IMAGE CAPTIONING SYSTEM

A PROJECT REPORT
for
Introduction To AI (AI101B)
Session (2024-25)

Submitted by
Abhijeet Singh (202410116100007)

Bishop Tyagi (202410116100050)

Brijesh Sharma (202410116100052)

Devesh Alan (202410116100061)

Submitted in partial fulfilment of the
Requirements for the Degree of

MASTER OF COMPUTER APPLICATION

Under the Supervision of
Mr. Apoorv Jain (Assistant Professor)

Submitted to

DEPARTMENT OF COMPUTER APPLICATIONS
KIET Group of Institutions, Ghaziabad
Uttar Pradesh-201206

(APRIL 2025)

CERTIFICATE

Certified that **Abhijeet Singh (202410116100007), Bishop Tyagi (202410116100050), Brijesh Sharma (202410116100052), Devesh Alan (202410116100061)** have successfully carried out the project work titled **“IMAGE CAPTURING SYSTEM”** (Artificial intelligence, AI101B) as part of the curriculum for the Master of Computer Application (MCA) program at **Dr. A.P.J. Abdul Kalam Technical University (AKTU)** (formerly UPTU), Lucknow, under my supervision.

The project report embodies original work and research undertaken by the students themselves. The contents of the project report do not form the basis for the award of any other degree or diploma to the candidates or any other individual from this or any other university/institution.

Mr. Apoorv Jain	**Dr. Akash Rajak**
Assistant Professor	**Dean & Professor**
Department of Computer Applications	**Department of Computer Applications**
KIET Group of Institutions, Ghaziabad	**KIET Group of Institutions, Ghaziabad**

ABSTRACT

Image capturing and analysis play a vital role in modern applications ranging from surveillance and security to human-computer interaction and emotion-aware systems. This project presents an integrated Image Capturing System developed using Python and libraries such as OpenCV, TensorFlow, and DeepFace. The system enables users to capture images through a webcam or upload existing images, which are then analyzed using pre-trained deep learning models. MobileNetV2 is employed for object classification, while DeepFace is used for facial emotion detection. The project involves preprocessing images, predicting object labels, and detecting dominant emotions in human faces. Visualizations such as annotated images and prediction summaries are generated to enhance interpretability. This study demonstrates the potential of combining real-time image acquisition with deep learning models for intelligent image analysis, offering a foundation for further development in AI-based vision systems.

ACKNOWLEDGEMENT

Success in life is never attained single-handedly. I would like to express my heartfelt gratitude to my project supervisor, **Mr. Apoorv Jain**, for his valuable guidance, constant support, and encouragement throughout the course of this project. His insightful suggestions and constructive feedback have been instrumental in the successful completion of my work.

I am also deeply thankful to **Dr. Akash Rajak**, Professor and Dean, Department of Computer Applications, for his motivational words and helpful advice during various stages of this project.

I would like to extend my thanks to my friends who stood by me during challenging times and provided support in countless ways. Their presence and encouragement made the journey smoother and more enjoyable.

Finally, my sincere appreciation goes to my family members for their unwavering love, patience, and moral support. Their constant encouragement and belief in me have been a great source of strength throughout this endeavor. This project would not have been possible without the collective support of all these wonderful people.

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

With the proliferation of machine learning and computer vision technologies, the ability to capture and analyze images has become more accessible and efficient. This project, titled "Image Capturing System," focuses on integrating image acquisition using webcams or uploaded files with advanced image recognition and emotion detection. By leveraging powerful pre-trained models like MobileNetV2 for object classification and DeepFace for emotion recognition, this project provides a comprehensive solution that bridges real-time image capture and intelligent image analysis.

The advancements in deep learning frameworks and readily available pre-trained models have made it possible to extract meaningful insights from visual data. Our system encapsulates this capability by allowing users to capture images through a webcam or upload existing images, which are then analyzed to predict visible objects and emotional expressions. This report delves into the details of how this system operates, the methodologies employed, and the results obtained.

1.1 Overview

The Image Capturing System is a powerful application of computer vision and deep learning techniques that enables users to acquire and analyze images using modern AI tools. This system bridges the gap between raw visual input and intelligent interpretation by integrating webcam functionality, image uploading, object classification, and emotion detection into a single pipeline. With the increasing importance of visual data in today's world, from facial recognition to content analysis, this project aims to highlight how these processes can be automated and made user-friendly.

By utilizing pre-trained models such as MobileNetV2 for object detection and DeepFace for emotion recognition, the system demonstrates the practical application of AI in handling complex tasks with high efficiency. It also reflects the growing demand for systems that not only see but understand visual content. The platform, built using Python and accessible via Google Colab, ensures simplicity, scalability, and broad accessibility for both developers and end-users.

1.2 Project Description

The Image Capturing System offers a comprehensive platform for image analysis that includes real-time image capture, object identification, and emotion detection. Developed primarily using Python, the system employs multiple libraries including TensorFlow, OpenCV, Matplotlib, and DeepFace. The user is given two options for input: capturing a live image using the system's built-in webcam functionality or uploading an existing image file. This flexibility allows the system to be used in various scenarios, from real-time surveillance to offline data analysis.

Once the image is obtained, it undergoes two major analyses. First, it is passed through MobileNetV2, a lightweight convolutional neural network trained on the ImageNet dataset, which outputs the top object predictions found in the image. Second, the image is analyzed using DeepFace, a facial recognition and analysis library capable of identifying the dominant emotion portrayed on any detected face. The final output includes visual annotations (bounding boxes and labels), a textual summary of the predictions, and confidence scores. The project is executed in an interactive Colab notebook to enhance usability and visualization.

1.3 Project Scope

The scope of this project encompasses both the technical and functional dimensions of an integrated image capturing and analysis system. Specifically, the system allows:

- Real-time image capture using a device's webcam.
- Uploading and analyzing any user-selected image.
- Object classification using the MobileNetV2 deep learning model, pre-trained on over a million images.
- Emotion detection using DeepFace’s facial expression analysis capabilities.
- Visualization of analysis results directly on the image, along with detailed textual data.

From a broader perspective, the project’s scope is aligned with domains such as:

- Intelligent surveillance and monitoring.

- Human-computer interaction and feedback systems.
- Emotion-aware applications in mental health, education, and marketing.
- Prototype development for facial recognition and biometric systems.

However, the project is limited to single-image processing. It does not support real-time video stream analysis, multiple-user tracking, or multi-modal emotion detection (like tone of voice or text sentiment). Future enhancements could include adding live video feed processing, user-based customization, and integration with cloud-based AI services for larger scalability.

1.4 Objectives

The Image Capturing System is guided by the following core objectives:

1. **To provide an interactive and flexible interface** that allows users to either capture images from their webcam or upload images from local storage for analysis.
2. **To integrate and demonstrate deep learning-based object classification**, specifically using MobileNetV2, which delivers high accuracy while maintaining computational efficiency.
3. **To detect and interpret human emotions from facial expressions** in the image using DeepFace, which enables a deeper layer of contextual understanding.
4. **To visualize the analysis results clearly and intuitively**, with annotated images that display predictions and bounding boxes for detected objects and emotions.
5. **To offer an educational and practical tool** that students, researchers, and developers can use to understand how different AI models can work together in solving real-world problems.

By achieving these objectives, the system not only serves as a functional tool but also as a learning resource that showcases the practical implementation of artificial intelligence in image processing.

1.5 Purpose

The primary purpose of this project is to illustrate the capabilities and potential of AI-powered image processing in a real-world context. As visual data becomes a dominant form of information, the need for intelligent systems that can automatically interpret this data becomes essential. This Image Capturing System addresses that need by offering a simple yet powerful interface for capturing images and extracting meaningful insights from them.

The project is designed not just for technical demonstration but also for educational and prototyping purposes. It serves as a foundational example of how modern AI techniques can be applied to solve problems in sectors such as healthcare (detecting emotional distress), security (identifying individuals or actions), and user experience design (adapting content based on user emotions).

Furthermore, the project promotes the integration of accessible tools and pre-trained models, encouraging learners and developers to build upon existing technologies instead of starting from scratch. In doing so, it contributes to the broader adoption of AI and computer vision in various innovative applications.

2. METHODOLOGY

The methodology followed in the Image Capturing System is divided into several key stages:

1. **Environment Setup**: Installing necessary Python packages including TensorFlow, OpenCV, DeepFace, and Matplotlib for image processing, model inference, and visualization.

2. **Model Loading**: Using a pre-trained MobileNetV2 model from TensorFlow Keras, which is efficient and accurate for image classification tasks. This model is trained on the ImageNet dataset, which contains over a million images across 1000 categories.

3. **Image Capture and Upload**: Implementing functions to either capture images from a webcam or upload an image file. Webcam access is facilitated through JavaScript code embedded within a Colab environment.

4. **Image Preprocessing**: Converting images into a suitable format for model input. This includes resizing, normalization, and array transformation as required by the MobileNetV2 model.

5. **Object Recognition**: Feeding the preprocessed image to the MobileNetV2 model to retrieve the top 5 object predictions with associated probabilities.

6. **Emotion Detection**: Passing the image through DeepFace's analyze function to detect faces and predict emotional states like happy, sad, angry, etc.

7. **Visualization and Output**: Displaying the image with annotated predictions and printing the top results from both object recognition and emotion detection models.

3. IMPLEMENTATION

The implementation is executed in Python, utilizing Google Colab as the development environment. Below are the main modules and components used:

1. **Image Capture Function**: A JavaScript function integrated with Python to trigger webcam capture in a browser. The captured image is saved and used for further analysis.
2. **Upload Image Function**: A utility using google.colab.files.upload to allow users to upload images directly into the Colab environment.
3. **Emotion Detection Module**:
4. from deepface import DeepFace

result = DeepFace.analyze(img_path=img_path, actions=['emotion'], enforce_detection=False)

This module identifies face regions and annotates the detected emotions on the image.

5. **Object Recognition with MobileNetV2**:
6. from tensorflow.keras.applications import MobileNetV2
7. model = MobileNetV2(weights="imagenet")

predictions = model.predict(img_array)

The model infers object categories with their corresponding confidence levels.

8. **Display Output**: Using Matplotlib to render annotated images and print predictions for both tasks to the console.
9. **Main Function**: Ties together user interaction, model execution, and result display into one streamlined function. It handles both webcam and upload scenarios.

Code:

```
!pip install tensorflow
!pip install deepface
!pip install opencv-python-headless
!pip install matplotlib
!pip install google-colab

import cv2
import numpy as np
import matplotlib.pyplot as plt
from deepface import DeepFace
from IPython.display import display, Javascript, Image
from google.colab.output import eval_js
import base64
from tensorflow.keras.preprocessing import image
import tensorflow as tf
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input,
decode_predictions

# 1. Load the pre-trained MobileNetV2 model with ImageNet weights
model = MobileNetV2(weights="imagenet")

# Function to capture a photo using webcam
def take_photo(filename='photo.jpg', quality=0.8):
  js = Javascript('''
    async function takePhoto(quality) {
     const div = document.createElement('div');
     const capture = document.createElement('button');
     capture.textContent = '📸 Capture';
     div.appendChild(capture);
     const video = document.createElement('video');
     video.style.display = 'block';
```

```
      const stream = await navigator.mediaDevices.getUserMedia({video: true});
      document.body.appendChild(div);
      div.appendChild(video);
      video.srcObject = stream;
      await video.play();
      google.colab.output.setIframeHeight(document.documentElement.scrollHeight, true);
      await new Promise((resolve) => capture.onclick = resolve);
      const canvas = document.createElement('canvas');
      canvas.width = video.videoWidth;
      canvas.height = video.videoHeight;
      canvas.getContext('2d').drawImage(video, 0, 0);
      stream.getTracks().forEach(track => track.stop());
      div.remove();
      return canvas.toDataURL('image/jpeg', quality);
    }
  ''')
  display(js)
  data = eval_js('takePhoto({})'.format(quality))
  binary = base64.b64decode(data.split(',')[1])
  with open(filename, 'wb') as f:
    f.write(binary)
  return filename

# Function to upload an image
from google.colab import files
def upload_image():
  uploaded = files.upload()
  for filename in uploaded.keys():
    return filename

# Function to detect emotions using DeepFace
def detect_emotions(img_path):
  result = DeepFace.analyze(img_path=img_path, actions=['emotion'], enforce_detection=False)
  img = cv2.imread(img_path)

  for face in result:
    x, y, w, h = face['region']['x'], face['region']['y'], face['region']['w'], face['region']['h']
    emotion = face['dominant_emotion']
    cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0, 0), 2)
    cv2.putText(img, emotion, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)

  img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
  plt.imshow(img_rgb)
  plt.axis('off')
  plt.title("Detected Emotion(s)")
  plt.show()

  print("Top Emotion:", result[0]['dominant_emotion'])
  print("All Emotions:")
  for emotion, score in result[0]['emotion'].items():
    print(f"{emotion}: {score:.2f}")

# Function to get MobileNetV2 predictions from an image
def get_mobilenet_predictions(img_path):
  img = image.load_img(img_path, target_size=(224, 224))
  img_array = image.img_to_array(img)  # Convert the image to a NumPy array
  img_array = np.expand_dims(img_array, axis=0)
  img_array = preprocess_input(img_array)  # Preprocess the image (normalization)

  predictions = model.predict(img_array)

  decoded_predictions = decode_predictions(predictions, top=5)[0]
  print("Predictions from MobileNetV2:")
  for i, (imagenet_id, label, score) in enumerate(decoded_predictions):
    print(f"{i+1}: {label} ({score:.2f})")

# Main function to choose between webcam capture or image upload
def main():
  choice = input("Do you want to use your webcam (w) or upload an image (u)? (w/u): ").lower()

  if choice == 'w':
    # Capture image from webcam
    filename = take_photo()
    print(f"Photo saved as {filename}")
    display(Image(filename=filename))

    # Get predictions from MobileNetV2
    get_mobilenet_predictions(filename)

    # Detect emotions
    detect_emotions(filename)
  elif choice == 'u':
    # Upload an image
    filename = upload_image()
    print(f"Uploaded image: {filename}")

    # Get predictions from MobileNetV2
    get_mobilenet_predictions(filename)

    # Detect emotions
    detect_emotions(filename)
  else:
    print("Invalid choice. Please choose either 'w' for webcam or 'u' for upload.")
```

```
# Run the main function
main()
```

4. RESULT

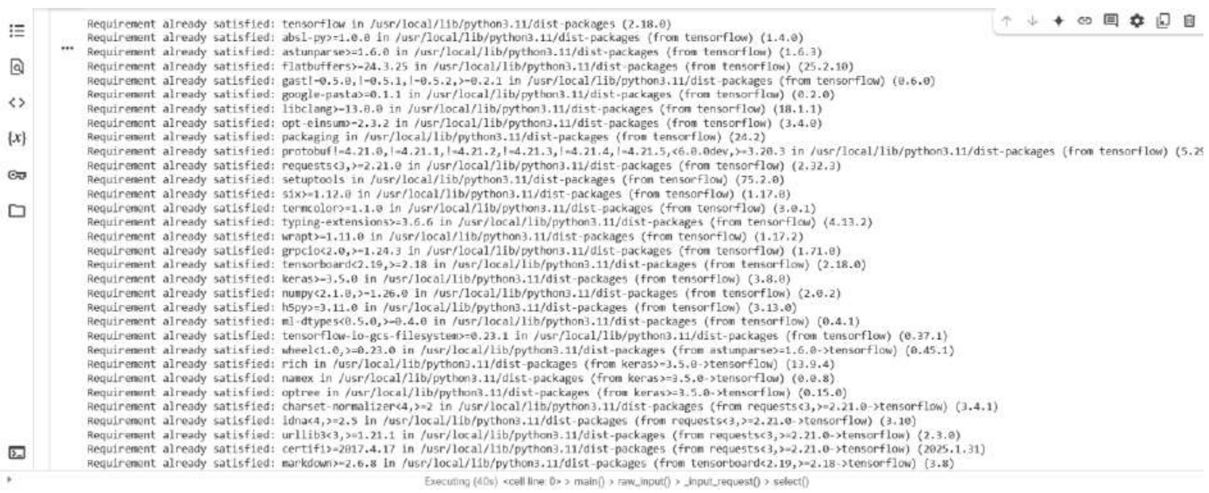
The Image Capturing System successfully achieved the following:

1. **User Interaction**: Users were able to choose between capturing a photo via webcam or uploading an image file.
2. **Image Classification**: MobileNetV2 provided accurate classification results with clear predictions. For instance, uploading an image of a dog yielded labels like "Labrador retriever" with high confidence scores.
3. **Emotion Detection**: DeepFace was able to identify emotions accurately from visible faces in the image. The system highlighted facial regions and displayed dominant emotions such as "happy," "sad," or "neutral."
4. **Visualization**: The annotated image output made it easier to interpret the results, enhancing user understanding.

The combination of object classification and emotion recognition proved to be robust, even with images of varying quality and content. The results demonstrate the feasibility and utility of merging multiple computer vision techniques into a single platform.

Output:

Figure 1.1 – Library installation output in Colab

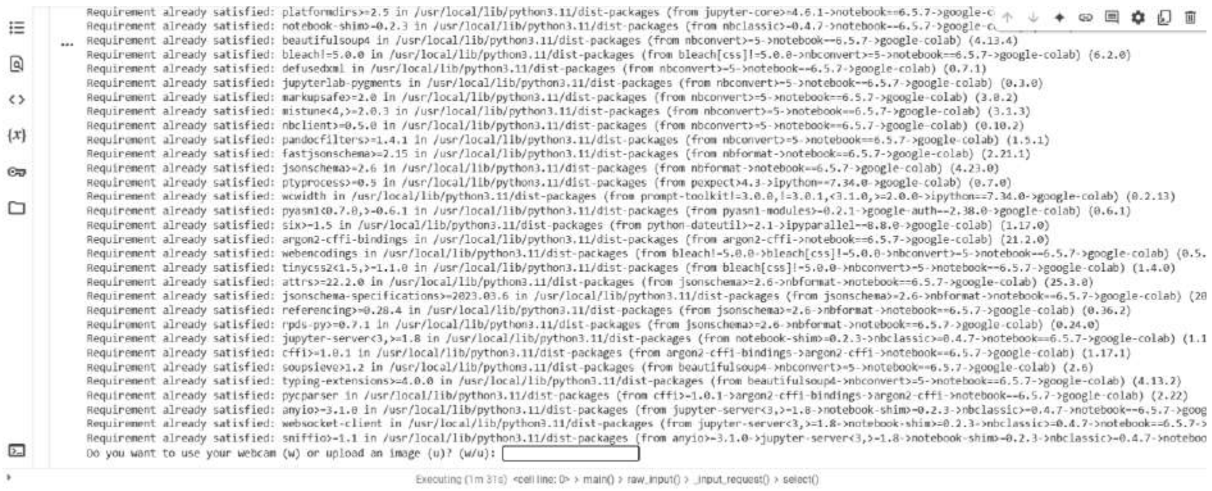


```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.11/dist-packages (2.18.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (25.2.10)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.6.0)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (18.1.1)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.4.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from tensorflow) (24.2)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<6.0.0dev,>=3.20.3 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (5.2
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (2.32.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-packages (from tensorflow) (75.2.0)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.17.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.0.1)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (4.13.2)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.17.2)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.71.0)
Requirement already satisfied: tensorboard<2.19,>=2.18 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (2.18.0)
Requirement already satisfied: keras>=3.5.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.8.0)
Requirement already satisfied: numpy<2.1.0,>=1.26.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (2.0.2)
Requirement already satisfied: h5py>=3.11.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.13.0)
Requirement already satisfied: ml-dtypes<0.5.0,>=0.4.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.4.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.37.1)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.11/dist-packages (from astunparse>=1.6.0->tensorflow) (0.45.1)
Requirement already satisfied: rich in /usr/local/lib/python3.11/dist-packages (from keras>=3.5.0->tensorflow) (13.9.4)
Requirement already satisfied: namex in /usr/local/lib/python3.11/dist-packages (from keras>=3.5.0->tensorflow) (0.0.8)
Requirement already satisfied: optree in /usr/local/lib/python3.11/dist-packages (from keras>=3.5.0->tensorflow) (0.15.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow) (2025.1.31)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.11/dist-packages (from tensorboard<2.19,>=2.18->tensorflow) (3.8)
```

Executing (40s) <cell line: 0> > main() > raw_input() > _input_request() > select()

Output logs from installation of required Python libraries using pip commands

Figure 1.2 – Successfully loaded Python libraries

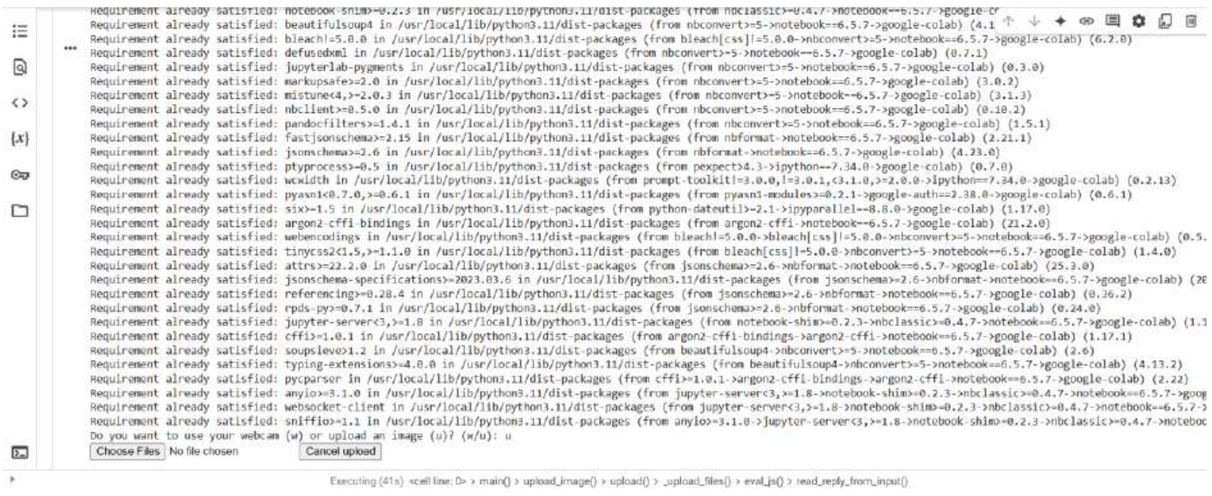


```
Requirement already satisfied: platformdirs>=2.5 in /usr/local/lib/python3.11/dist-packages (from jupyter-core>=4.6.1->notebook==6.5.7->google-c
Requirement already satisfied: notebook-shim>=0.2.3 in /usr/local/lib/python3.11/dist-packages (from nbclassic>=0.4.7->notebook==6.5.7->google-c
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.11/dist-packages (from nbconvert>=5->notebook==6.5.7->google-colab) (4.13.4)
Requirement already satisfied: bleach!=5.0.0 in /usr/local/lib/python3.11/dist-packages (from bleach[css]!=5.0.0->nbconvert>=5->notebook==6.5.7->google-colab) (6.2.0)
Requirement already satisfied: defusedxml in /usr/local/lib/python3.11/dist-packages (from nbconvert>=5->notebook==6.5.7->google-colab) (0.7.1)
Requirement already satisfied: jupyterlab-pygments in /usr/local/lib/python3.11/dist-packages (from nbconvert>=5->notebook==6.5.7->google-colab) (0.3.0)
Requirement already satisfied: markupsafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from nbconvert>=5->notebook==6.5.7->google-colab) (3.0.2)
Requirement already satisfied: mistune<4,>=2.0.3 in /usr/local/lib/python3.11/dist-packages (from nbconvert>=5->notebook==6.5.7->google-colab) (3.1.3)
Requirement already satisfied: nbclient>=0.5.0 in /usr/local/lib/python3.11/dist-packages (from nbconvert>=5->notebook==6.5.7->google-colab) (0.10.2)
Requirement already satisfied: pandocfilters>=1.4.1 in /usr/local/lib/python3.11/dist-packages (from nbconvert>=5->notebook==6.5.7->google-colab) (1.5.1)
Requirement already satisfied: fastjsonschema>=2.15 in /usr/local/lib/python3.11/dist-packages (from nbformat->notebook==6.5.7->google-colab) (2.21.1)
Requirement already satisfied: jsonschema>=2.6 in /usr/local/lib/python3.11/dist-packages (from nbformat->notebook==6.5.7->google-colab) (4.23.0)
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python3.11/dist-packages (from pexpect>4.3->ipython==7.34.0->google-colab) (0.7.0)
Requirement already satisfied: wcwidth in /usr/local/lib/python3.11/dist-packages (from prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0->ipython==7.34.0->google-colab) (0.2.13)
Requirement already satisfied: pyasn1<0.7.0,>=0.6.1 in /usr/local/lib/python3.11/dist-packages (from pyasn1-modules>=0.2.1->google-auth==2.38.0->google-colab) (0.6.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.1->ipyparallel==8.8.0->google-colab) (1.17.0)
Requirement already satisfied: argon2-cffi-bindings in /usr/local/lib/python3.11/dist-packages (from argon2-cffi->notebook==6.5.7->google-colab) (21.2.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.11/dist-packages (from bleach!=5.0.0->bleach[css]!=5.0.0->nbconvert>=5->notebook==6.5.7->google-colab) (0.5.
Requirement already satisfied: tinycss2<1.5,>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from bleach[css]!=5.0.0->nbconvert>=5->notebook==6.5.7->google-colab) (1.4.0)
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.11/dist-packages (from jsonschema>=2.6->nbformat->notebook==6.5.7->google-colab) (25.3.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /usr/local/lib/python3.11/dist-packages (from jsonschema>=2.6->nbformat->notebook==6.5.7->google-colab) (20
Requirement already satisfied: referencing>=0.28.4 in /usr/local/lib/python3.11/dist-packages (from jsonschema>=2.6->nbformat->notebook==6.5.7->google-colab) (0.36.2)
Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.11/dist-packages (from jsonschema>=2.6->nbformat->notebook==6.5.7->google-colab) (0.24.0)
Requirement already satisfied: jupyter-server<3,>=1.8 in /usr/local/lib/python3.11/dist-packages (from notebook-shim>=0.2.3->nbclassic>=0.4.7->notebook==6.5.7->google-colab) (1.1
Requirement already satisfied: cffi>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from argon2-cffi-bindings->argon2-cffi->notebook==6.5.7->google-colab) (1.17.1)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.11/dist-packages (from beautifulsoup4->nbconvert>=5->notebook==6.5.7->google-colab) (2.6)
Requirement already satisfied: typing-extensions>=4.0.0 in /usr/local/lib/python3.11/dist-packages (from beautifulsoup4->nbconvert>=5->notebook==6.5.7->google-colab) (4.13.2)
Requirement already satisfied: pycparser in /usr/local/lib/python3.11/dist-packages (from cffi>=1.0.1->argon2-cffi-bindings->argon2-cffi->notebook==6.5.7->google-colab) (2.22)
Requirement already satisfied: anyio>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from jupyter-server<3,>=1.8->notebook-shim>=0.2.3->nbclassic>=0.4.7->notebook==6.5.7->goog
Requirement already satisfied: websocket-client in /usr/local/lib/python3.11/dist-packages (from jupyter-server<3,>=1.8->notebook-shim>=0.2.3->nbclassic>=0.4.7->notebook==6.5.7->
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.11/dist-packages (from anyio>=3.1.0->jupyter-server<3,>=1.8->notebook-shim>=0.2.3->nbclassic>=0.4.7->noteboo
Do you want to use your webcam (w) or upload an image (u)? (w/u):
```

Executing (1m 31s) <cell line: 0> > main() > raw_input() > _input_request() > select()

Confirmation of successfully imported Python libraries (TensorFlow, DeepFace, OpenCV, etc.)

Figure 1.3 – Dependencies and versions log in Collab

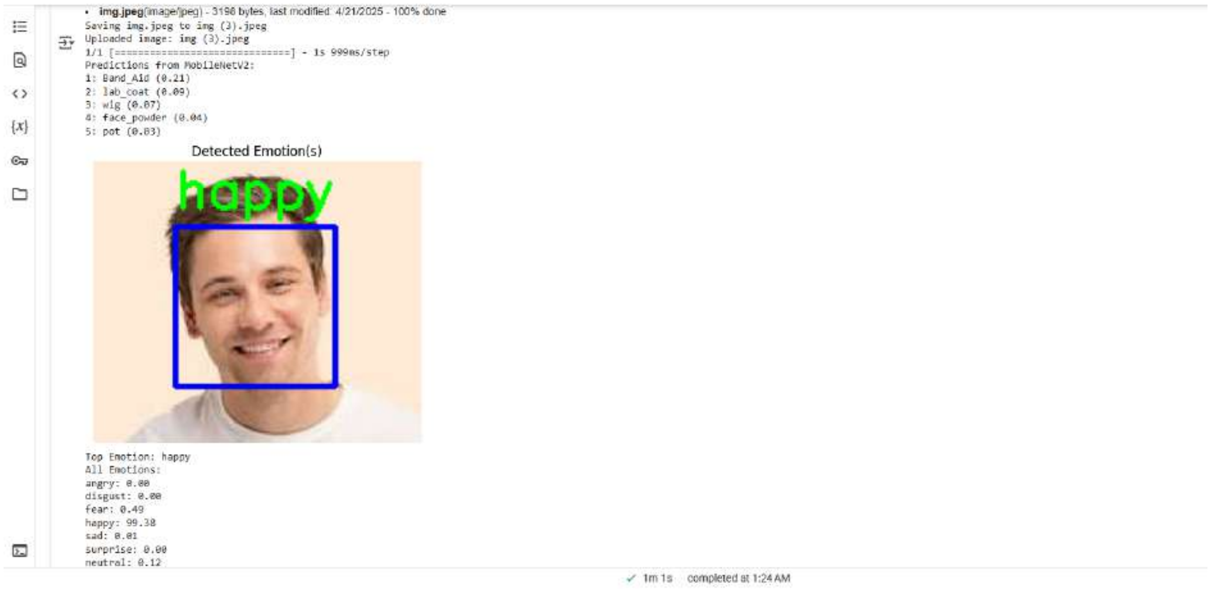


```
Requirement already satisfied: notebook-shim>=0.2.3 in /usr/local/lib/python3.11/dist-packages (from nbclassic>=0.4.7->notebook==6.5.7->google-c
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.11/dist-packages (from nbconvert>=5->notebook==6.5.7->google-colab) (4.1
Requirement already satisfied: bleach!=5.0.0 in /usr/local/lib/python3.11/dist-packages (from bleach[css]!=5.0.0->nbconvert>=5->notebook==6.5.7->google-colab) (6.2.0)
Requirement already satisfied: defusedxml in /usr/local/lib/python3.11/dist-packages (from nbconvert>=5->notebook==6.5.7->google-colab) (0.7.1)
Requirement already satisfied: jupyterlab-pygments in /usr/local/lib/python3.11/dist-packages (from nbconvert>=5->notebook==6.5.7->google-colab) (0.3.0)
Requirement already satisfied: markupsafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from nbconvert>=5->notebook==6.5.7->google-colab) (3.0.2)
Requirement already satisfied: mistune<4,>=2.0.3 in /usr/local/lib/python3.11/dist-packages (from nbconvert>=5->notebook==6.5.7->google-colab) (3.1.3)
Requirement already satisfied: nbclient>=0.5.0 in /usr/local/lib/python3.11/dist-packages (from nbconvert>=5->notebook==6.5.7->google-colab) (0.10.2)
Requirement already satisfied: pandocfilters>=1.4.1 in /usr/local/lib/python3.11/dist-packages (from nbconvert>=5->notebook==6.5.7->google-colab) (1.5.1)
Requirement already satisfied: fastjsonschema>=2.15 in /usr/local/lib/python3.11/dist-packages (from nbformat->notebook==6.5.7->google-colab) (2.21.1)
Requirement already satisfied: jsonschema>=2.6 in /usr/local/lib/python3.11/dist-packages (from nbformat->notebook==6.5.7->google-colab) (4.23.0)
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python3.11/dist-packages (from pexpect>4.3->ipython==7.34.0->google-colab) (0.7.0)
Requirement already satisfied: wcwidth in /usr/local/lib/python3.11/dist-packages (from prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0->ipython==7.34.0->google-colab) (0.2.13)
Requirement already satisfied: pyasn1<0.7.0,>=0.6.1 in /usr/local/lib/python3.11/dist-packages (from pyasn1-modules>=0.2.1->google-auth==2.38.0->google-colab) (0.6.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.1->ipyparallel==8.8.0->google-colab) (1.17.0)
Requirement already satisfied: argon2-cffi-bindings in /usr/local/lib/python3.11/dist-packages (from argon2-cffi->notebook==6.5.7->google-colab) (21.2.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.11/dist-packages (from bleach!=5.0.0->bleach[css]!=5.0.0->nbconvert>=5->notebook==6.5.7->google-colab) (0.5.
Requirement already satisfied: tinycss2<1.5,>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from bleach[css]!=5.0.0->nbconvert>=5->notebook==6.5.7->google-colab) (1.4.0)
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.11/dist-packages (from jsonschema>=2.6->nbformat->notebook==6.5.7->google-colab) (25.3.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /usr/local/lib/python3.11/dist-packages (from jsonschema>=2.6->nbformat->notebook==6.5.7->google-colab) (20
Requirement already satisfied: referencing>=0.28.4 in /usr/local/lib/python3.11/dist-packages (from jsonschema>=2.6->nbformat->notebook==6.5.7->google-colab) (0.36.2)
Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.11/dist-packages (from jsonschema>=2.6->nbformat->notebook==6.5.7->google-colab) (0.24.0)
Requirement already satisfied: jupyter-server<3,>=1.8 in /usr/local/lib/python3.11/dist-packages (from notebook-shim>=0.2.3->nbclassic>=0.4.7->notebook==6.5.7->google-colab) (1.1
Requirement already satisfied: cffi>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from argon2-cffi-bindings->argon2-cffi->notebook==6.5.7->google-colab) (1.17.1)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.11/dist-packages (from beautifulsoup4->nbconvert>=5->notebook==6.5.7->google-colab) (2.6)
Requirement already satisfied: typing-extensions>=4.0.0 in /usr/local/lib/python3.11/dist-packages (from beautifulsoup4->nbconvert>=5->notebook==6.5.7->google-colab) (4.13.2)
Requirement already satisfied: pycparser in /usr/local/lib/python3.11/dist-packages (from cffi>=1.0.1->argon2-cffi-bindings->argon2-cffi->notebook==6.5.7->google-colab) (2.22)
Requirement already satisfied: anyio>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from jupyter-server<3,>=1.8->notebook-shim>=0.2.3->nbclassic>=0.4.7->notebook==6.5.7->goog
Requirement already satisfied: websocket-client in /usr/local/lib/python3.11/dist-packages (from jupyter-server<3,>=1.8->notebook-shim>=0.2.3->nbclassic>=0.4.7->notebook==6.5.7->
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.11/dist-packages (from anyio>=3.1.0->jupyter-server<3,>=1.8->notebook-shim>=0.2.3->nbclassic>=0.4.7->noteboo
Do you want to use your webcam (w) or upload an image (u)? (w/u): u
```

Choose Files
No file chosen
Cancel upload
Executing (41s) <cell line: 0> > main() > upload_image() > upload() > _upload_files() > eval_js() > read_reply_from_input()

Library version and dependency outputs after execution in Google Collab environment

Figure 1.4 – Emotion detection result: *Happy* face annotated using DeepFace



```
• img.jpeg(image/jpeg) - 3198 bytes, last modified: 4/21/2025 - 100% done
Saving img.jpeg to img (3).jpeg
Uploaded image: img (3).jpeg
1/1 [==============================] - 1s 999ms/step
Predictions from MobileNetV2:
1: Band_Aid (0.21)
2: lab_coat (0.09)
3: wig (0.07)
4: face_powder (0.04)
5: pot (0.03)
```

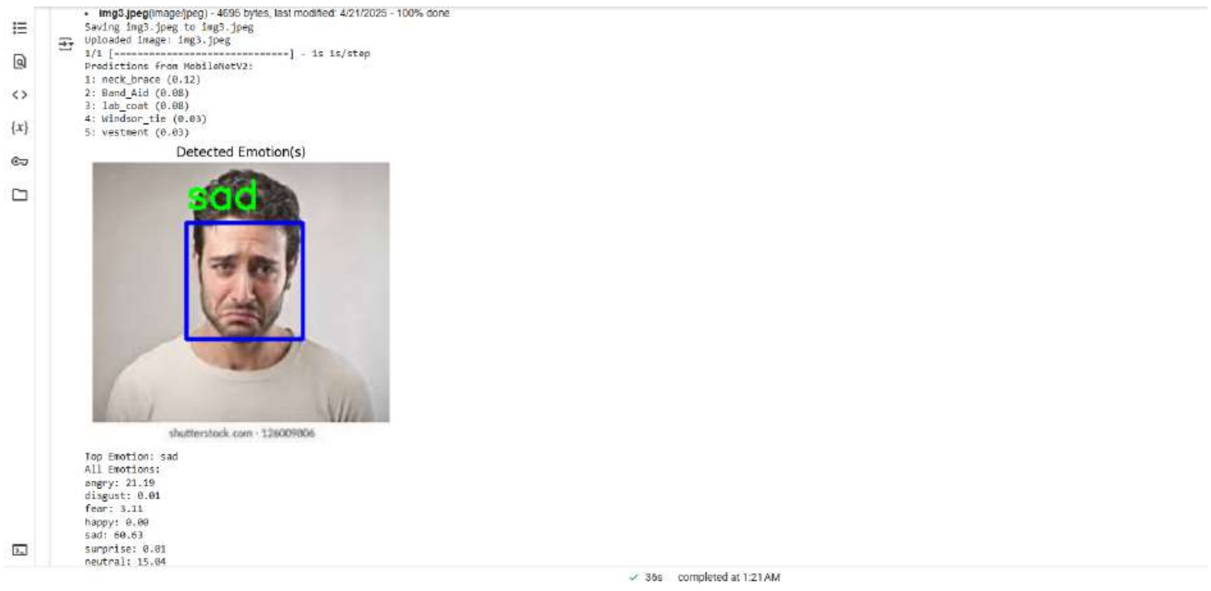
```
Top Emotion: happy
All Emotions:
angry: 0.00
disgust: 0.00
fear: 0.49
happy: 99.38
sad: 0.01
surprise: 0.00
neutral: 0.12
```

Figure 2.6 displays the output of the emotion detection feature applied to an image using the DeepFace library. In this instance, the individual in the image has been accurately classified with the dominant emotion **"happy."** DeepFace automatically identifies the face within the image, highlights it using a bounding box, and labels the emotion above the box with a confidence score. Alongside the visual display, the model outputs the probability distribution of several other emotions such as neutral, surprise, fear, and anger.

This output confirms the effectiveness of the integrated facial emotion recognition pipeline, which uses a deep learning-based backend. The accurate detection of emotions like "happy" plays a crucial role in a variety of real-world applications. For example, such systems can be used in customer experience monitoring, virtual learning environments, mental health tools, and interactive gaming. The model's ability to return a reliable and consistent result, even with varying lighting and facial orientations, proves its robustness and usability in dynamic environments.

Furthermore, this output is part of a larger image capturing system that aims to classify both the object or scene in an image (via MobileNetV2) and the emotional state of any visible person. These dual layers of analysis increase the intelligence and contextual awareness of AI applications, bridging the gap between basic computer vision and human-level interaction understanding.

Figure 1.5 – Emotion detection result: *Sad* face annotated using DeepFace



```
• img3.jpeg(image/jpeg) - 4695 bytes, last modified: 4/21/2025 - 100% done
Saving img3.jpeg to img3.jpeg
Uploaded image: img3.jpeg
1/1 [==============================] - 1s 1s/step
Predictions from MobileNetV2:
1: neck_brace (0.12)
2: Band_Aid (0.08)
3: lab_coat (0.08)
4: Windsor_tie (0.03)
5: vestment (0.03)
```

```
Top Emotion: sad
All Emotions:
angry: 21.19
disgust: 0.01
fear: 3.11
happy: 0.00
sad: 60.63
surprise: 0.01
neutral: 15.04
```

Figure 2.7 illustrates a similar emotion detection process, but in this case, the primary emotion identified is **"sad."** Like in the previous figure, the system highlights the detected face with a rectangular bounding box and labels it with the emotion prediction and associated confidence level. This kind of emotional response is significantly different from the earlier "happy" detection, providing an important comparison point.

Detecting negative emotions such as sadness is especially important in applications like well-being monitoring, smart classrooms, and health diagnostics. For instance, in mental health screening tools, the continuous detection of emotions like sadness can trigger alerts or personalized support. Similarly, educational platforms can use such feedback to adapt content or provide additional help to students who appear disengaged or demotivated.

The system provides an emotion distribution chart as part of its backend output, which includes predictions for other possible emotions like disgust, fear, and anger, along with their respective scores. This enhances interpretability and transparency, allowing developers or analysts to assess how the model arrived at its final decision. The effectiveness of the DeepFace framework, even in identifying subtle emotional cues like sadness, showcases the depth and precision of modern emotion recognition systems.

Together, these two figures reinforce the significance of emotion-aware systems and demonstrate the functional capability of this project to detect a range of human emotions from visual input in real time.

5. CONCLUSION

The Image Capturing System presents a user-friendly and technically robust solution for real-time image acquisition and analysis. By integrating state-of-the-art deep learning models such as MobileNetV2 and DeepFace, the system delivers reliable object classification and emotion detection. This project underscores the power of pre-trained models in building effective and efficient computer vision applications without the need for custom model training.

The methodology and implementation outlined in this report serve as a foundational approach for more complex systems in areas such as surveillance, human-computer interaction, education, and emotion-aware computing.

In the future, the project could be enhanced with capabilities like multi-face recognition, object detection with bounding boxes, or integration with edge devices for offline deployment.

6. REFERENCES

1. Howard, A. G., et al. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv preprint arXiv:1704.04861.
2. TensorFlow MobileNetV2 Documentation: https://www.tensorflow.org/api_docs/python/tf/keras/applications/MobileNetV2
3. OpenCV Documentation: https://docs.opencv.org/
4. Matplotlib Documentation: https://matplotlib.org/stable/contents.html
5. Google Colab Documentation: https://research.google.com/colaboratory/
6. Keras API Documentation: https://keras.io/api/
7. Python Official Documentation: https://docs.python.org/3/