**A close-up of a logo

Description automatically generated**

**KARPAGAM INSTITUTE OF TECHNOLOGY**

**CLOUD APPLICATION DEVELOPMENT**

**Project: Image Recognition with IBM Cloud Visual Recognition**

**Phase 4 Submission Document**

**A person smiling with her hand on her chin

Description automatically generated**

**Introduction:**

* In this second development phase, we embark on the integration of face detection and emotion recognition, creating a holistic image recognition system.
* At the core of this effort lies the "Haar Cascade Classifier," an advanced computer vision technology primarily designed for detecting frontal faces in images.
* This phase also includes a set of code snippets designed to streamline the development of this system.
* It commences with the preparation of a machine learning model for facial emotion recognition, configuring data generators, building and fine-tuning the deep learning model, and preparing it for training.
* Users are provided with the choice of selecting from a variety of pre-trained deep learning models for emotion recognition, each with its unique architecture.
* The training and evaluation process constantly monitors the model's performance, with early stopping mechanisms and informative visualizations of accuracy and loss.
* Moreover, the option to save the trained model and associated performance metrics enhances the practicality of this multifaceted image recognition project.
* This phase bridges the realms of face detection and emotion recognition, offering a comprehensive solution for image analysis and comprehension.

**Data Sources and Setup:**

To prepare for running the scripts in the image recognition project, it's essential to configure the necessary datasets. This document offers insight into the datasets utilized in our project and provides instructions on acquiring them.

**Datasets:**

**CK+ (Cohn-Kanade Extended+):**

**Source:**

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

**Description**:

The CK+ dataset comprises facial expressions recorded in controlled laboratory settings, offering a valuable resource for training and evaluating emotion recognition models due to its inclusion of seven distinct emotion labels.

**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 compilation of images depicting a range of facial expressions, encompassing diverse emotional states. This dataset facilitates thorough training and testing of emotion recognition models.

**FERPlus:**

**Source:**

<https://github.com/microsoft/FERPlus>

**Description**:

FERPlus is an extension of the FER-13 dataset, offering enhanced emotion annotations. It includes additional labels, providing a higher level of detail and granularity for emotion recognition tasks.

**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:**

* Data preprocessing holds a crucial position in the realm of deep learning-based emotion recognition, significantly impacting the efficacy of the models.
* This document provides a comprehensive overview of the fundamental data preprocessing procedures required 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. change sentence.
* format.

**Emotions are mapped into seven primary categories:**

neutral, happy, surprise, sad, angry, disgust, and fear.

**Data Reorganization:**

**Transfer Images:**

* Images are relocated from the original FERPlus directory structure to conform with the FER-2013 dataset structure.
* The categorization of images into training and test sets is determined by the "Usage" attribute specified in the CSV file.

**Emotion-Based Sorting:**

* Images are subsequently organized into subfolders within the training and test sets, based on the primary emotion category they represent.

**Execution:**

* The provided Python script automates the data preprocessing procedures detailed above, ensuring 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 refined our focus on critical aspects of data preparation, employed the Haar Cascade Classifier for effective face detection, and dedicated efforts to craft and optimize deep learning models for emotion recognition. We deeply acknowledged the pivotal role of well-organized and preprocessed datasets, which form a robust foundation for training highly accurate image recognition models.
* Our exploration into the Haar Cascade Classifier unveiled its exceptional proficiency in swiftly detecting faces within images, positioning it as a valuable tool with a multitude of applications in the realm of computer vision. Our journey further delved into the creation of potent deep learning models, capitalizing on pre-existing architectures such as VGG16, ResNet50, Xception, and Inception, and fine-tuning them for precise emotion recognition. These models are poised to deliver remarkable levels of accuracy.
* As we advance, these acquired skills open doors to a myriad of practical applications, ranging from enhancing human-computer interaction to catalyzing transformations in domains like healthcare and beyond. Our ongoing journey continues to reveal the vast potential inherent in image recognition technology.