**Machine Learning Applications Project**

**Image Class Prediction & Viewer (Flet + TensorFlow)**

**Project Descriptions:**

This is a desktop/web application built with Flet, TensorFlow, and MobileNetV2 that allows you to:

* Upload an image
* Predict its class using a pre-trained classifier
* Display similar images from a local or database-backed dataset
* Retrain the model directly from the interface

**Features**

* Upload and view images instantly
* Predict class using a Keras model
* Feature extraction using **MobileNetV2**
* Show related images based on prediction
* Retrain model on-the-fly with a button click
* SQLite or PostgreSQL-compatible backend (via db\_utils\_CNN.py)

**UI Overview**

* **Image Preview**: Displays the uploaded image.
* **Prediction Text**: Shows predicted class and confidence.
* **Slider**: Select how many related images to show.
* **Related Images Grid**: Shows thumbnails from dataset.
* **Retrain Button**: Trigger training script (retrain.py) from UI.

**Requirements**

* Python 3.8+
* flet
* tensorflow
* pillow
* numpy
* psycopg2-binary (if using PostgreSQL)
* Pre-trained classifier saved (e.g., model.pkl via joblib)
* Optional: PostgreSQL with a photo\_dataset table (see db\_utils\_CNN.py)

**Project Code Files :**

1. **app\_CNN.py**
2. **db\_utils\_CNN.py**
3. **model\_utils.py**
4. **retrain.py**

**Project Working in Detail and the Codes:**

1. db\_utils\_CNN.py

This python code is used to load the data from the database using CNN to extract features .

***Code***:

import sqlite3

import random

import numpy as np

from PIL import Image

from io import BytesIO

from tensorflow.keras.applications import MobileNetV2

from tensorflow.keras.applications.mobilenet\_v2 import preprocess\_input

DB\_PATH = "photo\_database.sqlite"

cnn\_model = MobileNetV2(include\_top=False, weights='imagenet', pooling='avg', input\_shape=(128, 128, 3))

def load\_dataset\_from\_db():

    X, y = [], []

    conn = sqlite3.connect(DB\_PATH)

    cur = conn.cursor()

    cur.execute("SELECT image\_label, image\_data FROM photo\_dataset")

    for image\_label, image\_data in cur.fetchall():

        try:

            img = Image.open(BytesIO(image\_data)).convert('RGB').resize((128, 128))

            img\_array = np.array(img).astype("float32")

            img\_array = preprocess\_input(img\_array)

            features = cnn\_model.predict(np.expand\_dims(img\_array, axis=0), verbose=0)[0]

            X.append(features)

            y.append(image\_label)

        except Exception as e:

            print(f"Error processing image from DB: {e}")

    conn.close()

    return np.array(X), np.array(y)

def get\_random\_images\_by\_label(image\_label, count=5):

    conn = sqlite3.connect(DB\_PATH)

    cur = conn.cursor()

    cur.execute("SELECT image\_data FROM photo\_dataset WHERE image\_label = ?", (image\_label,))

    rows = cur.fetchall()

    conn.close()

    selected = random.sample(rows, min(count, len(rows)))

    return [row[0] for row in selected]

1. model\_utils.py

This python code is used to train a randomforest model and store it into a file named “model.pkl”

***Code:***

import pickle

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

import numpy as np

MIN\_ACCURACY = 0.35

MODEL\_PATH = "model.pkl"

def train\_model(X, y):

    model = RandomForestClassifier(n\_estimators=100, random\_state=42)

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, stratify=y)

    model.fit(X\_train, y\_train)

    acc = accuracy\_score(y\_test, model.predict(X\_test))

    print(f"Training Accuracy: {acc:.2%}")

    if acc >= MIN\_ACCURACY:

        with open(MODEL\_PATH, "wb") as f:

            pickle.dump(model, f)

        return model, True

    return None, False

def load\_model():

    with open(MODEL\_PATH, "rb") as f:

        return pickle.load(f)

1. app\_CNN.py

This Python code is the main interface for an image classification app using Flet. It allows users to upload an image, predicts its class using a pre-trained model, and displays similar images from a database. It also includes a slider to control image count and a button to retrain the model via a subprocess call to update predictions.

***Code:***

import flet as ft

import numpy as np

from PIL import Image

import base64

import subprocess

from io import BytesIO

from model\_utils import load\_model

from db\_utils\_CNN import get\_random\_images\_by\_label

from tensorflow.keras.applications import MobileNetV2

from tensorflow.keras.applications.mobilenet\_v2 import preprocess\_input

import tensorflow as tf

clf = load\_model()

class\_labels = clf.classes\_

cnn\_model = MobileNetV2(include\_top=False, weights='imagenet', pooling='avg', input\_shape=(128, 128, 3))

def preprocess\_image(path):

    img = Image.open(path).convert('RGB').resize((128, 128))

    img\_array = np.array(img).astype("float32")

    img\_array = preprocess\_input(img\_array)

    features = cnn\_model.predict(np.expand\_dims(img\_array, axis=0), verbose=0)

    return features

def main(page: ft.Page):

    page.title = "Image Class Prediction & Viewer"

    page.scroll = ft.ScrollMode.AUTO

    uploaded\_img = ft.Image(width=300, height=300, fit=ft.ImageFit.CONTAIN)

    status\_text = ft.Text()

    num\_slider = ft.Slider(min=1, max=10, divisions=9, label="{value}", value=5)

    image\_grid = ft.Column(scroll=ft.ScrollMode.AUTO, auto\_scroll=False)

    def display\_images\_in\_rows(images):

        image\_grid.controls.clear()

        row = ft.Row(wrap=True, spacing=10, run\_spacing=10)

        for i, img\_data in enumerate(images):

            encoded = base64.b64encode(img\_data).decode("utf-8")

            img\_widget = ft.Image(

                src\_base64=encoded,

                width=200,

                height=200,

                fit=ft.ImageFit.CONTAIN

            )

            row.controls.append(img\_widget)

            if (i + 1) % 3 == 0:

                image\_grid.controls.append(row)

                row = ft.Row(wrap=True, spacing=10, run\_spacing=10)

        if row.controls:

            image\_grid.controls.append(row)

        page.update()

    def on\_upload(e: ft.FilePickerResultEvent):

        if e.files:

            path = e.files[0].path

            uploaded\_img.src = path

            uploaded\_img.update()

            image\_grid.controls.clear()

            status\_text.value = "Predicting..."

            page.update()

            input\_array = preprocess\_image(path)

            probs = clf.predict\_proba(input\_array)[0]

            best\_idx = np.argmax(probs)

            confidence = probs[best\_idx]

            predicted\_label = class\_labels[best\_idx]

            if confidence < 0.30:

                status\_text.value = f"Prediction confidence too low ({confidence:.2%}). No image shown."

                page.update()

                return

            status\_text.value = f"Predicted Label: {predicted\_label} (Confidence: {confidence:.2%})"

            images = get\_random\_images\_by\_label(predicted\_label, count=int(num\_slider.value))

            display\_images\_in\_rows(images)

            page.update()

    def retrain\_model(e):

        status\_text.value = "Retraining model..."

        page.update()

        result = subprocess.run(["python", "Image\_recognition\\retrain.py"], capture\_output=True, text=True)

        if result.returncode == 0:

            status\_text.value = "Model retrained successfully. Reloading..."

            global clf

            clf = load\_model()

            status\_text.value = "Model reloaded and ready to use."

        else:

            status\_text.value = f"Retraining failed:\n{result.stderr}"

        page.update()

    file\_picker = ft.FilePicker(on\_result=on\_upload)

    page.overlay.append(file\_picker)

    page.add(

        ft.Text("Upload an image:"),

        ft.ElevatedButton("Select Image", on\_click=lambda \_: file\_picker.pick\_files(allow\_multiple=False)),

        uploaded\_img,

        status\_text,

        ft.Text("Number of images to display:"),

        num\_slider,

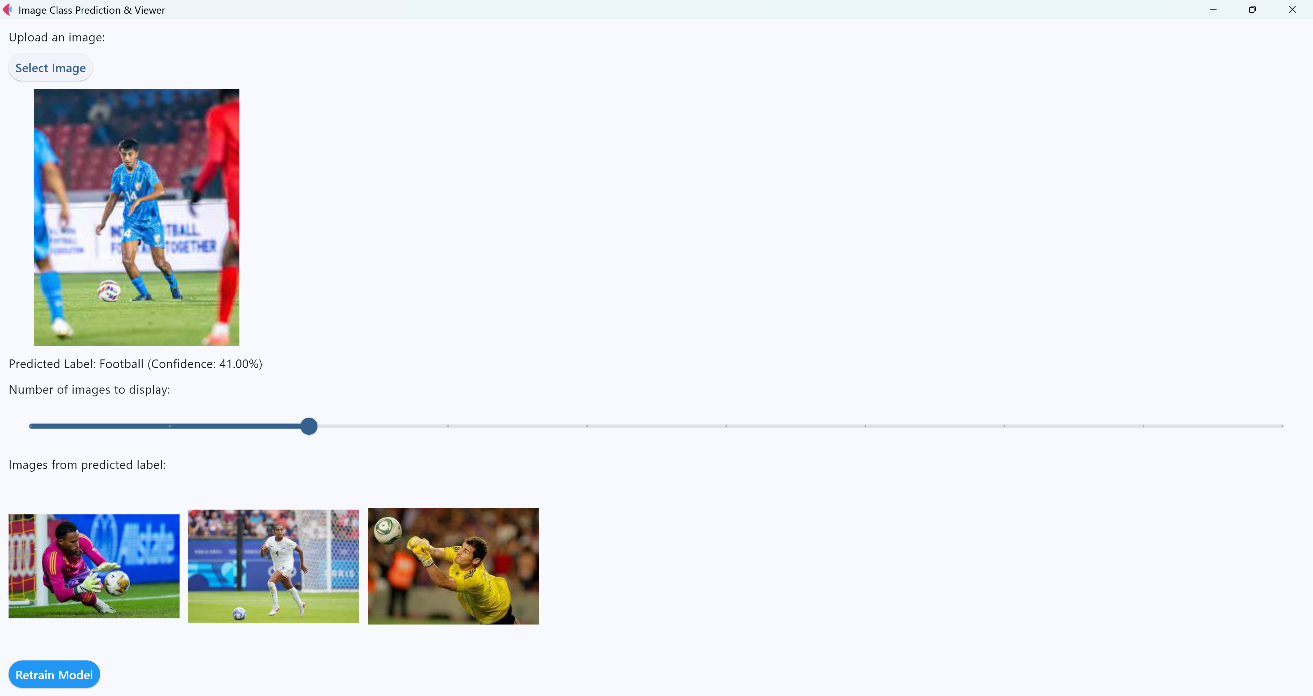
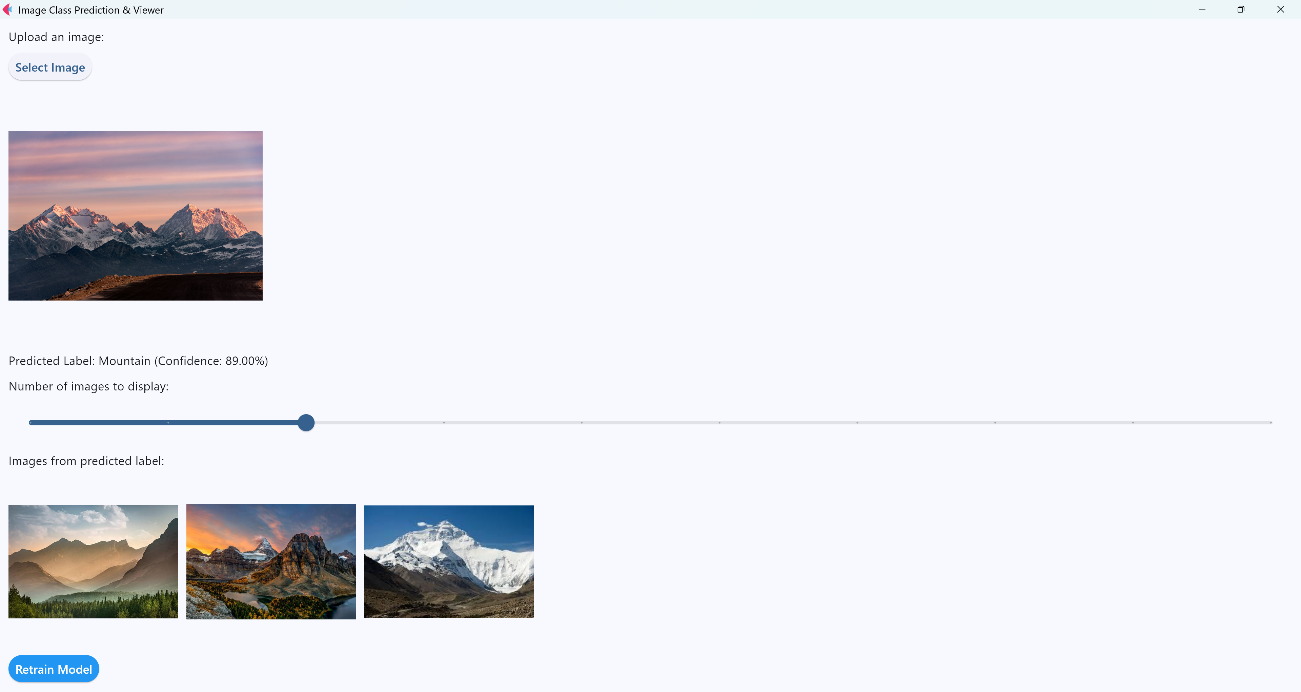
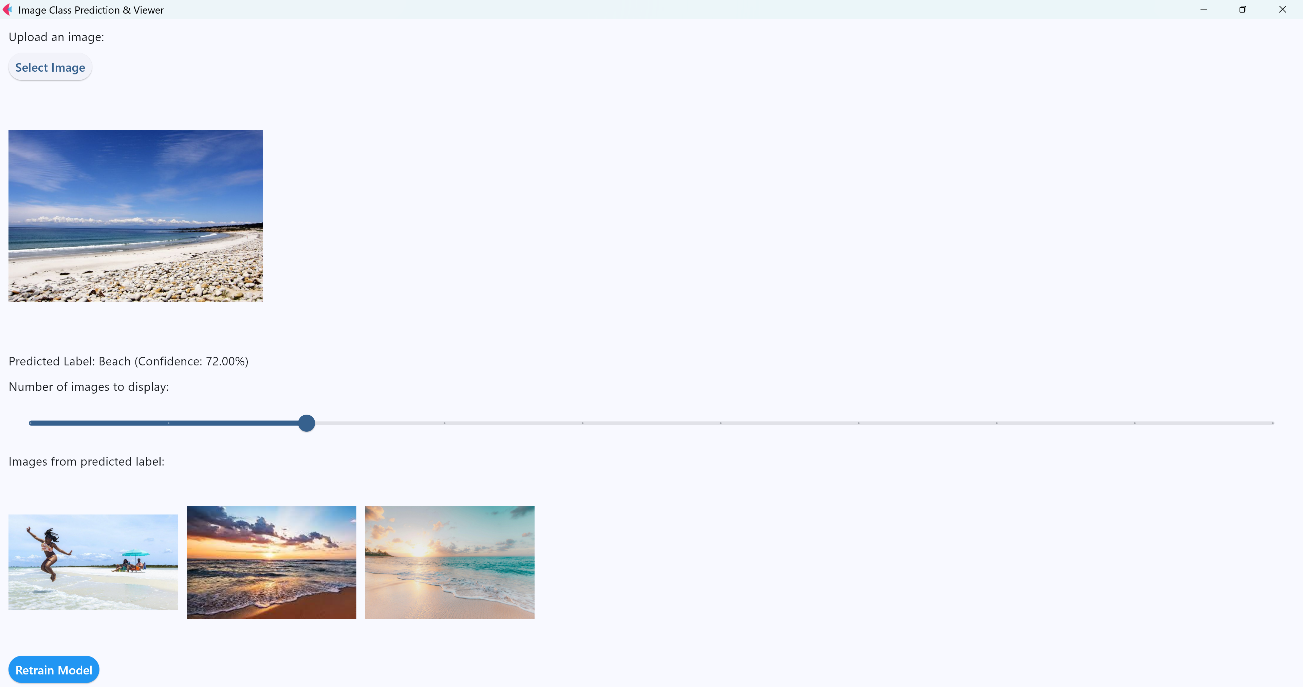
        ft.Text("Images from predicted label:"),

        image\_grid,

        ft.ElevatedButton("Retrain Model", on\_click=retrain\_model, bgcolor=ft.colors.BLUE, color=ft.colors.WHITE),

    )

ft.app(target=main)

**Sample Run of the Project :**