IBM NAAN MUDHALVAN

IMAGE RECOGNITION WITH IBM CLOUD VISUAL RECOGNITION

Introduction:

Image recognition, a subset of computer vision, involves the identification and classification of objects, people, places, or elements within an image. It encompasses various techniques like convolutional neural networks (CNNs) and deep learning models to analyse and interpret visual data. This submission aims to demonstrate the efficacy of image recognition in accurately identifying and categorizing objects within images, showcasing its potential applications in diverse fields like healthcare, automotive, security, and more.

Overview:

Image recognition is the task of identifying objects of interest in an image and identifying to which class the image belongs. Although various human vision simulation methods have been developed, a common goal of image recognition machine learning projects is classifying recognized objects into different classes, otherwise known as object detection.

The adoption of image recognition applications has been on the rise and has accelerated even further due to COVID-19. The global image recognition market size is projected to reach USD 86.32 billion by 2027, exhibiting a growth of 17.6% during the forecast period according to the lates report by Fortune business insights. With 60% projected growth in North America alone and a 116% increase in job demand for computer vision engineers, practicing image recognition machine learning projects is important for any aspiring data scientist.

Key components of the project may include:

Data Collection: Gathering a diverse and representative dataset of images for training and testing the model.
Model Training: Utilizing IBM Cloud Visual Recognition's pre-trained models or customizing models to suit specific recognition needs.
Integration: Embedding the trained model into applications or systems to automate image recognition tasks.
Application Scenarios: Implementing image recognition in various domains, such as healthcare, e-commerce, security, and more.
Performance Evaluation: Assessing the accuracy, precision, recall, and other metrics to ensure the model's effectiveness

Problem Definition:

Identify the Problem:

Define the specific problem you want to solve with image recognition. For instance, you might want to build an image recognition system for a social media platform to help users identify objects, people, or places in photos they upload.

Scope and Objectives:

Clearly outline the scope of your image recognition project. Determine your main objectives, such as enhancing user experience, improving content discovery, or enabling accessibility.

Data Collection:

Gather a diverse and well-labelled dataset that aligns with your problem. Ensure that your dataset contains a variety of images relevant to the application. In our case, this dataset might include photos with various objects, scenes, and people.

Performance Metrics:

Define performance metrics like accuracy, precision, recall, and F1 score to evaluate the effectiveness of your image recognition model.

Design Thinking:

Empathize:

Understand the needs, preferences, and expectations of your users. Conduct user research, surveys, or interviews to gather insights about how they would like to interact with image recognition.

Define:

Create user personas and user stories to define the problem and its context from the user's perspective. Identify pain points and opportunities for improvement.

Ideate:

Brainstorm creative solutions for integrating image recognition into your application. Explore different ways to leverage IBM Cloud Visual Recognition, such as in content tagging, search, or recommendations.

Prototype:

Develop prototypes of your user interface that incorporate image recognition features. These prototypes should allow users to interact with the system and see how it works in practice. Consider wireframes and mock-ups.

Test and Iterate:

Collect user feedback on your prototypes. Iterate on your designs based on their input, refining the user interface and improving the overall user experience.

Image Recognition Setup:

In titrate IBM Cloud Visual Recognition into your application or platform. Set up the necessary APIs and authentication to enable image recognition capabilities.
Train your image recognition models using the collected dataset to ensure accurate classification and object detection.

User Interface:

Design a user-friendly interface that incorporates image recognition features seamlessly. Ensure that users can easily upload images and receive recognition results.

Consider incorporating visual feedback, such as highlighting recognized objects or providing information about them in a user-friendly format.

Image Classification:

Implement image classification using IBM Cloud Visual Recognition. When users upload images, the system should classify and identify objects or subjects within them.

Display classification results in an organized and intuitive manner within the user interface.

AI-Generated Captions:

Enhance user engagement by providing AI-generated captions or descriptions for images. This feature can be particularly helpful for users with visual impairments.

Use natural language processing (NLP) techniques to generate meaningful and contextually relevant captions.

User Engagement:

Foster user engagement by continuously improving the accuracy and speed of your image recognition system.
Encourage user feedback and iterate on your system based on their input. Consider gamification or social sharing features to enhance user involvement.

Empathize with users to understand their motivations and pain points related to user engagement with image recognition systems.

Define user engagement goals, such as increasing user interactions with an image-based mobile app.

Problem Statement:

The design for incorporating sentiment analysis into IBM Cloud Visual Recognition to enhance the image recognition capabilities.

The goal is to generate captions that not only describe the objects in the images but also capture the emotions and mood portrayed within them.

This innovative approach can be valuable for various applications, including social media, content creation, and sentiment analysis.

Traditional image recognition systems excel at identifying objects and patterns within images but often lack the ability to understand the emotional context. Users businesses require more than just object recognition; they want to know the sentiment and mood conveyed by images.

Proposed Solution:

To address this issue, we propose the integration of sentiment analysis into IBM Cloud Visual Recognition. This integration will allow the system to analyse the images and generate captions that include emotional and mood context.

Image Analysis:

Utilize IBM Cloud Visual Recognition's deep learning algorithms to analyse images and identify objects and patterns within them.

Sentiment Analysis:

Integrate sentiment analysis models (similar to those in IBM Watson NLP) to assess the emotional tone of the images.

Caption Generation:

Combine the results of image recognition and sentiment analysis to generate captions that provide a comprehensive description of the image, including emotional context.

User Interaction:

Allow users to access these image captions, making it easier for them to understand the image's content and mood.

Incorporating Sentiment Analysis for Image Captions:

Integration of Sentiment Analysis

To add a new layer of understanding to the images, we will integrate sentiment analysis using IBM Watson NLP or a similar tool.

This will analyse the textual content within images or associated metadata to determine the sentiment, i.e., whether the image conveys positive, negative, or neutral emotions.

Generating Emotion-Centric Captions:

After determining the sentiment, we will generate captions for images that not only describe their content but also capture the emotional tone.

For example, for an image with a positive sentiment, the caption might express happiness or positivity.

Customization and Fine-Tuning:

Depending on the specific use case, we can fine-tune the sentiment analysis model to align with the nuances of the domain.

Customizing the sentiment analysis ensures more accurate results.

Application in Various Sectors:

This innovation can find applications in diverse sectors such as e-commerce, social media, healthcare, and more.

For instance, in e-commerce, it can help in selecting product images with the right emotional appeal for marketing campaigns.

Evaluation and Optimization:

Continuously monitor and optimize the sentiment analysis model to improve accuracy and adapt to changing trends and user preferences.

Benefits

- **Enhanced Image Understanding**:

Users can gain deeper insights into the emotional context of images, making it useful for social media, marketing, and content creation.

- **Improved User Engagement**:

Images with mood-based captions are more engaging and can lead to higher user interaction.

- **Data-driven Decision Making**:

Businesses can use sentiment data from images for market research and sentiment analysis.

IBM Cloud account, set up the Visual Recognition service:

I can provide you with an overview of the steps to create an IBM Cloud account, set up the Visual Recognition service, and obtain API keys. Please note that the exact steps may vary over time, so it's essential to refer to the most up-to-date IBM Cloud documentation. As of my last update in September 2021, here are the general steps:

Creating an IBM Cloud Account:

1. **Visit IBM Cloud Website:**
   - Go to the IBM Cloud website.
2. **Sign Up:**
   - Click the "Sign Up" button. You'll need to provide your email address, name, and other required information.
3. **Verification:**
   - IBM will send a verification email to the address you provided. Follow the link in the email to verify your account.
4. **Create an IBM ID:**
   - If you don't already have an IBM ID, you'll be prompted to create one during the sign-up process.
5. **Provide Billing Information (If required):**
   - Depending on the services you plan to use, you may need to enter billing information even if you are using free services. IBM may ask for credit card information for verification purposes.

Setting Up Visual Recognition Service and Obtaining API Keys:

1. **Log in to IBM Cloud:**

- After creating an IBM Cloud account, log in to your IBM Cloud Dashboard.

2. Create a Visual Recognition Service:

- Click "Create Resource" or "Create Service" to access the IBM Cloud Catalog.
- In the Catalog, search for "Visual Recognition" and select it.
- Follow the prompts to create your service instance.

3. Service Configuration:

- Choose a plan, such as "Lite" if you want to start with the free plan.
- Assign a service name.
- Review the service configuration and make any necessary adjustments.

4**. Create Service Credentials:**

- Once your Visual Recognition service is created, go to your service instance from the IBM Cloud Dashboard.
- In the service dashboard, navigate to the "Service credentials" tab.

5**. Generate API Key and URL:**

- Click "New Credential" or "Create Credential" to generate API keys.
- The service will create a set of credentials that include an API Key, URL, and potentially other information.

6. Store Your API Key and URL:

- Make sure to securely store your API Key and URL. You will need these to authenticate and make API requests to the IBM Cloud Visual Recognition service

Design a simple web interface

Designing a simple web interface for users to upload images and view AI-generated captions involves combining front-end development for the user interface with back-end integration to process the images and obtain captions.

Here's a basic outline of how you can create such a web interface using HTML, CSS, and JavaScript for the front-end and Python with a pre-trained AI model for the back end. You can further refine and expand on this design as needed.

Front-End (HTML, CSS, JavaScript)

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.

<u>Development Part 1:</u>

<u>HTML code:</u>

```
<!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>
```

<u>Description:</u>

<!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.

<u>CSS CODE:</u>

```
.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.

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+):
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):
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:
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")
```

SUBMISSION

GitHub Repository Link
Share the GitHub repository link where your project's code and files are hosted.

Deployment Instructions
Provide step-by-step instructions on how to deploy the image recognition system using IBM Cloud. This should include setting up any necessary services and configuring the system.

Web Interface
Explain how to use the web interface, including how to upload images, receive AI-generated captions, and any other user interactions.

README File

Create a detailed README file that explains how to navigate the website, update content, and any dependencies. Include clear instructions for users and developers. Please make sure to include relevant code snippets, configurations, and screenshots in your documentation to make it easier for others to understand and use your project

CONCLUSION:

Incorporating sentiment analysis into IBM Cloud Visual Recognition is a powerful way to enrich image recognition capabilities.

By providing captions that capture emotions and mood, this innovation can be a game-changer for various industries.

It enhances user experience and enables data-driven decision-making.

To assess this design, we recommend a detailed implementation plan and collaboration with experts in both image recognition and sentiment analysis fields.

The combination of IBM Cloud Visual Recognition and sentiment analysis holds great potential for revolutionizing image understanding and content creation.