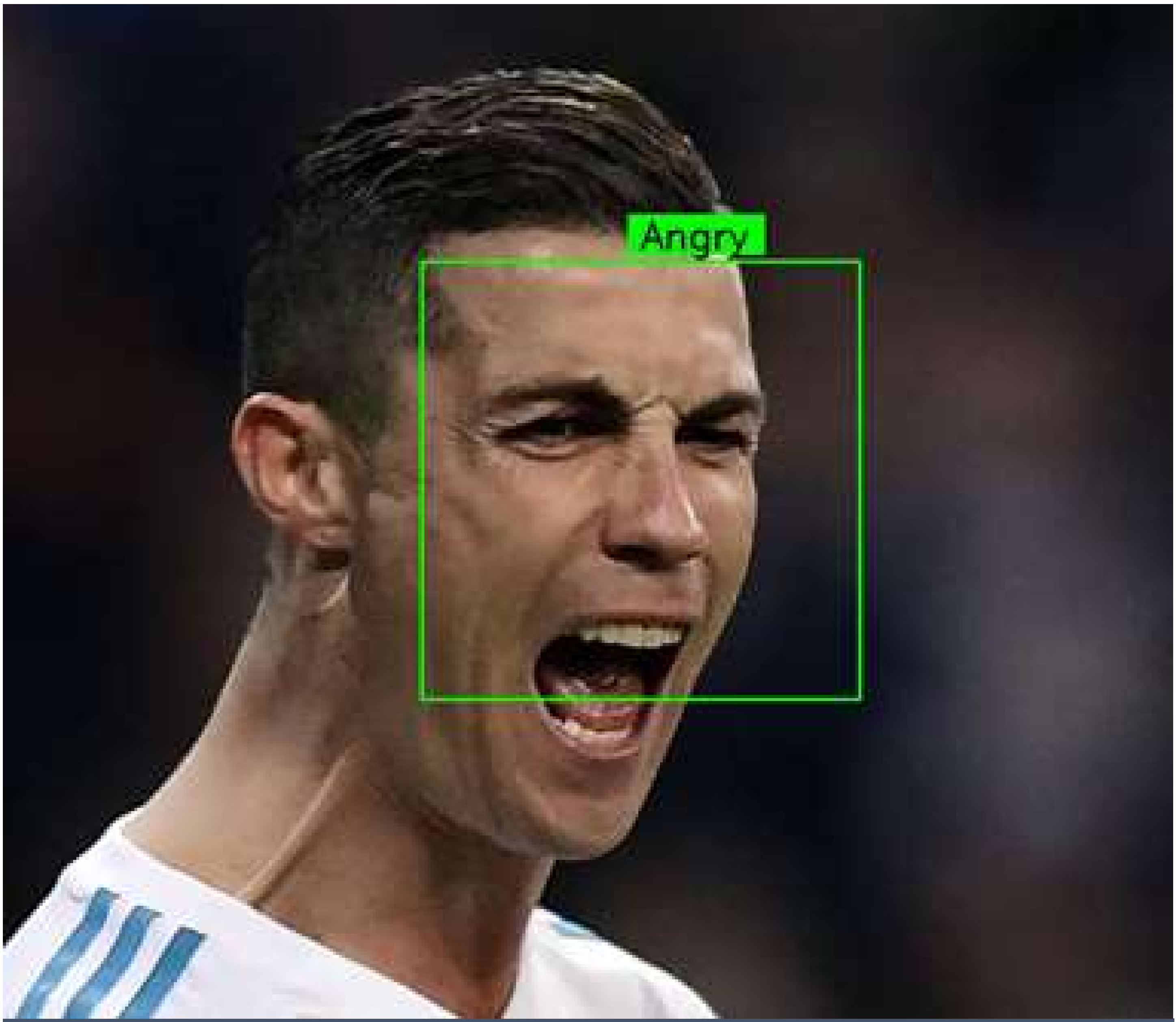


Realtime Emotion Detection using Deep Learning

Computer Vision/ CSOC’23

Aryan Prakhar/Civ Part 2

Introduction

The goal of this project is to build a real-time emotion detection system using deep learning techniques. We will use Convolutional Neural Networks (CNNs) to train a model from scratch to recognize facial emotions. The trained model will be deployed to perform real-time emotion detection in real-time through the camera feed.

Project Overview

This project aims to build a deep learning model for real-time emotion detection from facial expressions in images. We'll leverage popular Python libraries commonly used in computer vision and deep learning.

Libraries Used:

1. matplotlib.pyplot: Powerful plotting library for data and image visualization. It will display sample emotion images, training and validation accuracy/loss plots, and create real-time video streams with emotion labels.

2. numpy (np): Essential for numerical computing, enabling efficient array operations and mathematical computations. It will help convert images to arrays, calculate mean and standard deviation for data normalization, etc.

3. pandas (pd): A data manipulation library that can assist in reading and preprocessing data if needed, although it may not be heavily used in this project.

4. seaborn (sns): Enhances plot appearance, but not directly utilized in the provided code.

5. os: Facilitates interaction with the file system, crucial for accessing and navigating image directories for training and validation data.

6. keras.preprocessing.image: From Keras, it loads and preprocesses image data for deep learning models, converting images into arrays for further processing and training.

7. keras.layers: Contains various neural network layers like Conv2D, MaxPooling2D, Dense, BatchNormalization, Activation, and Dropout, essential for designing the emotion detection model.

8. keras.models: Provides classes for defining and compiling deep learning models. We'll use the `Sequential` model to construct the emotion detection architecture, compiled with the `Adam` optimizer and `categorical_crossentropy` loss function.

9. keras.optimizers: Offers various optimization algorithms for training neural networks. We'll utilize the `Adam` optimizer to guide the model's learning process during training, minimizing the loss function.

By combining these libraries, we will create an effective emotion detection system capable of recognizing facial emotions in real-time.

Understanding Dataset

The data consists of 48x48 pixel grayscale 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 data is properly labelled based on the emotion shown in the facial expression into one of seven categories Angry, Disgust, Fear, Happy, Sad, Surprise and Neutral.

The link to dataset is - https://www.kaggle.com/datasets/jonathanoheix/face-expression-recognition-dataset

Data Preprocessing

Prepared the dataset for training and validation by using the `ImageDataGenerator` from Keras. The goal is to feed the images into the deep learning model in batches, and not all at once.

The specifications are-

- batch_size = 128 the model will process 128 images at a time.

- ImageDataGenerator():applies various transformations to the images on-the-fly, such as rotation, zooming, flipping, etc., which helps in increasing the diversity of the training data and enhances the model's ability to generalize.

- We create two `ImageDataGenerator` objects, one for the training set (`datagen_train`) and another for the validation set (`datagen_val`). The purpose of having separate generators is to apply different transformations and augmentation techniques to the training and validation data, ensuring that the model is trained on a diverse range of data while validating on untouched samples.

- target_size = (picture_size, picture_size): The images are resized to a target size of 48x48 pixels. Resizing the images ensures that all images are of the same size, making it easier to feed them into the model.

- color_mode = "grayscale": We convert the images to grayscale, to make it computationally less expensive.

- class_mode = 'categorical': the labels will be one-hot encoded arrays representing each emotion.

- shuffle = True and shuffle = False: The training data is shuffled randomly in each epoch to introduce randomness and avoid overfitting. However, we do not shuffle the validation data, as we want to evaluate the model on fixed validation samples consistently.

Model Architecture

•Input Layer:

-Grayscale images of size 48x48 pixels are received, reducing computational complexity while capturing relevant facial information for emotion detection.

•1st CNN Layer:

-Convolutional Layer (Conv2D): Applies 64 filters of size (3,3) with 'same' padding to maintain spatial dimensions.

-Batch Normalization: Normalizes activations for faster convergence and reduced covariate shift.

-ReLU Activation: Introduces non-linearity to learn complex patterns for better emotion representation.

-MaxPooling: Reduces spatial dimensions using a (2x2) window, reducing computational complexity.

•2nd CNN Layer:

-Convolutional Layer (Conv2D): Applies 128 filters of size (5,5) for higher-level features.

-Batch Normalization, ReLU Activation: Enhances stable training and feature representation.

-MaxPooling, Dropout (rate=0.25): Further downsampling and prevents overfitting.

•3rd and 4th CNN Layers:

-Convolutional Layers: Two layers with 512 filters each and (3,3) size capture intricate patterns and fine details.

-Batch Normalization, ReLU Activation, MaxPooling, Dropout: Enhance learning and prevent overfitting.

•Flatten Layer:

-Converts convolutional outputs into a 1D vector for fully connected layers.

•Fully Connected Layers:

-Dense Layer (256 neurons): Learns higher-level representations from convolutional features.

-Batch Normalization, ReLU Activation, Dropout: Enhance robustness and prevent overfitting.

-Dense Layer (512 neurons): Further refines learned representations.

-Batch Normalization, ReLU Activation, Dropout: Improve feature learning and regularization.

•Output Layer:

-Dense Layer (7 neurons): Represents the emotion classes (angry, disgust, fear, happy, neutral, sad, surprise).

-Softmax Activation: Converts output to probabilities for emotion likelihoods.

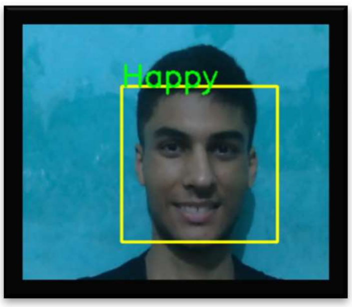
Model Training

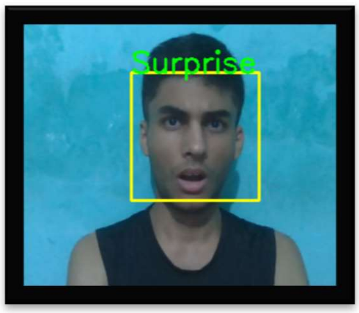
The code for model training defines the model architecture and compiles it with an optimizer (`Adam`), categorical cross-entropy loss function, and accuracy as the evaluation metric. The code uses three callback functions (`ModelCheckpoint`, `EarlyStopping`, and `ReduceLROnPlateau`) to improve training performance and avoid overfitting. The model is trained using the `fit_generator` function with training and validation data generators (`train_set` and `test_set`). The training is performed for 48 epochs.

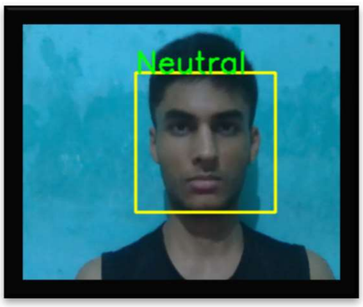
Plotting Accuracy and Loss

We display two subplots in a single figure: one for the training and validation loss, and the other for the training and validation accuracy. The plots help analyze the model's performance and its ability to learn from the data during training. The dark background style enhances the visibility of the graphs.

`Deploying for real-time detection







We utilize the trained deep learning model to detect facial emotions from a live video stream. Here is the process-

1. Importing Libraries:

-We import the libraries including `keras.models.load_model`, `time.sleep`, `keras.preprocessing.image.img_to_array`, `keras.preprocessing.image`, `cv2`, and `numpy`.

2. Loading Pre-trained Models:

-The code loads a pre-trained deep learning model for emotion detection using `keras.models.load_model`. It also loads the Haar Cascade classifier (`face_classifier`) used for detecting faces in the video stream.

3. Emotion Labels:

- The `emotion_labels` list contains the corresponding emotion labels ('Angry', 'Disgust', 'Fear', 'Happy', 'Neutral', 'Sad', 'Surprise') that the model can predict.

4. Video Capture:

- The code initializes the video capture using `cv2.VideoCapture(0)`, where '0' represents the default camera.

5. Real-time Emotion Detection Loop:

- The code enters a while loop to continuously process frames from the video stream.

- It detects faces in each frame using the Haar Cascade classifier (`face_classifier`).

- For each detected face, the code performs the following steps:

  - Draws a rectangle around the face using `cv2.rectangle`.

  - Preprocesses the face region by converting it to grayscale and resizing it to (48x48) pixels.

  - Normalizes the face region and converts it into an array for model prediction.

  - Predicts the emotion from the preprocessed face region using the loaded deep learning model (`classifier`).

  - Retrieves the predicted emotion label and places it on the frame using `cv2.putText`.

- If no faces are detected, it displays a message 'No Faces' on the frame.

- The processed frame with emotion labels is displayed in a window named 'Emotion Detector'.

6. Exiting the Loop:

- The code continues to process frames until the user presses the 'q' key.

- When 'q' is pressed, the loop breaks, and the video capture is released (`cap.release()`).

- The OpenCV windows are closed using `cv2.destroyAllWindows()`.

Practical use cases

1. **Mental Health Monitoring**: Emotion detection technology can be used in mental health applications to monitor patients' emotional states over time. It aids in early detection of mood disorders and provides valuable insights to healthcare professionals for treatment planning.
2. **Education & E-Learning-** The emotion detection technology can be used to gauge student’s response and engagement to learning materials and help tailor the teaching strategy by identifying tougher topics.
3. **Automotive industry-** The technology can help identifying signs of fatigue, weakness etc by assessing the emotions of driver and thus alert the driver to take necessary breaks.

How to run the program in your system

1. To run the code in your system, download all the files.
2. Open main.py in your IDE and change the file path string of ‘face_classifier’ variable and ‘classifier’ variable to the directories containing ‘haarcascade_frontalface_default.xml’ and ‘model.h5’ respectively.
3. Now save the main.py and run the script.
4. A window would open up showing your camera feed and labelling your emotions in real-time.
5. To close the window, stop the execution of the script.