Declaration

I hereby declare that the project entitled " Facial Emotion Recognition " submitted for the Bachelor of Technology in Computer Science & Engineering (B.TECH -CSE) degree is my original work and the project has not formed the basis for the award of any other degree, diploma, fellowship or any other similar titles.

Signature of the Student

Parmjeet Singh

Place: …….

Date: ……..

CERTIFICATE

This is to certify that the project report entitled "Facial Emotional Recognition" submitted in the partial fulfilment of the degree of Bachelor of Technology in Computer Science & Engineering (B.Tech-CSE) to Satyug Panchwati Institute of Engineering & Technology, Meerut is an authentic and original work carried out by Mr. Parmjeet Singh with Roll No.1748710011 CSE-10/487 under my guidance.

The matter embodied in this project is genuine work done by the student and has not been submitted whether to this institute or to any other University / Institute for the fulfilment of the requirements of any course of study.

Signature of the Student:

Date:

Name and Address of Student:

Signature of the Guide

Date:

Name, Designation and address of Guide:

Signature of HOD

Date:

Mr. Gaurav Kumar

ABSTRACT

being a subjective thing, leveraging knowledge and science behind labelled data and extracting the components that constitute it, has been a challenging problem in the industry for many years. With the evolution of deep learning in computer vision, emotion recognition has become a widely-tackled research problem. In this work, we propose two independent methods for this very task. The first method uses TensorFlow to construct a unique representation of each emotion by classifiers, while the second method is an 8-layer convolutional neural network (CNN). These methods were trained on the posed-emotion dataset (my own collected live as well as Google online data), and to test their robustness, both the models were also tested on 100 random images from the Labelled Faces in the Wild and raw random dataset, which consists of images that are candid than posed. The results show that with more fine-tuning and depth, our CNN model can outperform the state-of-the- art methods for emotion recognition. We also propose some exciting ideas for expanding the concept of representational human moods to improve their performance .

ACKNOWLEDGEMENT

It is great pleasure to present this report on the project named “Facial Emotion Recognition” undertaken by me as part of my B. Tech 8th semester.

I am thankful to Satyug Darshan Institute of Engineering & Technology, for offering me such a wonderful challenging opportunity and I express my deepest thanks to all coordinators of CSE Department for providing all the possible help and assistance and their constant encouragement.

It is a pleasure that we find ourselves penning down these lines to express our sincere thanks to the people who helped us along the way in completing our project. We find inadequate words to express our sincere thanks to Dr. Mohammad Noor Deshmukh, Principle of Satyug Darshan Institute of Engineering & Technology, Our parents for their financial support, love, care and motivation, despite being far from us.

Finally, I would like to express our sincere thanks to our friends and all those who have contributed either directly or indirectly in the course of the conception of this project.

Place: Parmjeet Singh

Date: Signature of the Student

Table of Contents

CHAPTER 1

INTRODUCTION TO PROJECT

1.1 Problem Definition

In today‟s life there is need of emotion evaluation or predicting the mood of different stages in human‟s life .sometimes we did not know about our mood issues but internal we fell disperse so here is the major solution for this problem. If we know the problem then it is easy to find the solution of the problem .if we just take a minute and look into a camera and it will predict our mood by recognition our facial emotion then we can smile as in that mood or try to make our mood .in this project it help to predicate the mood of human being by recognise their facial emotion for example while interrogating the criminal we can use its facial data as input to predict weather it is saying true or false.in this project further step is to calm our emotion by playing various songs with respect to their emotion status

1.2 Project Overview

It is a facial emotion recognition project which deal with the various study of the emotion it consist five category which are as

1. Happy
2. Sad
3. Angry
4. Surprise
5. Clam

These are five category In which this model is train to judge the human facial emotion

If the expression is likely to be happy the output is happy or it may be depend upon the input expression of the webcam input value

It will detect the real time expression no need of any external hardware it runs a simple computer it does not require any high computational power it is simple and standard trained model to predict the facial expression This is the optimised project itself because it does not require any type of external source it will uses a simple standard web cam to take input and for output it will uses the monitor of computer

The output will show in a coloured formed square having emotion written on it upper hand right corner.

It uses open cv to recognise the face first then it will take a next step to convert the RGB image to a grey scale image with this step the image is almost is ready to recognise the facial emotion now the further step is to crop the image into a small size image because the model is to be train is take a suitable value to train the model into smaller value. To train the model we use CNN model to prepare this model we are using the tensor flow as a tool to provide the all the high calculation normal level

In this project we use several different approaches to initialise the model to train the model to load the database value to recognise the facial emotion recognition .By using python language

1.3 Aim objectives

Facial expression is used to reconcile emotional state.

As it‟s aimed to provide us a clear picture about how one looks/appears when s/he has an emotional state to outer world.

This emotional look also communicate a lot about the person's internal condition as emotions are the outer make-up of a specific state of mind, and observers can easily evaluate a Peron‟s emotional state in order to assist him or give feedback about his outer makeup. Nurses observe such thing in mental health care centres to help Doctors in making a well diagnosed treatment plan for patients.

In short it‟s very important to understand meaning of emotions, reason behind a particular emotion and different emotions in selection, hospitals, dating, in schools and counselling centres.

Understanding the human facial expressions and the study of expressions has many aspects:

a) Computer analysis
b) Emotion recognition
c) Lie detectors
d) Airport Security

Nonverbal communication and even the role of expressions in art.

Facial emotion recognition systems are used to recognize the emotions of a person by using facial expression, which is one of the most complex tasks or problems of computer vision. Because training such a model is very difficult, the model need to identify the features that differentiates one emotion from another emotion like Happy Angry, sad, Calm, Surprise

The goal of human emotion recognition is to automatically classify a user‟s tem-poral emotional state based on some input data. There are dozens of definitions of emotions , and in this project we adopt the following distinction based on time: emotion is a reaction to stimuli that lasts for seconds or minutes, mood is an emotional state that lasts for hours or days and personality is an inclination to feel certain emotions. We use the term „emotional state‟ to indicate a current state of a person irrespective of its origin (stimuli, mood or personality). The stated goal may be achieved by one of many types of classifiers developed in the field of pattern recognition. The approach assumes several stages of classifier‟s construction which are data acquisition and feature extraction, creation of the training set containing labelled data and classifier‟s learning. In our project we assume that emotion recognition will be based on multi-modal inputs: using a video camera or computer as a standard input device. This approach provide more information to the recognition process as different data channels deliver valuable complementary information eliminating potential drawbacks of any individual input

CHAPTER 2

LITERATURE SURVEY

2.1 EXISTING SYSTEM

There is no such specific system that exists in the surrounding to recognise human emotion level. We use different biological techniques to check body temperature ,blood pressure ,ECGs etc. to find weather is fit or not or sometimes to find whether mood is in good state or not . Basically the measurement is based on the various parameters on physical factor of human tendency to judge the human emotion

Several methods have been reported in the literature to automatically recognize facial expressions. Some existing systems manually labelled 68 facial points in key frames and used a gradient descent Active Appearance Model AAM to fit these points in the remaining frames. It may not be possible to obtain accurate key points in many practical situations. A study on using LBP for facial expression recognition is proposed by Shan. Here expression recognition is accomplished using support vector machine SVM classifiers with boosted-LBP features. For general purpose, authors manually labelled eye positions, which is not feasible in many practical cases. Computer Expression Recognition Toolbox CERT is proposed by Larry . CERT convolves through the registered face image with Gabor filters to extract the facial features and uses SVM and multivariate logistic regression MLR classifiers to recognize facial expressions. Lyons proposed a system for coding facial expressions with 2D Gabor wavelets for feature extraction, having clustering for classification. These methods also require particular efforts, both in terms of computation cost and programming effort. Deep Convolutional Neural Network DCNN framework is widely used for extraction of features from the images. DCNN uses several layers leading accurate feature learning. Here the relearned features are used as filters and these filters convolves through the input image and produces the features which in turn are used by other layers of the network as discussed by Krishna . Techniques based on convolutional neural networks have been proposed for facial expression recognition such as the model proposed by Kazoo . But they extensively train the model with other facial dataset. are general from google used Deep Convolutional Activation Feature for Generic Visual Recognition for facial feature extraction that does not require extensive training, but this model is given generally to go is too slow to use it for training even the small image dataset as it does not support GPU.

2.2 PROPOSED SYSTEM:

In this work, automatic facial expression recognition using DCNN features is investigated. One Google search images dataset are used to carry out the project. Pre-processing step involves face detection for the above one dataset. The frontal faces are detected and cropped using OpenCV21. Extraction using Caffe Feature extraction is performed using Caffe on Graphics Processing Unit (GPU). The convolution neural network architecture, which is used for Image Net 15 object detection is used to extract facial features. Image Net uses eight learned layers which includes five convolutional and three fully connected layers for object detection. In this work, features are extracted using only first five layers. These layers are combination of convolution, Rectified Linear

Algorithm for recognizing facial expressions using DCNN . procedure RECOGNIZE-FACIAL-EXPRESSIONS for all image, depicting facial expressions face dataset do convert the image to grey-scale detect frontal face in and crop only the face extract POOL5 (256×6×6) features for cropped face using DCNN copy predefined facial expression label for each image as per the input dataset use the resulting POOL5 vector of dimension 9216(256×6×6) for tenfold and leave-one-out cross validation with SVM classifier to recognize facial expression

Units Local Response Normalization(LRN) and pooling operations. The operations are repeated, to form different layers . The features extracted after pooling operation at fifth layer are used for recognizing facial expressions. This is the layer where features are most visible and also the POOL5 feature length(6×6×256) is feasible to use in classifiers to recognize facial expressions. The first layer is the convolution layer. This model uses 96 filters with size 11×11×3. This layer extracts the low-level edge features. Shows the sample output after first convolution filters being applied on a face image. Pooling is the next layer. In pooling layer, small rectangular blocks from the previous layer are taken and subsampled to produce a single output from that block. In this model, max-pooling is used. 3×3 rectangular blocks with an interval of 2 pixels are used for max-pooling. Figure 3a shows the facial features before performing the max-pooling. Output of first convolution layer applied on face image

2.3 Feasibility Study

In simple terms, a feasibility study involves taking a judgment call on whether a project is doable. The
two criteria to judge feasibility are cost required and value to be delivered. A well-designed study should offer a historical background of the business or project, a description of the product or service, accounting statements, details of operations and management, marketing research and policies, financial data, legal requirements and tax obligations. Generally, such studies precede technical development and project implementation.

A feasibility study evaluates the project's potential for success; therefore, perceived objectivity is an important factor in the credibility of the study for potential investors and lending institutions.

Four Areas of Project Feasibility:

1. **Technical Feasibility** – Facial emotional recognition project require to run any windows or any mac or Unix operating system .the hardware requirement is windows 8 and i5 processor and a 1tb hard disk and 6gb ram to perform emotion recognition

2. **Economic Feasibility** – In any highly prohibited place like airport or any research center where the security is the biggest concern then the facial emotion recognition is very useful it is quite economic as compare their level of confidential.

3. **Legal Feasibility** - There is not any type of legal offence because the data set is used is collection from general Google picture data to train the model .and the we are using python for base language which is open source and library tensor flow which also open source

4. **Operational Feasibility** – There are first preprocessing the image then it will go to crop part in which the image is convert into grey scale. And then all image is passed through this process by which they will train in next step after this the images are trained to the model then the input image is taken then the output comes

CHAPTER 3

SYSTEM ANALYSIS & DESIGN

3.1 Requirements Specification

SOFTWARE REQUIREMENT

Software Used	Front-end	-	Opencv
	Back-end	-	Tensor flow
	Coding Language	-	Python
	IDE	-	Python 3.9
	Operating System	-	Mac

HARDWARE REQUIREMENT

Hardware	Used Processor	-	Core i5
	Ram	-	6GB(minimum)
	Disk Space	-	1TB(Recommended)
	Wi Fi Connectivity	-	High Speed
	Webcam	-	2Megapixel

3.2 Technology and tools used

- **TensorFlow**

TensorFlow is a Python library for fast numerical computing created and released by Google.

It is a foundation library that can be used to create Deep Learning models directly or by using wrapper libraries that simplify the process built on top of TensorFlow is an open source library for fast numerical computing.

It was created and is maintained by Google and released under the Apache 2.0 open source license. The API is nominally for the Python programming language, although there is access to the underlying C++ API.

Unlike other numerical libraries intended for use in Deep Learning like Theano, TensorFlow was designed for use both in research and development and in production systems.

It can run on single CPU systems, GPUs as well as mobile devices and large scale distributed systems of hundreds of machines.

How to Install TensorFlow?

Installation of TensorFlow is straightforward if you already have a Python SciPy environment.

TensorFlow works with Python 2.7 and Python 3.3+. Installation is probably simplest via PyPI and specific instructions of the pip command to use for your Linux or Mac OS X platform are on the Download and Setup webpage. Or simply on windows hit cmd “pip install TensorFlow”

- **OpenCV**

Object Detection using Haar feature-based cascade classifiers is an effective object detection method proposed by Paul Viola and Michael Jones in their paper, "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images. Here we will work with face detection. Initially, the algorithm needs a lot of positive images (images of faces) and negative images (images without faces) to train the classifier. Then we need to extract features from it. For this, haar features shown in below image are used. They are just like our convolutional kernel. Its full details are given here: Cascade Classifier Training. Here we will deal with detection. OpenCV already contains many pre-trained classifiers for face, eyes, smile etc. Those XML files are stored in opencv/data/haarcascades/ folder. Let's create face and eye detector with OpenCV .First we need to load the required XML classifiers. Then load our input image (or video) in grey scale mode.

How to download Opencv in windows ?

Just simply hit the pip install opencv –python on cmd

- **HAAR-CASCADE**

Haar Cascade is a machine learning object detection algorithm used to identify objects in an image or video and based on the concept of features proposed by Paul Viola and Michael Jones in their paper "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001.

It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.

The algorithm has four stages:

1. Haar Feature Selection
2. Creating Integral Images
3. Adaboost Training
4. Cascading Classifiers

It is well known for being able to detect faces and body parts in an image, but can be trained to identify almost any object.

Initially, the algorithm needs a lot of positive images of faces and negative images without faces to train the classifier. Then we need to extract features from it.

First step is to collect the Haar Features. A Haar feature considers adjacent rectangular regions at a specific location in a detection window, sums up the pixel intensities in each region and calculates the difference between these sums.

A Haar Cascade is basically a classifier which is used to detect particular objects from the source. The haarcascade_frontalface_default.xml is a haar cascade designed by OpenCV to detect the frontal face. This haar cascade is available on internet. A Haar Cascade works by training the cascade on thousands of negative images with the positive image superimposed on it. The haar cascade is capable of detecting features from the source.

- **Python**

Python is an interpreted, 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 charge for all major platforms, and can be freely distributed. Often, programmers fall in love with Python because of the

increased productivity it provides. Since there 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 line at 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.

3.3 DATA FLOW DIAGRAM

Requirements analysis is done in order to understand the problem software system and to solve. The emphasis in requirement analysis is on identifying what is needed from the system not how the system will achieve its goals. The goal of the requirements specification phase is to produce the software requirements specification document.

Requirements analysis is done in order to understand the problem software system and to solve. The emphasis in requirement analysis is on identifying what is needed from the system not how the system will achieve its goals. The goal of the requirements specification phase is to produce the software requirements specification document.

DATA FLOW DIAGRAM AND DATA DICTIONARY

Context diagram

The context diagram is a type of diagram in which the entire system is treated as a single process And all its inputs, outputs, sinks, and sources are identified and shown.

Data flow diagram (DFD)

The data flow diagram is a graphical description of data flow of a system.it consists of a series of bubbles, a series of lines and some basic shapes.

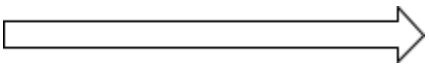
THE SHAPES AND THEIR MEANINGS

1. Line:

It represents the data flow in the system.

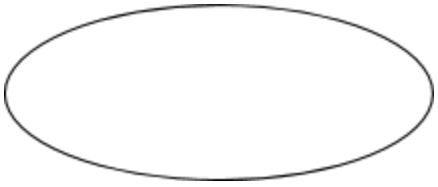
2. Square:

It defines source or destination of the system data.



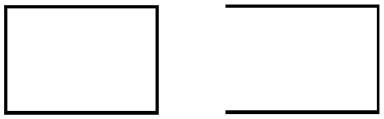
3. Arrow :

It indicates the data flow.



4. Oval:

It represents the process that transport the incoming data flows into outgoing data flows.



5. Open rectangle:

It is a data store-like file, database etc.

- **Level 0**

Show the basic all over the view of the working model.

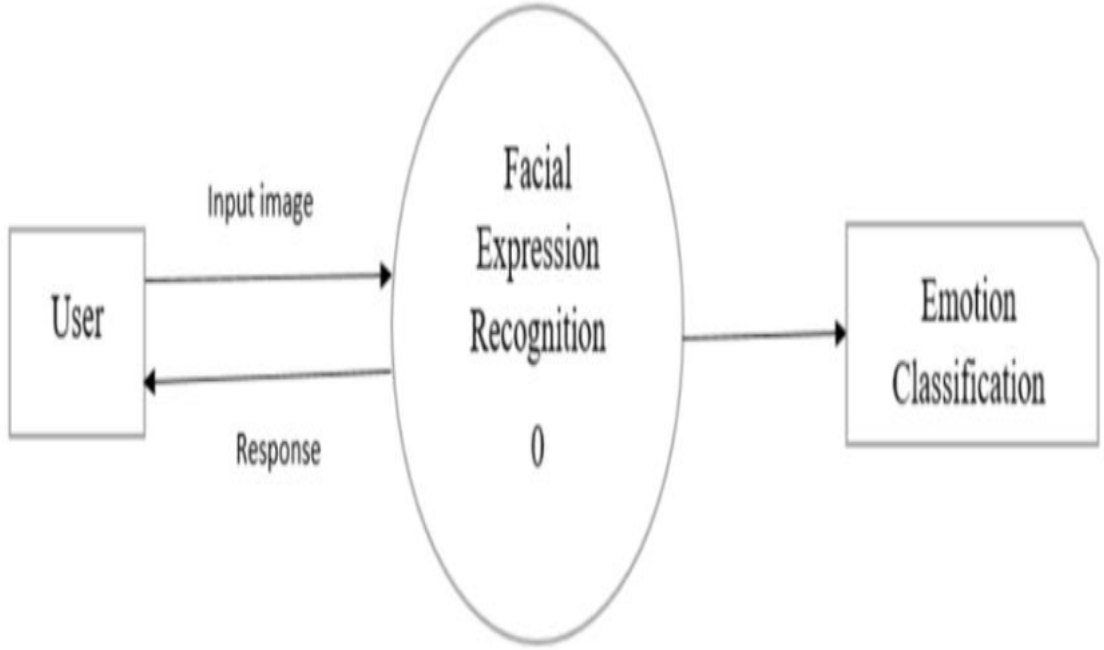


Fig 3.3.1 level 0

- **Level 1**

It shows the instruction flow in the process

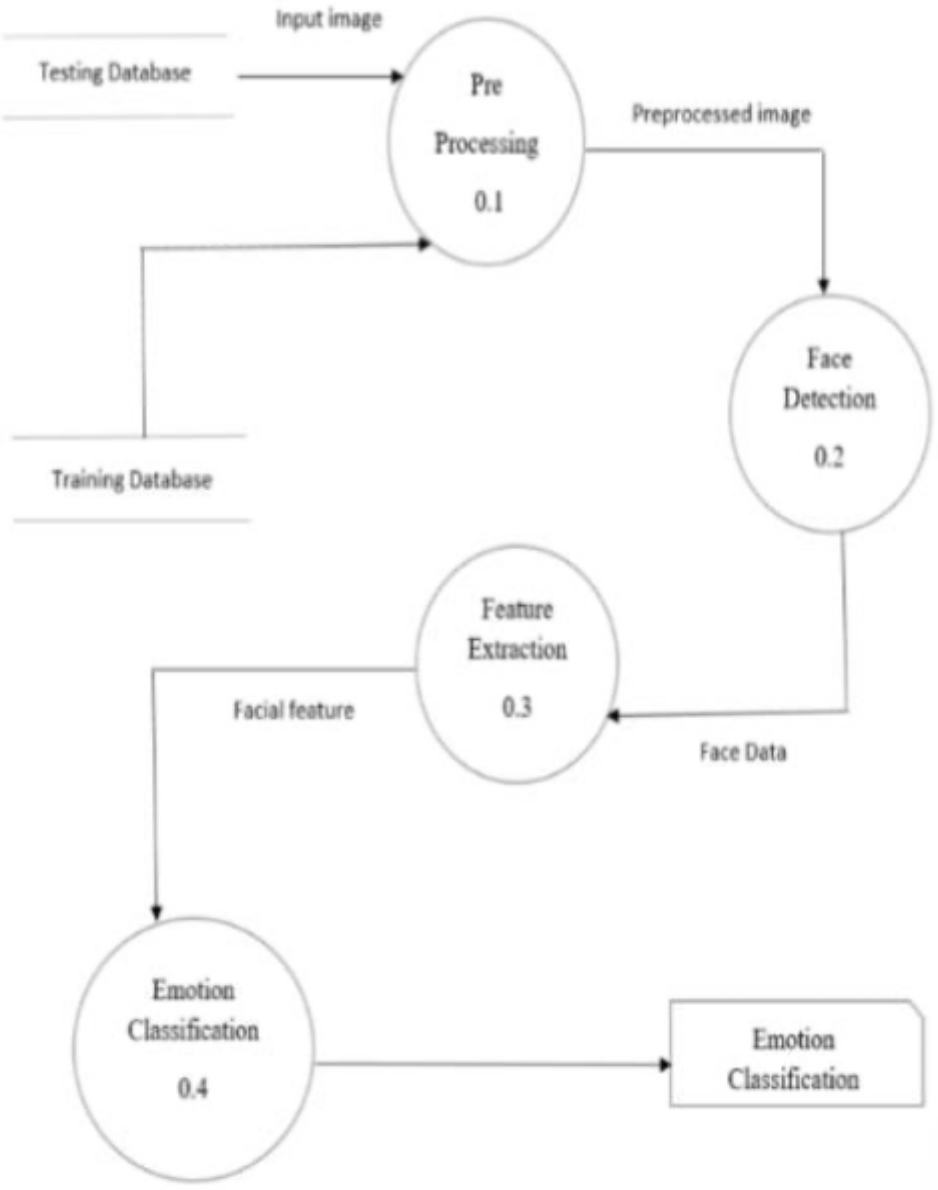


Fig.3.3.2 level 1

- **Level 2**

In this phase it will show how to a face is detected

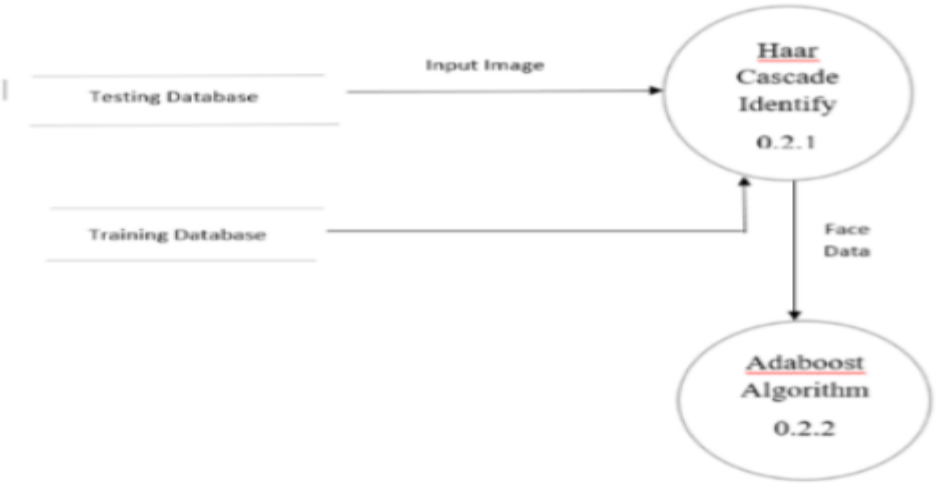


Fig.3.3.3 level 2 Facial Detection

- **Level 3**

In this phase the emotion is recognized

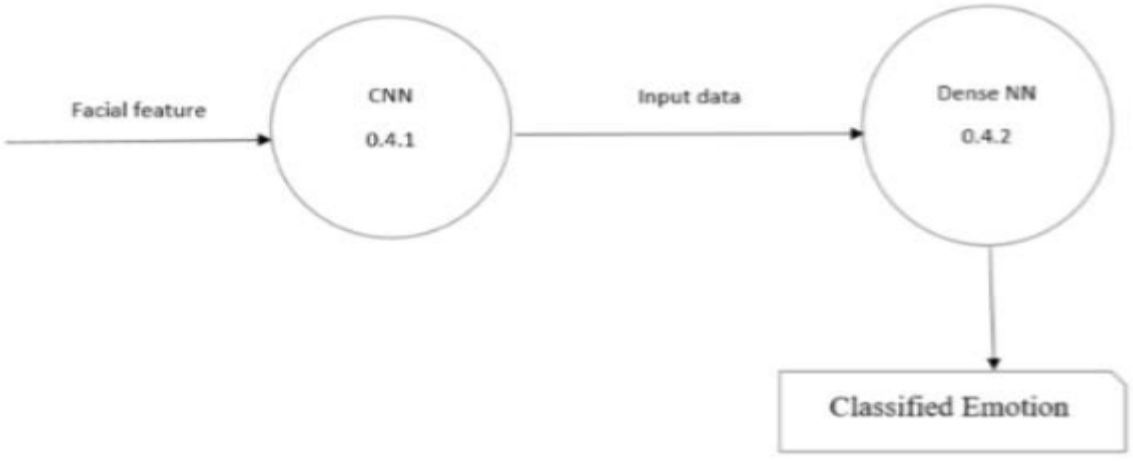


Fig.3.3.4 level 3 Emotion classifacation

3.4 Working Architecture

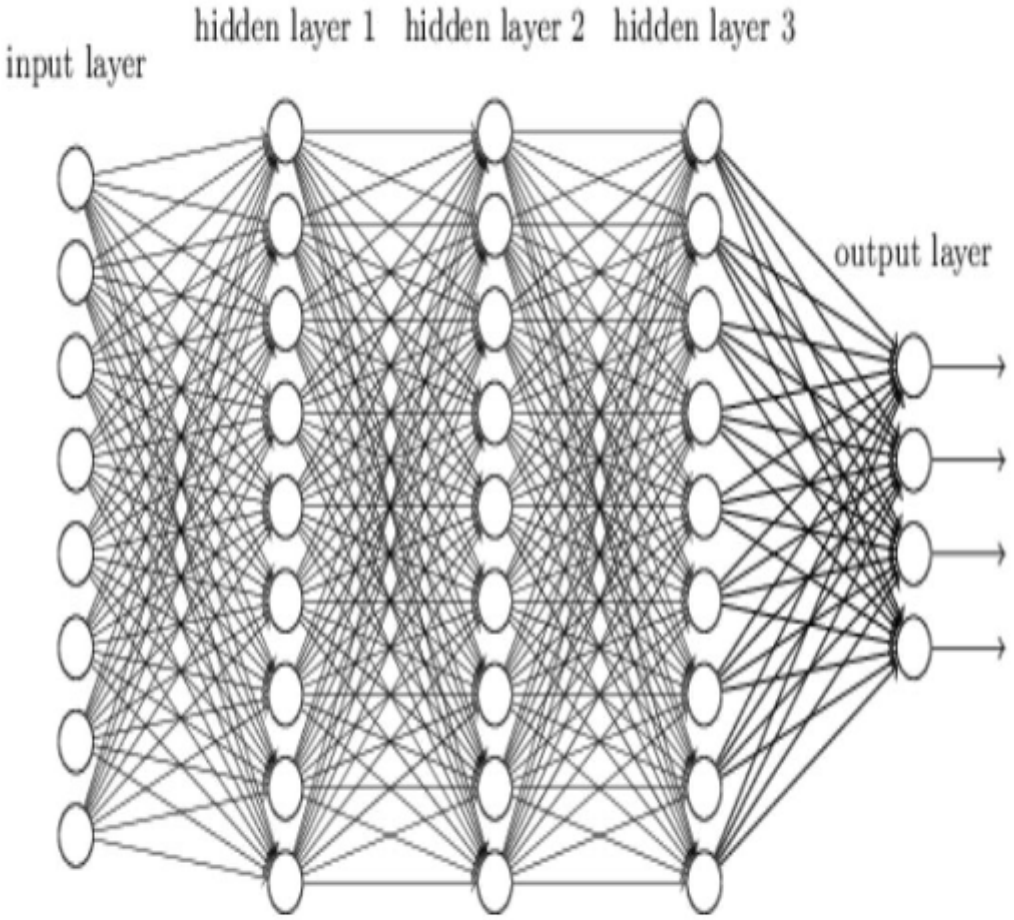


Fig 3.4.1 Neural Network

3.5 Flow Charts

Fig3.5.1 Flowchart

- **Control Flow**

This is the control flow of the image input to output

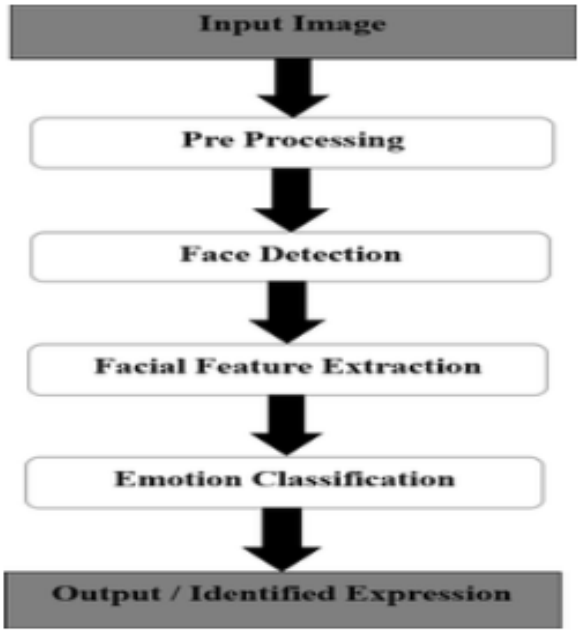


Fig.3.5.2 Control flow

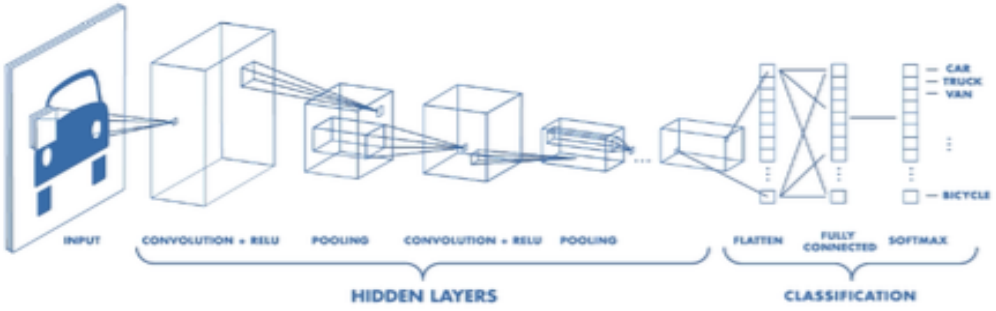
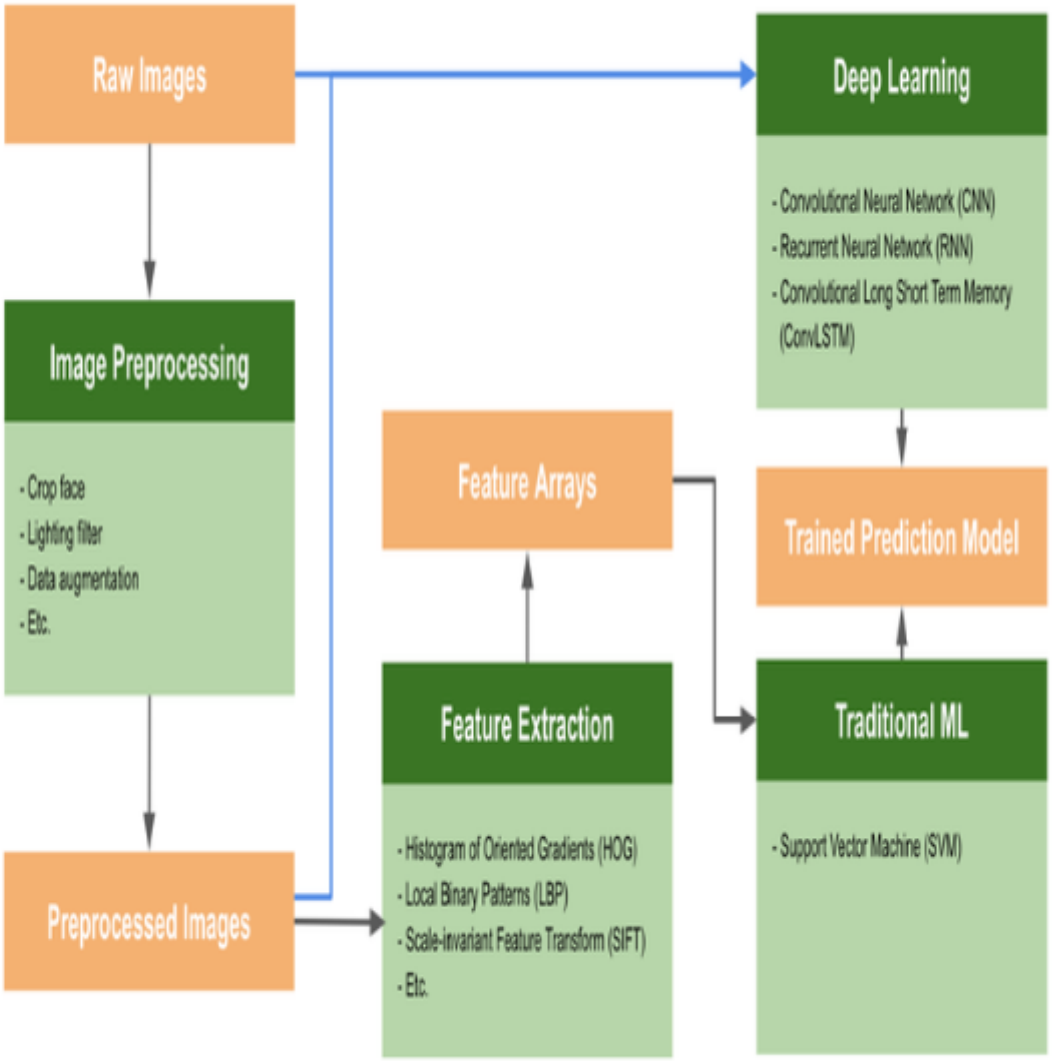


Fig.3.5.3 General classification

3.6 Trading Model



This is how the image classifier is work

CHAPTER 4

SOURCE CODE

4.1 Algorithm and Pseudo Code

To run main program

```
from __future__ import absolute_import from __future__ import division

from __future__ import print_function



import argparse
import sys
 import time



import numpy as np
import tensorflow as tf



def load_graph(model_file):
graph = tf.Graph()
graph_def = tf.GraphDef()



with open(model_file, "rb") as f:
graph_def.ParseFromString(f.read())
with graph.as_default():
tf.import_graph_def(graph_def)



return graph



def read_tensor_from_image_file(file_name, input_height=299, input_width=299,
input_mean=0,input_std=255):

input_name = "file_reader"



output_name = "normalized"
```

```
file_reader = tf.read_file(file_name, input_name)
if file_name.endswith(".png"):
image_reader = tf.image.decode_png(file_reader, channels = 3, name='png_reader')



elif file_name.endswith(".gif"):
image_reader = tf.squeeze(tf.image.decode_gif(file_reader,name='gif_reader'))

elif file_name.endswith(".bmp"):
image_reader = tf.image.decode_bmp(file_reader, name='bmp_reader')

else:

image_reader = tf.image.decode_jpeg(file_reader, channels = 3, name='jpeg_reader')

float_caster = tf.cast(image_reader, tf.float32)

dims_expander = tf.expand_dims(float_caster, 0);

resized = tf.image.resize_bilinear(dims_expander, [input_height, input_width])
normalized = tf.divide(tf.subtract(resized, [input_mean]), [input_std])

sess = tf.Session()

result = sess.run(normalized)


return result


def load_labels(label_file):
label = []
proto_as_ascii_lines = tf.gfile.GFile(label_file).readlines() for l in proto_as_ascii_lines:
label.append(l.rstrip())

return label




def main(img):

file_name = img
model_file = "retrained_graph.pb"
 label_file = "retrained_labels.txt"
input_height = 224
```

```
input_width = 224
input_mean = 128
input_std = 128
input_layer = "input"
output_layer = "final_result"

parser = argparse.ArgumentParser()

parser.add_argument("--image", help="image to be processed")
parser.add_argument("--graph", help="graph/model to be executed")
parser.add_argument("--labels", help="name of file containing labels")
parser.add_argument("--input_height", type=int, help="input height")
parser.add_argument("--input_width", type=int, help="input width")
parser.add_argument("--input_mean", type=int, help="input mean")
parser.add_argument("--input_std", type=int, help="input std")
parser.add_argument("--input_layer", help="name of input layer")
parser.add_argument("--output_layer", help="name of output layer")
args = parser.parse_args()



if args.graph:
        model_file = args.graph

if args.image:
        file_name = args.image

if args.labels:
        label_file = args.labels

if args.input_height:
        input_height = args.input_height



if args.input_width:
        input_width = args.input_width

if args.input_mean:
        input_mean = args.input_mean

if args.input_std:
        input_std = args.input_std

if args.input_layer:
        input_layer = args.input_layer
```

```
if args.output_layer:
        output_layer = args.output_layer

graph = load_graph(model_file)
        t = read_tensor_from_image_file(file_name,
                                                input_height=input_height,
                                                input_width=input_width,
                                                input_mean=input_mean,
                                                input_std=input_std)



input_name = "import/" + input_layer
output_name = "import/" + output_layer
input_operation = graph.get_operation_by_name(input_name);
output_operation = graph.get_operation_by_name(output_name);

with tf.Session(graph=graph) as sess:
        start = time.time()
        results = sess.run(output_operation.outputs[0],

                {input_operation.outputs[0]: t})
 end=time.time()

results = np.squeeze(results)

top_k = results.argsort()[-5:][::-1]
labels = load_labels(label_file)

for i in top_k:
        return labels[i]
```

There is the training code to create the model

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
```

```
import argparse
import collections
from datetime
import datetime
import hashlib
import os.path
import random
import re
import sys
import tarfile
import numpy as np

from six.moves import urllib
import tensorflow as tf

from tensorflow.python.framework import graph_util
from tensorflow.python.
framework import tensor_shape
from tensorflow.python.platform import gfile
from tensorflow.python.util import compat

FLAGS = None

#  These are all parameters that are tied to the particular model architecture
#  we're using for Inception v3. These include things like tensor names and their # sizes. If you
want to adapt this script to work with another model, you will
#  need to update these to reflect the values in the network you're using.
MAX_NUM_IMAGES_PER_CLASS = 2 ** 27 - 1 # ~134M

def create_image_lists(image_dir, testing_percentage, validation_percentage): """Builds a list of
training images from the file system.



Analyzes the sub folders in the image directory, splits them into stable
 training, testing, and validation sets, and returns a data structure
 describing the lists of images for each label and their paths.

Args:

image_dir: String path to a folder containing subfolders of images.
testing_percentage: Integer percentage of the images to reserve for tests.
validation_percentage: Integer percentage of images reserved for validation.

Returns:
```

```
A dictionary containing an entry for each label subfolder, with images split
 into training, testing, and validation sets within each label.
"""

if not gfile.Exists(image_dir):
tf.logging.error("Image directory '" + image_dir + "' not found.")

return None
result = collections.OrderedDict()
sub_dirs = [
os.path.join(image_dir,item)
for item in gfile.ListDirectory(image_dir)]
sub_dirs = sorted(item for item in sub_dirs
if gfile.IsDirectory(item))
for sub_dir in sub_dirs:
extensions = ['jpg', 'jpeg', 'JPG', 'JPEG']
file_list = []
dir_name = os.path.basename(sub_dir)
if dir_name == image_dir:

continue

tf.logging.info("Looking for images in '" + dir_name + "'")
for extension in extensions:
file_glob = os.path.join(image_dir, dir_name, '*.' + extension) file_list.extend(gfile.Glob(file_glob))
if not file_list:
tf.logging.warning('No files found')

continue


if len(file_list) < 20:
tf.logging.warning(

'WARNING: Folder has less than 20 images, which may cause issues.')

elif len(file_list) > MAX_NUM_IMAGES_PER_CLASS: tf.logging.warning(

'WARNING: Folder {} has more than {} images. Some images will '
never be selected.'.format(dir_name, MAX_NUM_IMAGES_PER_CLASS)) label_name =
re.sub(r'[^a-z0-9]+', ' ', dir_name.lower())

training_images = []
testing_images = []
validation_images = []
```

```
for file_name in file_list:
base_name = os.path.basename(file_name)

# We want to ignore anything after '_nohash_' in the file name when
# deciding which set to put an image in, the data set creator has a way of
# grouping photos that are close variations of each other. For example
# this is used in the plant disease data set to group multiple pictures of
# the same leaf.

hash_name = re.sub(r'_nohash_.*$', '', file_name)

# This looks a bit magical, but we need to decide whether this file should # go into the training,
testing, or validation sets, and we want to keep
# existing files in the same set even if more files are subsequently
# added.
# To do that, we need a stable way of deciding based on just the file name
# itself, so we do a hash of that and then use that to generate a
# probability value that we use to assign it.

hash_name_hashed = hashlib.sha1(compat.as_bytes(hash_name)).hexdigest()
percentage_hash = ((int(hash_name_hashed, 16) %
(MAX_NUM_IMAGES_PER_CLASS + 1)) *
(100.0 / MAX_NUM_IMAGES_PER_CLASS))

if percentage_hash < validation_percentage:
validation_images.append(base_name)
elif percentage_hash < (testing_percentage + validation_percentage):
testing_images.append(base_name)

else: training_images.append(base_name) result[label_name] = {
                                                    'dir': dir_name,
                                                    'training': training_images,
                                                    'testing': testing_images,
                                                    'validation': validation_images,
                                                }

return result


def get_image_path(image_lists, label_name, index, image_dir, category):



"""Returns a path to an image for a label at the given index.

Args:
```

image_lists: Dictionary of training images for each label.
label_name: Label string we want to get an image for.
index: Int offset of the image we want. This will be moduloed by the available number of images for the label, so it can be arbitrarily large. image_dir: Root folder string of the subfolders containing the training images.

category: Name string of set to pull images from - training, testing, or validation.

Returns:

File system path string to an image that meets the requested parameters.

"""

if label_name not in image_lists:
tf.logging.fatal('Label does not exist %s.', label_name) label_lists = image_lists[label_name]
if category not in label_lists:
tf.logging.fatal('Category does not exist %s.', category) category_list = label_lists[category]
if not category_list:
tf.logging.fatal('Label %s has no images in the category %s.', label_name, category)
mod_index = index % len(category_list)
base_name = category_list[mod_index]

sub_dir = label_lists['dir']
full_path = os.path.join(image_dir, sub_dir, base_name) return full_path

def get_bottleneck_path(image_lists, label_name, index, bottleneck_dir, category, architecture):

"""Returns a path to a bottleneck file for a label at the given index.

Args:
image_lists: Dictionary of training images for each label.
label_name: Label string we want to get an image for.
index: Integer offset of the image we want. This will be moduloed by the available number of images for the label, so it can be arbitrarily large. bottleneck_dir: Folder string holding cached files of bottleneck values. category: Name string of set to pull images from - training, testing, or validation.

architecture: The name of the model architecture.

Returns:

File system path string to an image that meets the requested parameters. """
return get_image_path(image_lists, label_name, index, bottleneck_dir, category) + '_' + architecture + '.txt'

```
def create_model_graph(model_info):

"""Creates a graph from saved GraphDef file and returns a Graph object.

Args:

model_info: Dictionary containing information about the model architecture.

Returns:

Graph holding the trained Inception network, and various tensors we'll be manipulating.

"""

with tf.Graph().as_default() as graph:
model_path = os.path.join(FLAGS.model_dir, model_info['model_file_name']) with
gfile.FastGFile(model_path, 'rb') as f:
graph_def = tf.GraphDef()
graph_def.ParseFromString(f.read())
bottleneck_tensor, resized_input_tensor =
(tf.import_graph_def(graph_def,name='',return_elements=[model_info['bottleneck_tensor_name'],
model_info['resized_input_tensor_name'],]))

return graph, bottleneck_tensor, resized_input_tensor

def run_bottleneck_on_image(sess, image_data, image_data_tensor,
 decoded_image_tensor,
resized_input_tensor,

bottleneck_tensor):

"""Runs inference on an image to extract the 'bottleneck' summary layer.

Args:

sess: Current active TensorFlow Session.
image_data: String of raw JPEG data.
image_data_tensor: Input data layer in the graph.
decoded_image_tensor: Output of initial image resizing and preprocessing. resized_input_tensor:
The input node of the recognition graph. bottleneck_tensor: Layer before the final softmax.



Returns:

Numpy array of bottleneck values.
```

"""

# First decode the JPEG image, resize it, and rescale the pixel values. resized_input_values = sess.run(decoded_image_tensor, {image_data_tensor: image_data})
# Then run it through the recognition network.
bottleneck_values = sess.run(bottleneck_tensor, {resized_input_tensor: resized_input_values})
bottleneck_values = np.squeeze(bottleneck_values)

return bottleneck_values

def maybe_download_and_extract(data_url): """Download and extract model tar file.

If the pretrained model we're using doesn't already exist, this function

downloads it from the TensorFlow.org website and unpacks it into a directory.

Args:

data_url: Web location of the tar file containing the pretrained model. """

dest_directory = FLAGS.model_dir
if not os.path.exists(dest_directory):
os.makedirs(dest_directory)
filename = data_url.split('/')[-1]
filepath = os.path.join(dest_directory, filename)
if not os.path.exists(filepath):

def _progress(count, block_size, total_size): sys.stdout.write('\r>> Downloading %s %.1f%%' % (filename,

float(count * block_size) / float(total_size) * 100.0)) sys.stdout.flush()

filepath, _ = urllib.request.urlretrieve(data_url, filepath, _progress) print()
statinfo = os.stat(filepath)
tf.logging.info('Successfully downloaded', filename, statinfo.st_size, 'bytes.')

```
tarfile.open(filepath, 'r:gz').extractall(dest_directory)

def ensure_dir_exists(dir_name):

"""Makes sure the folder exists on disk.

Args:

dir_name: Path string to the folder we want to create. """



if not os.path.exists(dir_name): os.makedirs(dir_name)
bottleneck_path_2_bottleneck_values = {}



def create_bottleneck_file(bottleneck_path, image_lists, label_name, index,
image_dir, category, sess, jpeg_data_tensor,
decoded_image_tensor, resized_input_tensor,
bottleneck_tensor):

"""Create a single bottleneck file."""

tf.logging.info('Creating bottleneck at ' + bottleneck_path) image_path =
get_image_path(image_lists, label_name, index, image_dir, category)
if not gfile.Exists(image_path):
tf.logging.fatal('File does not exist %s', image_path) image_data = gfile.FastGFile(image_path,
'rb').read()
Try:
bottleneck_values = run_bottleneck_on_image(
sess, image_data, jpeg_data_tensor, decoded_image_tensor,
 resized_input_tensor, bottleneck_tensor)

except Exception as e:

raise RuntimeError('Error during processing file %s (%s)' % (image_path, str(e)))

bottleneck_string = ','.join(str(x) for x in bottleneck_values)

with open(bottleneck_path, 'w') as bottleneck_file: bottleneck_file.write(bottleneck_string)



def get_or_create_bottleneck(sess, image_lists,
        label_name, index,
        image_dir, category, bottleneck_dir,
        jpeg_data_tensor,
```

decoded_image_tensor, resized_input_tensor,
bottleneck_tensor, architecture):

"""Retrieves or calculates bottleneck values for an image.

If a cached version of the bottleneck data exists on-disk, return that,
otherwise calculate the data and save it to disk for future use.

Args:

sess: The current active TensorFlow Session.
image_lists: Dictionary of training images for each label.
label_name: Label string we want to get an image for.

index: Integer offset of the image we want. This will be modulo-ed by the available number of images for the label, so it can be arbitrarily large. image_dir: Root folder string of the subfolders containing the training images.

category: Name string of which set to pull images from - training, testing,
or validation.

bottleneck_dir: Folder string holding cached files of bottleneck values. jpeg_data_tensor: The tensor to feed loaded jpeg data into. decoded_image_tensor: The output of decoding and resizing the image. resized_input_tensor: The input node of the recognition graph. bottleneck_tensor: The output tensor for the bottleneck values. architecture: The name of the model architecture.

Returns:

Numpy array of values produced by the bottleneck layer for the image. """

label_lists = image_lists[label_name]

sub_dir = label_lists['dir']

sub_dir_path = os.path.join(bottleneck_dir, sub_dir) ensure_dir_exists(sub_dir_path)

bottleneck_path = get_bottleneck_path(image_lists, label_name, index, bottleneck_dir, category, architecture)

if not os.path.exists(bottleneck_path): create_bottleneck_file(bottleneck_path, image_lists, label_name, index, image_dir, category, sess, jpeg_data_tensor,

decoded_image_tensor, resized_input_tensor,
bottleneck_tensor)
with open(bottleneck_path, 'r') as bottleneck_file:

bottleneck_string = bottleneck_file.read()
did_hit_error = False

Try:

bottleneck_values = [float(x) for x in bottleneck_string.split(',')]
except ValueError:

tf.logging.warning('Invalid float found, recreating bottleneck') did_hit_error = True

if did_hit_error:

create_bottleneck_file(bottleneck_path, image_lists, label_name, index, image_dir, category, sess, jpeg_data_tensor,

decoded_image_tensor, resized_input_tensor, bottleneck_tensor)

with open(bottleneck_path, 'r') as bottleneck_file: bottleneck_string = bottleneck_file.read()

Allow exceptions to propagate here, since they shouldn't happen after a # fresh creation

bottleneck_values = [float(x) for x in bottleneck_string.split(',')]

return bottleneck_values

def cache_bottlenecks(sess, image_lists, image_dir, bottleneck_dir, jpeg_data_tensor, decoded_image_tensor,

resized_input_tensor, bottleneck_tensor, architecture):

"""Ensures all the training, testing, and validation bottlenecks are cached.

Because we're likely to read the same image multiple times (if there are no distortions applied during training) it can speed things up a lot if we calculate the bottleneck layer values once for each image during preprocessing, and then just read those cached values repeatedly during training. Here we go through all the images we've found, calculate those values, and save them off.

Args:

sess: The current active TensorFlow Session.
image_lists: Dictionary of training images for each label.
image_dir: Root folder string of the subfolders containing the training images.

bottleneck_dir: Folder string holding cached files of bottleneck values. jpeg_data_tensor: Input tensor for jpeg data from file. decoded_image_tensor: The output of decoding and resizing the image. resized_input_tensor: The input node of the recognition graph. bottleneck_tensor: The penultimate output layer of the graph. architecture: The name of the model architecture.

Returns:

Nothing.

"""

how_many_bottlenecks = 0 ensure_dir_exists(bottleneck_dir)

for label_name, label_lists in image_lists.items(): for category in ['training', 'testing', 'validation']:
category_list = label_lists[category]

for index, unused_base_name in enumerate(category_list): get_or_create_bottleneck(
sess, image_lists, label_name, index, image_dir, category, bottleneck_dir, jpeg_data_tensor,
decoded_image_tensor, resized_input_tensor, bottleneck_tensor, architecture)

how_many_bottlenecks += 1
if how_many_bottlenecks % 100 == 0:
tf.logging.info(
str(how_many_bottlenecks) + ' bottleneck files created.')

def get_random_cached_bottlenecks(sess, image_lists, how_many, category, bottleneck_dir,
image_dir, jpeg_data_tensor,
decoded_image_tensor, resized_input_tensor,

bottleneck_tensor, architecture):.

Args:

sess: Current TensorFlow Session.
image_lists: Dictionary of training images for each label.
how_many: If positive, a random sample of this size will be chosen.
If negative, all bottlenecks will be retrieved.
category: Name string of which set to pull from - training, testing, or validation.
bottleneck_dir: Folder string holding cached files of bottleneck values. image_dir: Root folder string of the subfolders containing the training images.

jpeg_data_tensor: The layer to feed jpeg image data into. decoded_image_tensor: The output of decoding and resizing the image. resized_input_tensor: The input node of the recognition graph. bottleneck_tensor: The bottleneck output layer of the CNN graph. architecture: The name of the model architecture.

Returns:

List of bottleneck arrays, their corresponding ground truths, and the

relevant filenames.

"""

class_count = len(image_lists.keys())

bottlenecks = []

ground_truths = []

filenames = []

if how_many >= 0:

Retrieve a random sample of bottlenecks.

for unused_i in range(how_many):

label_index = random.randrange(class_count)

label_name = list(image_lists.keys())[label_index]

image_index = random.randrange(MAX_NUM_IMAGES_PER_CLASS + 1) image_name = get_image_path(image_lists, label_name, image_index, image_dir, category)

bottleneck = get_or_create_bottleneck(

sess, image_lists, label_name, image_index, image_dir, category, bottleneck_dir, jpeg_data_tensor, decoded_image_tensor, resized_input_tensor, bottleneck_tensor, architecture)

ground_truth = np.zeros(class_count, dtype=np.float32) ground_truth[label_index] = 1.0

bottlenecks.append(bottleneck)

ground_truths.append(ground_truth)

filenames.append(image_name)

else:

Retrieve all bottlenecks.

for label_index, label_name in enumerate(image_lists.keys()):
for image_index, image_name in enumerate(image_lists[label_name][category]):
image_name = get_image_path(image_lists, label_name, image_index, image_dir, category)

bottleneck = get_or_create_bottleneck(
sess, image_lists, label_name, image_index, image_dir, category, bottleneck_dir, jpeg_data_tensor, decoded_image_tensor, resized_input_tensor, bottleneck_tensor, architecture)

ground_truth = np.zeros(class_count, dtype=np.float32) ground_truth[label_index] = 1.0

bottlenecks.append(bottleneck)
ground_truths.append(ground_truth)
filenames.append(image_name)
return bottlenecks, ground_truths, filenames

def get_random_distorted_bottlenecks(

sess, image_lists, how_many, category, image_dir, input_jpeg_tensor, distorted_image, resized_input_tensor, bottleneck_tensor):

"""Retrieves bottleneck values for training images, after distortions.. Args:

sess: Current TensorFlow Session.

image_lists: Dictionary of training images for each label.
how_many: The integer number of bottleneck values to return. category: Name string of which set of images to fetch - training, testing, or validation.

image_dir: Root folder string of the subfolders containing the training images.

nput_jpeg_tensor: The input layer we feed the image data to. distorted_image: The output node of the distortion graph. resized_input_tensor: The input node of the recognition graph.
bottleneck_tensor: The bottleneck output layer of the CNN graph.

Returns:

List of bottleneck arrays and their corresponding ground truths.

"""

class_count = len(image_lists.keys())
bottlenecks = []
ground_truths = []
for unused_i in range(how_many):
label_index = random.randrange(class_count)
label_name = list(image_lists.keys())[label_index]

image_index = random.randrange(MAX_NUM_IMAGES_PER_CLASS + 1) image_path = get_image_path(image_lists, label_name, image_index, image_dir, category)
if not gfile.Exists(image_path):
tf.logging.fatal('File does not exist %s', image_path)
jpeg_data = gfile.FastGFile(image_path, 'rb').read()

Note that we materialize the distorted_image_data as a numpy array before
sending running inference on the image. This involves 2 memory copies and
might be optimized in other implementations.

distorted_image_data = sess.run(distorted_image,
{input_jpeg_tensor: jpeg_data})
bottleneck_values = sess.run(bottleneck_tensor,
bottleneck_values = np.squeeze(bottleneck_values) ground_truth = np.zeros(class_count, dtype=np.float32) ground_truth[label_index] = 1.0 bottlenecks.append(bottleneck_values) ground_truths.append(ground_truth)

return bottlenecks, ground_truths

def should_distort_images(flip_left_right, random_crop, random_scale, random_brightness):

"""Whether any distortions are enabled, from the input flags.

Args:

flip_left_right: Boolean whether to randomly mirror images horizontally. random_crop: Integer percentage setting the total margin used around the crop box.

random_scale: Integer percentage of how much to vary the scale by. random_brightness: Integer range to randomly multiply the pixel values by.

Returns:

Boolean value indicating whether any distortions should be applied. """

return (flip_left_right or (random_crop != 0) or (random_scale != 0) or (random_brightness != 0))

def add_input_distortions(flip_left_right, random_crop, random_scale, random_brightness, input_width, input_height, input_depth, input_mean, input_std):

Args:

flip_left_right: Boolean whether to randomly mirror images horizontally. random_crop: Integer percentage setting the total margin used around the crop box.

random_scale: Integer percentage of how much to vary the scale by. random_brightness: Integer range to randomly multiply the pixel values by. graph.

input_width: Horizontal size of expected input image to model. input_height: Vertical size of expected input image to model.

input_depth: How many channels the expected input image should have. input_mean: Pixel value that should be zero in the image for the graph. input_std: How much to divide the pixel values by before recognition.

Returns:

The jpeg input layer and the distorted result tensor. """

jpeg_data = tf.placeholder(tf.string, name='DistortJPGInput') decoded_image = tf.image.decode_jpeg(jpeg_data, channels=input_depth) decoded_image_as_float = tf.cast(decoded_image, dtype=tf.float32) decoded_image_4d = tf.expand_dims(decoded_image_as_float, 0) margin_scale = 1.0 + (random_crop / 100.0)

resize_scale = 1.0 + (random_scale / 100.0)

margin_scale_value = tf.constant(margin_scale)

resize_scale_value = tf.random_uniform(tensor_shape.scalar(), minval=1.0,

maxval=resize_scale)

scale_value = tf.multiply(margin_scale_value, resize_scale_value) precrop_width = tf.multiply(scale_value, input_width) precrop_height = tf.multiply(scale_value, input_height) precrop_shape = tf.stack([precrop_height, precrop_width]) precrop_shape_as_int = tf.cast(precrop_shape, dtype=tf.int32) precropped_image = tf.image.resize_bilinear(decoded_image_4d, precrop_shape_as_int)

precropped_image_3d = tf.squeeze(precropped_image, squeeze_dims=[0]) cropped_image = tf.random_crop(precropped_image_3d,

[input_height, input_width, input_depth])

```
if flip_left_right:


flipped_image = tf.image.random_flip_left_right(cropped_image) else:

flipped_image = cropped_image

brightness_min = 1.0 - (random_brightness / 100.0) brightness_max = 1.0 + (random_brightness /
100.0) brightness_value = tf.random_uniform(tensor_shape.scalar(), minval=brightness_min,


maxval=brightness_max)

brightened_image = tf.multiply(flipped_image, brightness_value) offset_image =
tf.subtract(brightened_image, input_mean) mul_image = tf.multiply(offset_image, 1.0 / input_std)

distort_result = tf.expand_dims(mul_image, 0, name='DistortResult') return jpeg_data, distort_result

def variable_summaries(var):

"""Attach a lot of summaries to a Tensor (for TensorBoard visualization)."""" with
tf.name_scope('summaries'):

mean = tf.reduce_mean(var)

tf.summary.scalar('mean', mean)

with tf.name_scope('stddev'):

stddev = tf.sqrt(tf.reduce_mean(tf.square(var - mean))) tf.summary.scalar('stddev', stddev)

tf.summary.scalar('max', tf.reduce_max(var))

tf.summary.scalar('min', tf.reduce_min(var)) tf.summary.histogram('histogram', var)


def add_final_training_ops(class_count, final_tensor_name, bottleneck_tensor,
bottleneck_tensor_size):

Args:

class_count: Integer of how many categories of things we're trying to recognize.

final_tensor_name: Name string for the new final node that produces results. bottleneck_tensor: The
output of the main CNN graph. bottleneck_tensor_size: How many entries in the bottleneck vector.
```

```
Returns:
The tensors for the training and cross entropy results, and tensors for the

bottleneck input and ground truth input.
"""
with tf.name_scope('input'):
bottleneck_input = tf.placeholder_with_default( bottleneck_tensor,

shape=[None, bottleneck_tensor_size], name='BottleneckInputPlaceholder')

ground_truth_input = tf.placeholder(tf.float32, [None, class_count], name='GroundTruthInput')

# Organizing the following ops as `final_training_ops` so they're easier # to see in TensorBoard
layer_name = 'final_training_ops'
with tf.name_scope(layer_name):

with tf.name_scope('weights'):
initial_value = tf.truncated_normal( [bottleneck_tensor_size, class_count], stddev=0.001)

layer_weights = tf.Variable(initial_value, name='final_weights')

variable_summaries(layer_weights)
with tf.name_scope('biases'):
```

```
layer_biases = tf.Variable(tf.zeros([class_count]), name='final_biases')
variable_summaries(layer_biases)

with tf.name_scope('Wx_plus_b'):

        logits = tf.matmul(bottleneck_input, layer_weights) + layer_biases
tf.summary.histogram('pre_activations', logits)


final_tensor = tf.nn.softmax(logits, name=final_tensor_name) tf.summary.histogram('activations',
final_tensor)


with tf.name_scope('cross_entropy'):

cross_entropy = tf.nn.softmax_cross_entropy_with_logits( labels=ground_truth_input, logits=logits)

with tf.name_scope('total'):

cross_entropy_mean = tf.reduce_mean(cross_entropy) tf.summary.scalar('cross_entropy',
cross_entropy_mean)


with tf.name_scope('train'):

optimizer = tf.train.GradientDescentOptimizer(FLAGS.learning_rate) train_step =
optimizer.minimize(cross_entropy_mean)


return (train_step, cross_entropy_mean, bottleneck_input, ground_truth_input, final_tensor)


def add_evaluation_step(result_tensor, ground_truth_tensor):

"""Inserts the operations we need to evaluate the accuracy of our results.


Args:

result_tensor: The new final node that produces results. ground_truth_tensor: The node we feed
ground truth data

into.
```

```
Returns:

Tuple of (evaluation step, prediction).

"""

with tf.name_scope('accuracy'):

with tf.name_scope('correct_prediction'):

prediction = tf.argmax(result_tensor, 1)

correct_prediction = tf.equal(

prediction, tf.argmax(ground_truth_tensor, 1))

with tf.name_scope('accuracy'):

evaluation_step = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
tf.summary.scalar('accuracy', evaluation_step)

return evaluation_step, prediction


def save_graph_to_file(sess, graph, graph_file_name): output_graph_def =
graph_util.convert_variables_to_constants( sess, graph.as_graph_def(), [FLAGS.final_tensor_name])

with gfile.FastGFile(graph_file_name, 'wb') as f: f.write(output_graph_def.SerializeToString())

return

def prepare_file_system():

# Setup the directory we'll write summaries to for TensorBoard if
tf.gfile.Exists(FLAGS.summaries_dir):

tf.gfile.DeleteRecursively(FLAGS.summaries_dir) tf.gfile.MakeDirs(FLAGS.summaries_dir)

if FLAGS.intermediate_store_frequency > 0:
ensure_dir_exists(FLAGS.intermediate_output_graphs_dir) return

def create_model_info(architecture):

Args:

architecture: Name of a model architecture.

Returns:
```

Dictionary of information about the model, or None if the name isn't recognized

Raises:

ValueError: If architecture name is unknown.

"""

architecture = architecture.lower()

if architecture == 'inception_v3':

pylint: disable=line-too-long

data_url = 'http://download.tensorflow.org/models/image/imagenet/inception-2015-12-05.tgz' # pylint: enable=line-too-long

bottleneck_tensor_name = 'pool_3/_reshape:0'

bottleneck_tensor_size = 2048

input_width = 299

input_height = 299

input_depth = 3

resized_input_tensor_name = 'Mul:0'

model_file_name = 'classify_image_graph_def.pb'

input_mean = 128

input_std = 128

elif architecture.startswith('mobilenet_'):

parts = architecture.split('_')

if len(parts) != 3 and len(parts) != 4:

tf.logging.error("Couldn't understand architecture name '%s'", architecture)

return None

version_string = parts[1]

if (version_string != '1.0' and version_string != '0.75' and version_string != '0.50' and version_string != '0.25'): tf.logging.error(

"""The Mobilenet version should be '1.0', '0.75', '0.50', or '0.25', but found '%s' for architecture '%s'""",

version_string, architecture)

return None

size_string = parts[2]

if (size_string != '224' and size_string != '192' and

size_string != '160' and size_string != '128'):

tf.logging.error(

"""The Mobilenet input size should be '224', '192', '160', or '128', but found '%s' for architecture '%s'""",

size_string, architecture)

return None

if len(parts) == 3:

is_quantized = False

else:

if parts[3] != 'quantized':

tf.logging.error(

"Couldn't understand architecture suffix '%s' for '%s'", parts[3], architecture)

return None

is_quantized = True

data_url = 'http://download.tensorflow.org/models/mobilenet_v1_' data_url += version_string + '_' + size_string + '_frozen.tgz' bottleneck_tensor_name = 'MobilenetV1/Predictions/Reshape:0' bottleneck_tensor_size = 1001

input_width = int(size_string)

input_height = int(size_string)

input_depth = 3

resized_input_tensor_name = 'input:0'

```
if is_quantized:

model_base_name = 'quantized_graph.pb'

else:

model_base_name = 'frozen_graph.pb'

model_dir_name = 'mobilenet_v1_' + version_string + '_' + size_string model_file_name =
os.path.join(model_dir_name, model_base_name) input_mean = 127.5

input_std = 127.5

else:

tf.logging.error("Couldn't understand architecture name '%s'", architecture) raise
ValueError('Unknown architecture', architecture)

return {

'data_url': data_url,

'bottleneck_tensor_name': bottleneck_tensor_name, 'bottleneck_tensor_size': bottleneck_tensor_size,
'input_width': input_width,

'input_height': input_height,

'input_depth': input_depth,

'resized_input_tensor_name': resized_input_tensor_name, 'model_file_name': model_file_name,

'input_mean': input_mean,

'input_std': input_std,

}


def add_jpeg_decoding(input_width, input_height, input_depth, input_mean, input_std):

"""Adds operations that perform JPEG decoding and resizing to the graph..

Args:

input_width: Desired width of the image fed into the recognizer graph. input_height: Desired width
of the image fed into the recognizer graph. input_depth: Desired channels of the image fed into the
recognizer graph. input_mean: Pixel value that should be zero in the image for the graph. input_std:
How much to divide the pixel values by before recognition.
```

```
Returns:

Tensors for the node to feed JPEG data into, and the output of the preprocessing steps.

"""

jpeg_data = tf.placeholder(tf.string, name='DecodeJPGInput') decoded_image =
tf.image.decode_jpeg(jpeg_data, channels=input_depth) decoded_image_as_float =
tf.cast(decoded_image, dtype=tf.float32) decoded_image_4d =
tf.expand_dims(decoded_image_as_float, 0) resize_shape = tf.stack([input_height, input_width])

resize_shape_as_int = tf.cast(resize_shape, dtype=tf.int32)

resized_image = tf.image.resize_bilinear(decoded_image_4d, resize_shape_as_int)

offset_image = tf.subtract(resized_image, input_mean)

mul_image = tf.multiply(offset_image, 1.0 / input_std)

return jpeg_data, mul_image



def main(_):

# Needed to make sure the logging output is visible.
# See https://github.com/tensorflow/tensorflow/issues/3047
tf.logging.set_verbosity(tf.logging.INFO)
# Prepare necessary directories that can be used during training prepare_file_system()
# Gather information about the model architecture we'll be using.

model_info = create_model_info(FLAGS.architecture) if not model_info:

tf.logging.error('Did not recognize architecture flag') return -1
# Set up the pre-trained graph. maybe_download_and_extract(model_info['data_url']) graph,
bottleneck_tensor, resized_image_tensor = ( create_model_graph(model_info))
# Look at the folder structure, and create lists of all the images.

image_lists = create_image_lists(FLAGS.image_dir, FLAGS.testing_percentage,
FLAGS.validation_percentage)

class_count = len(image_lists.keys())

if class_count == 0:

tf.logging.error('No valid folders of images found at ' + FLAGS.image_dir) return -1

if class_count == 1:
```

```
tf.logging.error('Only one valid folder of images found at ' +

FLAGS.image_dir +

' - multiple classes are needed for classification.')

return -1


# See if the command-line flags mean we're applying any distortions. do_distort_images =
should_distort_images(
FLAGS.flip_left_right, FLAGS.random_crop, FLAGS.random_scale, FLAGS.random_brightness)
with tf.Session(graph=graph) as sess:
# Set up the image decoding sub-graph.
jpeg_data_tensor, decoded_image_tensor = add_jpeg_decoding( model_info['input_width'],
model_info['input_height'], model_info['input_depth'], model_info['input_mean'],
model_info['input_std'])

if do_distort_images:

# We will be applying distortions, so setup the operations we'll need. (distorted_jpeg_data_tensor,
distorted_image_tensor) = add_input_distortions( FLAGS.flip_left_right, FLAGS.random_crop,
FLAGS.random_scale, FLAGS.random_brightness, model_info['input_width'],
model_info['input_height'], model_info['input_depth'], model_info['input_mean'],
model_info['input_std'])

else:

# We'll make sure we've calculated the 'bottleneck' image summaries and # cached them on disk.
cache_bottlenecks(sess, image_lists, FLAGS.image_dir, FLAGS.bottleneck_dir, jpeg_data_tensor,
decoded_image_tensor, resized_image_tensor,
bottleneck_tensor, FLAGS.architecture)
# Add the new layer that we'll be training.
(train_step, cross_entropy, bottleneck_input, ground_truth_input, final_tensor) =
add_final_training_ops(len(image_lists.keys()), FLAGS.final_tensor_name, bottleneck_tensor,
model_info['bottleneck_tensor_size'])

# Create the operations we need to evaluate the accuracy of our new layer. evaluation_step,
prediction = add_evaluation_step(final_tensor, ground_truth_input)

# Merge all the summaries and write them out to the summaries_dir merged =
tf.summary.merge_all()

train_writer = tf.summary.FileWriter(FLAGS.summaries_dir + '/train', sess.graph)
```

```
validation_writer = tf.summary.FileWriter( FLAGS.summaries_dir + '/validation')

# Set up all our weights to their initial default values. init = tf.global_variables_initializer()

sess.run(init)

# Run the training for as many cycles as requested on the command line. for i in
range(FLAGS.how_many_training_steps):
# Get a batch of input bottleneck values, either calculated fresh every
# time with distortions applied, or from the cache stored on disk.



if do_distort_images:

(train_bottlenecks,
train_ground_truth) = get_random_distorted_bottlenecks( sess, image_lists,
FLAGS.train_batch_size, 'training',

FLAGS.image_dir, distorted_jpeg_data_tensor,

distorted_image_tensor, resized_image_tensor, bottleneck_tensor)

else:

(train_bottlenecks,

train_ground_truth, _) = get_random_cached_bottlenecks(

sess, image_lists, FLAGS.train_batch_size, 'training', FLAGS.bottleneck_dir, FLAGS.image_dir,
jpeg_data_tensor, decoded_image_tensor, resized_image_tensor, bottleneck_tensor,
FLAGS.architecture)

# Feed the bottlenecks and ground truth into the graph, and run a training # step. Capture training
summaries for TensorBoard with the `merged` op. train_summary, _ = sess.run(

[merged, train_step],

feed_dict={bottleneck_input: train_bottlenecks,

ground_truth_input: train_ground_truth}) train_writer.add_summary(train_summary, i)

# Every so often, print out how well the graph is training.
is_last_step = (i + 1 == FLAGS.how_many_training_steps)
if (i % FLAGS.eval_step_interval) == 0 or is_last_step:
train_accuracy, cross_entropy_value = sess.run
( [evaluation_step, cross_entropy],
feed_dict={bottleneck_input: train_bottlenecks, ground_truth_input: train_ground_truth})
```

```
tf.logging.info('%s: Step %d: Train accuracy = %.1f%%' %
(datetime.now(), i, train_accuracy * 100))
 tf.logging.info('%s: Step %d: Cross entropy = %f' %

(datetime.now(), i, cross_entropy_value))

validation_bottlenecks, validation_ground_truth, _ = ( get_random_cached_bottlenecks(

sess, image_lists, FLAGS.validation_batch_size, 'validation', FLAGS.bottleneck_dir,
FLAGS.image_dir, jpeg_data_tensor, decoded_image_tensor, resized_image_tensor,
bottleneck_tensor, FLAGS.architecture))

# Run a validation step and capture training summaries for TensorBoard # with the `merged` op.

validation_summary, validation_accuracy = sess.run(

[merged, evaluation_step],

feed_dict={bottleneck_input: validation_bottlenecks, ground_truth_input: validation_ground_truth})
validation_writer.add_summary(validation_summary, i) tf.logging.info('%s: Step %d: Validation
accuracy = %.1f%% (N=%d)' % (datetime.now(), i, validation_accuracy * 100,
len(validation_bottlenecks)))



# Store intermediate results

intermediate_frequency = FLAGS.intermediate_store_frequency



if (intermediate_frequency > 0 and (i % intermediate_frequency == 0) and i > 0):

intermediate_file_name = (FLAGS.intermediate_output_graphs_dir + 'intermediate_' + str(i) + '.pb')



tf.logging.info('Save intermediate result to : ' + intermediate_file_name)

type=int,

default=0,

help="""\

How many steps to store intermediate graph. If "0" then will not store."""

)
```

```
parser.add_argument(
'--output_labels',
type=str,
default='/tmp/output_labels.txt',
help='Where to save the trained graph\'s labels.')

parser.add_argument('--summaries_dir',
type=str,
default='/tmp/retrain_logs',
help='Where to save summary logs for TensorBoard.' )
parser.add_argument(
'--how_many_training_steps',
type=int,
default=6000,
help='How many training steps to run before ending.' )
parser.add_argument('--learning_rate',
                    type=float,
                    default=0.01,
                    help='How large a learning rate to use when training.'
                    )

parser.add_argument(
                '--testing_percentage',type=int,
                default=10,
                help='What percentage of images to use as a test set.'
                )

parser.add_argument(
                    '--validation_percentage',
                    type=int,
                    default=10,
                    help='What percentage of images to use as a validation set.'

                    )

parser.add_argument(
                '--eval_step_interval',
                type=int,
                default=10,
                help='How often to evaluate the training results.'
                )

parser.add_argument(
'--train_batch_size',
type=int,
```

```
default=100,
help='How many images to train on at a time.' )


parser.add_argument(

'--test_batch_size',

type=int,

default=-1,

help="""\

How many images to test on. This test set is only used once, to evaluate the final accuracy of the
model after training completes.

A value of -1 causes the entire test set to be used, which leads to more stable results across runs.\

""")

parser.add_argument(

'--validation_batch_size',

type=int,

default=100,

help="""\

How many images to use in an evaluation batch. This validation set is used much more often than the
test set, and is an early indicator of how accurate the model is during training.

A value of -1 causes the entire validation set to be used, which leads to more stable results across
training iterations, but may be slower on large training sets.\

""")

parser.add_argument(

'--print_misclassified_test_images',

default=False,

help="""\

Whether to print out a list of all misclassified test images.\ """,
```

```
action='store_true'

)

parser.add_argument(

'--model_dir',

type=str,

default='/tmp/imagenet',

help="""\

Path to classify_image_graph_def.pb, imagenet_synset_to_human_label_map.txt, and
imagenet_2012_challenge_label_map_proto.pbtxt.\

"""

)

parser.add_argument(

'--bottleneck_dir',

type=str,

default='/tmp/bottleneck',

help='Path to cache bottleneck layer values as files.'

)

parser.add_argument(

'--final_tensor_name',

type=str,

default='final_result',

help="""\

The name of the output classification layer in the retrained graph.\ """)

parser.add_argument(

'--flip_left_right',
```

```
default=False,
help="""\
Whether to randomly flip half of the training images horizontally.\ """,
action='store_true'
) parser.add_argument( '--random_crop', type=int,
default=0,
help="""\
A percentage determining how much of a margin to randomly crop off the training images.\
"""
)
parser.add_argument(
'--random_scale',
type=int,
default=0,
help="""\
A percentage determining how much to randomly scale up the size of the training images by.\
"""
)
parser.add_argument(
```

```
'--random_brightness',
type=int,
default=0,
help="""\
A percentage determining how much to randomly multiply the training image
input pixels up or down by.\
""")
parser.add_argument(
'--architecture',
type=str,
default='inception_v3',

help="""\

Which model architecture to use. 'inception_v3' is the most accurate, but
also the slowest. For faster or smaller models, chose a MobileNet with the
form 'mobilenet_<parameter size>_<input_size>[_quantized]'. For example,
'mobilenet_1.0_224' will pick a model that is 17 MB in size and takes 224
pixel input images, while 'mobilenet_0.25_128_quantized' will choose a much
less accurate, but smaller and faster network that's 920 KB on disk and
takes 128x128 images. See https://research.googleblog.com/2017/06/mobilenets-open-
source-models-for.html
for more information on Mobilenet.\""")
FLAGS, unparsed = parser.parse_known_args()
tf.app.run(main=main, argv=[sys.argv[0]] + unparsed)
```

This the program by which the image is converted in grey scale and cropped

This program first ensures if the face of a person exists in the given image or not then if it exists, it crops
the image of the face and saves to the given directory.
Importing Modules import cv2

import os

##

##Make changes to these lines for getting the desired results.
DIRECTORY of the images
directory = "F:\project june\Facial-Expression-Detection-master\images_2\Sad"
directory where the images to be saved:

f_directory = "F:\project june\Facial-Expression-Detection-master\images\Sad"

##

def facecrop(image):

Crops the face of a person from any image!
OpenCV XML FILE for Frontal Facial Detection using HAAR CASCADES. facedata = "haarcascade_frontalface_alt.xml"

cascade = cv2.CascadeClassifier(facedata)
Reading the given Image with OpenCV img = cv2.imread(image)

Try:
Some downloaded images are of unsupported type and should be ignored while raising Exception, so for that

I'm using the try/except functions.

minisize = (img.shape[1],img.shape[0]) miniframe = cv2.resize(img, minisize)

faces = cascade.detectMultiScale(miniframe)

for f in faces:

x, y, w, h = [v for v in f]

cv2.rectangle(img, (x,y), (x+w,y+h), (0,255,0), 2)

```
sub_face = img[y:y+h, x:x+w]
f_name = image.split('/') f_name = f_name[-1]
## Change here the Desired directory. cv2.imwrite(f_directory + f_name, sub_face) print ("Writing: "
+ image)

except: pass

if __name__ == '__main__': images = os.listdir(directory) i= 0

for img in images:

file = directory + img print (i) facecrop(file)

i += 1
```

4.2 DATABASE

The dataset, used for training the model is from a random Google images Facial Expression Recognition . The data consists of 48x48 pixel grey scale images of faces. The faces have been automatically registered so that the face is more or less centred and occupies about the same amount of space in each image. The five emotion are categorized each face based on the emotion shown in the facial expression in to one of seven categories

0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise

The training set consists of 850 examples. The final test set, which was used to determine the winner of the competition, consists of another to level to judge .

Emotion labels in the dataset:

0: -850 images- Angry

1: -123 images- Disgust

2: -145 images- Happy

4: -345images- Sad

5: -112 images- Surprise

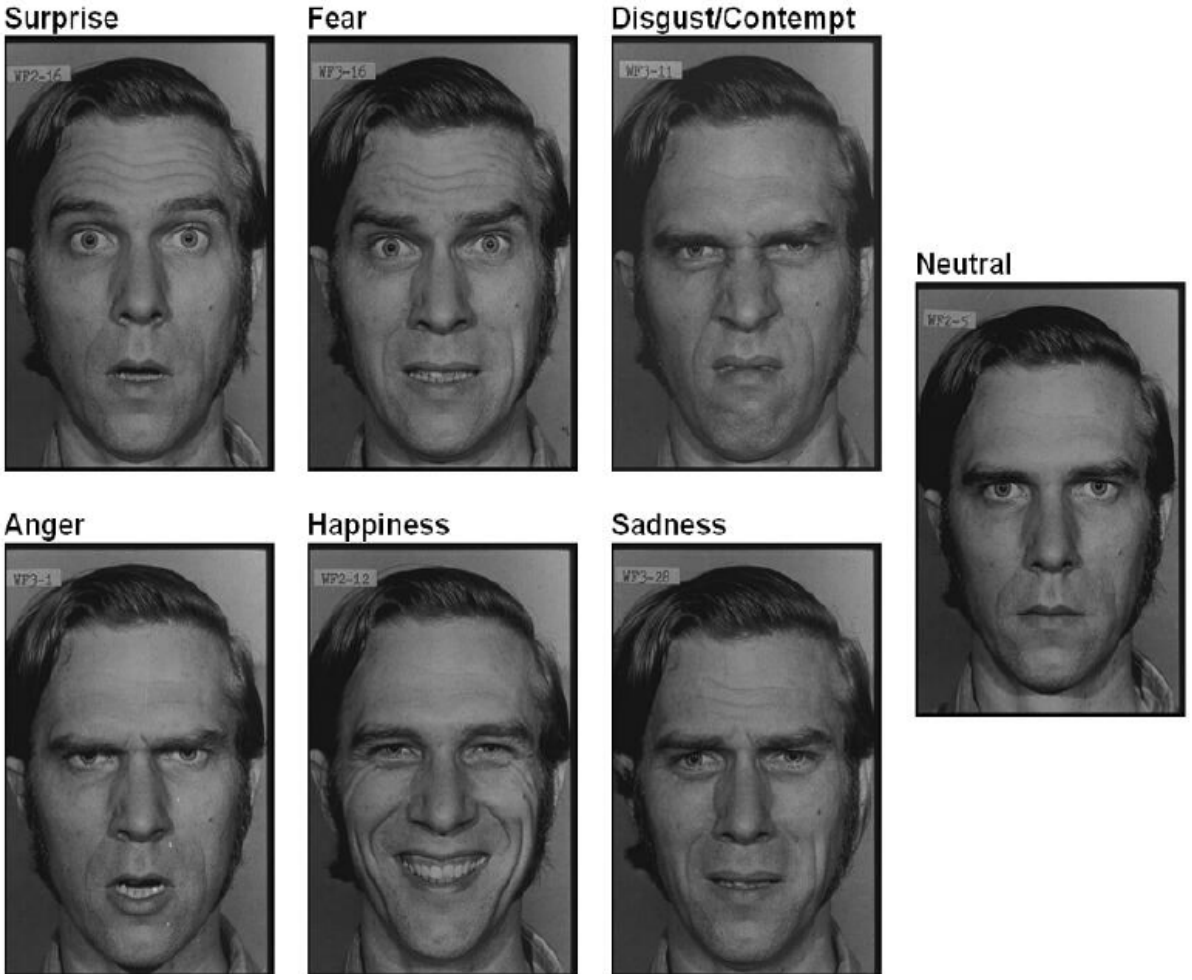
4.3 Some example from general database images :

- **Happy Emotion**

- **Sad Emotion**

- **Disgust Emotion**

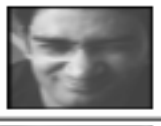
- **Other emotions**



CHAPTER 5

ANALYSIS

- **Happy Analysis**

Input Images	Expected Result	Actual Result
	TRUE	FALSE
	TRUE	TRUE
	TRUE	FALSE
	TRUE	FALSE
	TRUE	TRUE
	TRUE	TRUE
	TRUE	TRUE
	TRUE	TRUE
	TRUE	FALSE
	TRUE	TRUE

This is show the happy emotion Analysis

- **Happy Emotion**

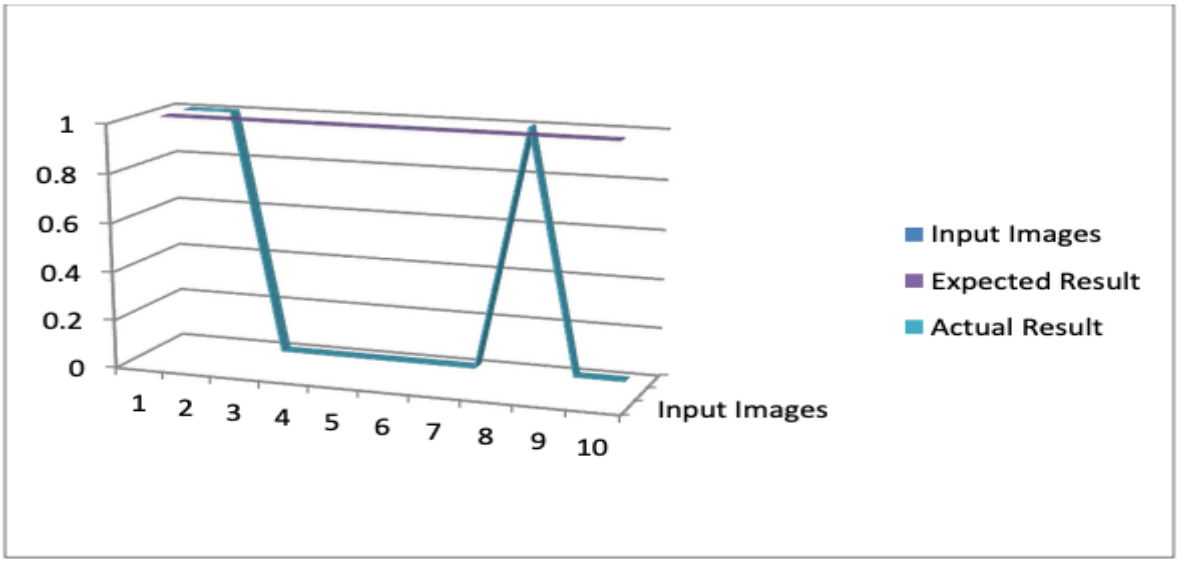


Fig.5.1 Happy Emotion Analysis

- **Happy Facial Recognition**

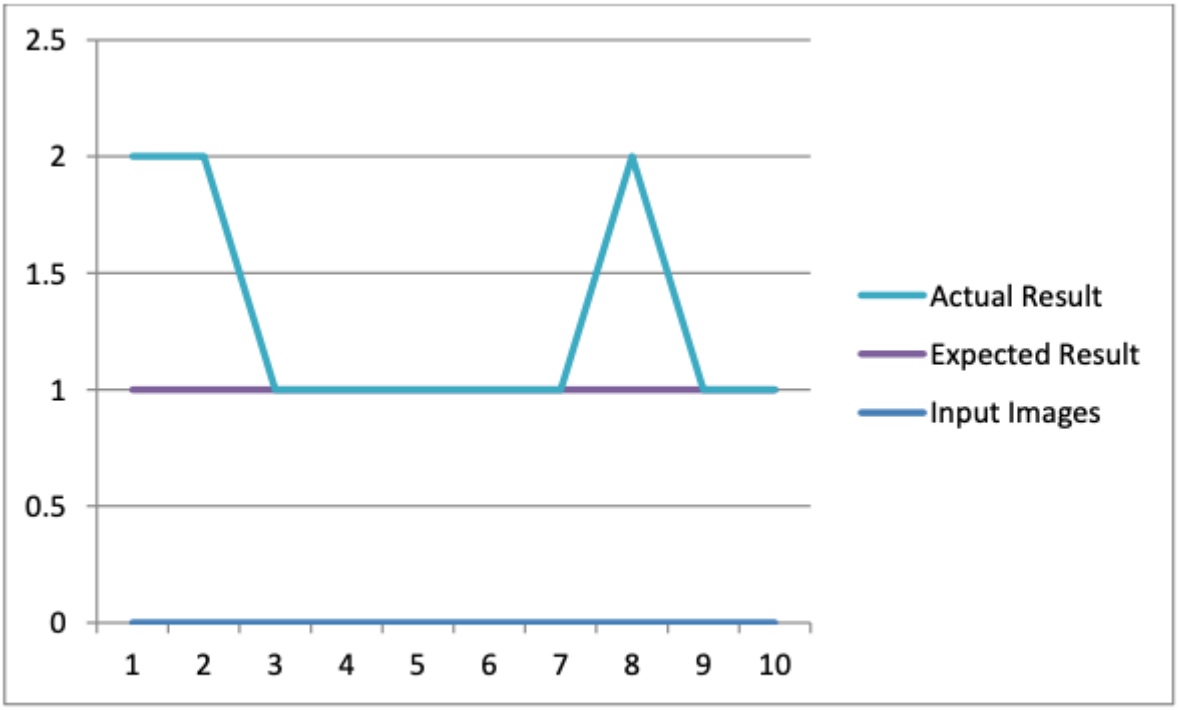


Fig.5.2 Facial Recognition

- **Sad Analysis**

Input Images	Expected Result	Actual Result
	TRUE	TRUE
	TRUE	TRUE
	TRUE	TRUE
	TRUE	TRUE
	TRUE	TRUE
	TRUE	TRUE
	TRUE	FALSE
	TRUE	TRUE
	TRUE	FALSE

Sad Emotion Analysis

- **Sad Emotion**

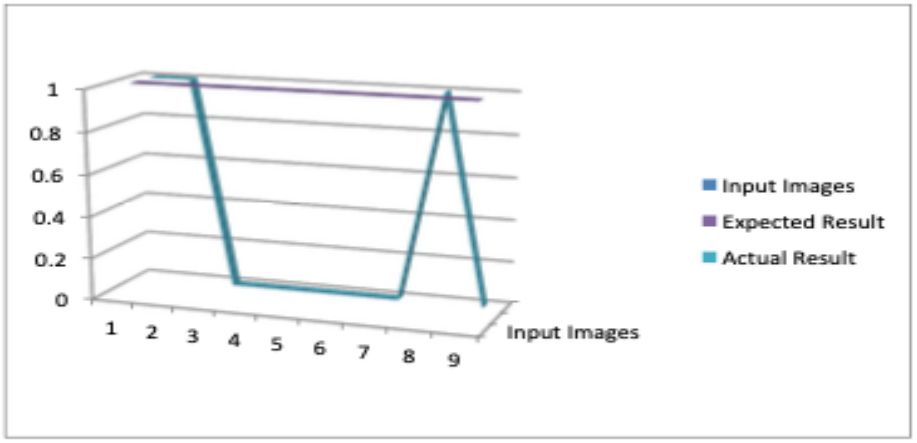


Fig.5.3 Sad Emotion Analysis

- **Sad Facial Recognition**

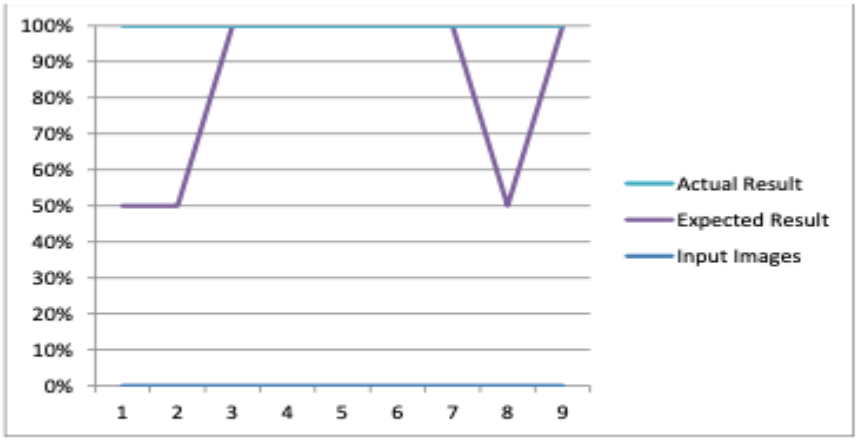


Fig.5.4 Facial Recognition

- **Angry Analysis**

Input Images	Expected Result	Actual Result
	TRUE	TRUE
	TRUE	TRUE
	TRUE	TRUE
	TRUE	TRUE
	TRUE	FALSE
	TRUE	TRUE
	TRUE	FALSE
	TRUE	TRUE
	TRUE	FALSE

Angry Emotion Analysis

- **Angry Emotion**

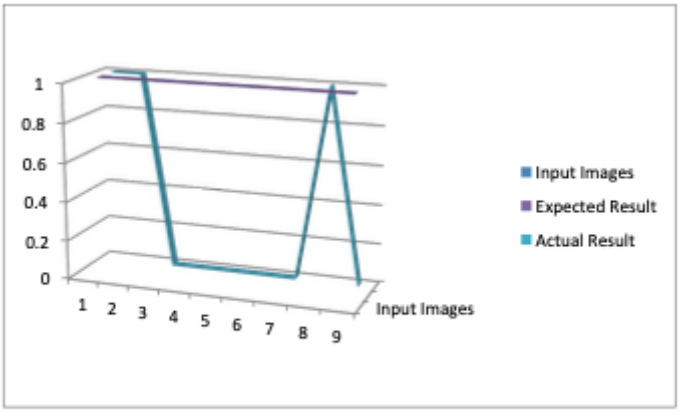


Fig.5.5 Angry Emotion Analysis

- **Angry Facial Emotion**

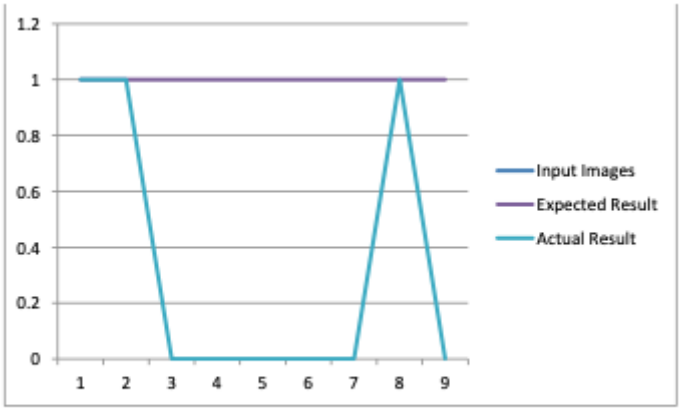


Fig.5.6 Angry Facial Recognition

- **Surprise Analysis**

Input Images	Expected Result	Actual Result
	TRUE	TRUE
	TRUE	TRUE
	TRUE	TRUE
	TRUE	TRUE
	TRUE	FALSE
	TRUE	TRUE
	TRUE	FALSE
	TRUE	TRUE
	TRUE	FALSE

Surprise Emotion Analysis

- **Surprise Emotion**

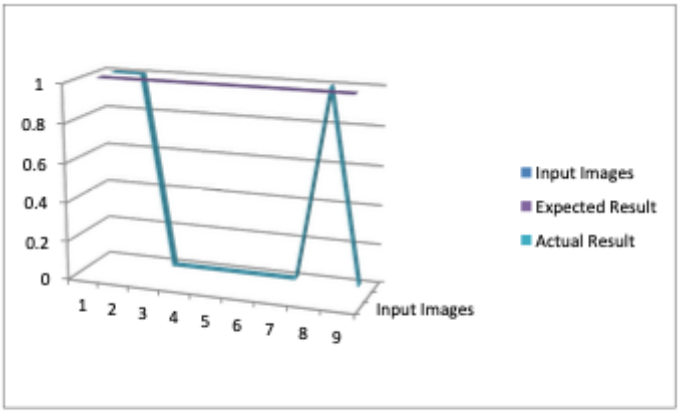


Fig 5.7 Surprise Emotion Analysis

- **Surprise Facial Recognition**

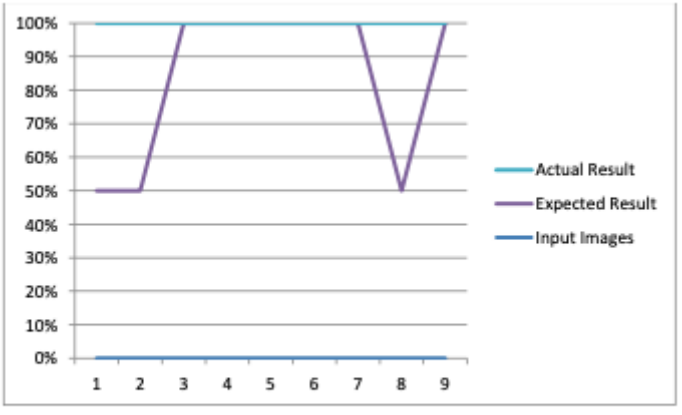


Fig 5.8 Surprise Facial Recognition

- **Disgust Analysis**

Input Images	Expected Result	Actual Result
	TRUE	TRUE
	TRUE	TRUE
	TRUE	FALSE
	TRUE	FALSE
	TRUE	FALSE
	TRUE	FALSE
	TRUE	FALSE
	TRUE	TRUE
	TRUE	FALSE

Disgust Emotion Analysis

- **Disgust Emotion**

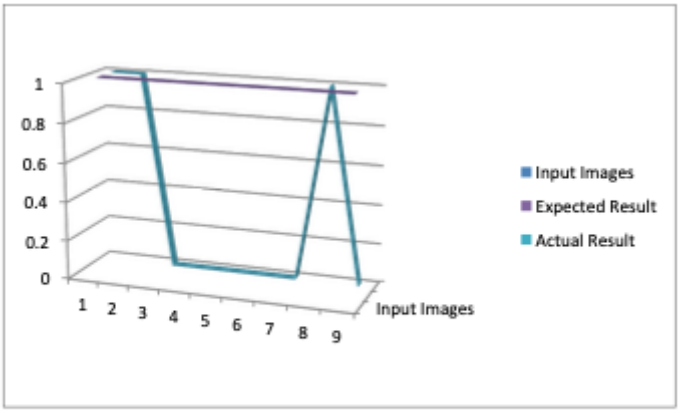


Fig 5.9 Disgust Emotion Analysis

- **Disgust Facial Recognition**

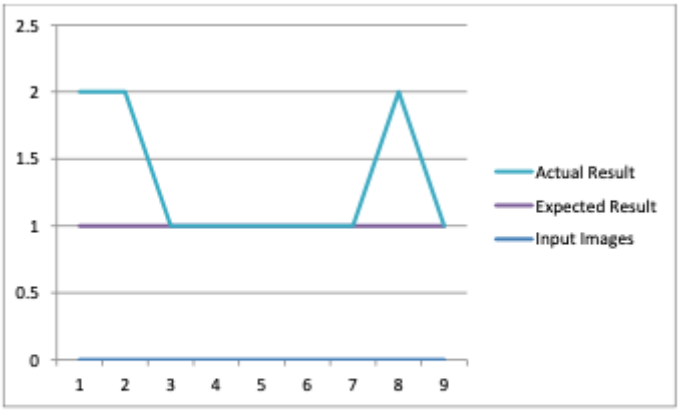


Fig 5.10 Disgust Facial Recognition

CHAPTER 6

TESTING

TESTING

6.1 TESTING PHASE

The basic goal of the software development process is to produce software that has no errors or very few errors. In an effort to detect errors soon after they are introduced, each phase ends with verification activity.

As testing is the last phase before the final product is delivered, it has the enormous responsibility of detecting any type of error that may be there in the software being developed. And to validate that a change has not affected some old functionality of software regression testing is performed.

- Levels of Testing
- Types of Testing
- Validation
- Test Strategy

6.2 Levels of Testing

The basic levels of testing are unit testing, integration testing and system testing and acceptance testing. These different levels of testing attempt to detect different types of faults.

Project need	**Acceptance Testing**
Requirements	**System Testing**
Design	**Integration Testing**
Code	**Unit Testing**

Table 6.2.1 Levels of Testing

6.3 Types of Testing

The purpose of system testing is to identify and correct errors in the system and also to judge if the system meets the requirements of the user or not.

There are various types of system testing:

● **Unit Testing**

This type of testing focuses on the verification of the smallest unit of software design namely the module. Using the procedural description as a guide, important control paths are tested to uncover the errors within the boundary of a module. Unit testing is normally white-box oriented, and the steps can be conducted in parallel for multiple modules. In this project the sub-modules have been individually tested.

Following checks were made:

- Given set of data was taken as input to the module and the output was observed.
- Logic and boundary conditions for input and output were also checked.
- Interfaces between two modules were also checked.
- Runtime Exceptions were thrown which were detected and rectified by taking the Stack Trace.
- Also all the possible situations were anticipated with the help of the user to conduct thorough tests of the system.

● **Integration Testing**

In this type of testing the main aim is to take the unit tested modules and build a program structure which can be directed and dictated by the design.

This includes:

- Top Down Testing
- Bottom-Up Testing
- Regression Testing

The last of the above was particularly important in this project which helped in ensuring changes without introducing unintended behaviour in addition.

● **System Testing:**

This type of testing consists of a series of tests whose primary purpose is to fully exercise the computer based system to verify that system elements have been properly integrated and that they perform their specified functions.

When the individual program modules are working, we combine the modules into single working system. This integration was planned and conducted in such a way that whenever an error occurs we get an idea about the source of it.

The entire system was viewed as a hierarchy of modules. We began with the module at the highest level of design and worked down. Then the next modules to be tested were those that called previously tested modules.

● **Functional Testing**

Once it was certain that, information passed between modules according to the design description, then the system was tested to assure whether the functions describing the requirement specification were performed.

● **Acceptance Testing**

When the functional test completes, the user gets involved to make sure that the system

works according to the user‟s expectation.

6.4 Validation

System validation checks the quality of the software in both simulated and live environments. It has two phases:

● **Alpha Testing:**

In this the software goes through a phase in which errors and failures based on simulated user requirements are verified and studied. The modified software is then subjected to Beta Testing.

● **Beta Testing:**

This is testing the software in the actual user‟s site or a live environment. The system is

Used regularly with live transaction. After a scheduled time, failures and errors are documented and final correction and enhancements are made before the package is released for use.

6.5 Checks & Constraints:

There are many checks and constraints to control input and navigational errors. Input validations are done for the following cases:

- Essential fields
- Character numeric input
- Choosing from certain range of values
- Non-repetition of primary key values
- No-input or wrong input

6.6 Test Strategy

Test Strategy is basically a list of test cases that need to be run on the system.
Given below is the test strategy of our project:

CHAPTER 7

CONCLUSION

Conclusion

We explored the CNN and ResNet50 architectures for recognizing facial emotions using deep learning. The results demonstrated that we were able to achieve acceptable results in comparison to other Kaggle contestants' dataset. We further improved these models by developing an ensemble model to combine the outputs from the two neural networks. Coupled with transfer learning, we achieved 67.2% accuracy on the Kaggle dataset and 78.3% accuracy on the KDEF dataset. For context, the winner of the Kaggle Facial Expression Recognition Challenge achieved an accuracy of 71.2% and the top 10 finalists achieved accuracies of at least 60%.

The facial expression recognition system presented in this research work contributes a resilient face recognition model based on the mapping of behavioural characteristics with the physiological biometric characteristics. The physiological characteristics of the human face with relevance to various expressions such as happiness, sadness, fear, anger, surprise and disgust are associated with geometrical structures which restored as base matching templates for the recognition system.

CHAPTER 8

REFERENCE

REFERENCE

[1] *Albert Mehrabian. Silent Messages, University of California Los Angeles, 1971.*

[2] *P. Ekman and W. V. Friesen. Emotional facial action coding system. Unpublished manuscript, University of California at San Francisco, 1983.*

[3] *Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, Lior Wolf. DeepFace: Closing the Gap to Human-Level Performance in Face Verification. The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2014, pp. 1701-1708.*

[4] *Very deep convolutional networks for large-scale image recognition. Visual Geometry Group, Department of Engineering Science, University of Oxford, 2015.*

[5] *Ruiz-Garcia A., Elshaw M., Altahhan A., Palade V. (2016) Deep Learning for Emotion Recognition in Faces. In: Villa A., Masulli P., Pons Rivero A. (eds) Artificial Neural Networks and Machine Learning – ICANN 2016. ICANN 2016. Lecture Notes in Computer Science, vol 9887. Springer, Cham.*

[6] *Hiranmayi Ranganathan, Shayok Chakraborty, Sethuraman Panchanathan, "Transfer of multimodal emotion features in deep belief networks", Signals Systems and Computers 2016 50th Asilomar Conference on, pp. 449-453, 2016.*

[7] *Liu W., Zheng WL., Lu BL. (2016) Emotion Recognition Using Multimodal Deep Learning. In: Hirose A., Ozawa S., Doya K., Ikeda K., Lee M., Liu D. (eds) Neural Information Processing. ICONIP 2016. Lecture Notes in Computer Science, vol 9948. Springer, Cham.*

[8] *Y. Tian, T. Kanade, and J. Cohn. Recognizing action units for facial expression analysis. IEEE Transactions on Pattern Analysis and Machine Intelligence, 23(2), 2001.*

[9] *M.S. Bartlett, G. Littlewort, M. Frank, C. Lain- scsek, I. Fasel, and J. Movellan. Fully automatic facial action recognition in spontaneous behavior. In Proceedings of the IEEE Conference on Automatic Facial and Gesture Recognition, 2006.*

[10] *M. Pantic and J.M. Rothkrantz. Facial action recognition for facial expression analysis from static face images. IEEE Transactions on Systems, Man and Cybernetics, 34(3), 2004*