**Image recognition with IBM cloud visual recognition**

**Procedure :**

Step 1: Login to Your Account

Step 2: Create a New Resource On your dashboard page, click on the Create a resource on the top right to create a new source.

Step 3: Create a Visual Recognition and Watson Studio Resource On the Catalog page, select the AI category from the left pane, and then select the Visual Recognition resource. On the next page, you will get to name your service instance and choose your region. Click on the arrow to reveal the drop- down menu of regions. Make sure to select the region that is closest to you. Then scroll down and make sure that the liteplan is selected, and click the Create button. On the Catalog page, select the AI category from the left pane, and then select the Watson Studio resource. On the next page, you will get to name your service instance and choose your region. Click on the arrow to reveal the drop-down menu of regions. Make sure to select the region that is closest to you. Then scroll down and make sure that the lite plan is selected, and click the Create button. On the next page, click the Get Started button to start using Watson Studio.Once the provisioning process is complete, click the Get Started button to start using Watson Studio.

Step 4: Create a Project

Once you land on the IBM Watson Studio main page, start by creating a project. Choose create empty project.

Step 5:

Set up Your Project .fill in some project details and click Create. The IBM Cloud Object Storage, which provides you storage for your images, should be automatically created for you but if not created you can click on add button below the heading on top right part of screen. if it is already created it will show you name and will not let you create a new one. Create the storage service with the lite plan and click on create button. Go to Add to project and choose Image classification.Then provision a visual recognition service.

Step 6:

Selecting Built-in Models for Watson Visual Recognition After creating your project, by default, you will land on the page where you can perform some advanced tasks but we will skip this for now and use the built-in models. To access the built-in models, click on the name of the service, as seen in the red box blow

Step 7: Choose the General model. Now you can see all the built-in image classification models that IBM Watson provides! Let’s try the General model.

Step 8: Try out the General model

To test the General model, click on Test tab on top of screen.

Step 9: Upload Your Images!.Now you can upload any images you’d like by clicking on Browse. Remember that you will have to upload all the images that you want to test in a single go as once you upload images then you cannot add more images one by one. you will have to go back and upload all images including the new ones.

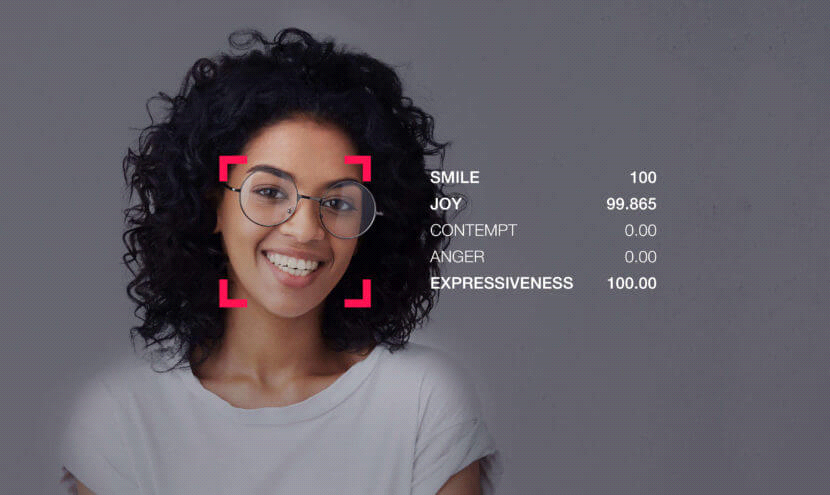
Step 10:

Check Out the Results! .Once you have uploaded your images, it will tell you what it found in your images! Beside each class of object (or color, age, etc.), it will also give you a confidence score (between 0 and 1) on how confident it thinks it found that particular object in your image (0 for lowest confidence and 1 for highest confidence).

Step 11:

Filter Your Results .You can even filter your results based on tag generated from images by selecting the tags from list in left hand side of screen.

**Facial-emotion-recognition-ai**



**Introduction:**

* In this phase of our image recognition project, we will be focusing on creating the user interface (UI) that allows users to interact with our AI-powered image recognition system.
* We will utilize HTML, CSS, and JavaScript to craft a visually appealing and user-friendly frontend that enables users to upload images and view the captions generated by our AI model.

**Project Context:**

* Our goal is to build a simple and intuitive web application that makes image recognition accessible to users.
* By leveraging the power of artificial intelligence, our system will provide descriptive captions for images, enhancing accessibility and understanding.

**Part 1 Objectives:**

**1.HTML Structure:**

We'll create an HTML structure that defines the layout of our web page. This structure will include components for image upload, caption display, and user interactions.

**2.CSS Styling:**

CSS will be used to style our interface, ensuring it's visually appealing, responsive, and aligns with our project's branding and usability standards.

**3.JavaScript Functionality:**

JavaScript will add interactivity to the UI. It will enable us to handle user actions, such as uploading images, and facilitate communication with the backend for image processing.

**Development Part 1:**

**HTML code:**

<!doctype html>

<html lang="fr">

<head>

<title>Emotion Recognition</title>

<link rel="stylesheet" href="../staticFiles/index.css" type="text/css">

</head>

<body>

<h1>Emotion Recognition</h1>

<div class="container">

<div class="left">

<p id="clock">no faces found</p>

<script>

const clock = document.getElementById("clock");

setInterval(() => {

fetch("{{ url\_for('time\_feed') }}")

.then(response => {

response.text().then(t => {

clock.innerHTML = t

})

});

}, 10);

</script>

</div>

<div class="right">

<div class="col-lg-8 offset-lg-2">

<img src="{{ url\_for('video\_feed') }}" height="80%" alt="please wait...">

</div>

</div>

</div>

</body>

</html>

**Description:**

**<!doctype html>:**

This declaration defines the document type as HTML5, indicating that the page follows modern HTML standards.

**<html lang="fr">:**

This opening tag specifies that the document is written in French (the "fr" language code). This helps search engines and browsers understand the language of the content.

**<head>:**

Within the head section, metadata and links to external resources are typically placed.

**<title>Emotion Recognition</title>:**

This sets the title of the web page, which is displayed in the browser's title bar or tab. In this case, the title is "Emotion Recognition."

**<link rel="stylesheet" href="../staticFiles/index.css" type="text/css">:**

This line links an external CSS stylesheet to the HTML document. The stylesheet is located at **"../staticFiles/index.css**" and is used to style the content of the web page.

**<body>:**

This is the main content area of the web page.

**<h1>Emotion Recognition</h1>:**

This is the main heading of the page, displaying "Emotion Recognition" as the title at the top of the web page.

**<div class="container">:**

This is a container that likely helps structure the content and layout of the page.

**<div class="left">:**

This div contains content that appears on the left side of the page.

**<p id="clock">no faces found</p>:**

This paragraph element with the ID "clock" initially displays "no faces found." It is likely intended to display real-time information.

The following JavaScript code updates the content of the "clock" element at regular intervals. It appears to fetch data from a URL (presumably related to time or real-time data) and updates the "clock" element with the fetched data. The fetch request is sent every 10 milliseconds.

**<div class="right">:**

This div contains content that appears on the right side of the page.

**<div class="col-lg-8 offset-lg-2">:**

This div is part of a grid or layout system (likely from a CSS framework like Bootstrap). It has a specific width ("col-lg-8") and offset ("offset-lg-2").

**<img src="{{ url\_for('video\_feed') }}" height="80%" alt="please wait...">:**

This image element displays an image sourced from the URL generated by the url\_for('video\_feed') Flask function. The image has a specified height of 80%, and the alt text is "please wait...," which is displayed if the image cannot be loaded or is still loading.

**CSS CODE:**

.container{

display: inline-flex;

width: 90%;

margin-left:5%;

margin-right:5%;

}

html{

background-color:#161a1b;

}

.left{

float: left;

background-color: lightgray;

width:30%;

padding:4%;

font-size:220%;

}

.right{

border:none;

float: right;

width:70%;

text-align: center;

}

.right img {

width:100%;

}

h1{

text-align: center;

color:white;

}

**Description:**

**.container:**

This CSS class is used to style a container element.

**display: inline-flex;:**

It sets the display property to "inline-flex," which makes the container an inline-level flex container. This allows its child elements to be displayed in a row, and they will adjust their sizes based on content.

**width: 90%;:**

The container occupies 90% of the available width of its parent element.

**margin-left: 5%; and margin-right: 5%;:**

These properties set left and right margins to 5%, which create spacing on the left and right sides of the container.

**html:**

These styles are applied to the <html> element, affecting the entire web page.

**background-color: #161a1b;:**

It sets the background color of the HTML element to a dark grayish color (#161a1b).

**.left:**

This class is used to style the left part of the content.

**float: left;:**

The "left" content is floated to the left side of its containing element. This allows the content on the right to flow around it.

**background-color: lightgray;:**

The background color of the left content is set to light gray.

**width: 30%;:**

The left content takes up 30% of the available width.

**padding: 4%;:**

It adds a 4% padding on all sides of the left content, creating spacing between the content and its container.

**font-size: 220%;:**

The font size for the left content is set to 220% of the default font size, making it quite large.

**.right:**

This class is used to style the right part of the content.

**border: none;:**

It removes the border from the right content.

**float: right;:**

The "right" content is floated to the right side of its containing element, allowing it to sit next to the left content.

**width: 70%;:**

The right content takes up 70% of the available width.

**text-align: center;:**

Text within the right content is centered horizontally.

**.right img:**

This selector targets <img> elements within the "right" content.

**width: 100%;:**

Images within the "right" content are scaled to occupy the full width of their container.

**h1:**

This selector targets <h1> elements.

**text-align: center;:**

The text in <h1> elements is centered horizontally.

**color: white;:**

The text color of <h1> elements is set to white.

**JS code:**

const clock = document.getElementById("clock");

setInterval(() => {

fetch("{{ url\_for('time\_feed') }}")

.then(response => {

response.text().then(t => {

clock.innerHTML = t

})

});

}, 2000);

**Description:**

**const clock = document.getElementById("clock");:**

This line retrieves an HTML element with the ID "clock" and stores it in a JavaScript variable called "clock." This element is likely a part of the webpage where you want to display real-time data, such as a clock or timestamp.

**setInterval(() => {...}, 2000);:**

This is a JavaScript function that repeatedly executes the provided code block at the specified interval in milliseconds. In this case, the code block is executed every 2000 milliseconds (2 seconds).

**fetch("{{ url\_for('time\_feed') }}"):**

Inside the interval function, a "fetch" request is made to a URL generated using Flask's url\_for function with the endpoint "time\_feed." The purpose of this request is to retrieve some real-time data, possibly a timestamp or clock data, from the server.

**.then(response => {...}):**

This is a promise that handles the response from the fetch request. It specifies what to do when the response is received.

**response.text().then(t => { clock.innerHTML = t });:**

This part of the code processes the response from the server. It converts the response to text and then updates the content of the HTML element with the ID "clock" (as stored in the "clock" variable) with the text received from the server. This action effectively updates the displayed content with the real-time data obtained from the server.

**Conclusion:**

By focusing on the frontend development in Part 1, we aim to provide users with an engaging and easy-to-use interface that seamlessly integrates with the backend, where our image recognition model does the heavy lifting. The result will be a user-friendly image recognition application that empowers users to understand and describe images with ease.

**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.