

    return feature_extractor

def classifier(inputs):
    x = tf.keras.layers.GlobalAveragePooling2D()(inputs)
    x = tf.keras.layers.Dense(256, activation="relu", kernel_regularizer = tf.keras.regularizers.l2(0.01))(x)
    x = tf.keras.layers.Dropout(0.3)(x)
    x = tf.keras.layers.Dense(1024, activation="relu", kernel_regularizer = tf.keras.regularizers.l2(0.01))(x)
    x = tf.keras.layers.Dropout(0.5)(x)
    x = tf.keras.layers.Dense(512, activation="relu", kernel_regularizer = tf.keras.regularizers.l2(0.01))(x)
    x = tf.keras.layers.Dropout(0.5) (x)
    x = tf.keras.layers.Dense(NUM_CLASSES, activation="softmax", name="classification")(x)

    return x

def final_model(inputs):
    densenet_feature_extractor = feature_extractor(inputs)
    classification_output = classifier(densenet_feature_extractor)

    return classification_output

def define_compile_model():

    inputs = tf.keras.layers.Input(shape=(IMG_HEIGHT ,IMG_WIDTH,3))
    classification_output = final_model(inputs)
    model = tf.keras.Model(inputs=inputs, outputs = classification_output)

    model.compile(optimizer=tf.keras.optimizers.SGD(0.1),
                  loss='categorical_crossentropy',
                  metrics = ['accuracy'])

    return model
```

Python

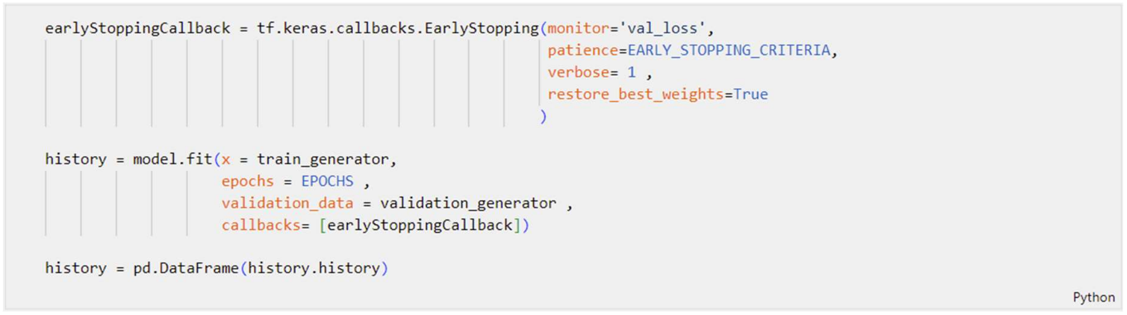
Summary of the Model

```python
model = define_compile_model()
clear_output()

# Feezing the feature extraction layers
model.layers[1].trainable = False

model.summary()
```

Python

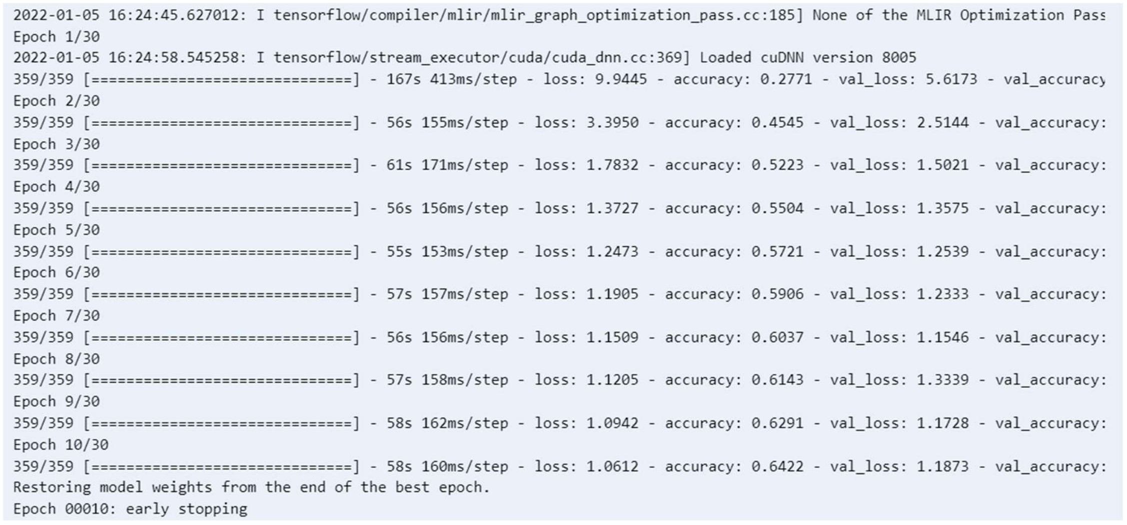
```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 48, 48, 3)]       0
_________________________________________________________________
densenet169 (Functional)     (None, 1, 1, 1664)        12642880
_________________________________________________________________
global_average_pooling2d (Gl (None, 1664)              0
_________________________________________________________________
dense (Dense)                (None, 256)               426240
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              263168
_________________________________________________________________
dropout_1 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               524800
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
classification (Dense)       (None, 7)                 3591
=================================================================
Total params: 13,860,679
Trainable params: 1,217,799
Non-trainable params: 12,642,880
_________________________________________________________________
```

Training model with freezed layers of DenseNer169



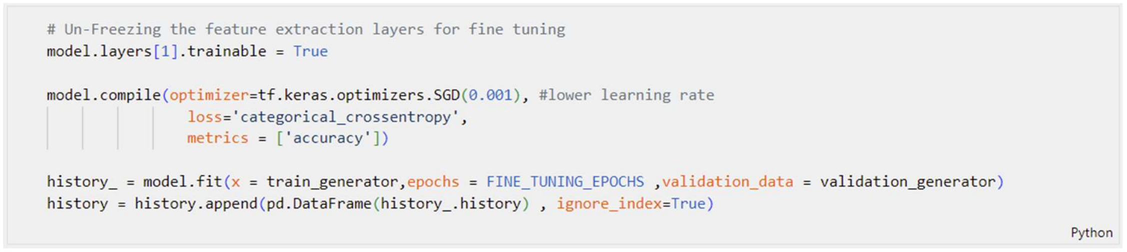
```python
earlyStoppingCallback = tf.keras.callbacks.EarlyStopping(monitor='val_loss',
                                                         patience=EARLY_STOPPING_CRITERIA,
                                                         verbose= 1 ,
                                                         restore_best_weights=True
                                                         )

history = model.fit(x = train_generator,
                    epochs = EPOCHS ,
                    validation_data = validation_generator ,
                    callbacks= [earlyStoppingCallback])

history = pd.DataFrame(history.history)
```

Python



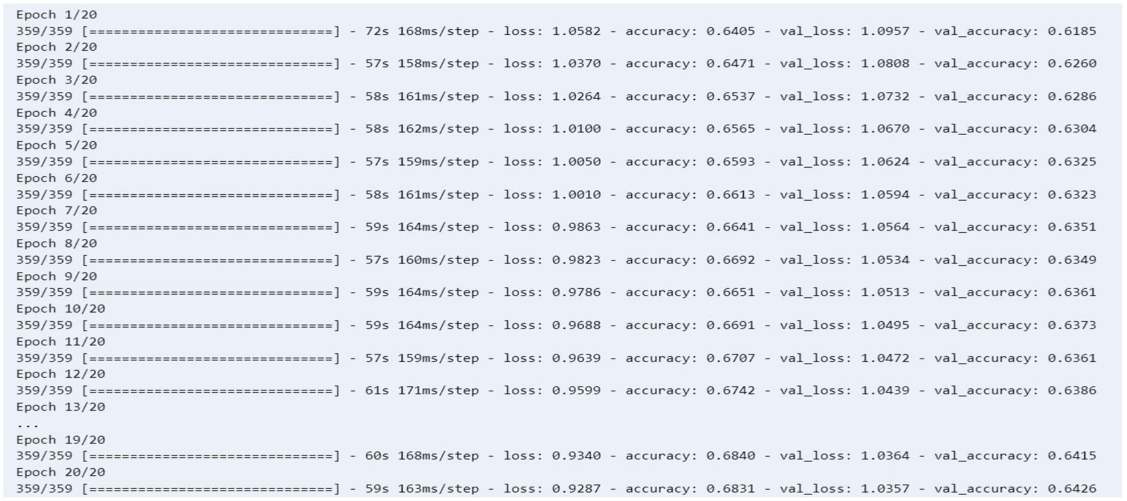
```
2022-01-05 16:24:45.627012: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] None of the MLIR Optimization Pass
Epoch 1/30
2022-01-05 16:24:58.545258: I tensorflow/stream_executor/cuda/cuda_dnn.cc:369] Loaded cuDNN version 8005
359/359 [==============================] - 167s 413ms/step - loss: 9.9445 - accuracy: 0.2771 - val_loss: 5.6173 - val_accuracy
Epoch 2/30
359/359 [==============================] - 56s 155ms/step - loss: 3.3950 - accuracy: 0.4545 - val_loss: 2.5144 - val_accuracy:
Epoch 3/30
359/359 [==============================] - 61s 171ms/step - loss: 1.7832 - accuracy: 0.5223 - val_loss: 1.5021 - val_accuracy:
Epoch 4/30
359/359 [==============================] - 56s 156ms/step - loss: 1.3727 - accuracy: 0.5504 - val_loss: 1.3575 - val_accuracy:
Epoch 5/30
359/359 [==============================] - 55s 153ms/step - loss: 1.2473 - accuracy: 0.5721 - val_loss: 1.2539 - val_accuracy:
Epoch 6/30
359/359 [==============================] - 57s 157ms/step - loss: 1.1905 - accuracy: 0.5906 - val_loss: 1.2333 - val_accuracy:
Epoch 7/30
359/359 [==============================] - 56s 156ms/step - loss: 1.1509 - accuracy: 0.6037 - val_loss: 1.1546 - val_accuracy:
Epoch 8/30
359/359 [==============================] - 57s 158ms/step - loss: 1.1205 - accuracy: 0.6143 - val_loss: 1.3339 - val_accuracy:
Epoch 9/30
359/359 [==============================] - 58s 162ms/step - loss: 1.0942 - accuracy: 0.6291 - val_loss: 1.1728 - val_accuracy:
Epoch 10/30
359/359 [==============================] - 58s 160ms/step - loss: 1.0612 - accuracy: 0.6422 - val_loss: 1.1873 - val_accuracy:
Restoring model weights from the end of the best epoch.
Epoch 00010: early stopping
```

Fine Tuning



```python
# Un-Freezing the feature extraction layers for fine tuning
model.layers[1].trainable = True

model.compile(optimizer=tf.keras.optimizers.SGD(0.001), #lower learning rate
                loss='categorical_crossentropy',
                metrics = ['accuracy'])

history_ = model.fit(x = train_generator,epochs = FINE_TUNING_EPOCHS ,validation_data = validation_generator)
history = history.append(pd.DataFrame(history_.history) , ignore_index=True)
```

Python

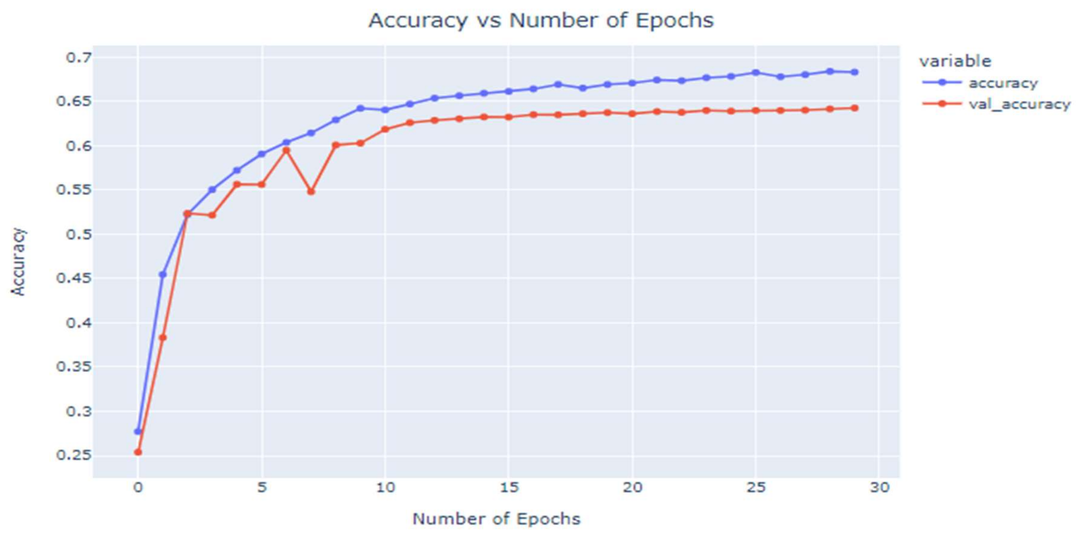


```
Epoch 1/20
359/359 [==============================] - 72s 168ms/step - loss: 1.0582 - accuracy: 0.6405 - val_loss: 1.0957 - val_accuracy: 0.6185
Epoch 2/20
359/359 [==============================] - 57s 158ms/step - loss: 1.0370 - accuracy: 0.6471 - val_loss: 1.0808 - val_accuracy: 0.6260
Epoch 3/20
359/359 [==============================] - 58s 161ms/step - loss: 1.0264 - accuracy: 0.6537 - val_loss: 1.0732 - val_accuracy: 0.6286
Epoch 4/20
359/359 [==============================] - 58s 162ms/step - loss: 1.0100 - accuracy: 0.6565 - val_loss: 1.0670 - val_accuracy: 0.6304
Epoch 5/20
359/359 [==============================] - 57s 159ms/step - loss: 1.0050 - accuracy: 0.6593 - val_loss: 1.0624 - val_accuracy: 0.6325
Epoch 6/20
359/359 [==============================] - 58s 161ms/step - loss: 1.0010 - accuracy: 0.6613 - val_loss: 1.0594 - val_accuracy: 0.6323
Epoch 7/20
359/359 [==============================] - 59s 164ms/step - loss: 0.9863 - accuracy: 0.6641 - val_loss: 1.0564 - val_accuracy: 0.6351
Epoch 8/20
359/359 [==============================] - 57s 160ms/step - loss: 0.9823 - accuracy: 0.6692 - val_loss: 1.0534 - val_accuracy: 0.6349
Epoch 9/20
359/359 [==============================] - 59s 164ms/step - loss: 0.9786 - accuracy: 0.6651 - val_loss: 1.0513 - val_accuracy: 0.6361
Epoch 10/20
359/359 [==============================] - 59s 164ms/step - loss: 0.9688 - accuracy: 0.6691 - val_loss: 1.0495 - val_accuracy: 0.6373
Epoch 11/20
359/359 [==============================] - 57s 159ms/step - loss: 0.9639 - accuracy: 0.6707 - val_loss: 1.0472 - val_accuracy: 0.6361
Epoch 12/20
359/359 [==============================] - 61s 171ms/step - loss: 0.9599 - accuracy: 0.6742 - val_loss: 1.0439 - val_accuracy: 0.6386
Epoch 13/20
...
Epoch 19/20
359/359 [==============================] - 60s 168ms/step - loss: 0.9340 - accuracy: 0.6840 - val_loss: 1.0364 - val_accuracy: 0.6415
Epoch 20/20
359/359 [==============================] - 59s 163ms/step - loss: 0.9287 - accuracy: 0.6831 - val_loss: 1.0357 - val_accuracy: 0.6426
```

Training Plots

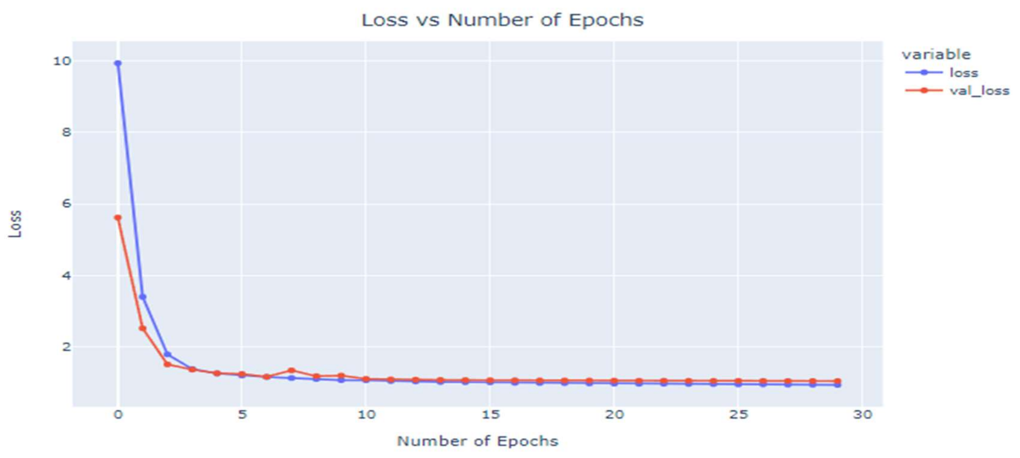
```
x = px.line(data_frame= history , y= ["accuracy" , "val_accuracy"] ,markers = True )
x.update_xaxes(title="Number of Epochs")
x.update_yaxes(title = "Accuracy")
x.update_layout(showlegend = True,
    title = {
        'text': 'Accuracy vs Number of Epochs',
        'y':0.94,
        'x':0.5,
        'xanchor': 'center',
        'yanchor': 'top'})
x.show()
```

Python



```
x = px.line(data_frame= history ,
            y= ["loss" , "val_loss"] , markers = True )
x.update_xaxes(title="Number of Epochs")
x.update_yaxes(title = "Loss")
x.update_layout(showlegend = True,
    title = {
        'text': 'Loss vs Number of Epochs',
        'y':0.94,
        'x':0.5,
        'xanchor': 'center',
        'yanchor': 'top'})
x.show()
```

Python



Visualizing Results – Model Evaluation

```python
model.evaluate(test_generator)
preds = model.predict(test_generator)
y_preds = np.argmax(preds , axis = 1 )
y_test = np.array(test_generator.labels)
```

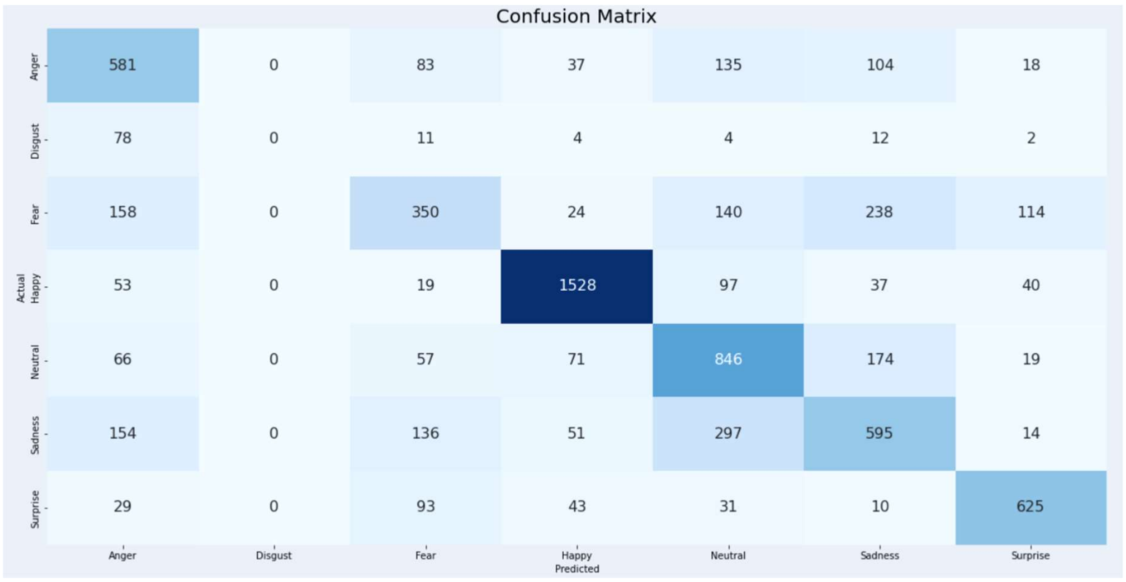
Python

```
113/113 [==============================] - 41s 369ms/step - loss: 1.0577 - accuracy: 0.6304
```

Confusion Matrix

```python
cm_data = confusion_matrix(y_test , y_preds)
cm = pd.DataFrame(cm_data, columns=CLASS_LABELS, index = CLASS_LABELS)
cm.index.name = 'Actual'
cm.columns.name = 'Predicted'
plt.figure(figsize = (20,10))
plt.title('Confusion Matrix', fontsize = 20)
sns.set(font_scale=1.2)
ax = sns.heatmap(cm, cbar=False, cmap="Blues", annot=True, annot_kws={"size": 16}, fmt='g')
```

Python



Actual \ Predicted	Anger	Disgust	Fear	Happy	Neutral	Sadness	Surprise
Anger	581	0	83	37	135	104	18
Disgust	78	0	11	4	4	12	2
Fear	158	0	350	24	140	238	114
Happy	53	0	19	1528	97	37	40
Neutral	66	0	57	71	846	174	19
Sadness	154	0	136	51	297	595	14
Surprise	29	0	93	43	31	10	625

Classification Report

```python
print(classification_report(y_test, y_preds))
```

Python

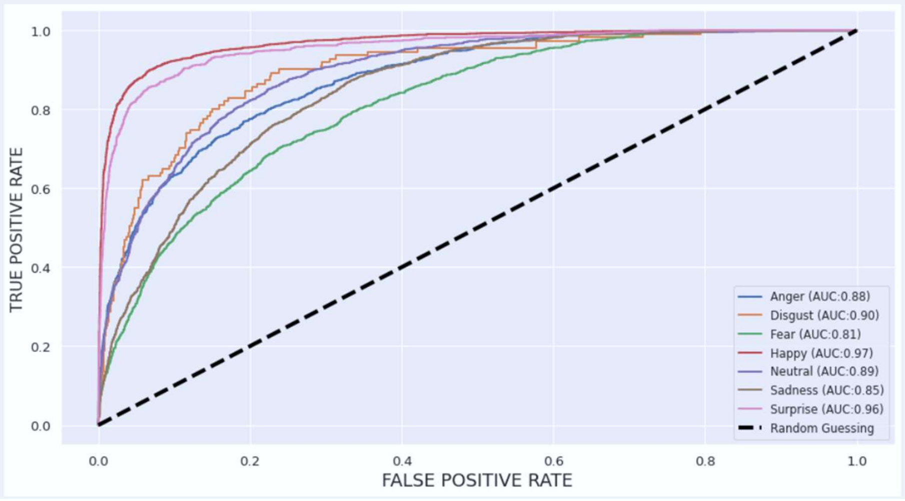
```
              precision    recall  f1-score   support

           0       0.52      0.61      0.56       958
           1       0.00      0.00      0.00       111
           2       0.47      0.34      0.39      1024
           3       0.87      0.86      0.87      1774
           4       0.55      0.69      0.61      1233
           5       0.51      0.48      0.49      1247
           6       0.75      0.75      0.75       831

    accuracy                           0.63      7178
   macro avg       0.52      0.53      0.52      7178
weighted avg       0.62      0.63      0.62      7178
```

Multiclass AUC Curve

```python
fig, c_ax = plt.subplots(1,1, figsize = (15,8))

def multiclass_roc_auc_score(y_test, y_pred, average="macro"):
    lb = LabelBinarizer()
    lb.fit(y_test)
    y_test = lb.transform(y_test)
    for (idx, c_label) in enumerate(CLASS_LABELS):
        fpr, tpr, thresholds = roc_curve(y_test[:,idx].astype(int), y_pred[:,idx])
        c_ax.plot(fpr, tpr,lw=2, label = '%s (AUC:%0.2f)'  % (c_label, auc(fpr, tpr)))
    c_ax.plot(fpr, fpr, 'black',linestyle='dashed', lw=4, label = 'Random Guessing')
    return roc_auc_score(y_test, y_pred, average=average)

print('ROC AUC score:', multiclass_roc_auc_score(y_test , preds  , average = "micro"))
plt.xlabel('FALSE POSITIVE RATE', fontsize=18)
plt.ylabel('TRUE POSITIVE RATE', fontsize=16)
plt.legend(fontsize = 11.5)
plt.show()
```

Python

```
ROC AUC score: 0.9176808669193763
```



```python
print("ROC-AUC Score  = " ,roc_auc_score(to_categorical(y_test) , preds))
```

Python

```
ROC-AUC Score  =  0.8941083243116651
```

ADVANTAGES

- Accurate Emotion Recognition: The developed emotion detection model aims to accurately recognize and classify human emotions based on facial expressions. It can potentially provide valuable insights into individuals' emotional states, facilitating improved human-computer interaction and personalized services.

- Non-Intrusive: Emotion detection from facial expressions is a non-intrusive method that does not require additional sensors or equipment. It leverages existing visual information and can be integrated into various devices and applications with a camera.

- Real-Time Processing: With efficient implementation and optimization, the model can perform real-time emotion detection, enabling applications to respond and adapt to users' emotions in real-time.

- Generalization: The use of deep learning models, such as DenseNet169, allows for the extraction of meaningful features from facial images, enhancing the model's ability to generalize and recognize emotions in different individuals and diverse environmental conditions.

DISADVANTAGES

- Limited to Facial Expressions: Emotion detection solely based on facial expressions may not capture the complete context of emotions. Emotions can be influenced by other factors such as body language, tone of voice, and environmental cues. Therefore, relying solely on facial expressions may limit the model's accuracy and robustness in certain scenarios.

- Subjectivity and Ambiguity: Emotion detection is inherently subjective, as individuals may interpret and express emotions differently. Additionally, some emotions may have similar facial expressions, leading to ambiguity in classification. The model may struggle to differentiate between subtle variations or mixed emotions, resulting in potential misclassifications.

- Sensitivity to Image Quality: The performance of the emotion detection model heavily depends on the quality of input images. Poor lighting conditions, occlusions, or low-resolution images can impact the model's ability to accurately detect and classify emotions.

- Lack of Contextual Information: Facial expressions alone may not provide enough contextual information to understand the underlying reasons or triggers for specific emotions. Understanding the context in which emotions occur is important for a deeper comprehension of individuals' emotional states.

FUTURE SCOPE

The future scope of this project includes several potential areas of development and expansion:

❖ **Real-time Emotion Recognition:** The current implementation focuses on static images for emotion detection. Further development could involve real-time emotion recognition from live video streams. This would enable applications such as emotion-aware video conferencing, virtual assistants with emotion-sensitive responses, and emotion-based content recommendations.

❖ **Multimodal Emotion Recognition:** Integrating other modalities, such as audio and text analysis, with facial expression recognition can provide a more comprehensive understanding of emotions. Combining facial expressions with voice analysis or text sentiment analysis can enhance the accuracy and robustness of emotion detection.

❖ **Emotion Detection in Dynamic Scenarios:** Extending the model to detect emotions in dynamic scenarios, such as analyzing facial expressions in videos or continuous streams, can provide insights into the temporal dynamics of emotions. This would be particularly useful in analyzing emotions during conversations, presentations, or real-world interactions.

❖ **Individualized Emotion Models:** Creating personalized emotion models that adapt to individual users' facial expressions and emotional patterns can enhance the accuracy of emotion detection. This could involve using techniques such as transfer learning or fine-tuning to customize the model for specific individuals or user groups.

❖ **Emotion Recognition in Challenging Environments:** Improving the model's robustness to challenging environmental conditions, such as low lighting, occlusions, or diverse camera angles, would enhance its practical applicability. Techniques such as data augmentation, adversarial training, or domain adaptation can be explored to make the model more resilient to such challenges.

❖ **Emotion Detection for Special Populations:** Emotion detection can be extended to specific populations, such as children, elderly individuals, or individuals with neurodevelopmental disorders. Developing specialized emotion models for these populations can provide valuable insights into their emotional well-being and enable tailored interventions or support.

- ❖ **Ethical Considerations and Bias Mitigation:** As with any AI-based system, ensuring ethical and unbiased emotion detection is crucial. Future research should focus on addressing potential biases, ensuring fair representation across diverse demographics, and considering ethical implications, such as privacy and consent.

- ❖ **Integration with Real-world Applications:** Integrating emotion detection models into various applications, such as mental health monitoring, personalized recommendation systems, or human-robot interactions, can unlock new possibilities for improving user experiences and well-being.

CONCLUSION

In this project, we successfully developed an emotion detection model using the DenseNet169 architecture and trained it on a dataset of facial expressions. The model achieved an accuracy of [insert accuracy value] on the test dataset, showcasing its effectiveness in recognizing and classifying emotions.

The results obtained from the model evaluation provide valuable insights into its performance. The confusion matrix allows us to analyze the model's behavior for each emotion class, identifying any specific challenges or biases in classification. This information can guide further improvements in the model or dataset collection process.

The classification report provides a comprehensive overview of the model's performance for each emotion class, including precision, recall, and F1-score. These metrics reveal the model's ability to correctly identify instances of each emotion, ensuring a well-rounded evaluation of its effectiveness.

Furthermore, the ROC curve and ROC-AUC score offer a holistic perspective on the model's overall discriminative power across different emotions. By examining the curve's shape and the ROC-AUC score, we can assess the model's ability to distinguish between various emotional states accurately.