**Step 1: Setup Kaggle API & download dataset**

!pip install kaggle

* Installs the Kaggle API package needed to download datasets from Kaggle within Colab.

from google.colab import files

* Imports the files module from Colab to allow file upload/download interactions.

print("Upload your kaggle.json API token file:")

files.upload() # Upload kaggle.json here

* Prompts you to upload your Kaggle API token (kaggle.json), which authorizes the Colab environment to download datasets from your Kaggle account.

!mkdir -p ~/.kaggle

* Creates a hidden .kaggle directory in your home folder, if it doesn’t already exist. This is where the API token file should be placed.

!cp kaggle.json ~/.kaggle/

* Copies the uploaded kaggle.json file into the .kaggle folder.

!chmod 600 ~/.kaggle/kaggle.json

* Changes file permission of kaggle.json to be readable/writable **only by you**, securing the API token.

print("Downloading dataset from Kaggle...")

!kaggle datasets download atulanandjha/lfwpeople -q

* Uses Kaggle API to download the dataset named “lfwpeople” quietly (-q flag suppresses verbose output).

!unzip -o -q lfwpeople.zip -d lfwpeople

* Unzips the downloaded dataset archive into a folder named lfwpeople.
* -o option overwrites existing files without asking (fixes the overwrite prompt problem).
* -q makes unzip silent.

print("Dataset downloaded and unzipped.")

* Confirms that the dataset is now ready for use.

**Step 2: Dummy training**

import tensorflow as tf

from tensorflow.keras import layers, models

import numpy as np

* Imports TensorFlow and Keras modules to build and train neural network models.
* Imports NumPy for numerical operations.

def create\_dummy\_model():

model = models.Sequential([

layers.Input(shape=(128,128,3)),

layers.Conv2D(16, (3,3), activation='relu'),

layers.MaxPooling2D(),

layers.Conv2D(32, (3,3), activation='relu'),

layers.MaxPooling2D(),

layers.Flatten(),

layers.Dense(64, activation='relu'),

layers.Dense(1, activation='sigmoid') # binary classification

])

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

return model

* Defines a function that creates a **simple CNN model**:
  + Input layer expects images sized 128x128 pixels with 3 color channels (RGB).
  + Two Conv2D + MaxPooling layers extract image features.
  + Flatten layer converts 2D features to 1D vector.
  + Dense layer with 64 neurons learns abstract representations.
  + Final Dense layer outputs a single value with sigmoid activation for **binary classification** (e.g., Male/Female or Glasses/No Glasses).
* Compiles the model with Adam optimizer and binary cross-entropy loss (appropriate for binary classification).

model\_gender = create\_dummy\_model()

model\_glasses = create\_dummy\_model()

* Creates two instances of this model: one to predict gender, one to predict glasses presence.

dummy\_X = np.random.rand(10,128,128,3)

dummy\_y = np.random.randint(0,2,size=(10,1))

* Generates dummy training data:
  + dummy\_X: 10 random images of size 128x128 with 3 color channels.
  + dummy\_y: 10 random binary labels (0 or 1).

model\_gender.fit(dummy\_X, dummy\_y, epochs=1)

model\_glasses.fit(dummy\_X, dummy\_y, epochs=1)

* Trains each model on the dummy data for 1 epoch (just for demonstration).

model\_gender.save('gender\_model.h5')

model\_glasses.save('glasses\_model.h5')

* Saves the trained models to files (.h5 format) for later use in prediction.

print("Dummy models trained and saved.")

* Prints confirmation.

**Step 3: Upload image and make predictions**

from PIL import Image

import matplotlib.pyplot as plt

import numpy as np

from google.colab import files

* Imports:
  + PIL for image loading/manipulation.
  + matplotlib.pyplot to display images.
  + numpy for numerical operations.
  + files from Colab for file upload.

print("Now upload an image to predict gender, glasses, and shirt color:")

uploaded = files.upload()

* Prompts user to upload an image, then reads the uploaded file(s).

model\_gender = tf.keras.models.load\_model('gender\_model.h5')

model\_glasses = tf.keras.models.load\_model('glasses\_model.h5')

* Loads previously saved gender and glasses detection models from disk.

def detect\_shirt\_color\_rgb(image):

width, height = image.size

shirt\_region = image.crop((0, height//2, width, height))

pixels = np.array(shirt\_region)

avg\_color = pixels.mean(axis=(0,1))

r, g, b = avg\_color

* Defines a function to detect shirt color:
  + Crops bottom half of image assuming shirt is there.
  + Converts cropped area to NumPy array to access pixels.
  + Calculates average RGB color values over all pixels.

standard\_colors = {

'Red': np.array([255, 0, 0]),

'Light Red': np.array([255, 102, 102]),

'Green': np.array([0, 255, 0]),

'Light Green': np.array([144, 238, 144]),

'Blue': np.array([0, 0, 255]),

'Light Blue': np.array([173, 216, 230]),

'Yellow': np.array([255, 255, 0]),

'White': np.array([255, 255, 255]),

'Black': np.array([0, 0, 0]),

'Orange': np.array([255, 165, 0]),

'Pink': np.array([255, 192, 203]),

'Light Pink': np.array([255, 182, 193]),

'Purple': np.array([128, 0, 128]),

'Brown': np.array([150, 75, 0]),

'Gray': np.array([128, 128, 128]),

'Light Gray': np.array([211, 211, 211]),

}

* Defines a dictionary of **standard colors** mapped to their RGB values for comparison.

avg\_rgb = np.array([r, g, b])

min\_dist = float('inf')

closest\_color = 'Unknown Color'

* Converts average color to a NumPy array, initializes variables to track closest color.

for color\_name, std\_rgb in standard\_colors.items():

dist = np.linalg.norm(avg\_rgb - std\_rgb)

if dist < min\_dist:

min\_dist = dist

closest\_color = color\_name

* For each standard color, calculates Euclidean distance between average shirt color and the standard color.
* Finds the standard color with the smallest distance (closest match).

max\_distance\_threshold = 100

if min\_dist > max\_distance\_threshold:

return 'Unknown Color'

return closest\_color

* If the closest color is still too far away (distance > threshold), returns ‘Unknown Color’.
* Otherwise, returns the closest matched color name.

**Prediction & Output**

for filename in uploaded.keys():

print(f"Uploaded file: {filename}")

img = Image.open(filename).convert('RGB')

* For each uploaded file:
  + Prints filename.
  + Opens the image and converts to RGB format.

plt.imshow(img)

plt.axis('off')

plt.show()

* Displays the uploaded image.

img\_resized = img.resize((128,128))

img\_array = np.array(img\_resized) / 255.0

img\_array = np.expand\_dims(img\_array, axis=0)

* Resizes image to 128x128 pixels (model input size).
* Converts image to NumPy array and normalizes pixel values to [0,1].
* Adds a batch dimension for model input shape (1, 128, 128, 3).

gender\_pred = model\_gender.predict(img\_array)

glasses\_pred = model\_glasses.predict(img\_array)

* Runs prediction using gender and glasses models.

gender\_label = 'Male' if gender\_pred[0][0] > 0.5 else 'Female'

glasses\_label = 'Wearing Glasses' if glasses\_pred[0][0] > 0.5 else 'No Glasses'

* Converts prediction probabilities to labels using 0.5 threshold.

shirt\_color = detect\_shirt\_color\_rgb(img)

* Calls the shirt color detection function on the original uploaded image.

print(f"Predicted Gender: {gender\_label}")

print(f"Glasses Detection: {glasses\_label}")

print(f"Detected Shirt Color: {shirt\_color}")

* Prints the final results for each uploaded image.

**How this fits our project:**

* **Step 1** prepares our **training dataset** (balanced, diverse face images from Kaggle).
* **Step 2** shows how to create and train models for **gender and glasses detection** (we can replace dummy training with our real training).
* **Step 3** builds the **inference pipeline**, allowing us to upload any face image and get:
  + Gender prediction
  + Glasses detection
  + Shirt color detection (using RGB average color from the shirt area)

This end-to-end flow aligns perfectly with our project requirements of scraping data, training models, and running predictions on uploaded images to return those three attributes.