Introduction:

- In this second development phase, we venture into the integration of face detection and emotion recognition, forming a comprehensive image recognition system.

- At the heart of this endeavor lies the "Haar Cascade Classifier," a cutting-edge technology in computer vision primarily designed for detecting frontal faces within images.

- Its role in identifying and locating faces is pivotal, providing a foundation for subsequent emotion recognition tasks.

- The phase also involves a series of code snippets designed to facilitate the development of this system.

- It begins with the preparation of a machine learning model for facial emotion recognition, configuring data generators, creating and fine-tuning the deep learning model, and preparing it for training.

- Users have the option to select from a range of pre-trained deep learning models for emotion recognition, each with its unique architecture.

- The training and evaluation process monitors the model's performance, with early stopping mechanisms and insightful visualizations of accuracy and loss.
- Additionally, the option to save the trained model and associated performance metrics adds a layer of practicality to this multifaceted image recognition project.

- This phase bridges the world of face detection and emotion recognition, offering a comprehensive solution for image analysis and understanding.

Data Sources and Setup

Before running the scripts for the image recognition project, it's crucial to set up the required datasets. This document provides an overview of the datasets used in our project and how to obtain them.

Datasets:

CK+ (Cohn-Kanade Extended+):

Source:

https://www.kaggle.com/datasets/shawon10/ckplus

Description: The CK+ dataset contains facial expressions captured in lab-controlled environments. It includes seven different emotion labels, making it suitable for training and testing emotion recognition models.

FER-13 (Facial Expression Recognition 2013):

Sources:

https://www.kaggle.com/datasets/msambare/fer2013

https://www.kaggle.com/datasets/deadskull7/fer2013

Description: The FER-13 dataset is a collection of images representing facial expressions. It contains various emotional states, enabling comprehensive training and testing for emotion recognition.

FERPlus:

Source:

https://github.com/microsoft/FERPlus

Description: FERPlus is an extension of the FER-13 dataset, providing a more refined annotation of emotions. It includes additional labels, offering improved granularity in emotion recognition.

Data Setup:

To run the image recognition scripts successfully, follow these steps:

- Download the CK+, FER-13, and FERPlus datasets from their respective sources.
- Organize the dataset files according to your project's directory structure.

Data Preprocessing for Emotion Recognition Model

- In the field of emotion recognition using deep learning, data preprocessing plays a pivotal role in shaping the effectiveness of the models.
- This document outlines essential data preprocessing steps to prepare the FERPlus dataset for training emotion recognition models.

Data Cleaning and Transformation:

Read and Clean CSV:

- The process begins with reading the FERPlus dataset's CSV file, which contains labels and information about the images.
- Any rows with missing values (NaN) are removed to ensure data integrity.

Mapping Emotions:

- The FERPlus dataset provides emotion labels in a detailed format.

Emotions are mapped into seven primary categories: neutral, happy, surprise, sad, angry, disgust, and fear.

Data Reorganization:

3. Transfer Images:

- Images are transferred from the original FERPlus directory structure to match the FER-2013 structure.
- Images are categorized into training and test sets based on the "Usage" attribute in the CSV file.

Emotion-Based Sorting:

Images are further sorted into subfolders within the training and test sets based on the dominant emotion category they represent.

Execution:

The Python script provided automates the data preprocessing steps mentioned above. It ensures that the FERPlus dataset aligns with the FER-2013 dataset's structure and emotion categories.

```
import os
import shutil


import cv2
import numpy as np
import pandas as pd



def get_best_emotion(list_of_emotions, emotions):
    best_emotion = np.argmax(emotions)
    if best_emotion == "neutral" and sum(emotions[1::]) > 0:
        emotions[best_emotion] = 0
        best_emotion = np.argmax(emotions)
    return list_of_emotions[best_emotion]



def read_and_clean_csv(path):
    # we read the csv and we delete all the rows which contains NaN
    df = pd.read_csv(path)
    df = df.dropna()
```

```
    return df



def rewrite_image_from_df(df):
    print("Moving images from FERPlus inside FER-2013")
    # we setup an accumulator to print if we have finished a task
    acc = ""
    emotions = [
        "neutral",
        "happy",
        "surprise",
        "sad",
        "angry",
        "disgust",
        "fear",
        "contempt",
        "unknown",
        "NF",
    ]

    # we rewrite all the image files
    for row in range(len(df)):
        item = df.iloc[row]
        if item["Usage"] not in ["", acc]:
            print(f"{item['Usage']} done")
        if (item['Usage'] == "Training"):
            image = cv2.imread(f"./FERPlus/output/FER2013Train/{item['Image name']}")
        elif item['Usage'] == "PublicTest":
            image = cv2.imread(f"./FERPlus/output/FER2013Valid/{item['Image name']}")
        else:
            image = cv2.imread(f"./FERPlus/output/FER2013Test/{item['Image name']}")
        acc = item["Usage"]
        if acc == "Training":
            cv2.imwrite(
```

```
            f"./FER-2013/train/{get_best_emotion(emotions, item[2::])}/{item['Image name']}",
            image,
        )
    else:
        cv2.imwrite(
            f"./FER-2013/test/{get_best_emotion(emotions, item[2::])}/{item['Image name']}",
            image,
        )


if __name__ == "__main__":
    os.system('python ./FERPLUS/src/generate_training_data.py -d ./FERPLUS/output -fer ./FER-2013/fer2013.csv -ferplus ./FERPLUS/fer2013new.csv')
    df = read_and_clean_csv("./FERPlus/fer2013new.csv")
    rewrite_image_from_df(df)
```

Haar Cascade Classifier for Face Detection:

The provided XML code represents a machine learning model for detecting frontal faces in images. This specific model appears to be a Haar Cascade Classifier for face detection. Haar Cascade Classifiers are a type of object detection algorithm used in computer vision for detecting objects (in this case, faces) in images.

The XML code outlines various parameters and configurations for the classifier. It specifies details about the weak classifiers, stages, and thresholds used for face detection. Additionally, it defines the dimensions of the detection window (24x24 pixels).

The code includes a licensing agreement indicating that the software is provided by Intel Corporation. It highlights the terms

and conditions for using the software, including redistribution requirements and disclaimers of warranty.


```
<?xml version="1.0"?>
<opencv_storage>
<cascade type_id="opencv-cascade-classifier"><stageType>BOOST</stageType>
 <featureType>HAAR</featureType>
 <height>24</height>
 <width>24</width>
 <stageParams>
  <maxWeakCount>211</maxWeakCount></stageParams>
 <featureParams>
  <maxCatCount>0</maxCatCount></featureParams>
 <stageNum>25</stageNum>
 <stages>
  <_>
   <maxWeakCount>9</maxWeakCount>
   <stageThreshold>-5.0425500869750977e+00</stageThreshold>
   <weakClassifiers>
    <_>
     <internalNodes>
      0 -1 0 -3.1511999666690826e-02</internalNodes>
     <leafValues>
      2.0875380039215088e+00 -2.2172100543975830e+00</leafValues></_>
    <_>
     <internalNodes>
      0 -1 1 1.2396000325679779e-02</internalNodes>
     <leafValues>
      -1.8633940219879150e+00 1.3272049427032471e+00</leafValues></_>
    <_>
     <internalNodes>
      0 -1 2 2.1927999332547188e-02</internalNodes>
     <leafValues>
      -1.5105249881744385e+00 1.0625729560852051e+00</leafValues></_>
    <_>
     <internalNodes>
```

```
      0 -1 3 5.7529998011887074e-03</internalNodes>
      <leafValues>
      -8.7463897466659546e-01 1.1760339736938477e+00</leafValues></_>
    <_>
      <internalNodes>
      0 -1 4 1.5014000236988068e-02</internalNodes>
      <leafValues>
      -7.7945697307586670e-01 1.2608419656753540e+00</leafValues></_>
    <_>
      <internalNodes>
      0 -1 5 9.9371001124382019e-02</internalNodes>
      <leafValues>
      5.5751299858093262e-01 -1.8743000030517578e+00</leafValues></_>
    <_>
      <internalNodes>
      0 -1 6 2.7340000960975885e-03</internalNodes>
      <leafValues>
      -1.6911929845809937e+00 4.4009700417518616e-01</leafValues></_>
    <_>
      <internalNodes>
      0 -1 7 -1.8859000876545906e-02</internalNodes>
      <leafValues>
      -1.4769539833068848e+00 4.4350099563598633e-01</leafValues></_>
    <_>
      <internalNodes>
      0 -1 8 5.9739998541772366e-03</internalNodes>
      <leafValues>
      -8.5909199714660645e-01 8.5255599021911621e-01</leafValues></_></weakClassifiers></_>
  <_>
    <maxWeakCount>16</maxWeakCount>
    <stageThreshold>-4.9842400550842285e+00</stageThreshold>
    <weakClassifiers>
      <_>
        <internalNodes>
        0 -1 9 -2.1110000088810921e-02</internalNodes>
        <leafValues>
```

1.2435649633407593e+00 -1.5713009834289551e+00</leafValues></_>
<_>
<internalNodes>
0 -1 10 2.0355999469757080e-02</internalNodes>
<leafValues>
-1.6204780340194702e+00 1.1817760467529297e+00</leafValues></_>
<_>
<internalNodes>
0 -1 11 2.1308999508619308e-02</internalNodes>
<leafValues>
-1.9415930509567261e+00 7.0069098472595215e-01</leafValues></_>
<_>
<internalNodes>
0 -1 12 9.1660000383853912e-02</internalNodes>
<leafValues>
-5.5670100450515747e-01 1.7284419536590576e+00</leafValues></_>
<_>
<internalNodes>
0 -1 13 3.6288000643253326e-02</internalNodes>
<leafValues>
2.6763799786567688e-01 -2.1831810474395752e+00</leafValues></_>
<_>
<internalNodes>
0 -1 14 -1.9109999760985374e-02</internalNodes>
<leafValues>
-2.6730210781097412e+00 4.5670801401138306e-01</leafValues></_>
<_>
<internalNodes>
0 -1 15 8.2539999857544899e-03</internalNodes>
<leafValues>
-1.0852910280227661e+00 5.3564202785491943e-01</leafValues></_>
<_>
<internalNodes>
0 -1 16 1.8355000764131546e-02</internalNodes>
<leafValues>
-3.5200199484825134e-01 9.3339198827743530e-01</leafValues></_>

```
<_>
  <internalNodes>
    0 -1 17 -7.0569999516010284e-03</internalNodes>
  <leafValues>
    9.2782098054885864e-01 -6.6349899768829346e-01</leafValues></_>
<_>
  <internalNodes>
    0 -1 18 -9.8770000040531158e-03</internalNodes>
  <leafValues>
    1.1577470302581787e+00 -2.9774799942970276e-01</leafValues></_>
<_>
  <internalNodes>
    0 -1 19 1.5814000740647316e-02</internalNodes>
  <leafValues>
    -4.1960600018501282e-01 1.3576040267944336e+00</leafValues></_>
<_>
  <internalNodes>
    0 -1 20 -2.0700000226497650e-02</internalNodes>
  <leafValues>
    1.4590020179748535e+00 -1.9739399850368500e-01</leafValues></_>
<_>
  <internalNodes>
    0 -1 21 -1.3760800659656525e-01</internalNodes>
  <leafValues>
    1.1186759471893311e+00 -5.2915501594543457e-01</leafValues></_>
<_>
  <internalNodes>
    0 -1 22 1.4318999834358692e-02</internalNodes>
  <leafValues>
    -3.5127198696136475e-01 1.1440860033035278e+00</leafValues></_>
<_>
  <internalNodes>
    0 -1 23 1.0253000073134899e-02</internalNodes>
  <leafValues>
    -6.0850602388381958e-01 7.7098500728607178e-01</leafValues></_>
<_>
```

```
      <internalNodes>
        0 -1 24 9.1508001089096069e-02</internalNodes>
      <leafValues>
        3.8817799091339111e-01 -1.5122940540313721e+00</leafValues></_></weakClassifiers></_>
    <_>
      <maxWeakCount>27</maxWeakCount>
      <stageThreshold>-4.6551899909973145e+00</stageThreshold>
      <weakClassifiers>
        <_>
          <internalNodes>
            0 -1 25 6.9747000932693481e-02</internalNodes>
          <leafValues>
            -1.0130879878997803e+00 1.4687349796295166e+00</leafValues></_>
        <_>
          <internalNodes>
            0 -1 26 3.1502999365329742e-02</internalNodes>
          <leafValues>
            -1.6463639736175537e+00 1.0000629425048828e+00</leafValues></_>
        <_>
          <internalNodes>
            0 -1 27 1.4260999858379364e-02</internalNodes>
          <leafValues>
            4.6480301022529602e-01 -1.5959889888763428e+00</leafValues></_>
        <_>
          <internalNodes>
            0 -1 28 1.4453000389039516e-02</internalNodes>
          <leafValues>
            -6.5511900186538696e-01 8.3021801710128784e-01</leafValues></_>
        <_>
          <internalNodes>
            0 -1 29 -3.0509999487549067e-03</internalNodes>
          <leafValues>
            -1.3982310295104980e+00 4.2550599575042725e-01</leafValues></_>
        <_>
          <internalNodes>
            0 -1 30 3.2722998410463333e-02</internalNodes>
```

<leafValues>
-5.0702601671218872e-01 1.0526109933853149e+00</leafValues></_>
<_>
<internalNodes>
0 -1 31 -7.2960001416504383e-03</internalNodes>
<leafValues>
3.6356899142265320e-01 -1.3464889526367188e+00</leafValues></_>
<_>
<internalNodes>
0 -1 32 5.0425000488758087e-02</internalNodes>
<leafValues>
-3.0461400747299194e-01 1.4504129886627197e+00</leafValues></_>
<_>
<internalNodes>
0 -1 33 4.6879000961780548e-02</internalNodes>
<leafValues>
-4.0286201238632202e-01 1.2145609855651855e+00</leafValues></_>
<_>
<internalNodes>
0 -1 34 -6.9358997046947479e-02</internalNodes>
<leafValues>
1.0539360046386719e+00 -4.5719701051712036e-01</leafValues></_>
<_>
<internalNodes>
0 -1 35 -4.9033999443054199e-02</internalNodes>
<leafValues>
-1.6253089904785156e+00 1.5378999710083008e-01</leafValues></_>
<_>
<internalNodes>
0 -1 36 8.4827996790409088e-02</internalNodes>
<leafValues>
2.8402999043464661e-01 -1.5662059783935547e+00</leafValues></_>
.
(30000 lines in between)
.
.

<_>
<rects>
<_>
0 13 18 3 -1.</_>
<_>
0 14 18 1 3.</_></rects></_>
<_>
<rects>
<_>
15 17 9 6 -1.</_>
<_>
15 19 9 2 3.</_></rects></_>
<_>
<rects>
<_>
0 17 9 6 -1.</_>
<_>
0 19 9 2 3.</_></rects></_>
<_>
<rects>
<_>
12 17 9 6 -1.</_>
<_>
12 19 9 2 3.</_></rects></_>
<_>
<rects>
<_>
3 17 9 6 -1.</_>
<_>
3 19 9 2 3.</_></rects></_>
<_>
<rects>
<_>
16 2 3 20 -1.</_>
<_>
17 2 1 20 3.</_></rects></_>

```
<_>
<rects>
<_>
0 13 24 8 -1.</_>
<_>
0 17 24 4 2.</_></rects></_>
<_>
<rects>
<_>
9 1 6 22 -1.</_>
<_>
12 1 3 11 2.</_>
<_>
9 12 3 11 2.</_></rects></_></features></cascade>
</opencv_storage>
```


Developing and Fine-Tuning Deep Learning Models for Emotion Recognition:

Emotion recognition is a pivotal area of computer vision with diverse applications. This article focuses on the development and fine-tuning of deep learning models for accurate emotion recognition, outlining the essential steps in the process,

Data Preprocessing:

The article begins by illustrating the importance of data preprocessing in training emotion recognition models. It discusses techniques such as data augmentation using Keras' ImageDataGenerator to enhance the model's ability to recognize emotions from various facial expressions.

Architecture Selection:

Readers are introduced to a variety of pre-trained architectures, including VGG16, ResNet50, Xception, and Inception, which serve as the foundation for emotion recognition models. The choice of architecture depends on the specific requirements of the application.

Fine-Tuning for Optimal Performance:

A critical step in model development is fine-tuning, which involves making the model adaptable to the task at hand. The article outlines the process of selecting and configuring the layers that need to be retrained.

Monitoring and Evaluation:

Monitoring model performance is emphasized throughout the article. It showcases the use of Matplotlib for visualizing training and validation metrics, providing developers with insights into how their models are progressing.

Saving and Reusing Models:

Developers are guided on saving their trained models for future use, enabling them to deploy these models in various applications with consistent performance.

```
from glob import glob


from keras import Model
from keras.callbacks import EarlyStopping
from keras.layers import Flatten, Dense
from keras.models import save_model
from keras.optimizer_v2.gradient_descent import SGD
from keras_preprocessing.image import ImageDataGenerator




def get_data(parameters, preprocess_input: object) -> tuple:
    image_gen = ImageDataGenerator(
        # rescale=1 / 127.5,
        rotation_range=20,
        zoom_range=0.05,
        shear_range=10,
        horizontal_flip=True,
        fill_mode="nearest",
        validation_split=0.20,
```

```
        preprocessing_function=preprocess_input,
    )

    # create generators
    train_generator = image_gen.flow_from_directory(
        parameters["train_path"],
        target_size=parameters["shape"],
        shuffle=True,
        batch_size=parameters["batch_size"],
    )

    test_generator = image_gen.flow_from_directory(
        parameters["test_path"],
        target_size=parameters["shape"],
        shuffle=True,
        batch_size=parameters["batch_size"],
    )

    return (
        glob(f"{parameters['train_path']}/*/*.jp*g"),
        glob(f"{parameters['test_path']}/*/*.jp*g"),
        train_generator,
        test_generator,
    )


def fine_tuning(model: Model, parameters):
    # fine tuning
    for layer in model.layers[: parameters["number_of_last_layers_trainable"]]:
        layer.trainable = False
    return model


def create_model(architecture, parameters):
    model = architecture(
        input_shape=parameters["shape"] + [3],
```

```
        weights="imagenet",
        include_top=False,
        classes=parameters["nbr_classes"],
    )

    # Freeze existing VGG already trained weights
    for layer in model.layers[: parameters["number_of_last_layers_trainable"]]:
        layer.trainable = False

    # get the VGG output
    out = model.output

    # Add new dense layer at the end
    x = Flatten()(out)
    x = Dense(parameters["nbr_classes"], activation="softmax")(x)

    model = Model(inputs=model.input, outputs=x)

    opti = SGD(
        lr=parameters["learning_rate"],
        momentum=parameters["momentum"],
        nesterov=parameters["nesterov"],
    )

    model.compile(loss="categorical_crossentropy", optimizer=opti, metrics=["accuracy"])

    # model.summary()

    return model


def fit(model, train_generator, test_generator, train_files, test_files, parameters):
    early_stop = EarlyStopping(monitor="val_accuracy", patience=2)
    return model.fit(
        train_generator,
        validation_data=test_generator,
```

```
        epochs=parameters["epochs"],
        steps_per_epoch=len(train_files) // parameters["batch_size"],
        validation_steps=len(test_files) // parameters["batch_size"],
        callbacks=[early_stop],
    )



def evaluation_model(model, test_generator):
    score = model.evaluate_generator(test_generator)
    print("Test loss:", score[0])
    print("Test accuracy:", score[1])
    return score



def saveModel(filename, model):
    save_model(model=model, filepath=f"./trained_models/{filename}")
    model.save_weights(f"./trained_models/{filename}.h5")
```

Conclusion:

In the second phase of our image recognition development, we honed our focus on essential aspects of data preparation, Haar Cascade Classifier for face detection, and the creation and fine-tuning of deep learning models for emotion recognition. We recognized the paramount importance of well-organized and preprocessed datasets, which provide a solid foundation for training accurate image recognition models. The exploration of the Haar Cascade Classifier shed light on its exceptional capability in efficiently detecting faces in images, making it a valuable tool for a multitude of computer vision applications. Our journey also delved deep into the development of powerful deep learning models, leveraging pre-trained architectures like VGG16, ResNet50, Xception, and Inception, and refining them for emotion recognition. These models are poised to deliver impressive levels of accuracy. As we progress, these acquired skills open doors to a myriad of practical applications, from improving human-computer interaction to revolutionizing fields like healthcare and more. Our journey

continues to uncover the vast potential of image recognition technology.