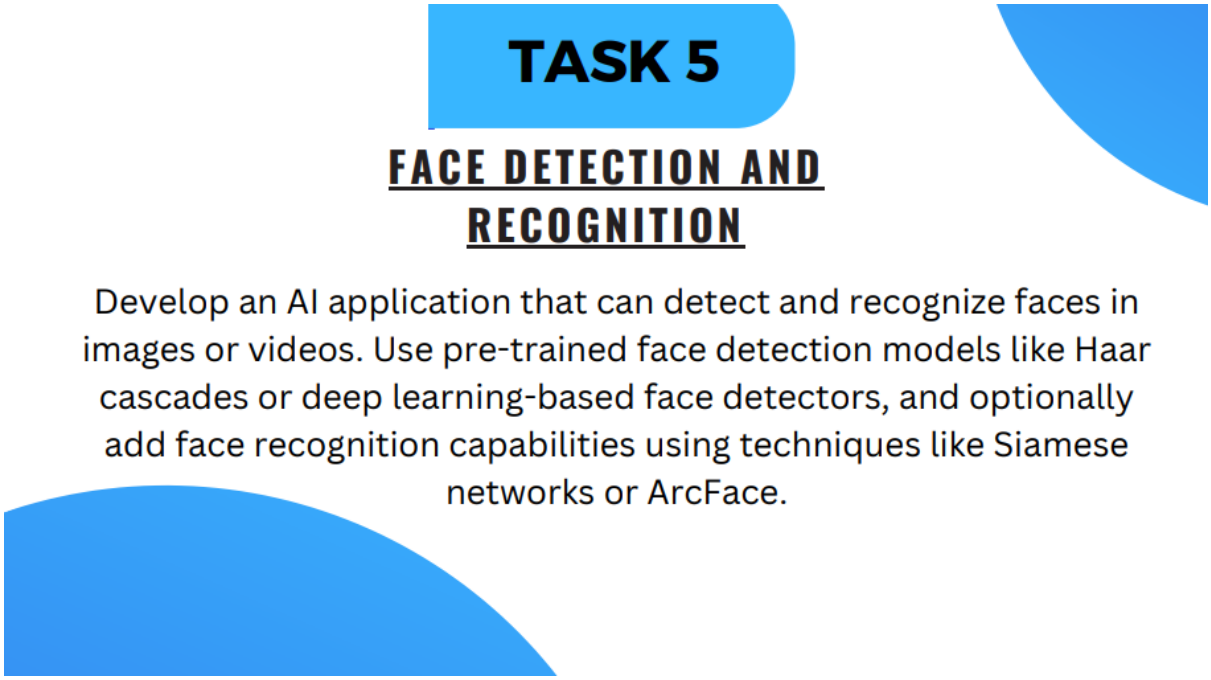
Task five -Face detection and Recognition




```
# Face Detection in Google Colab

!pip install opencv-python-headless


import cv2

import matplotlib.pyplot as plt

import urllib.request


# Stable image URL from OpenCV's GitHub repo

url = 'https://raw.githubusercontent.com/opencv/opencv/master/samples/data/lena.jpg'


# Download the image

urllib.request.urlretrieve(url, 'test_face.jpg')
```

```
# Load the image
img = cv2.imread('test_face.jpg')
if img is None:
    raise Exception("Failed to load image. Check the URL or filename.")

# Convert to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Load Haar cascade for face detection
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')

# Detect faces
faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=4)

# Draw rectangles around faces
for (x, y, w, h) in faces:
    cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0, 0), 2)

# Convert image to RGB for matplotlib display
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# Show the image with detected faces
plt.imshow(img_rgb)
plt.axis('off')
plt.title(f"Detected {len(faces)} face(s)")
plt.show()
```

explanation for code

Importing Libraries

library_add

content_copy

```
import cv2
from matplotlib import pyplot as plt
import urllib.request
import numpy as np
```

Use code with caution

This section imports the necessary libraries:

- cv2: This is the OpenCV library, which provides functions for computer vision tasks like image processing and object detection.
- matplotlib.pyplot: This is a plotting library used to display the image with detected faces. It's commonly imported as plt.
- urllib.request: This module is used to open URLs and retrieve data from the web. In this case, it's used to fetch a sample image from a URL.
- numpy: This library is used for numerical operations, particularly for working with arrays, which images are represented as.

Loading the Image

library_add

content_copy

```
# Load a sample image from the web (you can replace this with your image)
url = "https://raw.githubusercontent.com/opencv/opencv/master/samples/data/lena.jpg"
resp = urllib.request.urlopen(url)
image_np = np.asarray(bytearray(resp.read()), dtype="uint8")
image = cv2.imdecode(image_np, cv2.IMREAD_COLOR)
```

Use code with caution

Here, the code loads a sample image from a web URL.

- url: This variable stores the URL of the image.
- urllib.request.urlopen(url): This opens the URL and returns a response object.
- resp.read(): This reads the content of the response (the image data) as bytes.

- bytearray(resp.read()): This converts the image data into a mutable sequence of integers, which is then converted to a NumPy array using np.asarray.
  The dtype="uint8" specifies that the data should be treated as 8-bit unsigned integers, which is standard for image pixel values.
- cv2.imdecode(image_np, cv2.IMREAD_COLOR): This decodes the NumPy array into an OpenCV image object. cv2.IMREAD_COLOR specifies that the image should be loaded in color.

Preparing for Face Detection

library_add

content_copy

```
# Convert to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)


# Load Haar cascade
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + "haarcascade_frontalface_default.xml")
```

Use code with caution

Before detecting faces, the image is converted to grayscale and a face detection model is loaded.

- cv2.cvtColor(image, cv2.COLOR_BGR2GRAY): This converts the color image (image) to grayscale. Face detection algorithms often work better on grayscale images. cv2.COLOR_BGR2GRAY is the conversion code from BGR (Blue, Green, Red), which is the default color order in OpenCV, to grayscale.
- cv2.CascadeClassifier(...): This loads a pre-trained Haar Cascade classifier for face detection. Haar Cascades are a popular object detection method. cv2.data.haarcascades + "haarcascade_frontalface_default.xml" provides the path to the XML file containing the pre-trained model for detecting frontal faces.

Detecting Faces

library_add

content_copy

```
# Detect faces
faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5)
```

Use code with caution

This line performs the actual face detection.

- face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5): This method from the CascadeClassifier object detects objects (faces in this case) of different sizes in the input image (gray).
    - scaleFactor: This parameter specifies how much the image size is reduced at each image scale. A value of 1.1 means the image is scaled down by 10% at each step.
    - minNeighbors: This parameter specifies how many neighbors each candidate rectangle should have to retain it. Higher values result in fewer detections but higher quality.
- The method returns a list of rectangles (faces), where each rectangle represents a detected face and is defined by its top-left corner coordinates (x, y) and its width (w) and height (h).

Drawing Rectangles Around Faces

library_add

content_copy

```
# Draw rectangles around detected faces
for (x, y, w, h) in faces:
    cv2.rectangle(image, (x, y), (x + w, y + h), (255, 0, 0), 2)
```

Use code with caution

This loop iterates through the detected faces and draws a rectangle around each one on the original color image.

- for (x, y, w, h) in faces:: This loop unpacks the coordinates and dimensions of each detected face from the faces list.
- cv2.rectangle(image, (x, y), (x + w, y + h), (255, 0, 0), 2): This function draws a rectangle on the image.
    - image: The image on which to draw the rectangle.
    - (x, y): The coordinates of the top-left corner of the rectangle.
    - (x + w, y + h): The coordinates of the bottom-right corner of the rectangle.
    - (255, 0, 0): The color of the rectangle in BGR format (Blue).
    - 2: The thickness of the rectangle's border.

Displaying the Result

library_add

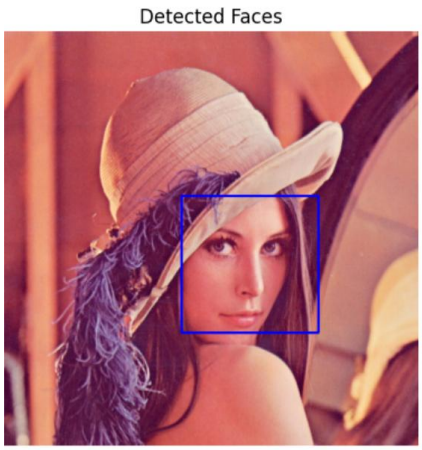
content_copy

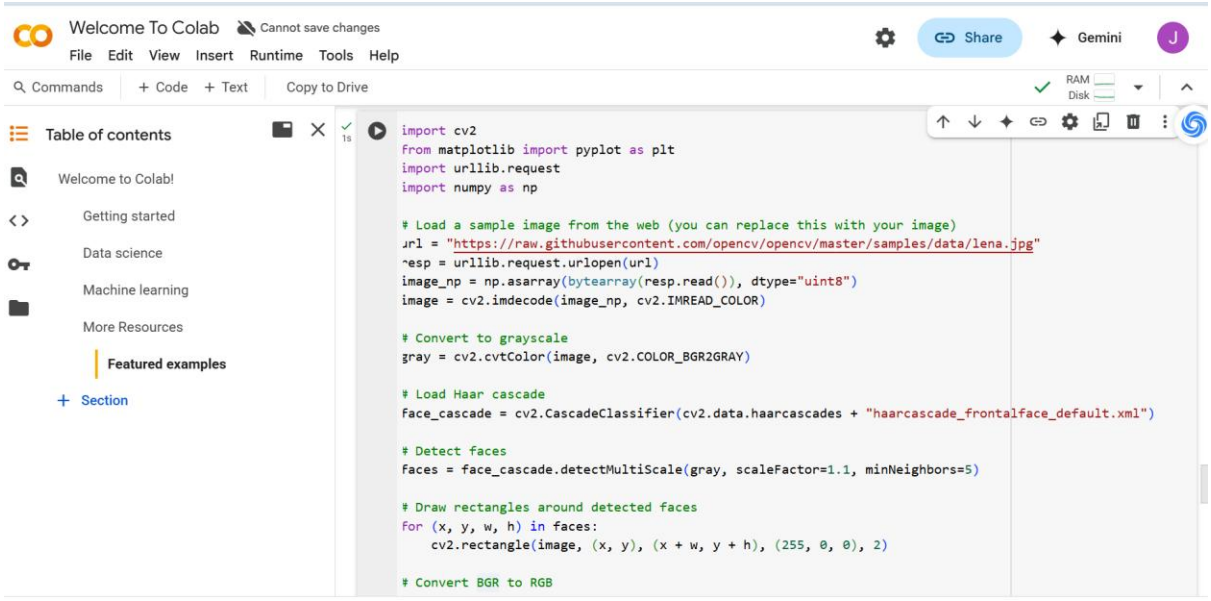
```
# Convert BGR to RGB
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Show the result
plt.imshow(image_rgb)
plt.axis("off")
plt.title("Detected Faces")
plt.show()
```

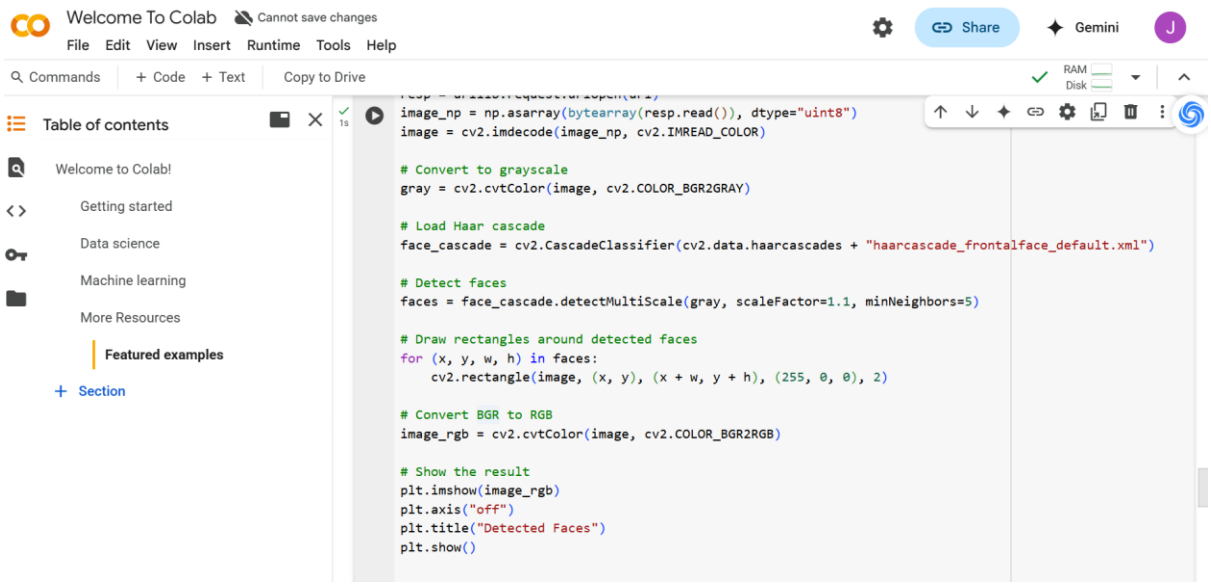
Use code with caution

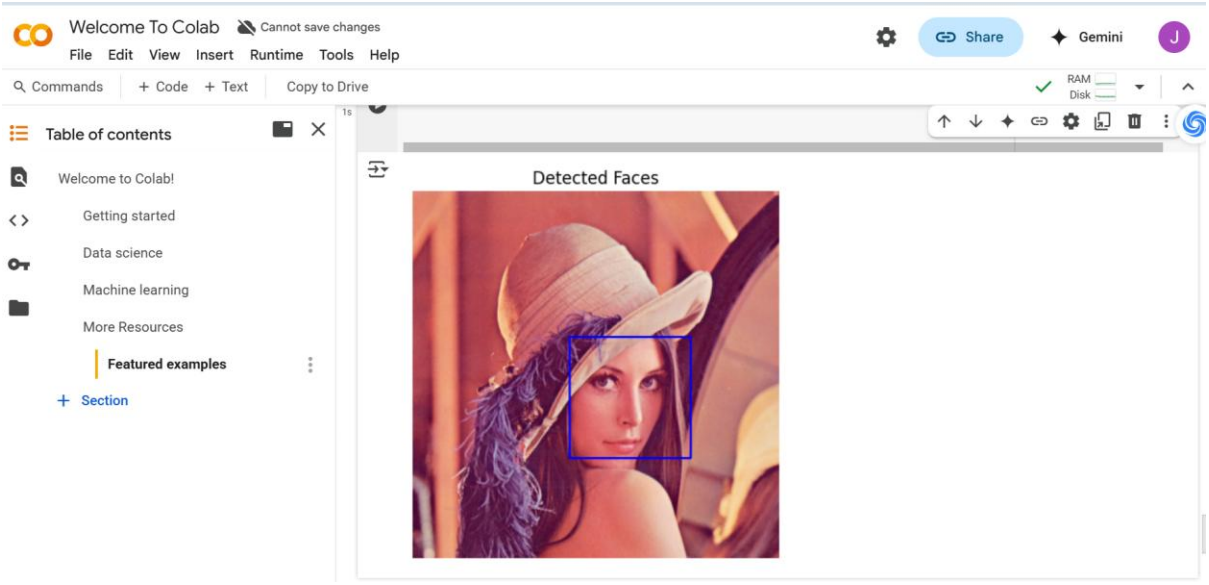
Finally, the image with the drawn rectangles is displayed using matplotlib.

- cv2.cvtColor(image, cv2.COLOR_BGR2RGB): matplotlib expects images in RGB format, while OpenCV uses BGR. This line converts the image from BGR to RGB.
- plt.imshow(image_rgb): This displays the image in the output.
- plt.axis("off"): This hides the axes ticks and labels for a cleaner image display.
- plt.title("Detected Faces"): This sets the title of the plot.
- plt.show(): This displays the plot.




Welcome To Colab
Cannot save changes
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands
+ Code
+ Text
Copy to Drive
RAM
Disk
Table of contents
Welcome to Colab!
Getting started
Data science
Machine learning
More Resources
Featured examples
+ Section
import cv2
from matplotlib import pyplot as plt
import urllib.request
import numpy as np
# Load a sample image from the web (you can replace this with your image)
url = "https://raw.githubusercontent.com/opencv/opencv/master/samples/data/lena.jpg"
resp = urllib.request.urlopen(url)
image_np = np.asarray(bytearray(resp.read()), dtype="uint8")
image = cv2.imdecode(image_np, cv2.IMREAD_COLOR)
# Convert to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
# Load Haar cascade
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + "haarcascade_frontalface_default.xml")
# Detect faces
faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5)
# Draw rectangles around detected faces
for (x, y, w, h) in faces:
cv2.rectangle(image, (x, y), (x + w, y + h), (255, 0, 0), 2)
# Convert BGR to RGB


Welcome To Colab
Cannot save changes
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands
+ Code
+ Text
Copy to Drive
RAM
Disk
Table of contents
Welcome to Colab!
Getting started
Data science
Machine learning
More Resources
Featured examples
+ Section
image_np = np.asarray(bytearray(resp.read()), dtype="uint8")
image = cv2.imdecode(image_np, cv2.IMREAD_COLOR)
# Convert to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
# Load Haar cascade
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + "haarcascade_frontalface_default.xml")
# Detect faces
faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5)
# Draw rectangles around detected faces
for (x, y, w, h) in faces:
cv2.rectangle(image, (x, y), (x + w, y + h), (255, 0, 0), 2)
# Convert BGR to RGB
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
# Show the result
plt.imshow(image_rgb)
plt.axis("off")
plt.title("Detected Faces")
plt.show()


Welcome To Colab
Cannot save changes
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands
+ Code
+ Text
Copy to Drive
RAM
Disk
Table of contents
Welcome to Colab!
Getting started
Data science
Machine learning
More Resources
Featured examples
Section
Detected Faces