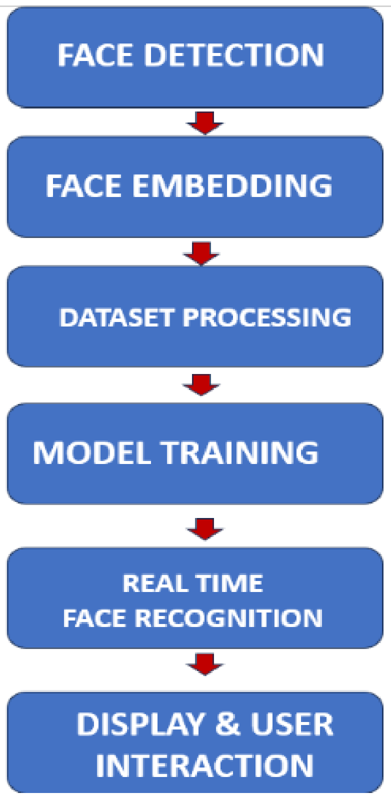
INTRODUCTION

Face recognition:

In the rapidly advancing landscape of technology, face recognition has become an indispensable component, influencing various aspects of our daily lives, from unlocking smartphones to bolstering security measures. Our primary objective is to develop a robust system capable of accurately identifying individuals across diverse scenarios.

Leveraging deep learning models, particularly Convolutional Neural Networks (CNNs), we aim to enhance the precision and real-time processing capabilities of the system. The project's significance lies in its contribution to the field of face recognition technology, with Python serving as a powerful tool for practical applications in security, access control, and personalized device authentication. Our commitment is to deliver a system that excels in accuracy, efficiency, and user-friendliness, thereby contributing to the evolution of face recognition technology.



We are using deep neural networking for this (DNN)

Deep Neural Network (DNN):

A Deep Neural Network (DNN) is a type of artificial neural network that consists of multiple layers, allowing it to learn hierarchical representations of data.

1. Learning Features:

• DNNs are good at figuring out basic things like edges and textures. For faces, they learn to recognize features like eyes, nose, and mouth by putting these basic pieces together.

2. Handling Differences:

• Faces can look different in various situations, like when the lighting changes or when someone makes a different expression. DNNs can handle these differences because they learn from lots of examples.

3. No Manual Rules:

• DNNs don't need us to tell them how to find faces. They can learn directly from pictures without us specifying exact rules. This makes them flexible for different face appearances.

4. Ready-made Models:

• There are already trained DNN models available for finding faces. It's like having a smart friend who knows a lot about faces and can help us quickly identify them in pictures.

5. Quick and Real-time:

• Some DNNs are designed to work really fast, almost in real-time. This is important for applications where we need to find faces quickly, like in video calls or security systems.

6. Adapting to New Tasks:

• DNNs can be trained for specific face detection tasks easily. We can use what they already know about faces and fine-tune them for particular situations, saving time and effort.

Related Work

Face detection:

Face detection in DNN (Deep Neural Network) refers to the process of using deep learning models to automatically locate and identify faces within images or video frames. DNN-based face detection has become highly effective and widely used due to its ability to handle complex patterns and variations in facial appearances. Here's how face detection in DNN typically works

1. Load Pre-trained Face Detection Model:

• Choose a pre-trained face detection model. Common choices include models based on the Single Shot MultiBox Detector (SSD) or You Only Look Once (YOLO) architecture. In this example, we'll use a model based on SSD.

• Load the model using the cv2.dnn.readNetFromCaffe function, providing the prototxt (architecture) and caffemodel (weights) files.

2. Prepare Image and Create Blob:

• Read the input image.

• Preprocess the image by resizing it to the required input size for the model.

• Create blob from the preprocessed image using the cv2.dnn.blobFromImage function.

3. Perform Face Detection:

• Set the blob as input to the network using net.setInput.

• Perform a forward pass to obtain face detections using net.forward.

• Interpret the output to get the bounding box coordinates and confidence scores for each detected face.

4. Process Detections:

• Loop through the detected faces and extract relevant information.

• Filter detections based on confidence scores or other criteria.

Face embedding:

Face embedding using DNN involves converting a face image into a fixed-size vector (embedding) that captures essential facial features.

Here's a simplified explanation of the face embedding process using a DNN:

1. Load Pre-trained Face Embedding Model:

• Choose a pre-trained face embedding model. Common choices include models trained on large face datasets using deep architectures. In the example below, we assume the use of the OpenFace model.

2. Prepare Face Image:

• Given a detected face region from an image, prepare the face image for input to the face embedding model.

3. Create Blob for Face Embedding Model:

• Convert the prepared face image into a blob suitable for input to the face embedding model.

4. Pass Blob Through Embedding Model:

• Set the input blob for the face embedding model and perform a forward pass to obtain the face embedding.

5. Use the Embedding Vector:

• The 'embedding' variable now contains the face embedding vector, which is a fixed-size representation of the essential facial features.

Dataset processing:

Dataset processing involves preparing and organizing the data before it is fed into the network for training or evaluation. This step is crucial for the success of the DNN, as the quality of the dataset directly influences the model's ability to learn and generalize.

1. Data Loading:

• Load the dataset into the system, ensuring it contains a representative sample of the problem you want the DNN to solve. This can include labelled examples for supervised learning tasks.

2. Data Cleaning:

• Clean the data by handling missing values, correcting errors, and addressing any inconsistencies. Ensuring the dataset is clean helps prevent the model from learning spurious patterns.

3. Data Splitting:

• Split the dataset into training, validation, and testing sets. The training set is used to train the model, the validation set helps tune hyperparameters and prevent overfitting, and the testing set evaluates the model's performance on unseen data.

4. Data Augmentation:

• For image-based tasks, apply data augmentation techniques to artificially increase the size of the dataset. This involves creating new training examples by applying random transformations such as rotations, flips, and zooms to the existing images.

5. Normalization:

• Normalize the data to bring it to a standard scale. For instance, in image processing, pixel values are often scaled to the range [0, 1] or [-1, 1].

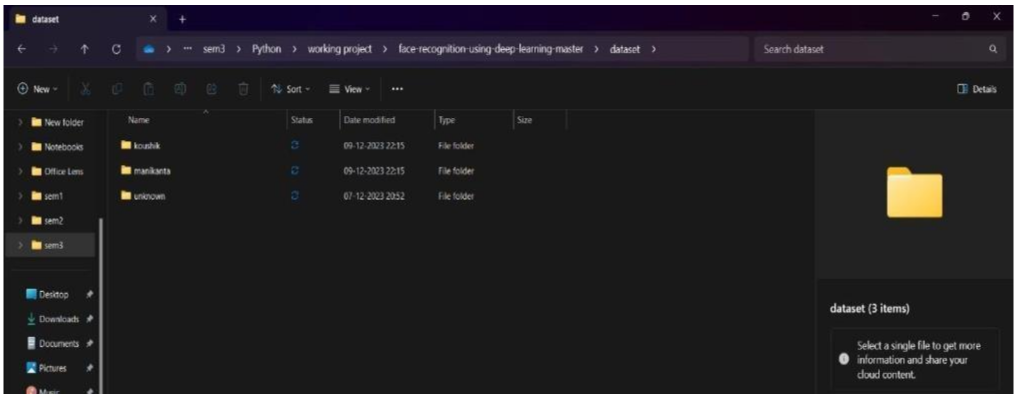
Model training:

1. Training the SVM Model:

• A linear Support Vector Machine (SVM) classifier (SVC) is instantiated with specified parameters (C=1.0, kernel="linear", and probability=True). The model is then trained using the facial embeddings (data["embeddings"]) and their corresponding encoded labels.

2. Saving the Trained Model and Label Encoder:

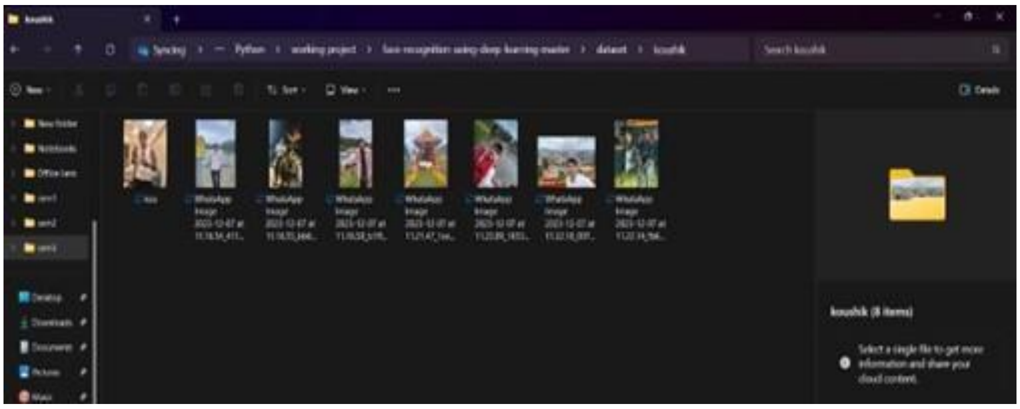
• Once the training is complete, the trained SVM recognizer model and the Label Encoder are saved using the pickle module. These saved models can be later used for face recognition tasks.



sem3 › Python › working project › face-recognition-using-deep-learning-master › dataset ›

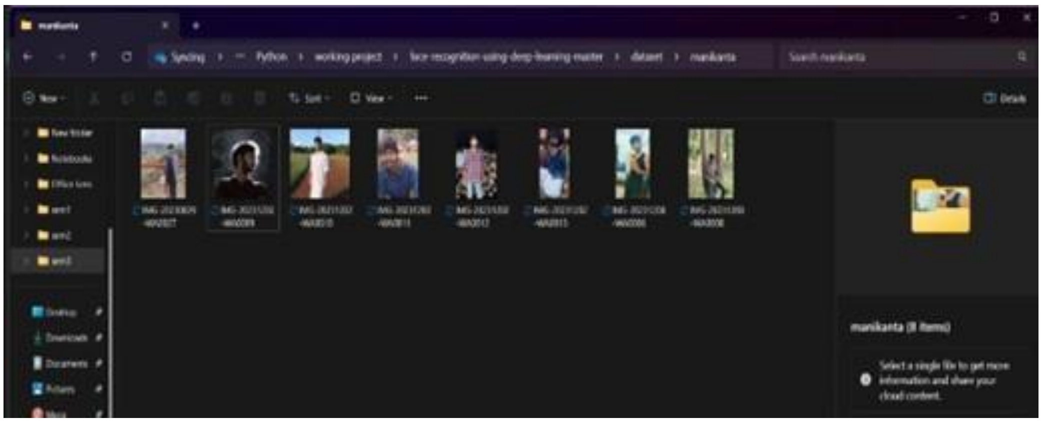
Name	Status	Date modified	Type	Size
koushik		09-12-2023 22:15	File folder	
manikanta		09-12-2023 22:15	File folder	
unknown		07-12-2023 20:52	File folder	

dataset (3 items)



Python › working project › face-recognition-using-deep-learning-master › dataset › koushik

koushik (8 items)


manikanta
Syncing › Python › working project › face recognition using deep learning master › dataset › manikanta
Search manikanta
New
Sort
View
Details
New folder
Notebooks
Desktop
Downloads
Documents
Pictures
manikanta (8 items)
Select a single file to get more information and share your cloud content.

Real time face recognition :

Real-time face recognition using the provided code involves continuously capturing video frames from a camera stream and processing each frame to detect and recognize faces in real-time.

1. Video Stream Initialization:

- The script initializes a video stream using the **VideoStream** class from the **imutils** library. The camera is started with **VideoStream(src=0)**.

2. FPS Estimation:

- Frames Per Second (FPS) is estimated using the **FPS** class from **imutils.video**. This class is used to measure the processing speed and display it at the end of the script.

3. Face Detection in Real-time:

- The code continuously captures frames from the video stream and resizes each frame to a width of 600 pixels to facilitate processing. The OpenCV deep learning-based face detector is then applied to localize faces in each frame.

4. Face Recognition:

- For each detected face, the script extracts the face region and ensures its width and height are sufficiently large. The face is then passed through a face embedding model to obtain a 128-dimensional vector (embedding) that quantifies the face.

5. Prediction and Display:

- The obtained face embedding is used for classification with a pre-trained Support Vector Machine (SVM) classifier (**recognizer**). The prediction probabilities are used to determine the recognized face, and the bounding box and associated probability are displayed on the frame in real-time.

6. FPS Display and Quitting:

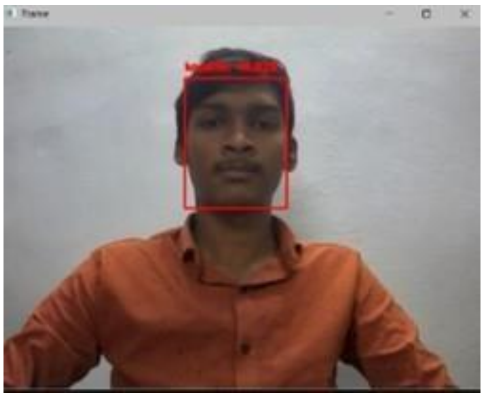
- The FPS information is displayed at the end of processing. Pressing the 'q' key breaks out of the loop, stopping the video stream and closing the display window.

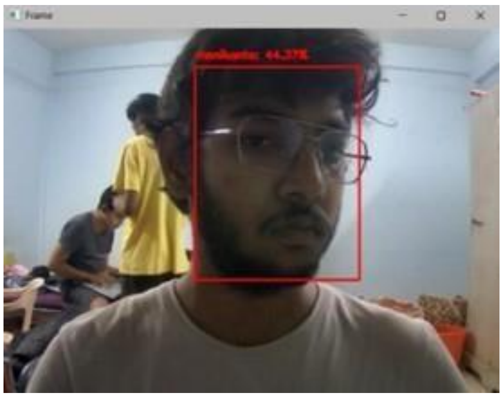
7. Cleanup:

- The script performs cleanup operations, closing the OpenCV display window and stopping the video stream.

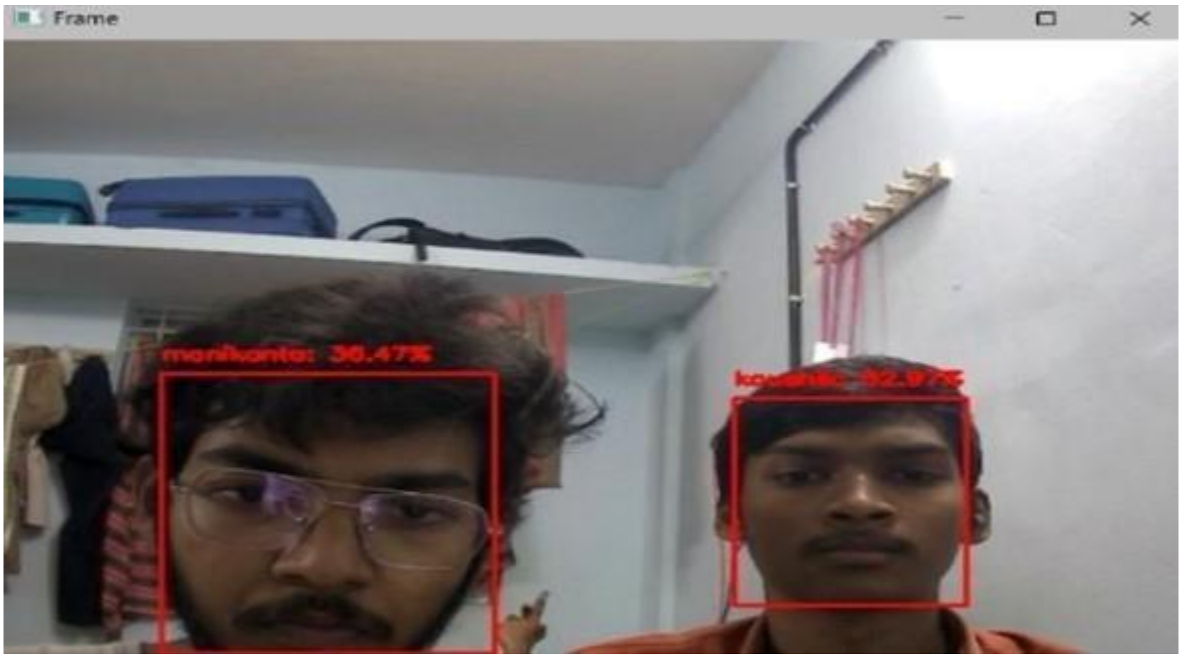
Display and user interface

- **Individual:**





- **Group:**



Codes:

- ## Face detection and Embedding:

```
# import libraries
from imutils import paths
import numpy as np
import argparse
import imutils
import pickle
import cv2 import
os
print("Loading Face Detector...")
protoPath = "face_detection_model/deploy.prototxt"
modelPath =
"face_detection_model/res10_300x300_ssd_iter_140000.caffemodel"
detector = cv2.dnn.readNetFromCaffe(protoPath, modelPath)
print("Loading Face Recognizer...")
embedder = cv2.dnn.readNetFromTorch("openface_nn4.small2.v1.t7")
print("Quantifying Faces...")
imagePaths = list(paths.list_images("dataset"))


knownEmbeddings = []
knownNames = []
total = 0

for (i, imagePath) in enumerate(imagePaths): if

(i%50 == 0):
print("Processing image {}/{}".format(i, len(imagePaths))) name =
imagePath.split(os.path.sep)[-2]



image = cv2.imread(imagePath)
image = imutils.resize(image, width=600) (h,
w) = image.shape[:2]


imageBlob = cv2.dnn.blobFromImage( cv2.resize(image,
(300, 300)), 1.0, (300, 300),
(104.0, 177.0, 123.0), swapRB=False, crop=False)


detector.setInput(imageBlob)
detections = detector.forward()


    if len(detections) > 0:

            i = np.argmax(detections[0, 0, :, 2])

            confidence = detections[0, 0, i, 2]
            if confidence > 0.5:

                box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
                (startX, startY, endX, endY) = box.astype("int")


                face = image[startY:endY,
                startX:endX] (fH, fW) = face.shape[:2]


                if fW < 20 or fH < 20:
                    continue



                faceBlob = cv2.dnn.blobFromImage(face, 1.0 / 255,

                    (96, 96), (0, 0, 0), swapRB=True, crop=False)
                embedder.setInput(faceBlob)

                vec = embedder.forward()


                knownNames.append(name)
                knownEmbeddings.append(vec.flatte
                n()) total += 1

            print("[INFO] serializing {}
    encodings...".format(total)) data = {"embeddings":
    knownEmbeddings, "names": knownNames}
```

Explanation (Intermediary Steps):

➢ Load libraries and models:

- Import necessary libraries like cv2, imutils, and pickle.
- Load pre-trained face detector and face embedding models.

➢ Process images in dataset:

- Iterate over each image in the specified dataset directory.
- Extract the person name from the image path.
- Load and resize the image.
- Convert the image to a blob for deep learning processing.

- **Defect faces:**
  - Pass the image blob to the face detector to identify faces.
  - Locate the bounding box with the highest probability if a face is detected.
  - Ensure the detection confidence exceeds a minimum threshold.
- **Extract facial embeddings:**
  - Extract the face region of interest (ROI) from the original image.
  - Create a blob from the face ROI and pass it to the face embedding model.
  - Obtain the 128-dimensional facial embedding vector representation.
- **Store extracted data:**
  - Append the extracted facial embedding and corresponding name to lists.
  - After processing all images, save the lists of names and embeddings as a serialized file.

    stored embeddings.

Model training:

```
print("[INFO] encoding
labels...") le =LabelEncoder()

labels = le.fit_transform(data["names"])


print("[INFO] training model...")

recognizer = SVC(C=1.0, kernel="linear", probability=True)

recognizer.fit(data["embeddings"], labels)
```

Explanation

Intermediary Steps:

- **Import libraries and arguments:**

  Import necessary libraries for label encoding, SVM classification, and serialization.
  Parse command-line arguments specifying paths for the pre- extracted data.

- **Encode labels:**

  Load the pre-extracted data containing facial embeddings and corresponding names.
  Create a LabelEncoder object to map the names to numerical labels. Encode the names in the data using the LabelEncoder.

- **Train the SVM model:**

  Initialize an SVM classifier with specific parameters (C, kernel, probability).
  Train the model using the encoded labels and pre-extracted facial embeddings as input.

- **Serialize the model:**

  Save the trained SVM model and the LabelEncoder for future use in facial recognition tasks.

Real time face recognition:

```
print("Starting          Video
Stream...")          vs          =
VideoStream(src=1).start()
time.sleep(2.0)

fps = FPS().start()

while True:


    frame = vs.read()

    frame = imutils.resize(frame,
    width=600) (h, w) =
    frame.shape[:2]

    imageBlob = cv2.dnn.blobFromImage(
        cv2.resize(frame, (300, 300)), 1.0, (300, 300),

        (104.0, 177.0, 123.0), swapRB=False, crop=False)


    detector.setInput(imageBlob)
    detections =detector.forward()


for i in range(0, detections.shape[2]):


    confidence = detections[0, 0, i, 2]


        if confidence > 0.5:



            box = detections[0, 0, i, 3:7] * np.array([w, h, w,
            h]) (startX, startY, endX, endY) = box.astype("int")
        face = frame[startY:endY,
        startX:endX] (fH, fW) =
        face.shape[:2]

        if fW < 20 or fH <
            20: continue

        faceBlob = cv2.dnn.blobFromImage(face, 1.0 / 255,
            (96, 96), (0, 0, 0), swapRB=True, crop=False)
        embedder.setInput(faceBlob)

        vec = embedder.forward()

        preds =
        recognizer.predict_proba(vec)[0] j
        = np.argmax(preds)

        proba =
        preds[j] name
        = le.classes_[j]

        text = "{}: {:.2f}%".format(name, proba * 100)

        y = startY - 10 if startY - 10 > 10 else startY + 10
        cv2.rectangle(frame, (startX, startY), (endX, endY),
            (0, 0, 255), 2)
        cv2.putText(frame, text, (startX, y),
            cv2.FONT_HERSHEY_SIMPLEX, 0.45, (0, 0, 255), 2)

fps.update()

cv2.imshow("Frame",fram
e) key = cv2.waitKey(1) &
0xFF

if key
    ==ord("q"):
    break
```

```
fps.stop()

print("Elasped time: {:.2f}".format(fps.elapsed()))

print("Approx. FPS: {:.2f}".format(fps.fps()))


cv2.destroyAllWindows(
) vs.stop()
```

Explanation:

Intermediary Steps:

Initialize video stream and models:

- Start the primary webcam video stream and allow the camera to warm up.
- Load pre-trained face detector and embedding models.
- Load the trained SVM classifier and LabelEncoder from serialized files.

Process video frames:

- Capture frames from the webcam video stream.
- Resize the frames for efficient processing.
- Convert the frames to blobs for deep learning models.

Detect faces:

- Pass the image blob to the face detector to identify faces.
- Extract the bounding boxes and confidence scores for detected faces.

Extract facial embeddings:

For each detected face, extract the corresponding region of interest (ROI).
Create a blob from the face ROI and pass it through the embedding model.
Obtain the 128-dimensional facial embedding representation for each face.

Recognize faces:

- Predict the identity of each face using the pre-trained SVM classifier and extracted embeddings.
- Obtain the predicted label (name) and its associated confidence score.

Draw results:

- Draw bounding boxes around the detected faces on the original frame.
- Display the predicted name and its confidence score to each bounding box.

Update and display:

- Update the FPS counter for performance monitoring.
- Display the processed frame with detections and labels.

Handle user interaction and cleanup:

- Listen for keyboard input ('q' to quit).
- Stop the video stream and FPS timer.
- Close all windows and release resources.

Proposed work:

The face recognition project harnesses the power of deep learning and machine learning to create a robust and accurate system for identifying individuals from facial images. The implementation involves two fundamental components: face detection using a deep neural network (DNN) and face recognition through a Support Vector Machine (SVM) classifier.

In conclusion, this project showcases the synergy of deep learning and machine learning techniques in the realm of face recognition. By combining the capabilities of a DNN-based face detector with an SVM classifier, the system achieves a balance between accuracy and efficiency, paving the way for practical and reliable face recognition solutions in Python.

EMOTION DETECTION:

Introduction:

Emotion detection using computer vision and deep learning is a fascinating application that allows machines to understand and respond to human emotions based on facial expressions. This project utilizes two powerful Python libraries: cv2 (OpenCV) and Keras.
These libraries, in conjunction, enable the creation of an effective emotion detection system that can be integrated into various applications.

OpenCV (cv2):
OpenCV is an open-source computer vision library that provides a wide array of tools for image and video processing. In the context of this project, OpenCV is used for capturing and processing real-time video frames from a camera feed. It facilitates tasks such as face detection, image filtering, and other operations crucial for extracting meaningful information from visual data.

Keras:
Keras is a high-level neural networks API written in Python and capable of running on top of TensorFlow, Microsoft Cognitive Toolkit (CNTK), or Theano. It simplifies the process of building, training, and deploying deep learning models. In this project, Keras is employed to construct a deep neural network for facial emotion recognition. The pre-built models and utilities in Keras make it easier to implement complex architectures without delving into the intricacies of low-level neural network programming.

Dataset Preparation:

For emotion detection, you need a labeled dataset containing images of faces with corresponding emotion labels. You can use existing datasets like the Facial Expression Recognition Challenge (FERC) dataset or create your own.

Ensure that the dataset is diverse and covers a range of emotions, lighting conditions, and facial expressions.

The dataset should be divided into training and testing sets. A common split is around 80% for training and 20% for testing.

Image Preprocessing:

Before training this emotion detection model, preprocessing the images is required to ensure the consistency of the high load, common steps included are:

Resizing: Resize images to a fixed size suitable for the model input.

Normalization: Normalize pixel values to a range between 0 and 1.

Grayscale Conversion: Convert images to grayscale if the color information is not necessary for your task.

Model Training:

Model training using Keras involves the process of optimizing a neural network's parameters to make accurate predictions on a given task. In the context of emotion detection, training the model means adjusting the weights and biases of the neural network to effectively map facial features to corresponding emotion labels. Here's a step-by- step breakdown of the model training process using Keras:

Importing Libraries:

```
import matplotlib.pyplot as
plt import numpy as np

import pandas
as pd import
seaborn as sns
import os
```

Importing Deep Learning Libraries

```
from keras.preprocessing.image import load_img,
img_to_array from keras.preprocessing.image import
ImageDataGenerator from keras.layers import

Dense,Input,Dropout,GlobalAveragePooling2D,Flatten,Conv2D,BatchNormalization,Activa
ti on,MaxPooling2D

from keras.models import Model,Sequential

from keras.optimizers import Adam,SGD,RMSprop
```

Displaying Images:

```
picture_size = 48

folder_path = "../input/face-expression-recognition-
dataset/images/" expression = 'disgust'

plt.figure(figsize= (12,12))

for i in range(1, 10, 1):

    plt.subplot(3,3,i)

    img = load_img(folder_path+"train/"+expression+"/"+

                os.listdir(folder_path + "train/" + expression)[i], target_size=(picture_size,
picture_size))

    plt.imshow(im
g) plt.show()
```

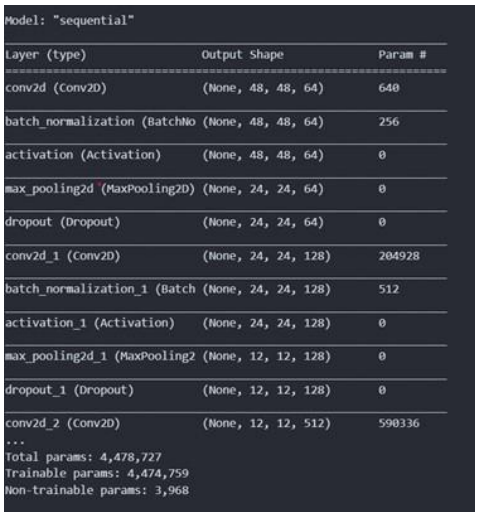
Training and Validating the data:

```
batch_size  = 128

datagen_train = ImageDataGenerator()
datagen_val = ImageDataGenerator()

train_set = datagen_train.flow_from_directory(folder_path+"train",
                                              target_size =
                                              (picture_size,picture_size),
                                              color_mode = "grayscale",
                                              batch_size=batch_size,
                                              class_mode='categorical',
                                              shuffle=True)


test_set = datagen_val.flow_from_directory(folder_path+"validation",
                                              target_size =
                                              (picture_size,picture_size),
                                              color_mode = "grayscale",
                                              batch_size=batch_size,
                                              class_mode='categorical',
                                              shuffle=False)
```

Model Building:

```
from keras.optimizers import Adam,SGD,RMSprop

no_of_classes = 7


model = Sequential()


#1st CNN layer

model.add(Conv2D(64,(3,3),padding = 'same',input_shape = (48,48,1)))
model.add(BatchNormalization())

model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Dropout(0.25))

#2nd CNN layer
model.add(Conv2D(128,(5,5),padding =
'same')) model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size =
(2,2)))

model.add(Dropout  (0.25))
```

```
#3rd CNN layer
model.add(Conv2D(512,(3,3),padding =
'same')) model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size =
(2,2)))

model.add(Dropout  (0.25))

#4th CNN layer

model.add(Conv2D(512,(3,3),
padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,
2))) model.add(Dropout(0.25))


model.add(Flatten())

#Fully connected 1st layer
model.add(Dense(256))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))

# Fully connected layer 2nd layer
model.add(Dense(512))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))


model.add(Dense(no_of_classes, activation='softmax'))

opt = Adam(lr = 0.0001)
model.compile(optimizer=opt,loss='categorical_crossentropy',
metrics=['accuracy']) model.summary()
```



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 48, 48, 64)        640
_________________________________________________________________
batch_normalization (BatchNo (None, 48, 48, 64)        256
_________________________________________________________________
activation (Activation)      (None, 48, 48, 64)        0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 24, 24, 64)        0
_________________________________________________________________
dropout (Dropout)            (None, 24, 24, 64)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 24, 24, 128)       204928
_________________________________________________________________
batch_normalization_1 (Batch (None, 24, 24, 128)       512
_________________________________________________________________
activation_1 (Activation)    (None, 24, 24, 128)       0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 128)       0
_________________________________________________________________
dropout_1 (Dropout)          (None, 12, 12, 128)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 12, 12, 512)       590336
...
Total params: 4,478,727
Trainable params: 4,474,759
Non-trainable params: 3,968
```

Fitting the model with the trained data:

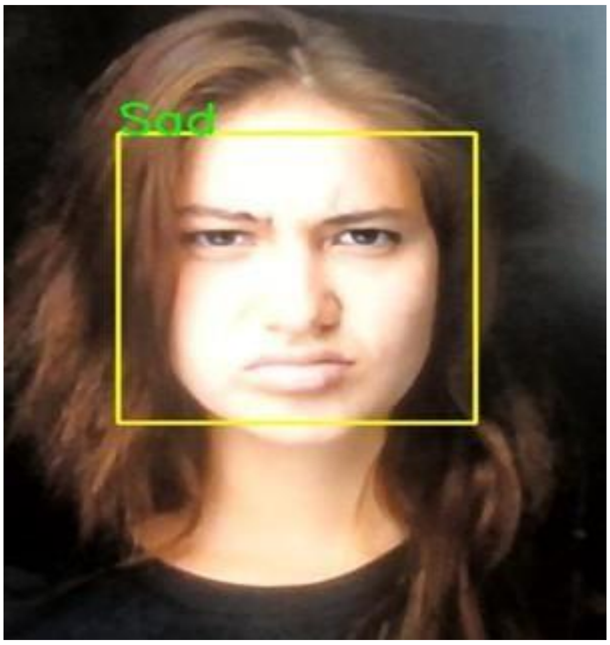
```
from keras.optimizers import RMSprop,SGD,Adam

from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau

checkpoint = ModelCheckpoint("./model.h5", monitor='val_acc', verbose=1,
save_best_only=True, mode='max')

early_stopping = EarlyStopping(monitor='val_loss',

                    min_del
                    ta=0,
                    patienc
                    e=3,
                    verbose
                    =1,

                    restore_best_weights=True

                    )

reduce_learningrate = ReduceLROnPlateau(monitor='val_loss',
                      factor=0.2,

                      patience=3,
                      verbose=1,
                      min_delta=0.0
                      001)

callbacks_list = [early_stopping,checkpoint,reduce_learningrate]

epochs = 48


model.compile(loss='categorical_crossentropy',
        optimizer = Adam(lr=0.001),
        metrics=['accuracy'])


history = model.fit_generator(generator=train_set,

                       steps_per_epoch=train_set.n//train_set.batc
                       h_size, epochs=epochs,

                       validation_data = test_set,

                       validation_steps =
                       test_set.n//test_set.batch_size,
                       callbacks=callbacks_list

                       )
```

Plotting the accuracy and its loss:

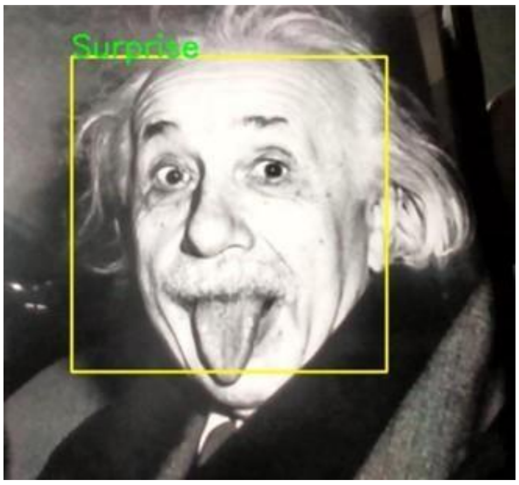
```
plt.style.use('dark_background')

plt.figure(figsize=(20,10)
) plt.subplot(1, 2, 1)

plt.suptitle('Optimizer : Adam', fontsize=10)
plt.ylabel('Loss', fontsize=16)
plt.plot(history.history['loss'],  label='Training  Loss')
plt.plot(history.history['val_loss'],   label='Validation
Loss') plt.legend(loc='upper right')

plt.subplot(1, 2, 2)
plt.ylabel('Accuracy',
fontsize=16)

plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation
Accuracy') plt.legend(loc='lower right')

plt.show()
```

Displaying the Output:

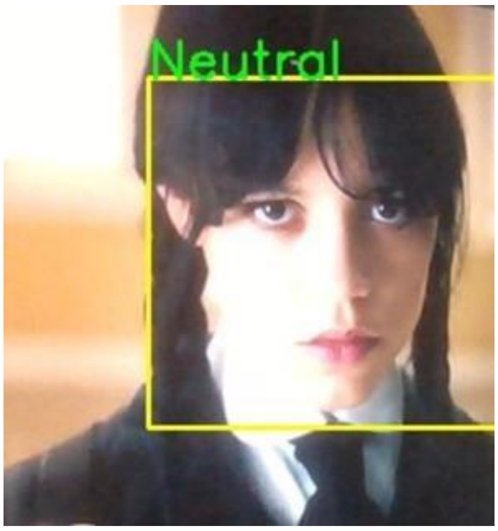
Sad:



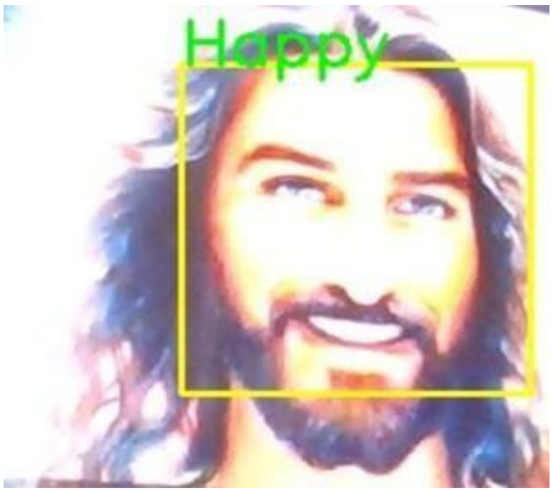
Surprise:



Neutral:



Happy:



Code:

```
from keras.models import
load_model from time import sleep

from keras.preprocessing.image import
img_to_array from keras.preprocessing import
image

import cv2

import numpy as np

face_classifier =
cv2.CascadeClassifier(r"C:\Users\vijay\Documents\python\Emotion_Detection_C
NN- main\haarcascade_frontalface_default.xml")

classifier
=load_model(r"C:\Users\vijay\Documents\python\Emotion_Detection_CNN-
main\model.h5")

emotion_labels = ['Angry','Disgust','Fear','Happy','Neutral', 'Sad',

'Surprise'] cap = cv2.VideoCapture(0)


while True:

  _, frame = cap.read()
  labels = []

  gray = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
  faces = face_classifier.detectMultiScale(gray)

  for (x,y,w,h) in faces:
    cv2.rectangle(frame,(x,y),(x+w,y+h),(0,255,255),2)
    roi_gray = gray[y:y+h,x:x+w]

    roi_gray = cv2.resize(roi_gray,(48,48),interpolation=cv2.INTER_AREA)



    if np.sum([roi_gray])!=0:

      roi =
      roi_gray.astype('float')/255.0
      roi = img_to_array(roi)

      roi = np.expand_dims(roi,axis=0)

      prediction = classifier.predict(roi)[0]
      label=emotion_labels[prediction.argmax()]
      label_position = (x,y)

cv2.putText(frame,label,label_position,cv2.FONT_HERSHEY_SIMPLEX,1,(0,255,0),2)
    else:
            cv2.putText(frame,'No
Faces',(30,80),cv2.FONT_HERSHEY_SIMPLEX,1,(0,255,0),2)

    cv2.imshow('Emotion
    Detector',frame) if
    cv2.waitKey(1) & 0xFF ==
    ord('q'):

        break

cap.release()
cv2.destroyAllWindows
()
```

Code explanation:

Importing libraries:

```
from keras.models import
load_model from time import
sleep

from keras.preprocessing.image import
img_to_array from keras.preprocessing import
image

import cv2

import numpy as np
```

Face Classifier:

```
face_classifier =
cv2.CascadeClassifier(r"C:\Users\vijay\Documents\python\Emotion_Detection_C
NN- main\haarcascade_frontalface_default.xml")
```

CascadeClassifier is used to load a pre-trained Haar Cascade classifier for face detection. This classifier is utilized to detect faces in
the video frames.

```
classifier =load_model(r"C:\Users\vijay\Documents\python\Emotion_Detection_CNN-
main\model.h5")
```

Load model is used to load the pretrained models which can detect the emotions of the inputted images or by using the camera.

Lables:

```
emotion_labels = ['Angry','Disgust','Fear','Happy','Neutral', 'Sad', 'Surprise']
```

These lables contains different emotions.

Video capture:

```
cap = cv2.VideoCapture(0)
```

VideoCapture(0) initializes a video capture object, capturing frames from the default camera (index 0).

Main loop:

```
while True:

   _, frame =
   cap.read() labels =
   []

   gray = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
   faces = face_classifier.detectMultiScale(gray)

   for (x,y,w,h) in faces:
      cv2.rectangle(frame,(x,y),(x+w,y+h),(0,255,255),
      2) roi_gray = gray[y:y+h,x:x+w]

      roi_gray = cv2.resize(roi_gray,(48,48),interpolation=cv2.INTER_AREA)



      if np.sum([roi_gray])!=0:

         roi =
         roi_gray.astype('float')/255.0
         roi = img_to_array(roi)

         roi = np.expand_dims(roi,axis=0)


         prediction = classifier.predict(roi)[0]
         label=emotion_labels[prediction.argmax()]
         label_position = (x,y)

cv2.putText(frame,label,label_position,cv2.FONT_HERSHEY_SIMPLEX,1,(0,255,0),2)
      else:

         cv2.putText(frame,'No
Faces',(30,80),cv2.FONT_HERSHEY_SIMPLEX,1,(0,255,0),2)

   cv2.imshow('Emotion
   Detector',frame) if
```

```
  cv2.waitKey(1) & 0xFF ==
  ord('q'):

  break

cap.release()
cv2.destroyAllWindows(
)
```

→cap.read() captures a frame from the video feed.
→face_classifier.detectMultiScale(gray) detects faces in the grayscale frame.
→For each detected face:
A rectangle is drawn around the face.
The region of interest (ROI) is extracted and resized to 48x48 pixels.
The ROI is preprocessed and fed into the pre-trained CNN model for emotion prediction.
The predicted emotion label is displayed on the frame.
→If no faces are detected, a message indicating "No Faces" is displayed.

→The processed frame with emotion labels is shown in a window named 'Emotion Detector.'
→The loop continues until the user presses the 'q' key.

Proposed Work

In this project involves utilizing the trained emotion detection model to make real-time predictions about the emotions expressed in the faces detected by the Haar Cascade classifier. The results are then displayed for user interaction in the live video feed.

Overview diagram:

Here's an overview diagram of a Face Detection and Emotional Detection AI project:

```
+-----------------+     +-----------------+     +-----------------+     +-----------------+
|      Input      | ---> | Preprocessing  | ---> | Face Detection | --->  | Face Alignment | (Opt)
+-----------------+     +-----------------+     +-----------------+     +-----------------+
                  |
                  | (Resize, Grayscale, Normalize)
                  |
                  v
+-----------------+     +-----------------+     +-----------------+
| Detected Faces  | ---> | Emotional      |     | (Optional)      |
+-----------------+     | Detection       | ---> +-----------------+
                   |                     | (CNN, LSTM)       |
                   v                     v
+-----------------+      +-----------------+      +-----------------+
| Emotion Labels   |       | Output (Image)  |       | (Happy, Sad,     |
+-----------------+      +-----------------+      +-----------------+
                                   | (Bounding boxes, labels)
                                   v
+-----------------+
| Visualization    | (Optional)
+-----------------+
```

Explanation:

- The diagram starts with the **Input**, which can be an image or a video frame.
- The **Preprocessing** block prepares the image for analysis by resizing it to a standard size, potentially converting it to grayscale, and applying normalization techniques.
- The **Face Detection** block uses a pre-trained deep learning model to identify faces in the image. This block outputs bounding boxes around the detected faces.
- **Face Alignment** (optional) takes the detected faces and applies transformations to normalize them for better emotional recognition. This might involve aligning key facial features like eyes and mouth.
- The **Emotional Detection** block utilizes another deep learning model to analyze the detected faces (aligned or non-aligned) and classify the emotions. This block outputs labels for the emotions (e.g., happy, sad, angry) for each face.
- **Postprocessing** (optional) refines the results by filtering out low-confidence predictions and potentially visualizing the results by drawing bounding boxes and emotion labels on the original image.
- Finally, the **Output** can be the processed image with visualized results or just the emotion labels associated with each detected face.

Result and analysis:

An analogous experiment was conducted and compared with three volunteers using a webcam. An already-existing API was contrasted. The API received an input image and used a box to identify faces. These feelings are expressed in a universal and cross-cultural manner through the same fundamental facial expressions, which are recognized by the Emotion API [64]. Microsoft's state-of-the-art cloud-based emotion identification algorithm was utilized by the API. A picture of the subject displaying the expression was captured with the webcam and processed. Here, the same characteristics of each subject—that is, the values of expressions such as happiness, melancholy, etc.—were considered for the purpose of comparison.

Using a Microsoft emotion API, the MHL and a webcam were utilized to identify emotions. The output from the regular webcam is totaled and presented as an emotion probability, with one representing a given emotion's perfect certainty. All five emotions' results were shown, and the feeling with the highest likelihood was chosen to be the observed emotion. A few other emotions (disgust and disdain, for example) that were supported by the API were not taken into account for this study since they were too difficult to act out. There were a few instances where the likelihood was extremely high and the accuracy of the emotions identified was good. When the emotion cheerful is recognized with a probability of 0.99999, this can be shown. In certain instances, the likelihood of identifying an emotion was unclear and nowhere close to 1, but it had a respectable degree of accuracy (>0.5), allowing it to be deduced which category the emotion represented belongs in. For example, image 15 of Table 2 indicates that the emotion was not identified with great accuracy. However, the likelihood of the detected emotion being angry is considerably greater than that of other emotions (0.575), thus the emotion angry can be deduced from that.

Comparison with existing work:

FUNCTIONS	Before this project	After this project
Accuracy This includes enhancing the quality and quantity of the training data, fine-tuning the underlying machine-learning models, and speeding up the feature extraction process.	Even though we have facial recognition and emotional detection algorithms, they have not been very accurate in terms of picture quality, and occlusions have occurred during capture during the last five years.	In this study, we improved the methods to address these challenges. We gathered a diversified and high-quality dataset that covered a wide range of demographics, lighting settings, and facial emotions and positions. Also, we used data preparation, which includes Preprocessing procedures such as alignment, augmentation, and normalization are used to increase the consistency and quality of the training set.
Robustness This entails strengthening facial recognition and emotional detection systems' resistance to changes in occlusions, lighting, facial expressions, and positions.	In the previous years, we lacked sophisticated feature extraction techniques, and we did not have any trained algorithms on diverse datasets that captured a wide range of real-world scenarios.	Thus, to recreate several real-world scenarios while also improving the robustness and efficiency of our system, we included some new methods, such as data augmentation, which involves enriching the training dataset with various transformations such as rotation, scaling, cropping, and flipping. Error Analysis and Feedback Loop: This technique iteratively improves the models and increases their resilience over time, incorporating user feedback and corrective actions based on real-world usage scenarios.

Emotional Granularity Emotional detection granularity allows for a greater range of emotions and subtle facial expressions.	In recent years, we have encountered difficulties in identifying facial expressions and emotions across a greater spectrum. That is, if the illumination were poor, we would not be able to identify the image effectively, and we would also be unable to recognize extreme brightness.	To address this issue, we used high-resolution data, which entails training models on high-resolution facial photographs or videos to extract minute information from faces and micro-expressions that hint at subconscious emotional states. Furthermore, we used deep learning architectures for fine-grained emotional analysis, such as convolutional neural networks.
Privacy and Security: We are strengthening security and privacy standards to protect the sensitive biometric information used in facial recognition software. To prevent unauthorized access to and misuse of personal data, strong encryption, anonymization schemes, and access control systems must be implemented.	Despite having many features in facial recognition and emotional detection, we encountered difficulties detecting the faces of elderly persons with wrinkles on their faces due to low-quality datasets. Because of this issue, we were unable to match the photos of the individuals, and privacy and security may have been jeopardized.	To address this issue, we used a range of techniques, including data encryption, which encodes sensitive information—such as images of faces and emotional traits—while in transit and at rest to protect the secrecy and prevent unauthorized access. To ensure the privacy of user data, powerful encryption techniques, and key management protocols should be utilized. We also used the access control technique, which employs access control mechanisms to restrict access to sensitive data and system functions based on user roles and permissions.

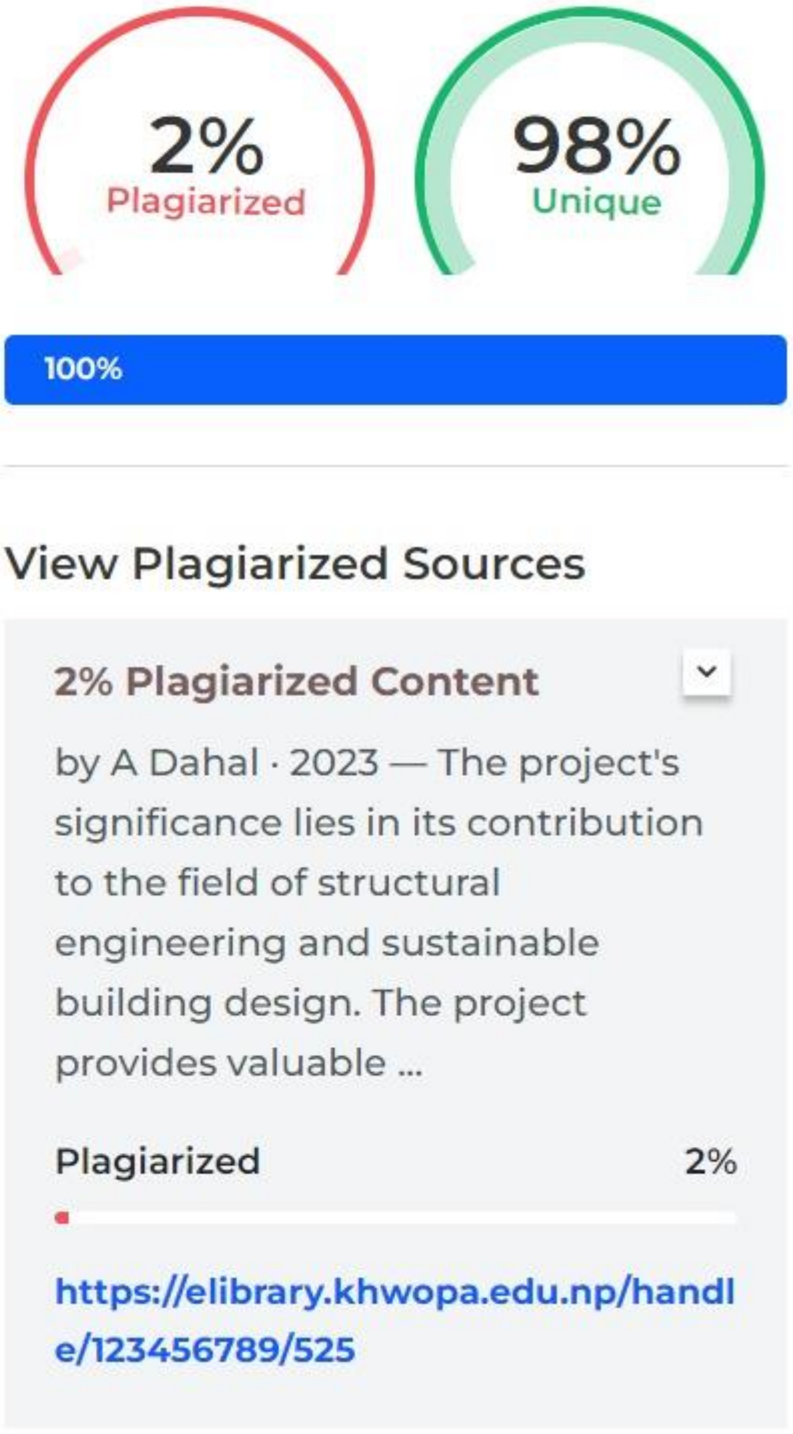
Conclusion:

Emotion recognition and facial detection are extremely difficult problems. To improve the performance measure of face detection and emotion recognition, they demand a lot of work. The field of emotion recognition is becoming more and more popular because of its uses in a variety of industries, including education, software engineering, and gaming. This study provided a thorough analysis of the several methods and strategies used in those methods to identify emotions by detecting facial expressions in people. Additionally, a succinct explanation of the methods involved in a geometric-based approach and a machine learning-based technique for face detection, emotion recognition, and classification were provided. A comparison of the accuracy of the databases used as training and testing datasets was done during the survey's reporting. To provide a brief overview of how the datasets were created, including whether they were posed or spontaneous, static or dynamic, and whether they were tested in a lab or outside, a detailed description of several database types was provided. This database survey leads to the conclusion that RGB databases are less convenient to do tests on and, consequently, have lower efficiency because they do not have intensity labels. The limitations of thermal databases include their inability to handle temperature variations, age, and varied scaling (such as identical twin problems). It is impossible to record disguises while the wearer is wearing glasses. The quality of the database is impacted by the extremely low resolution of thermal photos. There aren't enough 3D databases available to conduct experiments and increase accuracy. The accuracy of several algorithms using these databases were also discussed, indicating room for improvement in the detection of delicate micro-expressions and accuracy in the field of emotion recognition.

References:

https://www.analyticsvidhya.com/blog/2021/11/facial-emotiondetection-using-cn

Plagiarism Report:


2%
Plagiarized

98%
Unique

100%

View Plagiarized Sources

2% Plagiarized Content

by A Dahal · 2023 — The project's significance lies in its contribution to the field of structural engineering and sustainable building design. The project provides valuable ...

Plagiarized 2%

https://elibrary.khwopa.edu.np/handle/123456789/525