Reg No:D/BCS/22/0023

Practical 1

Shapes.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
pip install opencv-contrib-python
```

```
Requirement already satisfied: opencv-contrib-python in c:\users\asus\python data science\image processing\image_processing\lib\site-packages (4.9.0.80)
Requirement already satisfied: numpy>=1.17.3; python_version >= "3.8" in c:\users\asus\python data science\image processing\image_processing\lib\site-packages (from opencv-contrib-python) (1.24.4)
Note: you may need to restart the kernel to use updated packages.
WARNING: You are using pip version 20.1.1; however, version 23.3.2 is available.
You should consider upgrading via the 'C:\Users\asus\Python Data Science\Image Processing\Image_Processing\Scripts\python.exe -m pip install --upgrade pip' command.
```

+ Code + Text

```
pip install opencv-python
```

```
Requirement already satisfied: opencv-python in c:\users\asus\python data science\image processing\image_processing\lib\site-packages (4.9.0.80)
Requirement already satisfied: numpy>=1.17.3; python_version >= "3.8" in c:\users\asus\python data science\image processing\image_processing\lib\site-packages (from opencv-python) (1.24.4)
Note: you may need to restart the kernel to use updated packages.
WARNING: You are using pip version 20.1.1; however, version 23.3.2 is available.
You should consider upgrading via the 'C:\Users\asus\Python Data Science\Image Processing\Image_Processing\Scripts\python.exe -m pip install --upgrade pip' command.
```

```
pip install matplotlib
```

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

```
print(cv2.__version__)
```

```
4.9.0
```

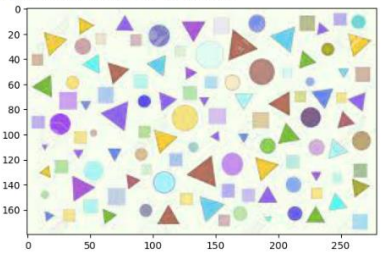
```
img = cv2.imread('picture.jpg')
plt.imshow(img)
```

Shapes.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

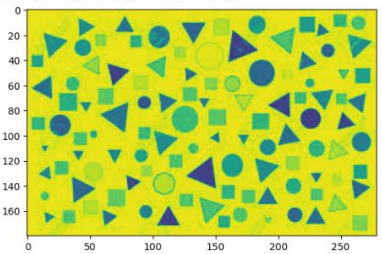
```
<matplotlib.image.AxesImage at 0x1fc6a88e430>
```



Here the Gray scale conversion is done to reduce the complexity of the image by converting it to flat colors.

```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
plt.imshow(gray)
```

```
<matplotlib.image.AxesImage at 0x1fc6cb0ff40>
```



Shapes.ipynb

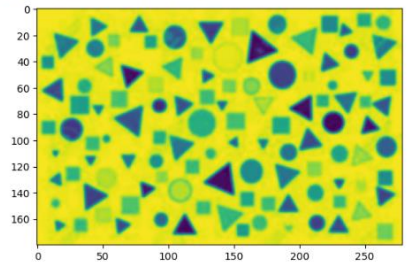
File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

What Blurring does is it further simplifies the image by smoothing. In here a 5 x 5 kernel is applied across the image matrix and pixel an averaging is done by this. After this as you can see the pixel count reduces, clarity of the image drops but mathematical representation of the image becomes much more simpler, which is helpful for the processing.

```
blurred = cv2.GaussianBlur(gray, (5, 5), 0)
plt.imshow(blurred)
```

```
<matplotlib.image.AxesImage at 0x1fc6a98f2b0>
```



Canny edge detection identifies areas in the image where there are rapid changes in intensity, which often correspond to edges or boundaries between different objects. Pixel intensity ranges between 50- 150 has been considered to be applied for this algorithm. This will superimpose the edges and bounderies making it easy to understand for the computer

```
[ ] edges = cv2.Canny(blurred, 50, 150)
plt.imshow(edges)
```

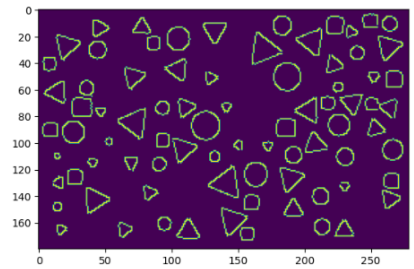
Shapes.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
edges = cv2.Canny(blurred, 50, 150)
plt.imshow(edges)
```

```
<matplotlib.image.AxesImage at 0x1fc6a979fd0>
```



Thresholding simplifies an image by dividing it into just two colors: black and white. If a pixel is darker than a medium gray (127), it becomes totally black (0); otherwise, it becomes totally white (255). It's like creating a simplified map where only certain details stand out, making it easier to focus on specific parts of the image.

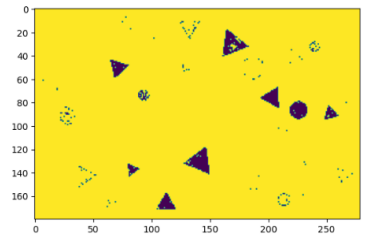
```
[ ] _, thresh = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)
plt.imshow(thresh)
```

Shapes.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
<matplotlib.image.AxesImage at 0x1fc6dd724c0>
```



Coutours are outlines drawn surrounding the shapes. It`s mainly used for shape analysis. cv2.RETR_EXTERNAL means consider only the external aspect of the object, not considering the inside.cv2. CHAIN_APPROX_SIMPLE compresses horizontal, vertical, and diagonal lines and provide a simple averaged representation.

```
[ ] # Find contours in the binary image
    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
```

```
[ ] # Draw contours on the original image
    cv2.drawContours(img, contours, -1, (0, 255, 0), 2)
```

Shapes.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
array([[[  0, 255,   0],
        [  0, 255,   0],
        [  0, 255,   0],
        ...,
        [  0, 255,   0],
        [  0, 255,   0],
        [  0, 255,   0]],

       [[  0, 255,   0],
        [  0, 255,   0],
        [  0, 255,   0],
        ...,
        [  0, 255,   0],
        [  0, 255,   0],
        [  0, 255,   0]],

       [[  0, 255,   0],
        [  0, 255,   0],
        [248, 255, 240],
        ...,
        [248, 255, 240],
        [  0, 255,   0],
        [  0, 255,   0]],

       ...,

       [[  0, 255,   0],
        [  0, 255,   0],
        [248, 255, 240],
        ...,
        [248, 255, 240],
        [  0, 255,   0],
        [  0, 255,   0]],

       [[  0, 255,   0],
        [  0, 255,   0],
        [  0, 255,   0],
        ...,
        [  0, 255,   0],
        [  0, 255,   0],
        [  0, 255,   0]],

       [[  0, 255,   0],
        [  0, 255,   0],
        [  0, 255,   0],
        ...,
        [  0, 255,   0],
        [  0, 255,   0],
        [  0, 255,   0]]], dtype=uint8)
```

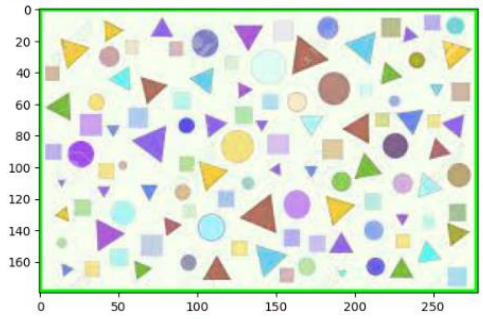
Shapes.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
[ ] plt.imshow(img)
```

```
<matplotlib.image.AxesImage at 0x1772a13c850>
```



```
[ ] def find_triangles(image_path):
    # Read the image
    img = cv2.imread(image_path)

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

    # Apply GaussianBlur to reduce noise and help contour detection
    blurred = cv2.GaussianBlur(gray, (7, 7), 0)

    # Use Canny edge detection to find edges in the image
    edges = cv2.Canny(blurred, 30, 150)

    # Find contours in the edged image
    contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    # Iterate through the contours
    for contour in contours:
        # Approximate the contour to a polygon
        epsilon = 0.04 * cv2.arcLength(contour, True)
        approx = cv2.approxPolyDP(contour, epsilon, True)

        # If the polygon has 3 vertices, it's a triangle
        if len(approx) == 4:
            # Draw a bounding box around the triangle (in red)
            x, y, w, h = cv2.boundingRect(contour)
            cv2.rectangle(img, (x, y), (x + w, y + h), (0, 0, 255), 2)

    # Display the result
    cv2.imshow('Triangles with Bounding Boxes', img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

```
[ ] # Replace 'your_image_path.jpg' with the path to your image
    find_triangles('picture.jpg')
```

Epsilon for deciding how much you care about the tiny details of the shape. Smaller epsilon means you really care about the details, while larger epsilon means you're okay with a simpler, more generalized shape. archLenth method calculate the total lenth of the surface contour and 4% of is applied for the Epsilon calculation.

True, denotes, we are looking for closed shapes, not open. appoxPolyDP method derives information of all poligon shapes, which satisfies the specified epsilon criteria.

Henceforth, via a condition we can check poligons having 3 contour lines connected as traingles.

Step-by-Step Explanation for the code above

Installation and Imports

First, the necessary libraries are installed: opencv-contrib-python and opencv-python for computer vision tasks, and matplotlib for plotting images. The cv2.__version__ statement prints the installed OpenCV version, ensuring compatibility and successful installation.

Reading and Displaying the Image

The image is read from the specified path using cv2.imread and displayed using plt.imshow. This provides an initial view of the original image.

Grayscale Conversion

The image is converted to grayscale using cv2.cvtColor. Grayscale conversion simplifies the image by reducing the number of color channels from three (BGR) to one, which aids in reducing computational complexity for subsequent processing steps.

Image Blurring

Gaussian Blurring is applied to the grayscale image using a 5x5 kernel. This smoothing operation helps in reducing noise and details in the image, making it easier to detect edges in the next step. The kernel size and standard deviation are parameters that control the extent of the blurring.

Edge Detection

Canny edge detection is applied to the blurred image. This algorithm identifies regions with rapid intensity changes, which typically correspond to edges. The specified thresholds (50 and 150) control the sensitivity of edge detection, defining the range of pixel intensity values considered for identifying edges.

Thresholding

Thresholding converts the grayscale image into a binary image, where pixels are either black or white. Pixels with intensity values above 127 are set to 255 (white), and those below are set to 0 (black). This simplification helps in focusing on specific parts of the image, especially useful for contour detection.

Contour Detection

Contours are detected in the binary image using cv2.findContours. The mode cv2.RETR_EXTERNAL retrieves only the external contours, ignoring inner contours. The method cv2.CHAIN_APPROX_SIMPLE compresses horizontal, vertical, and diagonal segments, storing only their end points for efficiency.

Drawing Contours

Detected contours are drawn on the original image using cv2.drawContours. The contours are outlined in green (0, 255, 0) with a thickness of 2 pixels. This visualization helps in verifying the detected contours.

Triangle Detection Function

This function, find_triangles, is designed to detect triangles in an image. below i have given a breakdown of the steps:

Reading the Image: The image is read from the specified path.

Grayscale Conversion: The image is converted to grayscale.

Blurring: Gaussian Blurring with a 7x7 kernel is applied to reduce noise.

Edge Detection: Canny edge detection is applied to the blurred image with adjusted thresholds.

Contour Detection: Contours are found in the edge-detected image.

Contour Approximation: Each contour is approximated to a polygon using cv2.approxPolyDP. The epsilon parameter determines the approximation accuracy, set as 4% of the contour's arc length.

Triangle Detection: Polygons with three vertices are identified as triangles.

Drawing Triangles: Detected triangles are drawn in green.

Displaying the Result: The final image with detected triangles is displayed using matplotlib.

Explanation of Parameters and Methods

Epsilon in cv2.approxPolyDP: Epsilon controls the approximation accuracy. A smaller epsilon results in a polygon that closely follows the contour's shape, while a larger epsilon produces a simpler, more generalized shape.

cv2.arcLength: This method calculates the perimeter of the contour, which is used to determine epsilon.

Contours with Three Vertices: The condition if len(approx) == 3 checks for triangles, as a triangle has three vertices.

The Improved code

Shapes.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

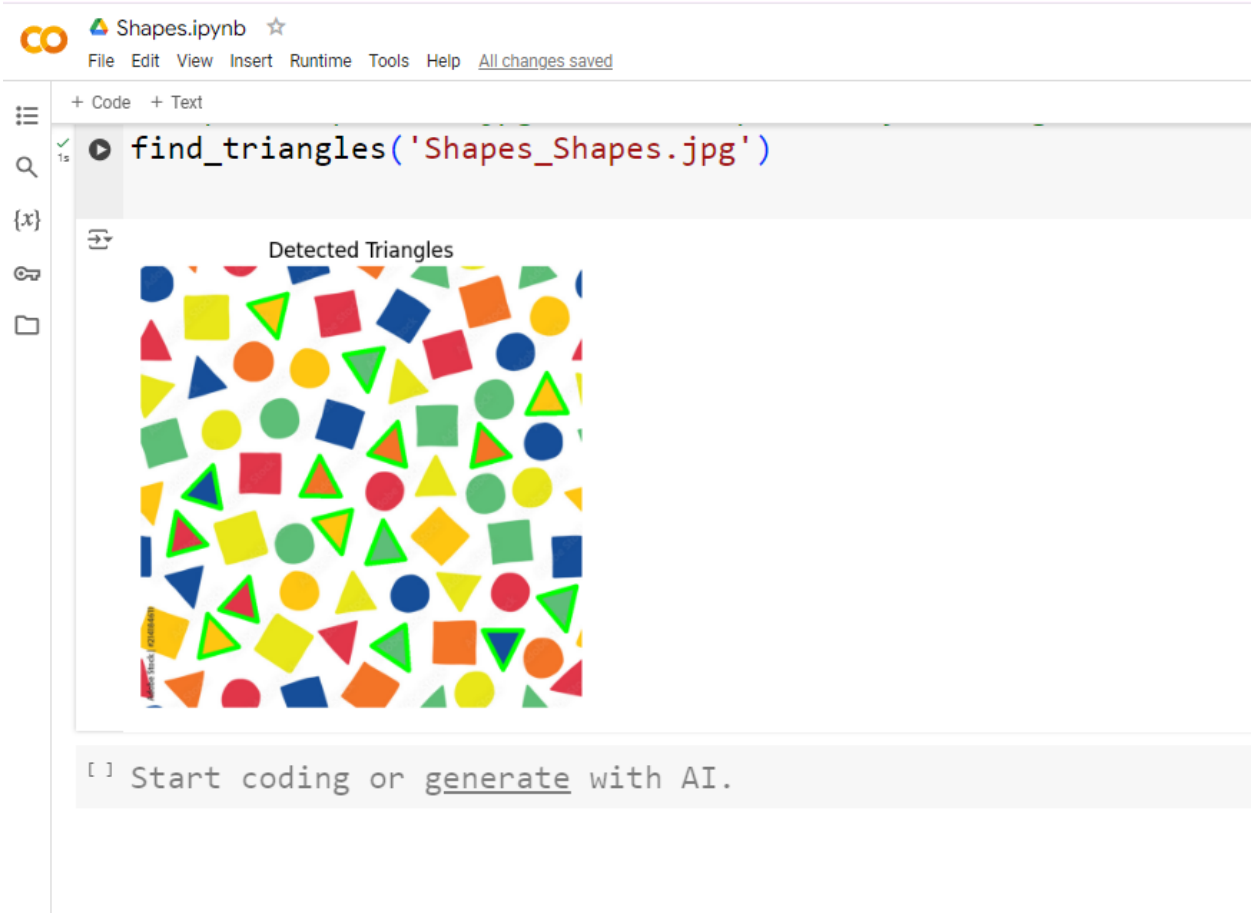
```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Install required packages
# pip install opencv-contrib-python
# pip install opencv-python
# pip install matplotlib

def find_triangles(image_path):
    # Read the image
    img = cv2.imread(image_path)

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

    # Apply GaussianBlur to reduce noise and help contour detection
    blurred = cv2.GaussianBlur(gray, (7, 7), 0)

    # Use Canny edge detection to find edges in the image
    edges = cv2.Canny(blurred, 50, 150)

    # Find contours in the edged image
    contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
```

Shapes.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```python
    # Iterate through the contours
    for contour in contours:
        # Approximate the contour to a polygon
        epsilon = 0.02 * cv2.arcLength(contour, True)  # Reduced epsilon for better accuracy
        approx = cv2.approxPolyDP(contour, epsilon, True)

        # If the polygon has 3 vertices, it's a triangle
        if len(approx) == 3:
            # Draw the triangle (in green)
            cv2.drawContours(img, [approx], 0, (0, 255, 0), 10)

    # Display the result
    plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    plt.title('Detected Triangles')
    plt.axis('off')
    plt.show()

# Replace 'picture.jpg' with the path to your image
find_triangles('Shapes_Shapes.jpg')
```



Detailed Justification for Improved Triangle Detection Implementation

The proposed improvements to the triangle detection algorithm involve fine-tuning several parameters and processes within the image preprocessing steps. These modifications are justified as follows:

Gaussian Blur Adjustment

Justification:

Gaussian Blur is used to reduce noise and detail in an image, which is crucial for reliable edge detection. By setting the kernel size to (7, 7), we increase the smoothing effect, effectively reducing noise. This larger kernel size is chosen to enhance the quality of the contours detected later, ensuring they are less influenced by minor irregularities in the image. The kernel size must balance between too much blurring, which can obscure important details, and too little, which can leave noise that affects edge detection.

Canny Edge Detection Thresholds

Justification:

The Canny edge detection algorithm identifies areas of rapid intensity change, corresponding to edges within the image. The thresholds of 50 and 150 are chosen to optimize edge detection sensitivity. The lower threshold (50) ensures that faint edges are detected, while the higher threshold (150) avoids detecting irrelevant noise as edges. These values are carefully selected to enhance edge clarity without introducing excessive noise, thereby improving the subsequent contour detection accuracy.

Contour Approximation with Reduced Epsilon

Justification:

Contour approximation simplifies the detected contours to polygons with fewer vertices. The epsilon parameter in cv2.approxPolyDP defines the maximum distance between the contour and the approximated polygon. By reducing epsilon to 0.02 times the arc length, we achieve a more precise approximation that closely follows the actual contour shape. This adjustment ensures that small details in the contour are preserved, which is essential for accurately identifying triangles (which have exactly three vertices).

Drawing Detected Triangles Directly

Justification:

Instead of drawing bounding boxes around detected triangles, we directly draw the triangles using cv2.drawContours. This method provides a clearer and more accurate visualization of the detected triangles. It ensures that the shapes are represented precisely as detected, avoiding any misinterpretation that might arise from using bounding boxes. This approach enhances the visual verification process, making it easier to confirm the accuracy of the detection algorithm.

Displaying Results Using Matplotlib

Justification:

Using matplotlib to display the final result within the script provides a more convenient and versatile visualization tool. Matplotlib allows for easy integration within Jupyter notebooks and other Python environments, offering a flexible and user-friendly interface for displaying images. This method also ensures that the displayed image is correctly converted from BGR to RGB format, preserving the correct color representation.

A brief summary of Improvements

The improvements in Gaussian Blurring, Canny edge detection thresholds, and contour approximation epsilon are designed to enhance the robustness and accuracy of triangle detection. By focusing on reducing noise, optimizing edge detection, and ensuring precise contour approximation, these adjustments contribute to more reliable identification of triangles in the image. The decision to draw triangles directly and use Matplotlib for display further enhances the clarity and usability of the results.

These enhancements are expected to improve the detection accuracy and visualization, making the algorithm more effective for various applications requiring precise shape detection. The parameters should be adjusted based on the specific characteristics of the input images to achieve optimal performance.

Practical 2

Matching.ipynb

File Edit View Insert Runtime Tools Help Last saved at 18:17

+ Code + Text

```
pip uninstall opencv-python
```

```
pip install face_recognition opencv-python
```

```
import face_recognition
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

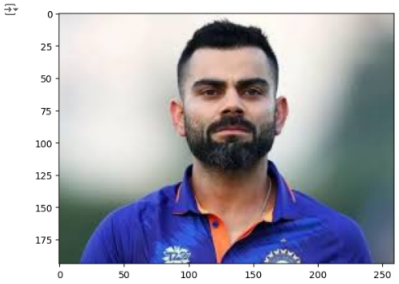
```
%matplotlib inline
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('duals.jpg')
imgplot = plt.imshow(img)
plt.show()
```

Matching.ipynb

File Edit View Insert Runtime Tools Help Last saved at 18:17

+ Code + Text

```
img2 = mpimg.imread('target_template_I.jpg')
imgplot = plt.imshow(img2)
plt.show()
```




```
def compare_and_mark_faces(main_image_path, target_image_path):

    # Load the main image and the target image
    main_image = face_recognition.load_image_file(main_image_path)
    target_image = face_recognition.load_image_file(target_image_path)

    # Find face locations and face encodings for the main and target images
    main_face_locations = face_recognition.face_locations(main_image)
    target_face_locations = face_recognition.face_locations(target_image)

    main_face_encodings = face_recognition.face_encodings(main_image, main_face_locations)
```

Matching.ipynb

File Edit View Insert Runtime Tools Help Last saved at 18:17

+ Code + Text

```
    target_face_encodings = face_recognition.face_encodings(target_image, target_face_locations)

    # Create a copy of the main image for visualization
    main_image_marked = np.copy(main_image)

    # Iterate through faces in the main image
    for main_face_location, main_face_encoding in zip(main_face_locations, main_face_encodings):
        # Compare the face encoding with all face encodings in the target image
        results = face_recognition.compare_faces(target_face_encodings, main_face_encoding)

        # Check if there is at least one match
        if True in results:
            # Find the index of the first matching face in the target image
            target_index = results.index(True)

            # Draw a red bounding box around the matching face in the main image
            top, right, bottom, left = main_face_location
            cv2.rectangle(main_image_marked, (left, top), (right, bottom), (0, 0, 255), 2)

    # Display the marked main image
    #cv2.imshow('Face Matching with Marking', main_image_marked)
    #cv2.waitKey(0)
    #cv2.destroyAllWindows()

    plt.imshow(main_image_marked)
    plt.axis('off')  # Turn off axis labels
    plt.show()
```

Matching.ipynb

File Edit View Insert Runtime Tools Help Last saved at 18:17

+ Code + Text

```
# Specify the paths to the main image and the target image
main_image_path = "duals.jpg"
#main_image_path = "duals.jpg"
target_image_path = "target_template_I.jpg"
#target_image_path = "target_template_II.jpg"
```

```
# Call the function for face comparison and marking
compare_and_mark_faces(main_image_path, target_image_path)
```


```
Start coding or generate with AI.
```

The improved code for this practical

Matching.ipynb

File Edit View Insert Runtime Tools Help Saving failed since 22:01

+ Code + Text

```
import face_recognition
import cv2
import numpy as np
import matplotlib.pyplot as plt

def display_image(image_path):
    img = plt.imread(image_path)
    plt.imshow(img)
    plt.axis('off')
    plt.show()

def compare_and_mark_faces(main_image_path, target_image_path):
    # Load images
    main_image = face_recognition.load_image_file(main_image_path)
    target_image = face_recognition.load_image_file(target_image_path)

    print(f"Main image shape: {main_image.shape}")
    print(f"Target image shape: {target_image.shape}")

    # Find face locations and encodings
    main_face_locations = face_recognition.face_locations(main_image, model='cnn')
    print(f"Main image face locations: {main_face_locations}")

    main_face_encodings = face_recognition.face_encodings(main_image, main_face_locations)
    print(f"Number of faces encoded in main image: {len(main_face_encodings)}")
```

Matching.ipynb

File Edit View Insert Runtime Tools Help Saving failed since 22:01

+ Code + Text

```
    target_face_locations = face_recognition.face_locations(target_image, model='cnn')
    print(f"Target image face locations: {target_face_locations}")

    target_face_encodings = face_recognition.face_encodings(target_image, target_face_locations)
    print(f"Number of faces encoded in target image: {len(target_face_encodings)}")

    if not main_face_encodings:
        print("No faces found in the main image.")
        return
    if not target_face_encodings:
        print("No faces found in the target image.")
        return

    # Assume the first face in the main image is Virat Kohli
    virat_encoding = main_face_encodings[0]

    # Create a copy of the target image for drawing
    target_image_marked = np.copy(target_image)

    # Compare faces
    for (top, right, bottom, left), face_encoding in zip(target_face_locations, target_face_encodings):
        matches = face_recognition.compare_faces([virat_encoding], face_encoding, tolerance=0.6)
        print(f"Match result for face at {(top, right, bottom, left)}: {matches[0]}")
```

Matching.ipynb

File Edit View Insert Runtime Tools Help Saving failed since 22:01

+ Code + Text

```
        if matches[0]:
            # Draw a green bounding box for a match
            cv2.rectangle(target_image_marked, (left, top), (right, bottom), (0, 255, 0), 2)
            print(f"Drew green box at {(left, top, right, bottom)}")
        else:
            # Draw a red bounding box for a non-match
            cv2.rectangle(target_image_marked, (left, top), (right, bottom), (0, 0, 255), 2)
            print(f"Drew red box at {(left, top, right, bottom)}")

    # Save the marked image
    cv2.imwrite('marked_target_image.jpg', cv2.cvtColor(target_image_marked, cv2.COLOR_RGB2BGR))
    print("Saved marked image as 'marked_target_image.jpg'")

    # Display the marked target image
    plt.figure(figsize=(12, 8))
    plt.imshow(cv2.cvtColor(target_image_marked, cv2.COLOR_BGR2RGB))
    plt.axis('off')
    plt.title("Detected Faces (Green: Match, Red: Non-match)")
    plt.show()

# Paths to your images
main_image_path = 'virat1.jpg'
target_image_path = 'two.jpg'
```

Matching.ipynb ☆

File Edit View Insert Runtime Tools Help Saving failed since 22:01

+ Code + Text

```
# Display original images
print("Main Image:")
display_image(main_image_path)
print("Target Image:")
display_image(target_image_path)

# Perform face comparison and marking
compare_and_mark_faces(main_image_path, target_image_path)
```

```
Main Image:
```

```
Target Image:
```


```
Main image shape: (652, 1000, 3)
Target image shape: (1080, 1920, 3)
Main image face locations: [(94, 568, 298, 364)]
Number of faces encoded in main image: 1
Target image face locations: [(225, 1146, 519, 852)]
Number of faces encoded in target image: 1
Match result for face at (225, 1146, 519, 852): True
Drew green box at (852, 225, 1146, 519)
Saved marked image as 'marked_target_image.jpg'
```

Detected Faces (Green: Match, Red: Non-match)

Now let's break down the code and its functionality step by step:

Imports and Function Definitions:

The code imports necessary libraries: face_recognition, cv2, numpy, and matplotlib.pyplot.

Two functions are defined:

display_image(image_path): Displays an image given its file path using matplotlib.

compare_and_mark_faces(main_image_path, target_image_path): This function compares faces between a main image and a target image, and marks the faces in the target image based on the comparison results.

Displaying Images:

display_image(image_path) is called twice to display the original main and target images using matplotlib.pyplot.

Face Recognition and Encoding:

face_recognition.load_image_file() is used to load images from file paths into numpy arrays (main_image and target_image).

face_recognition.face_locations() is used to detect face locations in both images using the 'cnn' model, which is a convolutional neural network model known for its accuracy in face detection.

face_recognition.face_encodings() generates face encodings (vectors representing facial features) for each detected face in both images.

Face Comparison:

The first detected face in the main image (virat_encoding) is chosen as the reference (assumed to be Virat Kohli in this example).

For each detected face in the target image (target_face_locations and target_face_encodings):

face_recognition.compare_faces() compares the face encoding of each target face with the reference face encoding (virat_encoding). It uses a tolerance of 0.6, meaning faces are considered a match if the distance between their encodings is within this threshold.

If a match is found (matches[0] is True), a green bounding box is drawn around the face in target_image_marked. Otherwise, a red bounding box is drawn.

Drawing Bounding Boxes:

Using cv2.rectangle(), the code draws either green or red rectangles (bounding boxes) around faces in the target_image_marked based on whether they match the reference face encoding.

Saving and Displaying Results:

The marked target image (target_image_marked) is saved as 'marked_target_image.jpg' using cv2.imwrite().

Finally, the marked image is displayed using matplotlib.pyplot, showing the original target image with green boxes around matching faces and red boxes around non-matching faces.

Output and Results:

Throughout the process, various print statements provide diagnostic information such as image shapes, detected face locations, and match results for each face in the target image.

This code I provided is more advanced and robust compared to the normal practical for several reasons:

1. Face detection model:

My code uses the CNN model for face detection (`model='cnn'` in face_recognition.face_locations()`), which is generally more accurate than the default HOG model used in your code. This can lead to better face detection, especially in challenging images.

2. Flexible matching:

My code assumes the first face in the main image is the target (Virat Kohli) and compares it to all faces in the target image. The normal practical's code compares all faces in the main image to all faces in the target image, which might lead to false positives if there are multiple people in both images.

3. Visual feedback:

My code marks faces in the target image, showing green boxes for matches and red for non-matches. This provides more informative visual feedback than just marking matches in the main image, as the practical's code does.

4. Error handling:

My code includes checks to ensure faces are found in both images before proceeding, preventing potential errors if no faces are detected.

5. Diagnostic information:

The improved version I provided includes extensive print statements for debugging, helping to identify where issues might occur in the face detection and matching process.

6. Image saving:

My code saves the marked image to a file, ensuring the results are preserved even if there are display issues.

7. Tolerance setting:

My code explicitly sets a face comparison tolerance (0.6), allowing for fine-tuning of the matching strictness.

8. Color handling:

My code includes explicit color space conversions (BGR to RGB) to ensure correct color display with matplotlib.

9. Modular design:

The code is structured into functions, making it more readable and easier to maintain or extend.

10. Use of numpy:

My code uses `np.copy()` for creating image copies, which is generally more efficient than standard Python copying for large arrays like images.

These improvements make the code more robust, informative, and flexible for face detection and matching tasks.

Practical 3

Dumb ROI-Number Plate Detection

In [3]:

```
import cv2
import easyocr
import numpy as np
from matplotlib import pyplot as plt
```

In [14]:

```
def find_number_plate(image_path):
    # Load the image
    img = cv2.imread(image_path)

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

    # Apply Gaussian blur to reduce noise and improve contour detection
    blurred = cv2.GaussianBlur(gray, (5, 5), 0)

    # Use edge detection to find potential contours
    # Edges are where there is a sudden change in brightness or color
    # Change range is provided as 50 - 150.
    # More smaller the value output becomes noisy.
    # More higher the value output becomes too strict as it could miss some
    # edges. Hence, start with a mid value like 50 and experiment.
    edges = cv2.Canny(blurred, 50, 450)

    # Find contours in the edge-detected image. It`s like connecting the edges
    # located by the Canny to come up with a closed surface area.
    contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    # Filter contours based on area to find potential number plate region.
    # Check and track cordinates of the closed regions ,approximating with
    # a size of a number plate.
    potential_plates = []
    for contour in contours:
        area = cv2.contourArea(contour)
        if 50000 < area < 100000:  # Adjust these thresholds based on your specific images
            potential_plates.append(contour)
```

```
    # Draw the potential plates on a copy of the original image
    plate_img = img.copy()
    cv2.drawContours(plate_img, potential_plates, -1, (0, 255, 0), 2)

    # Display the image with potential plates
    plt.imshow(cv2.cvtColor(plate_img, cv2.COLOR_BGR2RGB))
    plt.title("Potential Number Plates")
    plt.axis('off')
    plt.show()
```

15]:

```
# Example usage
image_path = 'vehicle_2.jpg'
find_number_plate(image_path)
```

1. Imports

cv2: OpenCV library for image processing tasks.

easyocr: EasyOCR library for Optical Character Recognition (though not used in this specific snippet).

numpy: Library for numerical operations.

pyplot: From the Matplotlib library, used for displaying images and plots.

2. Function to Find Number Plates

cv2.imread(image_path): Reads the image from the specified path.

cv2.cvtColor(img, cv2.COLOR_BGR2GRAY): Converts the image to grayscale to simplify further processing.

cv2.GaussianBlur(gray, (5, 5), 0): Applies Gaussian blur to reduce noise, which helps in better contour detection.

cv2.Canny(blurred, 50, 450): Uses the Canny edge detection algorithm to find edges in the blurred grayscale image. The thresholds of 50 and 450 are used to control the sensitivity of the edge detection.

cv2.findContours(edges, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE): Finds contours in the edge-detected image. Contours are continuous lines or curves that bound or cover the full boundary of an object in an image.

This loop iterates through all the detected contours. For each contour, it calculates the area using cv2.contourArea(contour).

If the area of the contour falls within a specified range (between 50,000 and 100,000 pixels in this example), it is considered a potential number plate and added to the potential_plates list. These thresholds can be adjusted based on the specific characteristics of the images being processed.

img.copy(): Creates a copy of the original image to draw contours on.

cv2.drawContours(plate_img, potential_plates, -1, (0, 255, 0), 2): Draws the potential plate contours on the copied image in green color with a thickness of 2 pixels.

cv2.cvtColor(plate_img, cv2.COLOR_BGR2RGB): Converts the image to RGB format for proper display using Matplotlib.

plt.imshow(), plt.title(), plt.axis('off'), plt.show(): Matplotlib functions to display the image with the title "Potential Number Plates" and no axis labels.

3. Example Usage

Specifies the path to the image file (vehicle_2.jpg).

Calls find_number_plate(image_path) to process the image and display potential number plates.

Vehicle number plate detection-OCR

In [1]:
```
pip install easyocr
```

```
Collecting easyocr
  Using cached easyocr-1.7.1-py3-none-any.whl (2.9 MB)
Requirement already satisfied: scikit-image in c:\users\asus\python data science\image processing\image_processing\lib\site-p
ackages (from easyocr) (0.21.0)
Collecting torchvision>=0.5
  Using cached torchvision-0.16.2-cp38-cp38-win_amd64.whl (1.1 MB)
Collecting pyclipper
  Using cached pyclipper-1.3.0.post5-cp38-cp38-win_amd64.whl (108 kB)
Collecting torch
  Downloading torch-2.1.2-cp38-cp38-win_amd64.whl (192.3 MB)
Requirement already satisfied: Pillow in c:\users\asus\python data science\image processing\image_processing\lib\site-package
s (from easyocr) (10.2.0)
Collecting Shapely
  Downloading shapely-2.0.2-cp38-cp38-win_amd64.whl (1.5 MB)
Requirement already satisfied: scipy in c:\users\asus\python data science\image processing\image_processing\lib\site-packages
(from easyocr) (1.10.1)
Collecting ninja
  Downloading ninja-1.11.1.1-py2.py3-none-win_amd64.whl (312 kB)
Collecting python-bidi
  Downloading python_bidi-0.4.2-py2.py3-none-any.whl (30 kB)
Collecting opencv-python-headless
  Downloading opencv_python_headless-4.9.0.80-cp37-abi3-win_amd64.whl (38.5 MB)
Requirement already satisfied: numpy in c:\users\asus\python data science\image processing\image_processing\lib\site-packages
(from easyocr) (1.24.4)
Collecting PyYAML
  Using cached PyYAML-6.0.1-cp38-cp38-win_amd64.whl (157 kB)
Requirement already satisfied: networkx>=2.8 in c:\users\asus\python data science\image processing\image_processing\lib\site-
packages (from scikit-image->easyocr) (3.1)
Requirement already satisfied: lazy_loader>=0.2 in c:\users\asus\python data science\image processing\image_processing\lib\si
te-packages (from scikit-image->easyocr) (0.3)
Requirement already satisfied: imageio>=2.27 in c:\users\asus\python data science\image processing\image_processing\lib\site-
packages (from scikit-image->easyocr) (2.33.1)
Requirement already satisfied: packaging>=21 in c:\users\asus\python data science\image processing\image_processing\lib\site-
packages (from scikit-image->easyocr) (23.2)
Requirement already satisfied: tifffile>=2022.8.12 in c:\users\asus\python data science\image processing\image_processing\lib
\site-packages (from scikit-image->easyocr) (2023.7.10)

\site-packages (from scikit-image->easyocr) (2023.7.10)
Requirement already satisfied: PyWavelets>=1.1.1 in c:\users\asus\python data science\image processing\image_processing\lib\s
ite-packages (from scikit-image->easyocr) (1.4.1)
Requirement already satisfied: requests in c:\users\asus\python data science\image processing\image_processing\lib\site-packa
ges (from torchvision>=0.5->easyocr) (2.31.0)
Collecting sympy
  Downloading sympy-1.12-py3-none-any.whl (5.7 MB)
Collecting fsspec
  Downloading fsspec-2023.12.2-py3-none-any.whl (168 kB)
Collecting filelock
  Downloading filelock-3.13.1-py3-none-any.whl (11 kB)
Requirement already satisfied: typing-extensions in c:\users\asus\python data science\image processing\image_processing\lib\s
ite-packages (from torch->easyocr) (4.9.0)
Collecting jinja2
  Downloading Jinja2-3.1.3-py3-none-any.whl (133 kB)
Requirement already satisfied: six in c:\users\asus\python data science\image processing\image_processing\lib\site-packages
(from python-bidi->easyocr) (1.16.0)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\asus\python data science\image processing\image_processing\lib
\site-packages (from requests->torchvision>=0.5->easyocr) (2023.11.17)
Requirement already satisfied: idna<4,>=2.5 in c:\users\asus\python data science\image processing\image_processing\lib\site-p
ackages (from requests->torchvision>=0.5->easyocr) (3.6)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\asus\python data science\image processing\image_processin
g\lib\site-packages (from requests->torchvision>=0.5->easyocr) (3.3.2)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\asus\python data science\image processing\image_processing\lib
\site-packages (from requests->torchvision>=0.5->easyocr) (2.1.0)
Collecting mpmath>=0.19
  Downloading mpmath-1.3.0-py3-none-any.whl (536 kB)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\asus\python data science\image processing\image_processing\lib\sit
e-packages (from jinja2->torch->easyocr) (2.1.3)
Installing collected packages: mpmath, sympy, fsspec, filelock, jinja2, torch, torchvision, pyclipper, Shapely, ninja, python
-bidi, opencv-python-headless, PyYAML, easyocr
Successfully installed PyYAML-6.0.1 Shapely-2.0.2 easyocr-1.7.1 filelock-3.13.1 fsspec-2023.12.2 jinja2-3.1.3 mpmath-1.3.0 ni
nja-1.11.1.1 opencv-python-headless-4.9.0.80 pyclipper-1.3.0.post5 python-bidi-0.4.2 sympy-1.12 torch-2.1.2 torchvision-0.16.
2
Note: you may need to restart the kernel to use updated packages.
```

In [1]:
```
import cv2
import easyocr
from matplotlib import pyplot as plt
```

In [2]:
```
def preprocess_image(image_path):
    # Load the image
    img = cv2.imread(image_path)

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

    # Apply Gaussian blur to reduce noise and improve OCR accuracy
    blurred = cv2.GaussianBlur(gray, (5, 5), 0)

    return blurred
```

In [3]:
```
def perform_ocr(image, reader):
    # Perform OCR on the preprocessed image
    result = reader.readtext(image)

    # Extract and return the recognized text
    recognized_text = ' '.join([entry[1] for entry in result])
    return recognized_text
```

In [4]:
```
def display_image_with_results(image, recognized_text):
    # Display the image with recognized text
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.title(f"Recognized Text: {recognized_text}")
    plt.axis('off')
    plt.show()
```

In [5]:
```
def number_plate_recognition(image_path):
    # Preprocess the image
    preprocessed_image = preprocess_image(image_path)

    # Create an OCR reader using the 'en' language for English text
    reader = easyocr.Reader(['en'])

    # Perform OCR on the preprocessed image
    recognized_text = perform_ocr(preprocessed_image, reader)

    # Display the results
    display_image_with_results(cv2.imread(image_path), recognized_text)
```

In [9]:
```
# Example usage
image_path = 'plate.jpg'
number_plate_recognition(image_path)
```

```
Neither CUDA nor MPS are available - defaulting to CPU. Note: This module is much faster with a GPU.
```

1. Imports

cv2: OpenCV library for image processing.

easyocr: EasyOCR library for Optical Character Recognition (OCR).

pyplot: From the Matplotlib library, used for displaying images and plots.

2. Image Preprocessing Function

preprocess_image(image_path): This function takes an image file path as input.

cv2.imread(image_path): Reads the image from the given path.

cv2.cvtColor(img, cv2.COLOR_BGR2GRAY): Converts the image to grayscale.

cv2.GaussianBlur(gray, (5, 5), 0): Applies Gaussian blur to reduce noise and improve OCR accuracy.

Returns the preprocessed (blurred grayscale) image.

3. OCR Function

perform_ocr(image, reader): This function takes a preprocessed image and an OCR reader as input.

reader.readtext(image): Uses the OCR reader to read text from the image.

Extracts the recognized text from the OCR result.

Returns the recognized text as a single string.

4. Display Function

display_image_with_results(image, recognized_text): This function takes an image and recognized text as input.

Displays the image with the recognized text as the title.

cv2.cvtColor(image, cv2.COLOR_BGR2RGB): Converts the image to RGB format for proper display using Matplotlib.

plt.imshow(), plt.title(), plt.axis('off'), plt.show(): Matplotlib functions to display the image without axis labels.

5. Main Function for Number Plate Recognition

number_plate_recognition(image_path): This function orchestrates the entire number plate recognition process.

Calls preprocess_image(image_path) to preprocess the image.

Creates an EasyOCR reader for English text (easyocr.Reader(['en'])).

Calls perform_ocr(preprocessed_image, reader) to perform OCR on the preprocessed image.

Calls display_image_with_results(cv2.imread(image_path), recognized_text) to display the original image with the recognized text.

Vehicle number plate detection - Ensembled OCR for imges with noise

Ensembled OCR for Images with noise.ipynb

File Edit View Insert Runtime Tools Help Last saved at 22:52

+ Code + Text

```
import cv2
import easyocr
import matplotlib.pyplot as plt
import numpy as np
```

```
def ensemble_ocr(image, readers):
    # Perform OCR using multiple OCR models
    ocr_results = []
    for reader in readers:
        try:
            results = reader.readtext(image)
            ocr_results.append(results)
        except Exception as e:
            print(f"Error in OCR with {reader}: {e}")

    return ocr_results
```

Ensembled OCR for Images with noise.ipynb

File Edit View Insert Runtime Tools Help Last saved at 22:52

+ Code + Text

```
def display_image_with_ocr(image_path, readers, confidence_threshold=0.5):
    # Read the image using OpenCV
    image = cv2.imread(image_path)

    # Convert BGR image to RGB for display in matplotlib
    image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    # Display the image using plt
    plt.imshow(image_rgb)
    plt.axis('off')  # Turn off axis labels
    plt.show()

    # Ensemble OCR using EasyOCR and Tesseract
    try:
        ocr_results = ensemble_ocr(image, readers)

        print("OCR Results:")
        for i, reader_results in enumerate(ocr_results):
            print(f"OCR Model {i + 1}:")
            for result in reader_results:
                text, confidence = result[1], result[2]
                print(f"  Text: {text}, Confidence: {confidence}")
```

Ensembled OCR for Images with noise.ipynb

File Edit View Insert Runtime Tools Help Last saved at 22:52

+ Code + Text

```
        # Draw bounding boxes on the image based on the ensemble OCR results
        for reader_results in ocr_results:
            for result in reader_results:
                (top_left, top_right, bottom_right, bottom_left) = result[0]
                pts = np.array([top_left, top_right, bottom_right, bottom_left], dtype=int)
                pts = pts.reshape((-1, 1, 2))
                cv2.polylines(image, [pts], isClosed=True, color=(0, 255, 0), thickness=2)

        # Display the image with ensemble OCR results
        plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
        plt.axis('off')
        plt.show()

    except Exception as e:
        print(f"Error in displaying results: {e}")
```

```
# Replace 'path/to/your/image.jpg' with the actual path to your image
image_path = 'vehicle.jpg'
```

```
# Initialize EasyOCR readers with different models/languages
reader_easyocr_en = easyocr.Reader(['en'])
reader_easyocr_fr = easyocr.Reader(['fr'])
```

Ensembled OCR for Images with noise.ipynb

File Edit View Insert Runtime Tools Help Last saved at 22:52

+ Code + Text

```
# Add more readers as needed
# Create a list of readers
readers = [reader_easyocr_en, reader_easyocr_fr]
```

```
# Display the image with ensemble OCR results
display_image_with_ocr(image_path, readers)
```

```
OCR Results:
OCR Model 1:
  Text: GLai :44, Confidence: 0.058805191813383
OCR Model 2:
  Text: Gplad ,144, Confidence: 0.039946071437862996
```

Start coding or generate with AI.

Now Let's delve into the code and its functionality in detail:

Imports and Function Definitions:

The code imports necessary libraries: cv2 for image processing, easyocr for optical character recognition (OCR), matplotlib.pyplot for image display, and numpy for numerical operations.

Two functions are defined:

ensemble_ocr(image, readers): Performs OCR on an image using multiple OCR models provided in the readers list.

display_image_with_ocr(image_path, readers, confidence_threshold=0.5): Loads an image using OpenCV (cv2.imread), displays it using matplotlib, performs OCR using multiple OCR models, and overlays bounding boxes on the image based on the OCR results.

Ensemble OCR Function (ensemble_ocr):

This function iterates through each OCR model (reader) provided in the readers list.

It attempts to read text from the image (reader.readtext(image)) and appends the OCR results (results) to ocr_results.

If an exception occurs during OCR, it prints an error message.

Display Image with OCR Function (display_image_with_ocr):

Loads an image specified by image_path using cv2.imread.

Converts the BGR image format (used by OpenCV) to RGB format suitable for matplotlib display (cv2.cvtColor(image, cv2.COLOR_BGR2RGB)).

Displays the original image using matplotlib.pyplot.imshow.

Ensemble OCR and Bounding Box Drawing:

Calls ensemble_ocr(image, readers) to obtain OCR results (ocr_results) from multiple OCR models.

Prints the OCR results, iterating over each model's results and displaying the recognized text and confidence level (result[1] and result[2]).

Iterates through each OCR result (reader_results) and extracts bounding box coordinates (result[0]).

Draws green bounding boxes (cv2.polylines) on the original image (image) around detected text regions based on the OCR results.

Displaying Image with Overlayed OCR Results:

Uses matplotlib.pyplot.imshow again to display the original image (image) with overlaid green bounding boxes representing detected text regions.

Turns off axis labels (plt.axis('off')) for cleaner visualization.

If any exceptions occur during this process, they are caught and an error message is printed.

Initialization and Execution:

Specifies the path to the image (image_path) that needs OCR processing ('vehicle.jpg' in this case).

Initializes easyocr.Reader instances (reader_easyocr_en and reader_easyocr_fr) for English and French OCR, respectively.

Creates a list readers containing these OCR readers to be passed to ensemble_ocr and display_image_with_ocr.

Execution of OCR and Visualization:

Calls display_image_with_ocr(image_path, readers) to execute OCR using the specified readers and display the original image with overlaid OCR results.

Improved code for the above practical

Ensembled OCR for Images with noise.ipynb

File Edit View Insert Runtime Tools Help Saving failed since 23:36

+ Code + Text

```
!pip install easyocr pytesseract fuzzywuzzy python-Levenshtein imutils

import cv2
import numpy as np
import matplotlib.pyplot as plt
import easyocr
import pytesseract
from PIL import Image
import re
from fuzzywuzzy import fuzz
from google.colab.patches import cv2_imshow
import imutils

!sudo apt install tesseract-ocr
!sudo apt install libtesseract-dev

def preprocess_for_lpd(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    bfilter = cv2.bilateralFilter(gray, 11, 17, 17)
    edged = cv2.Canny(bfilter, 30, 200)
    return edged

def find_license_plate(preprocessed):
    keypoints = cv2.findContours(preprocessed.copy(), cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
    contours = imutils.grab_contours(keypoints)
    contours = sorted(contours, key=cv2.contourArea, reverse=True)[:10]

    location = None
    for contour in contours:
        approx = cv2.approxPolyDP(contour, 10, True)
        if len(approx) == 4:
            location = approx
            break

    return location
```

Ensembled OCR for Images with noise.ipynb

File Edit View Insert Runtime Tools Help Saving failed since 23:36

+ Code + Text

```
def preprocess_plate(plate):
    gray = cv2.cvtColor(plate, cv2.COLOR_BGR2GRAY)
    thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)[1]
    return thresh

def ocr_plate(plate, readers):
    results = []

    # EasyOCR
    for reader in readers:
        easy_results = reader.readtext(plate)
        for res in easy_results:
            results.append((res[1], res[2], 'EasyOCR'))

    # Tesseract
    custom_config = r'--oem 3 --psm 6 -c tessedit_char_whitelist=ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789'
    tess_result = pytesseract.image_to_string(Image.fromarray(plate), config=custom_config)
    if tess_result.strip():
        results.append((tess_result.strip(), 1.0, 'Tesseract'))

    return results

def post_process_plate(results):
    processed = []
    for text, conf, engine in results:
        text = re.sub(r'[^A-Z0-9]', '', text.upper())
        if text:
            processed.append((text, conf, engine))
    return processed

def merge_results(results, threshold=80):
    merged = []
    for res in results:
        if not merged or all(fuzz.ratio(res[0], m[0]) < threshold for m in merged):
            merged.append(res)
```

```
        else:
            for i, m in enumerate(merged):
                if fuzz.ratio(res[0], m[0]) >= threshold:
                    if res[1] > m[1]:
                        merged[i] = res
                    break
    return merged

def recognize_license_plate(image_path, readers):
    image = cv2.imread(image_path)
    preprocessed = preprocess_for_lpd(image)
    plate_location = find_license_plate(preprocessed)

    if plate_location is None:
        print("No license plate found.")
        return

    mask = np.zeros(image.shape[:2], dtype="uint8")
    cv2.drawContours(mask, [plate_location], 0, 255, -1)

    (x, y) = np.where(mask == 255)
    (top_x, top_y) = (np.min(x), np.min(y))
    (bottom_x, bottom_y) = (np.max(x), np.max(y))
    plate = image[top_x:bottom_x+1, top_y:bottom_y+1]

    preprocessed_plate = preprocess_plate(plate)
    ocr_results = ocr_plate(preprocessed_plate, readers)
    post_processed = post_process_plate(ocr_results)
    final_results = merge_results(post_processed)

    # Display results
    plt.figure(figsize=(12, 4))
    plt.subplot(131)
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.title('Original Image')
    plt.axis('off')
```

Ensembled OCR for Images with noise.ipynb
File Edit View Insert Runtime Tools Help Saving failed since 23:36
+ Code + Text

```
    plt.subplot(132)
    plt.imshow(cv2.cvtColor(plate, cv2.COLOR_BGR2RGB))
    plt.title('Detected License Plate')
    plt.axis('off')

    plt.subplot(133)
    plt.imshow(preprocessed_plate, cmap='gray')
    plt.title('Preprocessed Plate')
    plt.axis('off')

    plt.tight_layout()
    plt.show()

    print("License Plate Recognition Results:")
    for text, conf, engine in final_results:
        print(f"Text: {text}, Confidence: {conf:.2f}, Engine: {engine}")

    return final_results

# Set the image path
image_path = 'car.jpg'

# Initialize EasyOCR reader
reader_easyocr = easyocr.Reader(['en'])

# Create a list of readers
readers = [reader_easyocr]

# Perform license plate recognition
results = recognize_license_plate(image_path, readers)
```

```
Requirement already satisfied: easyocr in /usr/local/lib/python3.10/dist-packages (1.7.1)
Requirement already satisfied: pytesseract in /usr/local/lib/python3.10/dist-packages (0.3.10)
Requirement already satisfied: fuzzywuzzy in /usr/local/lib/python3.10/dist-packages (0.18.0)
Requirement already satisfied: python-Levenshtein in /usr/local/lib/python3.10/dist-packages (0.25.1)
Requirement already satisfied: imutils in /usr/local/lib/python3.10/dist-packages (0.5.4)
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (from easyocr) (2.3.0+cu121)
Requirement already satisfied: torchvision>=0.5 in /usr/local/lib/python3.10/dist-packages (from easyocr) (0.18.0+cu121)
Requirement already satisfied: opencv-python-headless in /usr/local/lib/python3.10/dist-packages (from easyocr) (4.10.0.84)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from easyocr) (1.11.4)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from easyocr) (1.25.2)
Requirement already satisfied: Pillow in /usr/local/lib/python3.10/dist-packages (from easyocr) (9.4.0)
Requirement already satisfied: scikit-image in /usr/local/lib/python3.10/dist-packages (from easyocr) (0.19.3)
Requirement already satisfied: python-bidi in /usr/local/lib/python3.10/dist-packages (from easyocr) (0.4.2)
Requirement already satisfied: PyYAML in /usr/local/lib/python3.10/dist-packages (from easyocr) (6.0.1)
Requirement already satisfied: Shapely in /usr/local/lib/python3.10/dist-packages (from easyocr) (2.0.4)
Requirement already satisfied: pyclipper in /usr/local/lib/python3.10/dist-packages (from easyocr) (1.3.0.post5)
Requirement already satisfied: ninja in /usr/local/lib/python3.10/dist-packages (from easyocr) (1.11.1.1)
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.10/dist-packages (from pytesseract) (24.1)
Requirement already satisfied: Levenshtein==0.25.1 in /usr/local/lib/python3.10/dist-packages (from python-Levenshtein) (0.25.1)
Requirement already satisfied: rapidfuzz<4.0.0,>=3.8.0 in /usr/local/lib/python3.10/dist-packages (from Levenshtein==0.25.1->python-Levenshtein) (3.9.3)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch->easyocr) (3.15.3)
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/python3.10/dist-packages (from torch->easyocr) (4.12.2)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch->easyocr) (1.12.1)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch->easyocr) (3.3)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch->easyocr) (3.1.4)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch->easyocr) (2023.6.0)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.1.105 in /usr/local/lib/python3.10/dist-packages (from torch->easyocr) (12.1.105)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.1.105 in /usr/local/lib/python3.10/dist-packages (from torch->easyocr) (12.1.105)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.1.105 in /usr/local/lib/python3.10/dist-packages (from torch->easyocr) (12.1.105)
Requirement already satisfied: nvidia-cudnn-cu12==8.9.2.26 in /usr/local/lib/python3.10/dist-packages (from torch->easyocr) (8.9.2.26)
Requirement already satisfied: nvidia-cublas-cu12==12.1.3.1 in /usr/local/lib/python3.10/dist-packages (from torch->easyocr) (12.1.3.1)
Requirement already satisfied: nvidia-cufft-cu12==11.0.2.54 in /usr/local/lib/python3.10/dist-packages (from torch->easyocr) (11.0.2.54)
Requirement already satisfied: nvidia-curand-cu12==10.3.2.106 in /usr/local/lib/python3.10/dist-packages (from torch->easyocr) (10.3.2.106)
Requirement already satisfied: nvidia-cusolver-cu12==11.4.5.107 in /usr/local/lib/python3.10/dist-packages (from torch->easyocr) (11.4.5.107)
Requirement already satisfied: nvidia-cusparse-cu12==12.1.0.106 in /usr/local/lib/python3.10/dist-packages (from torch->easyocr) (12.1.0.106)
Requirement already satisfied: nvidia-nccl-cu12==2.20.5 in /usr/local/lib/python3.10/dist-packages (from torch->easyocr) (2.20.5)
Requirement already satisfied: nvidia-nvtx-cu12==12.1.105 in /usr/local/lib/python3.10/dist-packages (from torch->easyocr) (12.1.105)
Requirement already satisfied: triton==2.3.0 in /usr/local/lib/python3.10/dist-packages (from torch->easyocr) (2.3.0)
Requirement already satisfied: nvidia-nvjitlink-cu12 in /usr/local/lib/python3.10/dist-packages (from nvidia-cusolver-cu12==11.4.5.107->torch->easyocr) (12.5.40)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from python-bidi->easyocr) (1.16.0)
Requirement already satisfied: imageio>=2.4.1 in /usr/local/lib/python3.10/dist-packages (from scikit-image->easyocr) (2.31.6)
Requirement already satisfied: tifffile>=2019.7.26 in /usr/local/lib/python3.10/dist-packages (from scikit-image->easyocr) (2024.6.18)
Requirement already satisfied: PyWavelets>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-image->easyocr) (1.6.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch->easyocr) (2.1.5)
Requirement already satisfied: mpmath<1.4.0,>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from sympy->torch->easyocr) (1.3.0)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
tesseract-ocr is already the newest version (4.1.1-2.1build1).
0 upgraded, 0 newly installed, 0 to remove and 45 not upgraded.
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
libtesseract-dev is already the newest version (4.1.1-2.1build1).
0 upgraded, 0 newly installed, 0 to remove and 45 not upgraded.
WARNING:easyocr.easyocr:Neither CUDA nor MPS are available - defaulting to CPU. Note: This module is much faster with a GPU.
```

Code Explanation:

1. Libraries and Dependencies:

Libraries Installation: Installs necessary Python libraries (easyocr, pytesseract, fuzzywuzzy, python-Levenshtein, imutils) and system dependencies (tesseract-ocr and libtesseract-dev) for OCR and image processing tasks.

2. Image Preprocessing Functions:

Image Preprocessing:

preprocess_for_lpd: Converts the image to grayscale, applies bilateral filtering to preserve edges, and performs Canny edge detection.

preprocess_plate: Converts the license plate region to grayscale, applies Otsu's thresholding to binarize the image.

3. License Plate Localization and Extraction:

License Plate Localization: Uses contour detection (cv2.findContours) on the preprocessed image to find potential license plate contours. It then approximates the contour shape to a quadrilateral (approxPolyDP) and selects the largest suitable contour as the license plate location.

4. OCR Functions:

OCR Processing:

Utilizes EasyOCR and Tesseract for text extraction from the license plate region (plate).

ocr_plate: Performs OCR using EasyOCR and Tesseract. EasyOCR results include text, confidence score, and engine name ('EasyOCR'). Tesseract is configured with specific settings (custom_config) to recognize alphanumeric characters only.

5. Post-processing and Result Merging:

Post-processing and Result Fusion:

post_process_plate: Cleans up OCR results by removing non-alphanumeric characters and converting text to uppercase.

merge_results: Merges OCR results based on fuzzy matching similarity (fuzz.ratio). If a similar plate number is found, it retains the result with higher confidence.

6. Main Recognition Function:

Main Recognition Function (recognize_license_plate):

Loads the image specified by image_path and performs license plate recognition.

Detects the license plate region using find_license_plate and extracts it from the original image.

Preprocesses the plate image, performs OCR using EasyOCR and Tesseract (ocr_plate), cleans up and merges OCR results (post_process_plate and merge_results), and finally displays the original image, detected plate, and preprocessed plate with overlaid text regions.

Prints and returns the final recognized license plate results.

7. Initialization and Execution:

Initialization and Execution:

Sets the image path (image_path) and initializes EasyOCR with English language configuration (reader_easyocr).

Creates a list of OCR readers (readers) and performs license plate recognition using recognize_license_plate.

Reasons for Superiority of the improved code:

1. Modular Approach and Integration of Multiple OCR Engines:

Improved code: Utilizes a modular approach where each component (preprocessing, OCR, post-processing) is separated into distinct functions (preprocess_for_lpd, find_license_plate, ocr_plate, etc.). This allows for easier maintenance, debugging, and potential reuse in other projects.

Practical's code: Performs OCR using multiple models but lacks modular separation, making it harder to manage and modify.

2. Advanced Preprocessing and License Plate Localization:

Improved code: Implements advanced preprocessing techniques (bilateralFilter, Canny edge detection) and contour analysis (findContours, approxPolyDP) to robustly locate the license plate in varying conditions.

Practical's code: Directly applies OCR without detailed preprocessing or specific methods for license plate localization, potentially leading to less accurate results, especially in complex scenarios.

3. OCR Engine Selection and Customization:

Improved code: Integrates both EasyOCR and Tesseract with specific configurations (--oem 3 --psm 6 -c tessedit_char_whitelist=ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 for Tesseract) tailored for recognizing license plate characters effectively.

Practical's code: Uses EasyOCR and potentially other OCR models, but lacks specific configurations or customization options, limiting its adaptability and accuracy in different contexts.

4. Post-processing and Result Fusion:

Improved code: Applies post-processing to clean OCR results (post_process_plate) by removing non-alphanumeric characters and merging similar results (merge_results) using fuzzy matching (fuzz.ratio). This improves accuracy by refining the extracted text.

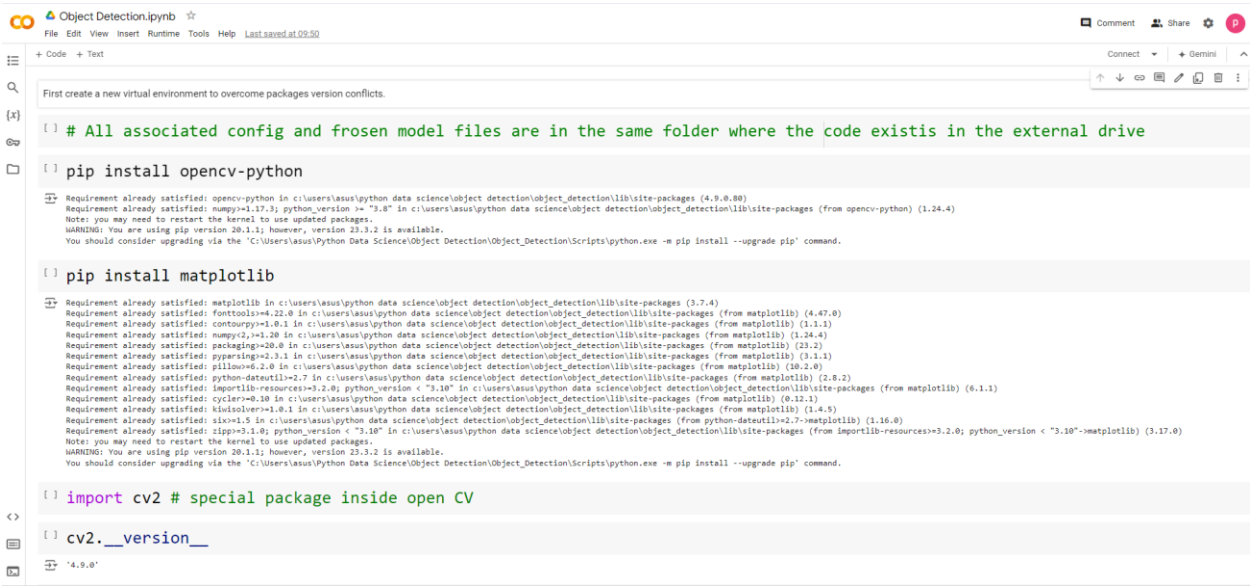
Practical's code: Does not include explicit post-processing or result merging steps, which could lead to more noisy or less coherent OCR outputs, especially when dealing with diverse fonts or degraded images.

5. Visualization and Debugging Tools:

Improved code: Includes visualization of intermediate results (original image, detected plate, preprocessed plate) using matplotlib, aiding in understanding the processing pipeline and facilitating debugging.

Practical's code: Also visualizes results but does not break down the process into distinct stages, making it harder to diagnose errors or optimize performance.

Practical 4 -Live object detection model



Object Detection.ipynb

File Edit View Insert Runtime Tools Help Last saved at 09:50

Comment Share

+ Code + Text

Connect + Gemini

```
import matplotlib.pyplot as plt
```

```
#mobile net is pre-trained model for image recognition and object analysis. It`s frozen model is avilable in public as a .pb
# file. We have used that, as mentioned below. It required some additional configurations, which is provided by the pbtext fi
config_file='ssd_mobilenet_v3_large_coco_2020_01_14.pbtxt'
frozen_model='frozen_inference_graph.pb'
```

```
model=cv2.dnn_DetectionModel(frozen_model,config_file)
```

```
# class label file has been downloaded. This is used as enumerator to make the int output provided into text.
# currently, mobile net works for 80 classes. Though we add new classes for the label file, it will not work, hecnce the
# model is not trained
classLabels=[]
file_name="Labels.txt"
with open (file_name,'rt') as fpt:
    classLabels=fpt.read().rstrip('\n').split('\n')
```

```
print(classLabels)
```

```
['person', 'bicycle', 'car', 'motorbike', 'aeroplane', 'bus', 'train', 'truck', 'boat', 'traffic light', 'fire hydrant', 'stop sign', 'parking meter', 'bench', 'bird', 'cat', 'dog', 'horse', 'sheep', 'cow', 'elephant', 'bear', 'zebra', 'giraffe', 'ba
```

```
# some adjustments are made to the input. Then the captured input will go through these adjustments. Becz, that1s how config
# file is organized and , once satisfying these adjusments only, the model can effectively asses the input images / videos
model.setInputSize(320,320)
model.setInputScale(1.0/127.5)
```

Object Detection.ipynb

File Edit View Insert Runtime Tools Help Last saved at 09:50

Comment

+ Code + Text

Connect

```
model.setInputSize(320,320)
model.setInputScale(1.0/127.5)
model.setInputMean((127.5,127.5,127.5))
model.setInputSwapRB(True)
```

```
< cv2.dnn.Model 000001EE457C8790>
```

```
img=cv2.imread('download.jpg')
```

```
plt.imshow(img) # this is why we need pyplotlib
```

```
<matplotlib.image.AxesImage at 0x1ee457f1d30>
```


```
plt.imshow(cv2.cvtColor(img,cv2.COLOR_BGR2RGB)) # some color change filter
```

Object Detection.ipynb

File Edit View Insert Runtime Tools Help Last saved at 09:50

Comment Share

+ Code + Text

Connect + Gemini

```
<matplotlib.image.AxesImage at 0x1ee468ea340>
```


```
ClassIndex,confidence,bbox=model.detect(img,confThreshold=0.5) # if the detection confidence is more than 50% display the o/p
```

```
print(ClassIndex) # numerical predications of classes. Check with the class label list, strating from 1. Then, those are
# man and car
```

```
[1 3]
```

```
font_scale=2
font=cv2.FONT_HERSHEY_PLAIN # bounding box text
for ClassInd,conf,boxes in zip(ClassIndex.flatten(),confidence.flatten(),bbox):
    # configuration of bounding box and associated text
    cv2.rectangle(img,boxes,(255,0,0),2)
    cv2.putText(img,classLabels[ClassInd-1],(boxes[0]+10,boxes[1]+40),font,fontScale=font_scale,color=(0,255,0),thickness=3)
```

Object Detection.ipynb

File Edit View Insert Runtime Tools Help Last saved at 09:50

Comment Share

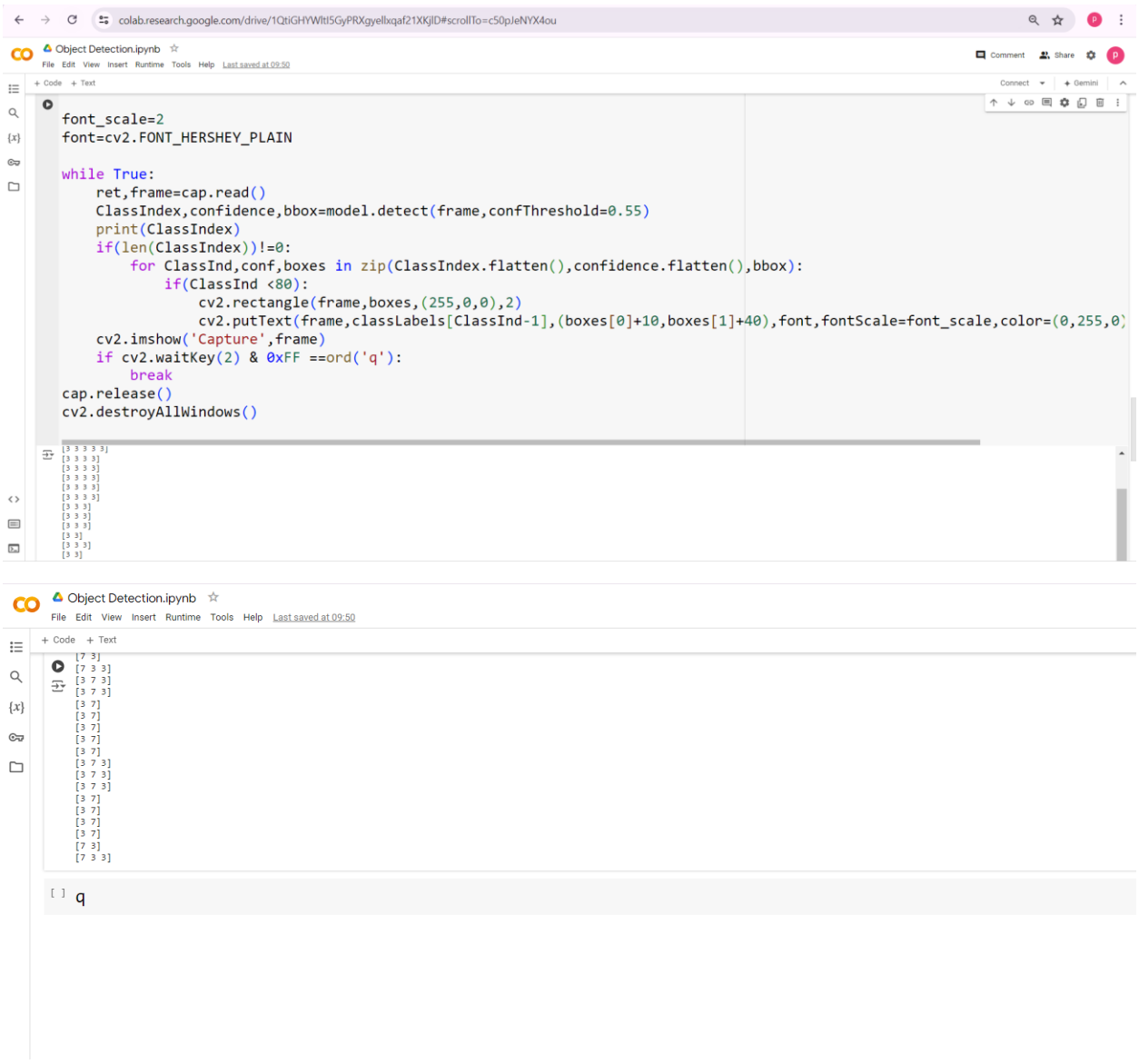
+ Code + Text

Connect + Gemini

```
plt.imshow(cv2.cvtColor(img,cv2.COLOR_BGR2RGB))
```

```
<matplotlib.image.AxesImage at 0x2a0e6b08430>
```


```
# same logic will work for both video and images.
# Type letter q to exist the execution on the image / video canvas

cap=cv2.VideoCapture('video.mp4') # for mp4 video feed  capture

#cap=cv2.VideoCapture(0) # for laptop or web camera capture. For external cameras, this 0 should be changed to 1.
if not cap.isOpened():
    cap=cv2.VideoCapture(0)
if not cap.isOpened():
    raise IOError("Cannot Open Video / Web Cam")
```



1. Virtual Environment Setup

Creating a new virtual environment ensures that package versions do not conflict. This is particularly important when dealing with machine learning models and their dependencies.

2. Installing Required Packages

opencv-python: Provides OpenCV functionalities for image and video processing.

matplotlib: Used for displaying images and plots.

3. Importing Libraries

cv2: Main module of OpenCV, which includes functions for image processing and computer vision.

matplotlib.pyplot: Used for plotting and displaying images.

4. Model Configuration and Loading

config_file: Path to the configuration file for the MobileNet model.

frozen_model: Path to the pre-trained model file (frozen graph).

model: Creates an OpenCV DNN detection model using the configuration and model files.

5. Loading Class Labels

classLabels: List to hold the names of object classes.

Labels.txt: File containing the class labels (80 classes for COCO dataset).

with open: Reads the class labels from the file and stores them in the list.

6. Model Input Configuration

setInputSize(320, 320): Resizes the input image to 320x320 pixels.

setInputScale(1.0 / 127.5): Scales the input image.

setInputMean((127.5, 127.5, 127.5)): Sets the mean values for image normalization.

setInputSwapRB(True): Swaps the red and blue channels (necessary for correct color representation).

7. Image Processing and Display

cv2.imread('download.jpg'): Reads the input image.

plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB)): Converts the BGR image to RGB and displays it using Matplotlib.

model.detect(img, confThreshold=0.5): Detects objects in the image with a confidence threshold of 50%.

8. Drawing Bounding Boxes on Detected Objects

for loop: Iterates over the detected objects.

cv2.rectangle: Draws a bounding box around each detected object.

cv2.putText: Puts the class label text on the image near the bounding box.

plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB)): Displays the image with bounding boxes and labels.

9. Video Processing and Real-Time Detection

cv2.VideoCapture('video.mp4'): Opens a video file.

cap = cv2.VideoCapture(0): If the video file cannot be opened, it tries to open the webcam.

if not cap.isOpened(): Checks if the video capture is successful, otherwise raises an error.

10. Real-Time Object Detection Loop

while True: Continuously captures frames from the video source.

cap.read(): Reads a frame from the video source.

model.detect(frame, confThreshold=0.55): Detects objects in the frame with a confidence threshold of 55%.

if len(ClassIndex) != 0: Checks if any objects are detected.

for loop: Iterates over detected objects and draws bounding boxes and labels.

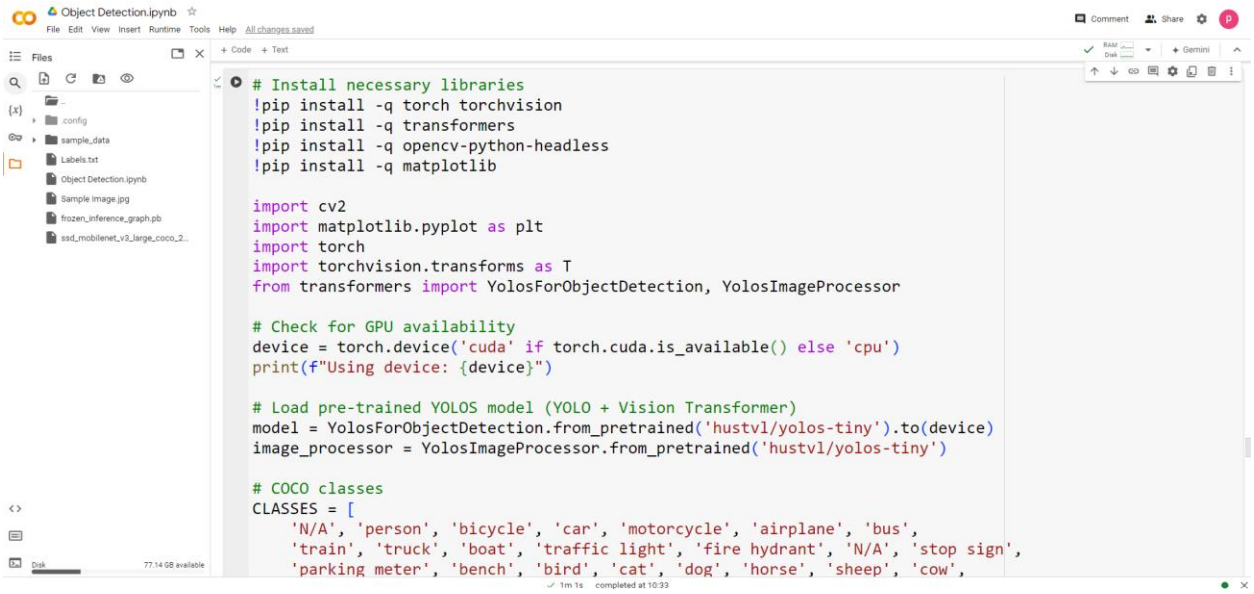
cv2.imshow('Capture', frame): Displays the video frame with detections.

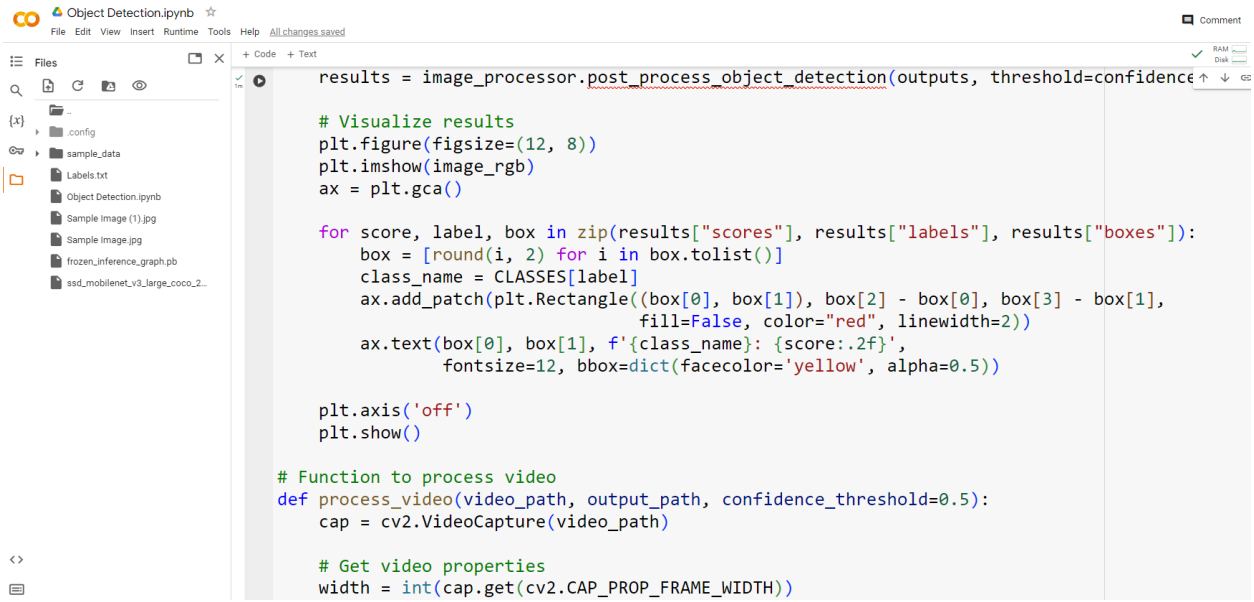
cv2.waitKey(2) & 0xFF == ord('q'): Waits for the 'q' key to be pressed to exit the loop.

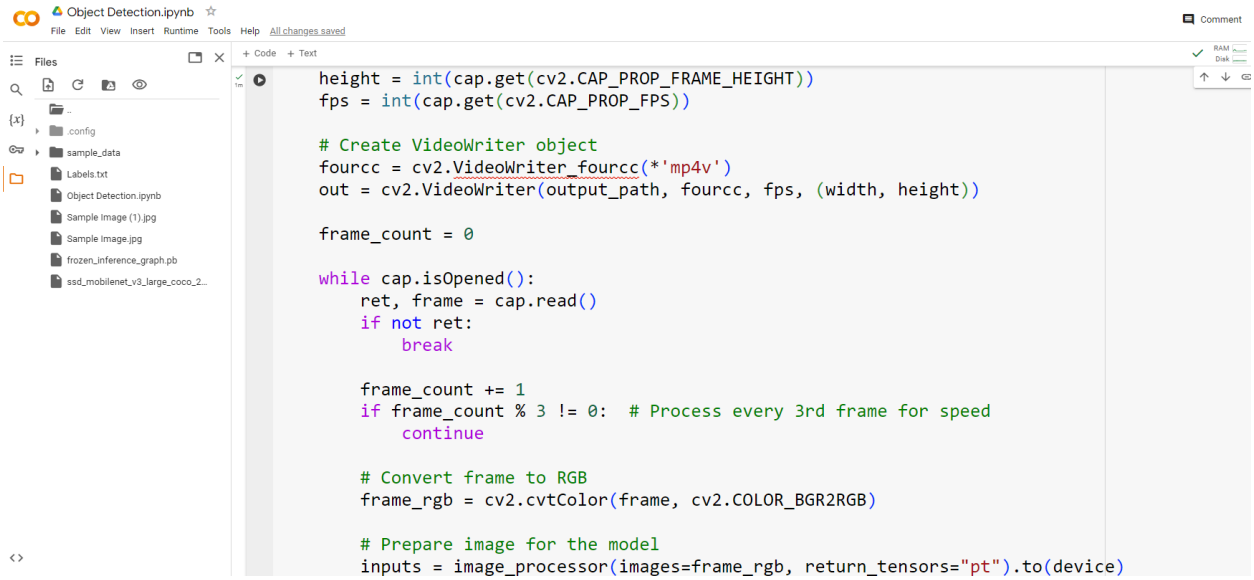
cap.release(): Releases the video capture object.

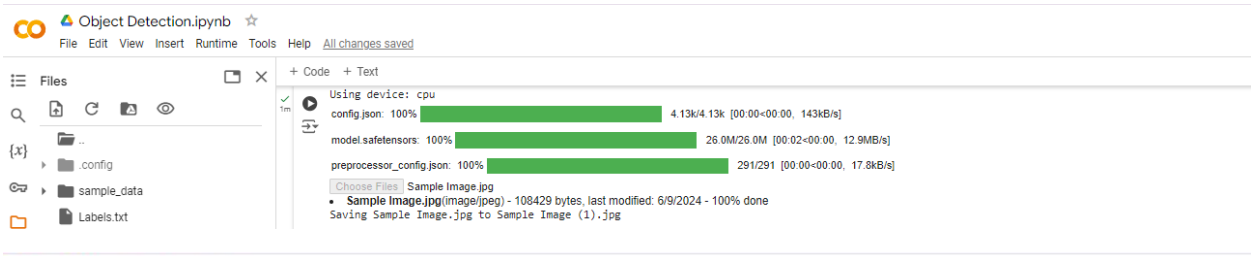
cv2.destroyAllWindows(): Closes all OpenCV windows.

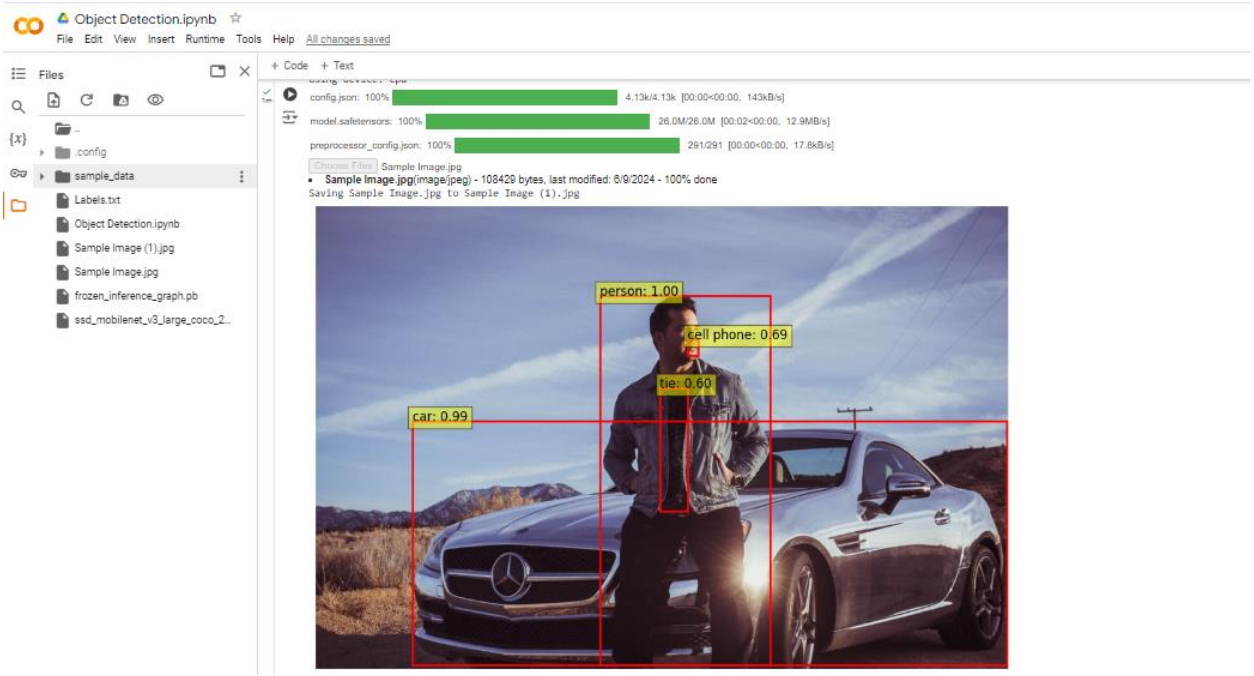
Improves version of this code




Object Detection.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Files
+ Code + Text
RAM
Disk
.config
sample_data
Labels.txt
Object Detection.ipynb
Sample Image (1).jpg
Sample Image.jpg
frozen_inference_graph.pb
ssd_mobilenet_v3_large_coco_2...
results = image_processor.post_process_object_detection(outputs, threshold=confidence
# Visualize results
plt.figure(figsize=(12, 8))
plt.imshow(image_rgb)
ax = plt.gca()
for score, label, box in zip(results["scores"], results["labels"], results["boxes"]):
box = [round(i, 2) for i in box.tolist()]
class_name = CLASSES[label]
ax.add_patch(plt.Rectangle((box[0], box[1]), box[2] - box[0], box[3] - box[1],
fill=False, color="red", linewidth=2))
ax.text(box[0], box[1], f'{class_name}: {score:.2f}',
fontsize=12, bbox=dict(facecolor='yellow', alpha=0.5))
plt.axis('off')
plt.show()
# Function to process video
def process_video(video_path, output_path, confidence_threshold=0.5):
cap = cv2.VideoCapture(video_path)
# Get video properties
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))


Object Detection.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Files
+ Code + Text
RAM
Disk
.config
sample_data
Labels.txt
Object Detection.ipynb
Sample Image (1).jpg
Sample Image.jpg
frozen_inference_graph.pb
ssd_mobilenet_v3_large_coco_2...
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
fps = int(cap.get(cv2.CAP_PROP_FPS))
# Create VideoWriter object
fourcc = cv2.VideoWriter_fourcc(*'mp4v')
out = cv2.VideoWriter(output_path, fourcc, fps, (width, height))
frame_count = 0
while cap.isOpened():
ret, frame = cap.read()
if not ret:
break
frame_count += 1
if frame_count % 3 != 0: # Process every 3rd frame for speed
continue
# Convert frame to RGB
frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
# Prepare image for the model
inputs = image_processor(images=frame_rgb, return_tensors="pt").to(device)





1. Library Installation

torch and torchvision: Libraries for deep learning, with torchvision providing datasets and models specifically for computer vision.

transformers: A library by Hugging Face that provides pre-trained models, including YOLOS.

opencv-python-headless: A version of OpenCV that is suitable for environments without GUI, like servers.

matplotlib: A library for plotting and visualizing images.

2. Importing Libraries

cv2: Provides functionalities for image and video processing.

matplotlib.pyplot: Used for displaying images and plots.

torch: Provides functionalities for tensor operations and model management.

torchvision.transforms as T: Contains common image transformations.

YolosForObjectDetection and YolosImageProcessor: Specific modules from the transformers library for using YOLOS.

3. Device Configuration

torch.device: Checks if a GPU is available and sets the device accordingly. Utilizing a GPU significantly speeds up model inference.

4. Model and Processor Initialization

YolosForObjectDetection.from_pretrained: Loads the pre-trained YOLOS model.

YolosImageProcessor.from_pretrained: Loads the corresponding image processor for preprocessing input images.

to(device): Moves the model to the specified device (CPU or GPU).

5. COCO Class Labels

CLASSES: List of object classes from the COCO (Common Objects in Context) dataset. These labels correspond to the output indices of the YOLOS model.

6. Object Detection in Images

cv2.imread: Reads the image from the specified path.

cv2.cvtColor: Converts the image from BGR (OpenCV default) to RGB format.

image_processor: Prepares the image for the model by resizing, normalizing, and converting it to a tensor.

torch.no_grad(): Disables gradient calculation to speed up inference and reduce memory usage.

**model(inputs): Feeds the preprocessed image into the model to get detection outputs.

target_sizes: Tensor containing the original image size for scaling the bounding boxes.

post_process_object_detection: Converts model outputs to bounding boxes, scores, and labels, filtering by the confidence threshold.

matplotlib: Used to draw bounding boxes and labels on the image and display it.

7. Object Detection in Videos

cv2.VideoCapture: Opens the video file.

cap.get: Retrieves video properties such as width, height, and frames per second (fps).

cv2.VideoWriter: Initializes the video writer for saving the processed video.

frame_count % 3 != 0: Processes every third frame to speed up processing.

cv2.cvtColor: Converts each frame to RGB format.

image_processor: Prepares each frame for the model.

torch.no_grad(): Disables gradient calculation for inference.

post_process_object_detection: Converts model outputs to bounding boxes, scores, and labels.

cv2.rectangle and cv2.putText: Draw bounding boxes and labels on each frame.

out.write: Writes the processed frame to the output video.

cap.release() and out.release(): Releases the video capture and writer objects.

cv2.destroyAllWindows(): Closes any OpenCV windows.

8. Example Usage

files.upload(): Allows the user to upload an image file in Google Colab.

detect_objects: Calls the function to perform object detection on the uploaded image.

Why the Improved code is superior,

1. Model Architecture and Performance

YOLOS (You Only Look One-level Series with Vision Transformer):

Advanced Architecture: YOLOS combines the strengths of the YOLO object detection framework with Vision Transformers (ViT). Vision Transformers provide superior feature extraction capabilities compared to traditional convolutional networks.

Pre-trained on Large Datasets: The YOLOS model has been pre-trained on large datasets like COCO, ensuring robust performance and higher accuracy in object detection tasks.

State-of-the-art Performance: Transformers have shown significant improvements in tasks like image classification and object detection, outperforming many traditional models.

MobileNet SSD:

Older Architecture: MobileNet SSD is a convolutional neural network designed for efficient object detection on mobile and embedded devices. While it is lightweight and fast, it may not achieve the same level of accuracy as newer models like YOLOS.

Fixed Number of Classes: The MobileNet SSD used here is limited to detecting 80 classes defined by the COCO dataset. Any changes or additions to classes require retraining the model, which is not straightforward.

2. Code Complexity and Flexibility

YOLOS-based Code:

Ease of Use: The code uses high-level libraries like transformers and torchvision, which abstract many complexities. Loading pre-trained models and processing images or videos is straightforward.

GPU Utilization: The code explicitly checks and uses GPU if available, significantly speeding up the processing, especially important for deep learning models.

Modular Design: The code is well-organized into functions (detect_objects and process_video), making it reusable and easy to extend or modify.

MobileNet SSD-based Code:

Manual Setup: The code requires manual setup of configurations and reading model files. This can be error-prone and less user-friendly.

Less Modularity: The code lacks modular functions, making it harder to manage and extend.

Limited to CPU Processing: The code does not check for or utilize GPU, which can slow down processing significantly for large images or videos.

3. Preprocessing and Postprocessing

YOLOS-based Code:

Automated Preprocessing: Uses YolosImageProcessor to handle image preprocessing, ensuring consistency and correctness.

Postprocessing with Confidence Thresholds: The results are filtered based on confidence thresholds, and bounding boxes are drawn with clear labels and scores, providing better visualization.

Scalability: The post-processing step uses efficient tensor operations, making it scalable for high-resolution images and videos.

MobileNet SSD-based Code:

Manual Preprocessing: Requires manual configuration of input preprocessing steps, which can introduce errors if not done correctly.

Simple Bounding Box Drawing: The bounding box and label drawing are done manually, which might not handle edge cases or overlapping detections well.

Fixed Confidence Threshold: The code uses a fixed confidence threshold, which might not be optimal for all use cases.

4. Visualization and User Interaction

YOLOS-based Code:

Advanced Visualization: Uses matplotlib for visualizing results, providing better control over the appearance of the output.

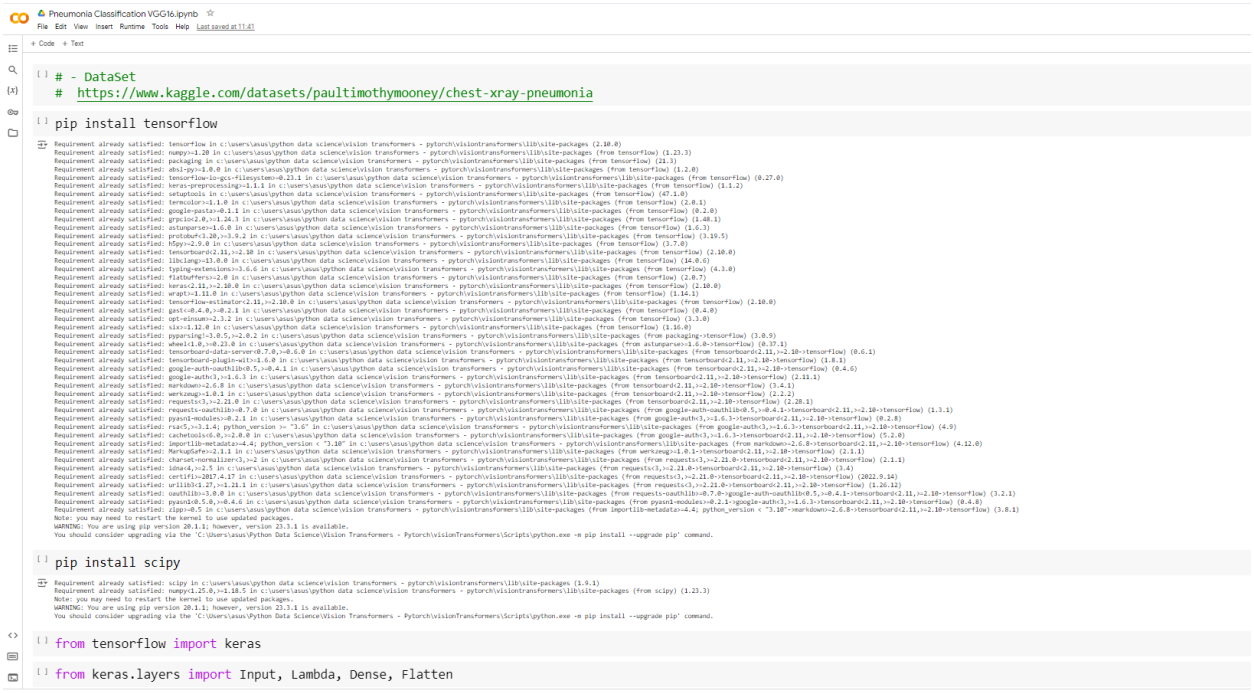
Interactive in Colab: Integrated with Google Colab's file upload and download features, making it easy for users to test with their images and videos.

MobileNet SSD-based Code:

Basic Visualization: Uses OpenCV for visualization, which is functional but less flexible compared to matplotlib.

No Colab Integration: Does not utilize Colab-specific features, making it less convenient for users in that environment.

Practical -Pneumonia Classification VGG16



Pneumonia Classification VGG16.ipynb

File Edit View Insert Runtime Tools Help Last saved at 11:41

+ Code + Text

```
# - DataSet
#  https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia
```

```
pip install tensorflow
```

```
Requirement already satisfied: tensorflow in c:\users\asus\python data science\vision transformers - pytorch\visiontransformers\lib\site-packages (2.10.0)
Requirement already satisfied: numpy>=1.20 in c:\users\asus\python data science\vision transformers - pytorch\visiontransformers\lib\site-packages (from tensorflow) (1.23.3)
Requirement already satisfied: packaging in c:\users\asus\python data science\vision transformers - pytorch\visiontransformers\lib\site-packages (from tensorflow) (21.3)
Requirement already satisfied: absl-py>=1.0.0 in c:\users\asus\python data science\vision transformers - pytorch\visiontransformers\lib\site-packages (from tensorflow) (1.2.0)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in c:\users\asus\python data science\vision transformers - pytorch\visiontransformers\lib\site-packages (from tensorflow) (0.27.0)
Requirement already satisfied: keras-preprocessing>=1.1.1 in c:\users\asus\python data science\vision transformers - pytorch\visiontransformers\lib\site-packages (from tensorflow) (1.1.2)
Requirement already satisfied: setuptools in c:\users\asus\python data science\vision transformers - pytorch\visiontransformers\lib\site-packages (from tensorflow) (67.1.0)
Requirement already satisfied: termcolor>=1.1.0 in c:\users\asus\python data science\vision transformers - pytorch\visiontransformers\lib\site-packages (from tensorflow) (2.0.1)
Requirement already satisfied: google-pasta>=0.1.1 in c:\users\asus\python data science\vision transformers - pytorch\visiontransformers\lib\site-packages (from tensorflow) (0.2.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in c:\users\asus\python data science\vision transformers - pytorch\visiontransformers\lib\site-packages (from tensorflow) (1.48.1)
Requirement already satisfied: astunparse>=1.6.0 in c:\users\asus\python data science\vision transformers - pytorch\visiontransformers\lib\site-packages (from tensorflow) (1.6.3)
Requirement already satisfied: protobuf<3.20,>=3.9.2 in c:\users\asus\python data science\vision transformers - pytorch\visiontransformers\lib\site-packages (from tensorflow) (3.19.5)
Requirement already satisfied: h5py>=2.9.0 in c:\users\asus\python data science\vision transformers - pytorch\visiontransformers\lib\site-packages (from tensorflow) (3.7.0)
Requirement already satisfied: tensorboard<2.11,>=2.10 in c:\users\asus\python data science\vision transformers - pytorch\visiontransformers\lib\site-packages (from tensorflow) (2.10.0)
Requirement already satisfied: libclang>=13.0.0 in c:\users\asus\python data science\vision transformers - pytorch\visiontransformers\lib\site-packages (from tensorflow) (14.0.6)
Requirement already satisfied: typing-extensions>=3.6.6 in c:\users\asus\python data science\vision transformers - pytorch\visiontransformers\lib\site-packages (from tensorflow) (4.3.0)
Requirement already satisfied: flatbuffers>=2.0 in c:\users\asus\python data science\vision transformers - pytorch\visiontransformers\lib\site-packages (from tensorflow) (2.0.7)
Requirement already satisfied: keras<2.11,>=2.10.0 in c:\users\asus\python data science\vision transformers - pytorch\visiontransformers\lib\site-packages (from tensorflow) (2.10.0)
Requirement already satisfied: wrapt>=1.11.0 in c:\users\asus\python data science\vision transformers - pytorch\visiontransformers\lib\site-packages (from tensorflow) (1.14.1)
Requirement already satisfied: tensorflow-estimator<2.11,>=2.10.0 in c:\users\asus\python data science\vision transformers - pytorch\visiontransformers\lib\site-packages (from tensorflow) (2.10.0)
Requirement already satisfied: gast<=0.4.0,>=0.2.1 in c:\users\asus\python data science\vision transformers - pytorch\visiontransformers\lib\site-packages (from tensorflow) (0.4.0)
Requirement already satisfied: opt-einsum>=2.3.2 in c:\users\asus\python data science\vision transformers - pytorch\visiontransformers\lib\site-packages (from tensorflow) (3.3.0)
Requirement already satisfied: six>=1.12.0 in c:\users\asus\python data science\vision transformers - pytorch\visiontransformers\lib\site-packages (from tensorflow) (1.16.0)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in c:\users\asus\python data science\vision transformers - pytorch\visiontransformers\lib\site-packages (from packaging->tensorflow) (3.0.9)
Requirement already satisfied: wheel<1.0,>=0.23.0 in c:\users\asus\python data science\vision transformers - pytorch\visiontransformers\lib\site-packages (from astunparse>=1.6.0->tensorflow) (0.37.1)
Requirement already satisfied: tensorboard-data-server<0.7.0,>=0.6.0 in c:\users\asus\python data science\vision transformers - pytorch\visiontransformers\lib\site-packages (from tensorboard<2.11,>=2.10->tensorflow) (0.6.1)
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in c:\users\asus\python data science\vision transformers - pytorch\visiontransformers\lib\site-packages (from tensorboard<2.11,>=2.10->tensorflow) (1.8.1)
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in c:\users\asus\python data science\vision transformers - pytorch\visiontransformers\lib\site-packages (from tensorboard<2.11,>=2.10->tensorflow) (0.4.6)
Requirement already satisfied: google-auth<3,>=1.6.3 in c:\users\asus\python data science\vision transformers - pytorch\visiontransformers\lib\site-packages (from tensorboard<2.11,>=2.10->tensorflow) (2.11.1)
Requirement already satisfied: markdown>=2.6.8 in c:\users\asus\python data science\vision transformers - pytorch\visiontransformers\lib\site-packages (from tensorboard<2.11,>=2.10->tensorflow) (3.4.1)
Requirement already satisfied: werkzeug>=1.0.1 in c:\users\asus\python data science\vision transformers - pytorch\visiontransformers\lib\site-packages (from tensorboard<2.11,>=2.10->tensorflow) (2.2.2)
Requirement already satisfied: requests<3,>=2.21.0 in c:\users\asus\python data science\vision transformers - pytorch\visiontransformers\lib\site-packages (from tensorboard<2.11,>=2.10->tensorflow) (2.28.1)
Requirement already satisfied: requests-oauthlib>=0.7.0 in c:\users\asus\python data science\vision transformers - pytorch\visiontransformers\lib\site-packages (from google-auth-oauthlib<0.5,>=0.4.1->tensorboard<2.11,>=2.10->tensorflow) (1.3.1)
Requirement already satisfied: pyasn1-modules>=0.2.1 in c:\users\asus\python data science\vision transformers - pytorch\visiontransformers\lib\site-packages (from google-auth<3,>=1.6.3->tensorboard<2.11,>=2.10->tensorflow) (0.2.8)
Requirement already satisfied: rsa<5,>=3.1.4; python_version >= "3.6" in c:\users\asus\python data science\vision transformers - pytorch\visiontransformers\lib\site-packages (from google-auth<3,>=1.6.3->tensorboard<2.11,>=2.10->tensorflow) (4.9)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in c:\users\asus\python data science\vision transformers - pytorch\visiontransformers\lib\site-packages (from google-auth<3,>=1.6.3->tensorboard<2.11,>=2.10->tensorflow) (5.2.0)
Requirement already satisfied: importlib-metadata>=4.4; python_version < "3.10" in c:\users\asus\python data science\vision transformers - pytorch\visiontransformers\lib\site-packages (from markdown>=2.6.8->tensorboard<2.11,>=2.10->tensorflow) (4.12.0)
Requirement already satisfied: MarkupSafe>=2.1.1 in c:\users\asus\python data science\vision transformers - pytorch\visiontransformers\lib\site-packages (from werkzeug>=1.0.1->tensorboard<2.11,>=2.10->tensorflow) (2.1.1)
Requirement already satisfied: charset-normalizer<3,>=2 in c:\users\asus\python data science\vision transformers - pytorch\visiontransformers\lib\site-packages (from requests<3,>=2.21.0->tensorboard<2.11,>=2.10->tensorflow) (2.1.1)
Requirement already satisfied: idna<4,>=2.5 in c:\users\asus\python data science\vision transformers - pytorch\visiontransformers\lib\site-packages (from requests<3,>=2.21.0->tensorboard<2.11,>=2.10->tensorflow) (3.4)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\asus\python data science\vision transformers - pytorch\visiontransformers\lib\site-packages (from requests<3,>=2.21.0->tensorboard<2.11,>=2.10->tensorflow) (2022.9.14)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in c:\users\asus\python data science\vision transformers - pytorch\visiontransformers\lib\site-packages (from requests<3,>=2.21.0->tensorboard<2.11,>=2.10->tensorflow) (1.26.12)
Requirement already satisfied: oauthlib>=3.0.0 in c:\users\asus\python data science\vision transformers - pytorch\visiontransformers\lib\site-packages (from requests-oauthlib>=0.7.0->google-auth-oauthlib<0.5,>=0.4.1->tensorboard<2.11,>=2.10->tensorflow) (3.2.1)
Requirement already satisfied: pyasn1>=0.5.0,>=0.4.6 in c:\users\asus\python data science\vision transformers - pytorch\visiontransformers\lib\site-packages (from pyasn1-modules>=0.2.1->google-auth<3,>=1.6.3->tensorboard<2.11,>=2.10->tensorflow) (0.4.8)
Requirement already satisfied: zipp>=0.5 in c:\users\asus\python data science\vision transformers - pytorch\visiontransformers\lib\site-packages (from importlib-metadata>=4.4; python_version < "3.10"->markdown>=2.6.8->tensorboard<2.11,>=2.10->tensorflow) (3.8.1)
Note: you may need to restart the kernel to use updated packages.
WARNING: You are using pip version 20.1.1; however, version 23.3.1 is available.
You should consider upgrading via the 'C:\Users\asus\Python Data Science\Vision Transformers - Pytorch\visionTransformers\Scripts\python.exe -m pip install --upgrade pip' command.
```

```
pip install scipy
```

```
Requirement already satisfied: scipy in c:\users\asus\python data science\vision transformers - pytorch\visiontransformers\lib\site-packages (1.9.1)
Requirement already satisfied: numpy<1.25.0,>=1.18.5 in c:\users\asus\python data science\vision transformers - pytorch\visiontransformers\lib\site-packages (from scipy) (1.23.3)
Note: you may need to restart the kernel to use updated packages.
WARNING: You are using pip version 20.1.1; however, version 23.3.1 is available.
You should consider upgrading via the 'C:\Users\asus\Python Data Science\Vision Transformers - Pytorch\visionTransformers\Scripts\python.exe -m pip install --upgrade pip' command.
```

```
from tensorflow import keras
```

```
from keras.layers import Input, Lambda, Dense, Flatten
```

```
from keras.models import Model
from keras.applications.vgg16 import VGG16
from keras.applications.vgg16 import preprocess_input
from keras.preprocessing import image
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
import numpy as np
from glob import glob
import matplotlib.pyplot as plt
```

```
vgg = VGG16(input_shape=(224,224,3), weights='imagenet', include_top=False)
# disreguard the final fully connected layer, which does the classification. We do this, in situations, if we need to customize
# the final outputs. Usually, VGG 16 has close to 1000 output classes. But here for pneumonia prediction we need only two classes
# as positive and negative.

# we use this strategy for situations, where we need the embeddings generated from the model to be plugged with another
# customized requirements.
```

```
# prevent weights adjustments of the VGG modle during the training procedure. So it will have the pretrained imagenet weights.
for layer in vgg.layers:
    layer.trainable = False
```

```
folders = glob('Pneumonia/train/*')
x = Flatten()(vgg.output) # flatten the embeddings generated to one single tensor, overl the 1000s of tensors output
```

```
# use the derived flattened tensor and re-use it to create a custom and trainable dense layer for us to get the binary class
# output we need as penumonia positive / negative. Hence it`s binary classed, we can use softmax either.
prediction = Dense(len(folders), activation='softmax')(x)
# create a model object
model = Model(inputs=vgg.input, outputs=prediction)
# view the structure of the model
model.summary()
```

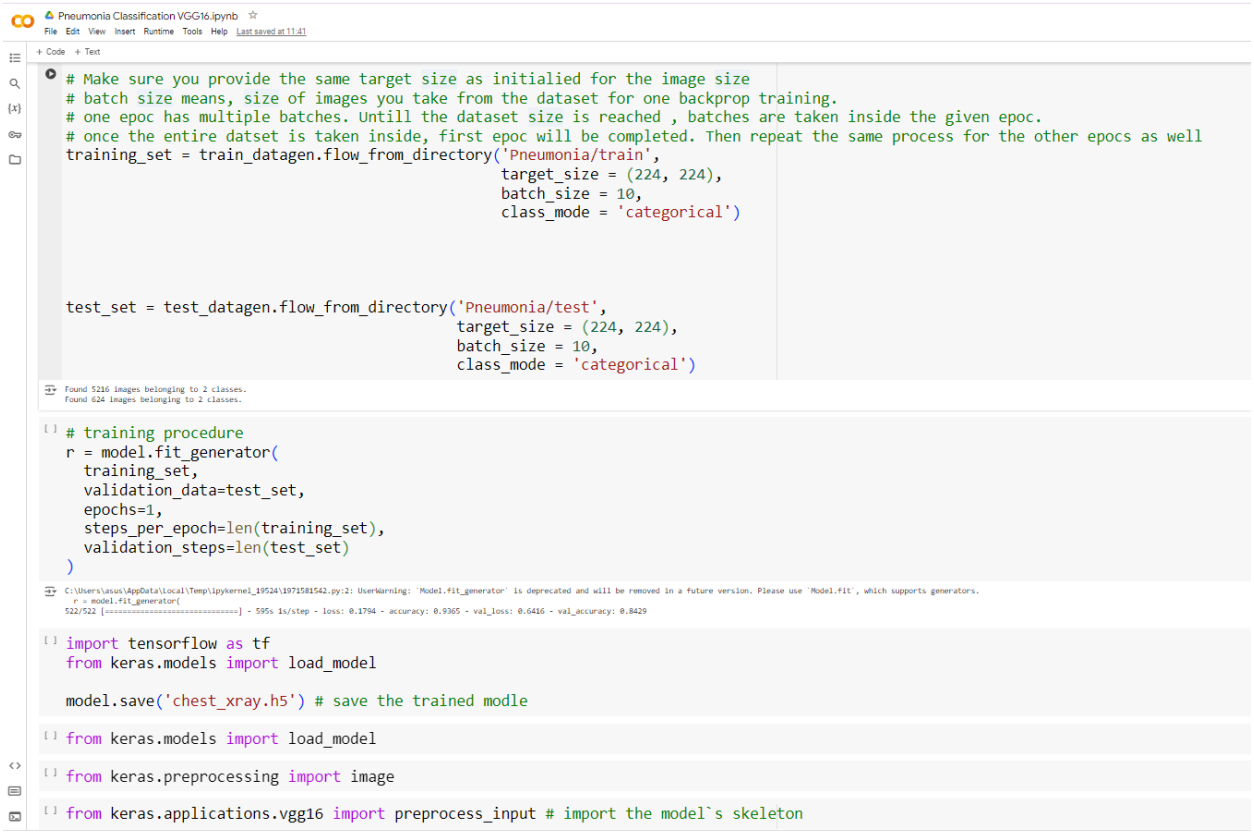


```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 2)                 50178

=================================================================
Total params: 14,764,866
Trainable params: 50,178
Non-trainable params: 14,714,688
_________________________________________________________________
```

```
# configure the back prop pipiline
model.compile(
  loss='categorical_crossentropy',
  optimizer='adam',
  metrics=['accuracy']
)
```

```
# used for the image augmentation process.
from keras.preprocessing.image import ImageDataGenerator
```

```
# augmentation pipeline
train_datagen = ImageDataGenerator(rescale = 1./255,
```



```
# Make sure you provide the same target size as initialied for the image size
# batch size means, size of images you take from the dataset for one backprop training.
# one epoc has multiple batches. Untill the dataset size is reached , batches are taken inside the given epoc.
# once the entire datset is taken inside, first epoc will be completed. Then repeat the same process for the other epocs as well
training_set = train_datagen.flow_from_directory('Pneumonia/train',
                                                 target_size = (224, 224),
                                                 batch_size = 10,
                                                 class_mode = 'categorical')



test_set = test_datagen.flow_from_directory('Pneumonia/test',
                                            target_size = (224, 224),
                                            batch_size = 10,
                                            class_mode = 'categorical')
```

```
Found 5216 images belonging to 2 classes.
Found 624 images belonging to 2 classes.
```

```
# training procedure
r = model.fit_generator(
  training_set,
  validation_data=test_set,
  epochs=1,
  steps_per_epoch=len(training_set),
  validation_steps=len(test_set)
)
```

```
C:\Users\asus\AppData\Local\Temp\ipykernel_19524\1971581542.py:2: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.
  r = model.fit_generator(
522/522 [==============================] - 595s 1s/step - loss: 0.1794 - accuracy: 0.9365 - val_loss: 0.6416 - val_accuracy: 0.8429
```

```
import tensorflow as tf
from keras.models import load_model

model.save('chest_xray.h5') # save the trained modle
```

```
from keras.models import load_model
```

```
from keras.preprocessing import image
```

```
from keras.applications.vgg16 import preprocess_input # import the model`s skeleton
```

```
import numpy as np
```

```
model=load_model('chest_xray.h5')
```

```
#load the test image
#tf.keras.utils.load_img
img=tf.keras.utils.load_img('C:/Users/asus/Python Data Science/Pneumonia/test/PNEUMONIA/person93_bacteria_453.jpeg',target_size=(224,224))

x=tf.keras.preprocessing.image.img_to_array(img) # image as a numpy array

x=np.expand_dims(x, axis=0)

img_data=preprocess_input(x) # organize for the prediction
```

```
classes=model.predict(img_data)
```

```
1/1 [==============================] - 2s 2s/step
```

```
result=int(classes[0][0])
```

```
if result== 0:
    print("Person is Affected By PNEUMONIA")
else:
    print("Result is Normal")
```

```
Person is Affected By PNEUMONIA
```

```
Start coding or generate with AI.
```

Data Preparation and Model Building

Install Necessary Libraries:

This installs TensorFlow and SciPy, which are necessary for building and training the neural network.

Import Libraries:

Here, we import various necessary modules from TensorFlow, Keras, and other libraries for image processing, model creation, and visualization.

Load Pre-trained VGG16 Model:

We load the VGG16 model pre-trained on the ImageNet dataset without the top fully connected layers. This allows us to use the feature extraction part of VGG16 and customize the classification layer.

Freeze Pre-trained Layers:

By setting trainable to False, we ensure that the weights of the pre-trained VGG16 layers are not updated during the training process. This is because the features learned by VGG16 on ImageNet are useful for our task and we don't want to alter them.

Customize the Model:

Flatten() converts the 3D feature maps to 1D feature vectors.

Dense(len(folders), activation='softmax') creates a dense layer with a number of units equal to the number of classes (2 classes: pneumonia and normal) and uses the softmax activation function for classification.

Model(inputs=vgg.input, outputs=prediction) constructs the final model by specifying the inputs and outputs.

Compile the Model:

The model is compiled using the Adam optimizer and categorical crossentropy loss function. Accuracy is used as the evaluation metric.

Data Augmentation and Training

Data Augmentation:

ImageDataGenerator is used for real-time data augmentation:

rescale=1./255 normalizes the pixel values.

shear_range, zoom_range, and horizontal_flip are used for random transformations to make the model robust.

Load Training and Test Data:

This loads the training and test data from the respective directories, with images resized to 224x224 and a batch size of 10.

Train the Model:

The model is trained using the augmented data. Here:

epochs=1 specifies one training epoch.

steps_per_epoch and validation_steps are set to the number of batches in the training and validation datasets.

Model Saving and Prediction

Save the Trained Model:

The trained model is saved to a file named chest_xray.h5.

Load and Predict:

The model is loaded from the saved file.

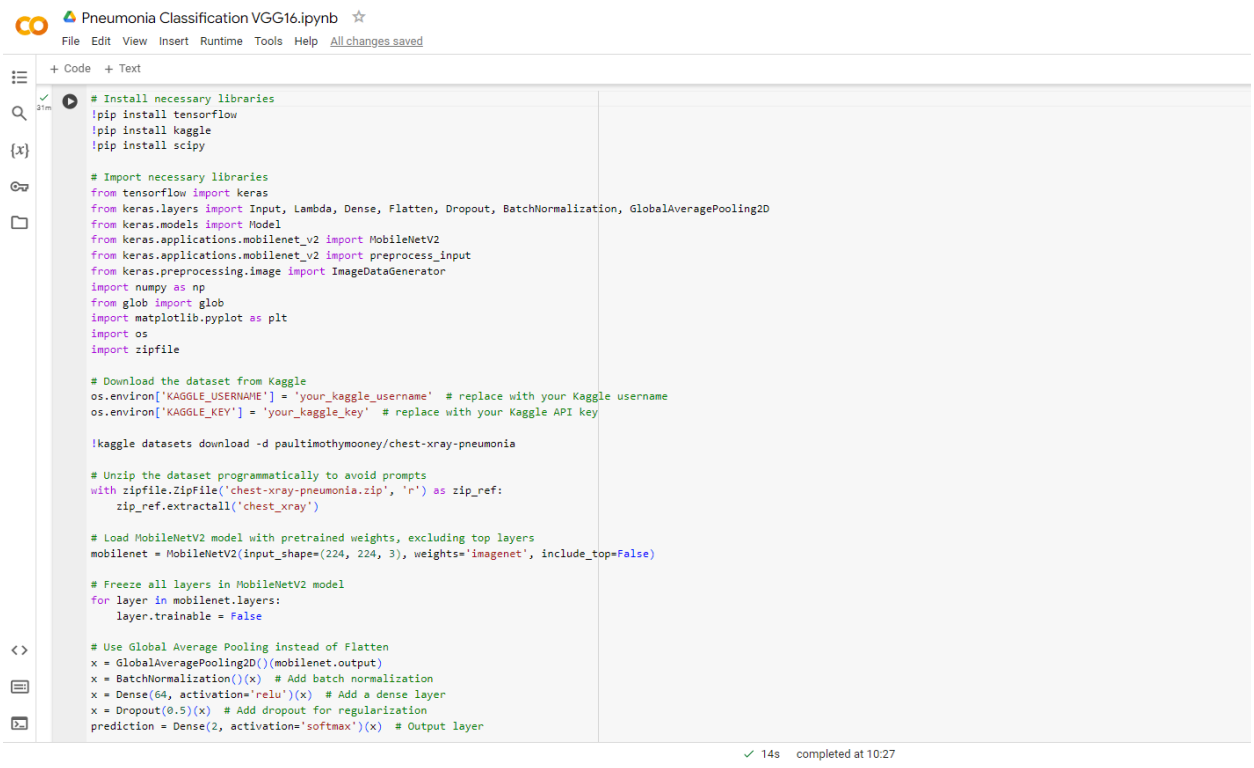
An image is loaded and resized to 224x224.

The image is converted to an array and expanded to match the input shape expected by the model.

The image is preprocessed and passed to the model for prediction.

Based on the prediction, it prints whether the person is affected by pneumonia or not.

Improved Model



Pneumonia Classification VGG16.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
# Create a model object
model = Model(inputs=mobilenet.input, outputs=prediction)

# View the structure of the model
model.summary()

# Compile the model
model.compile(
    loss='categorical_crossentropy',
    optimizer='adam',
    metrics=['accuracy']
)

# Image augmentation
train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)

test_datagen = ImageDataGenerator(rescale=1./255)

# Prepare training and validation datasets
training_set = train_datagen.flow_from_directory(
    'chest_xray/chest_xray/train',
    target_size=(224, 224),
    batch_size=32,  # Adjust batch size for your hardware
    class_mode='categorical'
)

test_set = test_datagen.flow_from_directory(
    'chest_xray/chest_xray/test',
    target_size=(224, 224),
    batch_size=32,  # Adjust batch size for your hardware
    class_mode='categorical'
)
```

Pneumonia Classification VGG16.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
# Train the model
r = model.fit(
    training_set,
    validation_data=test_set,
    epochs=5,  # Adjust number of epochs
    steps_per_epoch=len(training_set),
    validation_steps=len(test_set)
)

# Save the trained model
model.save('chest_xray_mobilenet_optimized.h5')

# Evaluate the model on validation data
loss, accuracy = model.evaluate(test_set)
print(f"Validation Accuracy: {accuracy * 100:.2f}%")

# Predict function for a new image
def predict_image(image_path, model):
    img = keras.utils.load_img(image_path, target_size=(224, 224))
    x = keras.preprocessing.image.img_to_array(img)
    x = np.expand_dims(x, axis=0)
    x = preprocess_input(x)
    prediction = model.predict(x)
    return prediction

# Load the trained model
model = keras.models.load_model('chest_xray_mobilenet_optimized.h5')

# Test prediction on a sample image
image_path = 'chest_xray/chest_xray/test/PNEUMONIA/person93_bacteria_453.jpeg'
prediction = predict_image(image_path, model)
result = np.argmax(prediction, axis=1)[0]

if result == 0:
    print("Person is Affected By PNEUMONIA")
else:
    print("Result is Normal")
```

Pneumonia Classification VGG16.ipynb ☆

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-packages (2.15.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=23.5.26 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (24.3.25)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.6.0)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.9.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (18.1.1)
Requirement already satisfied: ml-dtypes~=0.2.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: numpy<2.0.0,>=1.23.5 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.25.2)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.3.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from tensorflow) (24.1)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.20.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from tensorflow) (67.7.2)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.4.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (4.12.2)
Requirement already satisfied: wrapt<1.15,>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.14.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.37.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.64.1)
Requirement already satisfied: tensorboard<2.16,>=2.15 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.15.2)
Requirement already satisfied: tensorflow-estimator<2.16,>=2.15.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.15.0)
Requirement already satisfied: keras<2.16,>=2.15.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.15.0)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from astunparse>=1.6.0->tensorflow) (0.43.0)
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tensorflow) (2.27.0)
Requirement already satisfied: google-auth-oauthlib<2,>=0.5 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tensorflow) (1.2.0)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tensorflow) (3.6)
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tensorflow) (2.31.0)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tensorflow) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tensorflow) (3.0.3)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.16,>=2.15->tensorflow) (5.3.3)
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.16,>=2.15->tensorflow) (0.4.0)
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.16,>=2.15->tensorflow) (4.9)
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from google-auth-oauthlib<2,>=0.5->tensorboard<2.16,>=2.15->tensorflow) (1.3.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.16,>=2.15->tensorflow) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.16,>=2.15->tensorflow) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.16,>=2.15->tensorflow) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.16,>=2.15->tensorflow) (2024.6.2)
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packages (from werkzeug>=1.0.1->tensorboard<2.16,>=2.15->tensorflow) (2.1.5)
Requirement already satisfied: pyasn1<0.7.0,>=0.4.6 in /usr/local/lib/python3.10/dist-packages (from pyasn1-modules>=0.2.1->google-auth<3,>=1.6.3->tensorboard<2.16,>=2.15->tensorflow) (0.6.0)
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.10/dist-packages (from requests-oauthlib>=0.7.0->google-auth-oauthlib<2,>=0.5->tensorboard<2.16,>=2.15->tensorflow) (3.2.2)
Requirement already satisfied: kaggle in /usr/local/lib/python3.10/dist-packages (1.6.14)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.10/dist-packages (from kaggle) (1.16.0)
Requirement already satisfied: certifi>=2023.7.22 in /usr/local/lib/python3.10/dist-packages (from kaggle) (2024.6.2)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.8.2)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.31.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from kaggle) (4.66.4)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.10/dist-packages (from kaggle) (8.0.4)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.0.7)
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages (from kaggle) (6.1.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages (from bleach->kaggle) (0.5.1)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.10/dist-packages (from python-slugify->kaggle) (1.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->kaggle) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->kaggle) (3.7)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (1.11.4)
Requirement already satisfied: numpy<1.28.0,>=1.21.6 in /usr/local/lib/python3.10/dist-packages (from scipy) (1.25.2)
Dataset URL: https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia
License(s): other
```

Pneumonia Classification VGG16.ipynb ☆

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
chest-xray-pneumonia.zip: Skipping, found more recently modified local copy (use --force to force download)
Model: "model_3"
```

Layer (type)	Output Shape	Param #	Connected to
input_5 (InputLayer)	[(None, 224, 224, 3)]	0	[]
Conv1 (Conv2D)	(None, 112, 112, 32)	864	['input_5[0][0]']
bn_Conv1 (BatchNormalization)	(None, 112, 112, 32)	128	['Conv1[0][0]']
Conv1_relu (ReLU)	(None, 112, 112, 32)	0	['bn_Conv1[0][0]']
expanded_conv_depthwise (DepthwiseConv2D)	(None, 112, 112, 32)	288	['Conv1_relu[0][0]']
expanded_conv_depthwise_BN (BatchNormalization)	(None, 112, 112, 32)	128	['expanded_conv_depthwise[0][0]']
expanded_conv_depthwise_relu (ReLU)	(None, 112, 112, 32)	0	['expanded_conv_depthwise_BN[0][0]']
expanded_conv_project (Conv2D)	(None, 112, 112, 16)	512	['expanded_conv_depthwise_relu[0][0]']
expanded_conv_project_BN (BatchNormalization)	(None, 112, 112, 16)	64	['expanded_conv_project[0][0]']
block_1_expand (Conv2D)	(None, 112, 112, 96)	1536	['expanded_conv_project_BN[0][0]']
block_1_expand_BN (BatchNormalization)	(None, 112, 112, 96)	384	['block_1_expand[0][0]']
block_1_expand_relu (ReLU)	(None, 112, 112, 96)	0	['block_1_expand_BN[0][0]']
block_1_pad (ZeroPadding2D)	(None, 113, 113, 96)	0	['block_1_expand_relu[0][0]']
block_1_depthwise (DepthwiseConv2D)	(None, 56, 56, 96)	864	['block_1_pad[0][0]']
block_1_depthwise_BN (BatchNormalization)	(None, 56, 56, 96)	384	['block_1_depthwise[0][0]']
block_1_depthwise_relu (ReLU)	(None, 56, 56, 96)	0	['block_1_depthwise_BN[0][0]']
block_1_project (Conv2D)	(None, 56, 56, 24)	2304	['block_1_depthwise_relu[0][0]']
block_1_project_BN (BatchNormalization)	(None, 56, 56, 24)	96	['block_1_project[0][0]']
block_2_expand (Conv2D)	(None, 56, 56, 144)	3456	['block_1_project_BN[0][0]']

block_2_expand_BN (BatchNormalization)	(None, 56, 56, 144)	576	['block_2_expand[0][0]']
block_2_expand_relu (ReLU)	(None, 56, 56, 144)	0	['block_2_expand_BN[0][0]']
block_2_depthwise (DepthwiseConv2D)	(None, 56, 56, 144)	1296	['block_2_expand_relu[0][0]']
block_2_depthwise_BN (BatchNormalization)	(None, 56, 56, 144)	576	['block_2_depthwise[0][0]']
block_2_depthwise_relu (ReLU)	(None, 56, 56, 144)	0	['block_2_depthwise_BN[0][0]']
block_2_project (Conv2D)	(None, 56, 56, 24)	3456	['block_2_depthwise_relu[0][0]']
block_2_project_BN (BatchNormalization)	(None, 56, 56, 24)	96	['block_2_project[0][0]']
block_2_add (Add)	(None, 56, 56, 24)	0	['block_1_project_BN[0][0]', 'block_2_project_BN[0][0]']
block_3_expand (Conv2D)	(None, 56, 56, 144)	3456	['block_2_add[0][0]']
block_3_expand_BN (BatchNormalization)	(None, 56, 56, 144)	576	['block_3_expand[0][0]']
block_3_expand_relu (ReLU)	(None, 56, 56, 144)	0	['block_3_expand_BN[0][0]']
block_3_pad (ZeroPadding2D)	(None, 57, 57, 144)	0	['block_3_expand_relu[0][0]']
block_3_depthwise (DepthwiseConv2D)	(None, 28, 28, 144)	1296	['block_3_pad[0][0]']
block_3_depthwise_BN (BatchNormalization)	(None, 28, 28, 144)	576	['block_3_depthwise[0][0]']
block_3_depthwise_relu (ReLU)	(None, 28, 28, 144)	0	['block_3_depthwise_BN[0][0]']
block_3_project (Conv2D)	(None, 28, 28, 32)	4608	['block_3_depthwise_relu[0][0]']
block_3_project_BN (BatchNormalization)	(None, 28, 28, 32)	128	['block_3_project[0][0]']
block_4_expand (Conv2D)	(None, 28, 28, 192)	6144	['block_3_project_BN[0][0]']
block_4_expand_BN (BatchNormalization)	(None, 28, 28, 192)	768	['block_4_expand[0][0]']
block_4_expand_relu (ReLU)	(None, 28, 28, 192)	0	['block_4_expand_BN[0][0]']
block_4_depthwise (DepthwiseConv2D)	(None, 28, 28, 192)	1728	['block_4_expand_relu[0][0]']

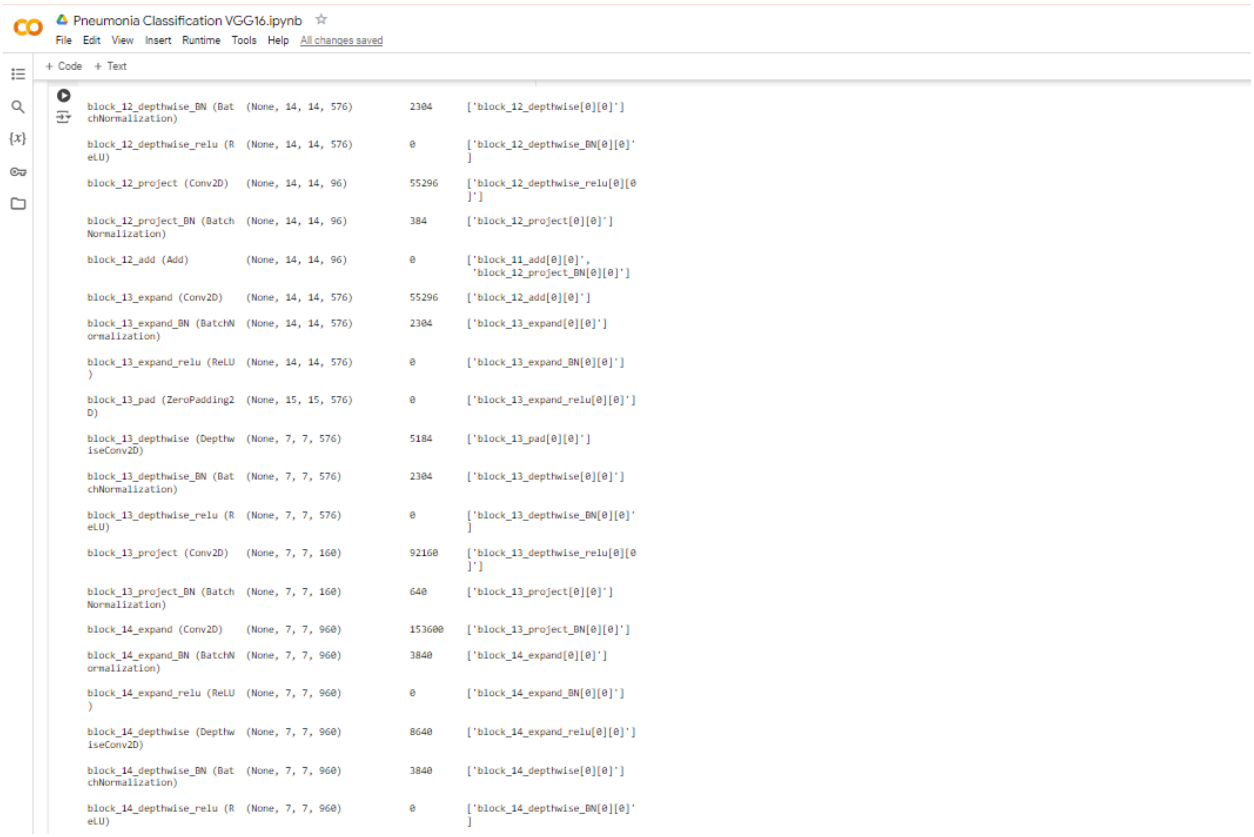
Pneumonia Classification VGG16.ipynb ☆
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text

block_4_depthwise_BN (BatchNormalization)	(None, 28, 28, 192)	768	['block_4_depthwise[0][0]']
block_4_depthwise_relu (ReLU)	(None, 28, 28, 192)	0	['block_4_depthwise_BN[0][0]']
block_4_project (Conv2D)	(None, 28, 28, 32)	6144	['block_4_depthwise_relu[0][0]']
block_4_project_BN (BatchNormalization)	(None, 28, 28, 32)	128	['block_4_project[0][0]']
block_4_add (Add)	(None, 28, 28, 32)	0	['block_3_project_BN[0][0]', 'block_4_project_BN[0][0]']
block_5_expand (Conv2D)	(None, 28, 28, 192)	6144	['block_4_add[0][0]']
block_5_expand_BN (BatchNormalization)	(None, 28, 28, 192)	768	['block_5_expand[0][0]']
block_5_expand_relu (ReLU)	(None, 28, 28, 192)	0	['block_5_expand_BN[0][0]']
block_5_depthwise (DepthwiseConv2D)	(None, 28, 28, 192)	1728	['block_5_expand_relu[0][0]']
block_5_depthwise_BN (BatchNormalization)	(None, 28, 28, 192)	768	['block_5_depthwise[0][0]']
block_5_depthwise_relu (ReLU)	(None, 28, 28, 192)	0	['block_5_depthwise_BN[0][0]']
block_5_project (Conv2D)	(None, 28, 28, 32)	6144	['block_5_depthwise_relu[0][0]']
block_5_project_BN (BatchNormalization)	(None, 28, 28, 32)	128	['block_5_project[0][0]']
block_5_add (Add)	(None, 28, 28, 32)	0	['block_4_add[0][0]', 'block_5_project_BN[0][0]']
block_6_expand (Conv2D)	(None, 28, 28, 192)	6144	['block_5_add[0][0]']
block_6_expand_BN (BatchNormalization)	(None, 28, 28, 192)	768	['block_6_expand[0][0]']
block_6_expand_relu (ReLU)	(None, 28, 28, 192)	0	['block_6_expand_BN[0][0]']
block_6_pad (ZeroPadding2D)	(None, 29, 29, 192)	0	['block_6_expand_relu[0][0]']
block_6_depthwise (DepthwiseConv2D)	(None, 14, 14, 192)	1728	['block_6_pad[0][0]']
block_6_depthwise_BN (BatchNormalization)	(None, 14, 14, 192)	768	['block_6_depthwise[0][0]']
block_6_depthwise_relu (ReLU)	(None, 14, 14, 192)	0	['block_6_depthwise_BN[0][0]']

block_6_project (Conv2D)	(None, 14, 14, 64)	12288	['block_6_depthwise_relu[0][0]']
block_6_project_BN (BatchNormalization)	(None, 14, 14, 64)	256	['block_6_project[0][0]']
block_7_expand (Conv2D)	(None, 14, 14, 384)	24576	['block_6_project_BN[0][0]']
block_7_expand_BN (BatchNormalization)	(None, 14, 14, 384)	1536	['block_7_expand[0][0]']
block_7_expand_relu (ReLU)	(None, 14, 14, 384)	0	['block_7_expand_BN[0][0]']
block_7_depthwise (DepthwiseConv2D)	(None, 14, 14, 384)	3456	['block_7_expand_relu[0][0]']
block_7_depthwise_BN (BatchNormalization)	(None, 14, 14, 384)	1536	['block_7_depthwise[0][0]']
block_7_depthwise_relu (ReLU)	(None, 14, 14, 384)	0	['block_7_depthwise_BN[0][0]']
block_7_project (Conv2D)	(None, 14, 14, 64)	24576	['block_7_depthwise_relu[0][0]']
block_7_project_BN (BatchNormalization)	(None, 14, 14, 64)	256	['block_7_project[0][0]']
block_7_add (Add)	(None, 14, 14, 64)	0	['block_6_project_BN[0][0]', 'block_7_project_BN[0][0]']
block_8_expand (Conv2D)	(None, 14, 14, 384)	24576	['block_7_add[0][0]']
block_8_expand_BN (BatchNormalization)	(None, 14, 14, 384)	1536	['block_8_expand[0][0]']
block_8_expand_relu (ReLU)	(None, 14, 14, 384)	0	['block_8_expand_BN[0][0]']
block_8_depthwise (DepthwiseConv2D)	(None, 14, 14, 384)	3456	['block_8_expand_relu[0][0]']
block_8_depthwise_BN (BatchNormalization)	(None, 14, 14, 384)	1536	['block_8_depthwise[0][0]']
block_8_depthwise_relu (ReLU)	(None, 14, 14, 384)	0	['block_8_depthwise_BN[0][0]']
block_8_project (Conv2D)	(None, 14, 14, 64)	24576	['block_8_depthwise_relu[0][0]']
block_8_project_BN (BatchNormalization)	(None, 14, 14, 64)	256	['block_8_project[0][0]']
block_8_add (Add)	(None, 14, 14, 64)	0	['block_7_add[0][0]', 'block_8_project_BN[0][0]']
block_9_expand (Conv2D)	(None, 14, 14, 384)	24576	['block_8_add[0][0]']

Pneumonia Classification VGG16.ipynb ☆
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text

block_9_expand_BN (BatchNormalization)	(None, 14, 14, 384)	1536	['block_9_expand[0][0]']
block_9_expand_relu (ReLU)	(None, 14, 14, 384)	0	['block_9_expand_BN[0][0]']
block_9_depthwise (DepthwiseConv2D)	(None, 14, 14, 384)	3456	['block_9_expand_relu[0][0]']
block_9_depthwise_BN (BatchNormalization)	(None, 14, 14, 384)	1536	['block_9_depthwise[0][0]']
block_9_depthwise_relu (ReLU)	(None, 14, 14, 384)	0	['block_9_depthwise_BN[0][0]']
block_9_project (Conv2D)	(None, 14, 14, 64)	24576	['block_9_depthwise_relu[0][0]']
block_9_project_BN (BatchNormalization)	(None, 14, 14, 64)	256	['block_9_project[0][0]']
block_9_add (Add)	(None, 14, 14, 64)	0	['block_8_add[0][0]', 'block_9_project_BN[0][0]']
block_10_expand (Conv2D)	(None, 14, 14, 384)	24576	['block_9_add[0][0]']
block_10_expand_BN (BatchNormalization)	(None, 14, 14, 384)	1536	['block_10_expand[0][0]']
block_10_expand_relu (ReLU)	(None, 14, 14, 384)	0	['block_10_expand_BN[0][0]']
block_10_depthwise (DepthwiseConv2D)	(None, 14, 14, 384)	3456	['block_10_expand_relu[0][0]']
block_10_depthwise_BN (BatchNormalization)	(None, 14, 14, 384)	1536	['block_10_depthwise[0][0]']
block_10_depthwise_relu (ReLU)	(None, 14, 14, 384)	0	['block_10_depthwise_BN[0][0]']
block_10_project (Conv2D)	(None, 14, 14, 96)	36864	['block_10_depthwise_relu[0][0]']
block_10_project_BN (BatchNormalization)	(None, 14, 14, 96)	384	['block_10_project[0][0]']
block_11_expand (Conv2D)	(None, 14, 14, 576)	55296	['block_10_project_BN[0][0]']
block_11_expand_BN (BatchNormalization)	(None, 14, 14, 576)	2304	['block_11_expand[0][0]']
block_11_expand_relu (ReLU)	(None, 14, 14, 576)	0	['block_11_expand_BN[0][0]']
block_11_depthwise (DepthwiseConv2D)	(None, 14, 14, 576)	5184	['block_11_expand_relu[0][0]']



block_12_depthwise_BN (BatchNormalization)	(None, 14, 14, 576)	2304	['block_12_depthwise[0][0]']
block_12_depthwise_relu (ReLU)	(None, 14, 14, 576)	0	['block_12_depthwise_BN[0][0]']
block_12_project (Conv2D)	(None, 14, 14, 96)	55296	['block_12_depthwise_relu[0][0]']
block_12_project_BN (BatchNormalization)	(None, 14, 14, 96)	384	['block_12_project[0][0]']
block_12_add (Add)	(None, 14, 14, 96)	0	['block_11_add[0][0]', 'block_12_project_BN[0][0]']
block_13_expand (Conv2D)	(None, 14, 14, 576)	55296	['block_12_add[0][0]']
block_13_expand_BN (BatchNormalization)	(None, 14, 14, 576)	2304	['block_13_expand[0][0]']
block_13_expand_relu (ReLU)	(None, 14, 14, 576)	0	['block_13_expand_BN[0][0]']
block_13_pad (ZeroPadding2D)	(None, 15, 15, 576)	0	['block_13_expand_relu[0][0]']
block_13_depthwise (DepthwiseConv2D)	(None, 7, 7, 576)	5184	['block_13_pad[0][0]']
block_13_depthwise_BN (BatchNormalization)	(None, 7, 7, 576)	2304	['block_13_depthwise[0][0]']
block_13_depthwise_relu (ReLU)	(None, 7, 7, 576)	0	['block_13_depthwise_BN[0][0]']
block_13_project (Conv2D)	(None, 7, 7, 160)	92160	['block_13_depthwise_relu[0][0]']
block_13_project_BN (BatchNormalization)	(None, 7, 7, 160)	640	['block_13_project[0][0]']
block_14_expand (Conv2D)	(None, 7, 7, 960)	153600	['block_13_project_BN[0][0]']
block_14_expand_BN (BatchNormalization)	(None, 7, 7, 960)	3840	['block_14_expand[0][0]']
block_14_expand_relu (ReLU)	(None, 7, 7, 960)	0	['block_14_expand_BN[0][0]']
block_14_depthwise (DepthwiseConv2D)	(None, 7, 7, 960)	8640	['block_14_expand_relu[0][0]']
block_14_depthwise_BN (BatchNormalization)	(None, 7, 7, 960)	3840	['block_14_depthwise[0][0]']
block_14_depthwise_relu (ReLU)	(None, 7, 7, 960)	0	['block_14_depthwise_BN[0][0]']

Pneumonia Classification VGG16.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
block_14_project (Conv2D)      (None, 7, 7, 160)     153600    ['block_14_depthwise_relu[0][0
                                                               ]']

block_14_project_BN (Batch     (None, 7, 7, 160)     640       ['block_14_project[0][0]']
Normalization)

block_14_add (Add)             (None, 7, 7, 160)     0         ['block_13_project_BN[0][0]',
                                                                'block_14_project_BN[0][0]']

block_15_expand (Conv2D)       (None, 7, 7, 960)     153600    ['block_14_add[0][0]']

block_15_expand_BN (BatchN     (None, 7, 7, 960)     3840      ['block_15_expand[0][0]']
ormalization)

block_15_expand_relu (ReLU     (None, 7, 7, 960)     0         ['block_15_expand_BN[0][0]']
)

block_15_depthwise (Depthw     (None, 7, 7, 960)     8640      ['block_15_expand_relu[0][0]']
iseConv2D)

block_15_depthwise_BN (Bat     (None, 7, 7, 960)     3840      ['block_15_depthwise[0][0]']
chNormalization)

block_15_depthwise_relu (R     (None, 7, 7, 960)     0         ['block_15_depthwise_BN[0][0]'
eLU)                                                           ]

block_15_project (Conv2D)      (None, 7, 7, 160)     153600    ['block_15_depthwise_relu[0][0
                                                               ]']

block_15_project_BN (Batch     (None, 7, 7, 160)     640       ['block_15_project[0][0]']
Normalization)

block_15_add (Add)             (None, 7, 7, 160)     0         ['block_14_add[0][0]',
                                                                'block_15_project_BN[0][0]']

block_16_expand (Conv2D)       (None, 7, 7, 960)     153600    ['block_15_add[0][0]']

block_16_expand_BN (BatchN     (None, 7, 7, 960)     3840      ['block_16_expand[0][0]']
ormalization)

block_16_expand_relu (ReLU     (None, 7, 7, 960)     0         ['block_16_expand_BN[0][0]']
)

block_16_depthwise (Depthw     (None, 7, 7, 960)     8640      ['block_16_expand_relu[0][0]']
iseConv2D)

block_16_depthwise_BN (Bat     (None, 7, 7, 960)     3840      ['block_16_depthwise[0][0]']
chNormalization)

block_16_depthwise_relu (R     (None, 7, 7, 960)     0         ['block_16_depthwise_BN[0][0]'
eLU)                                                           ]

block_16_project (Conv2D)      (None, 7, 7, 320)     307200    ['block_16_depthwise_relu[0][0
                                                               ]']

block_16_project_BN (Batch     (None, 7, 7, 320)     1280      ['block_16_project[0][0]']
Normalization)
```

Pneumonia Classification VGG16.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
Conv_1 (Conv2D)              (None, 7, 7, 1280)        409600    ['block_16_project_BN[0][0]']

Conv_1_bn (BatchNormalizat   (None, 7, 7, 1280)        5120      ['Conv_1[0][0]']
ion)

out_relu (ReLU)              (None, 7, 7, 1280)        0         ['Conv_1_bn[0][0]']

global_average_pooling2d_2   (None, 1280)              0         ['out_relu[0][0]']
 (GlobalAveragePooling2D)

batch_normalization_3 (Bat   (None, 1280)              5120      ['global_average_pooling2d_2[0
chNormalization)                                                 ][0]']

dense_6 (Dense)              (None, 64)                81984     ['batch_normalization_3[0][0]'
                                                                 ]

dropout_3 (Dropout)          (None, 64)                0         ['dense_6[0][0]']

dense_7 (Dense)              (None, 2)                 130       ['dropout_3[0][0]']

==================================================================================================
Total params: 2345218 (8.95 MB)
Trainable params: 84674 (330.76 KB)
Non-trainable params: 2260544 (8.62 MB)
__________________________________________________________________________________________________
Found 5216 images belonging to 2 classes.
Found 624 images belonging to 2 classes.
Epoch 1/5
163/163 [==============================] - 353s 2s/step - loss: 0.2477 - accuracy: 0.9051 - val_loss: 0.3305 - val_accuracy: 0.8638
Epoch 2/5
163/163 [==============================] - 340s 2s/step - loss: 0.1491 - accuracy: 0.9440 - val_loss: 0.2617 - val_accuracy: 0.8894
Epoch 3/5
163/163 [==============================] - 327s 2s/step - loss: 0.1265 - accuracy: 0.9532 - val_loss: 0.2311 - val_accuracy: 0.9054
Epoch 4/5
163/163 [==============================] - 337s 2s/step - loss: 0.1114 - accuracy: 0.9571 - val_loss: 0.3132 - val_accuracy: 0.8894
Epoch 5/5
163/163 [==============================] - 344s 2s/step - loss: 0.1163 - accuracy: 0.9567 - val_loss: 0.3100 - val_accuracy: 0.8926
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
20/20 [==============================] - 30s 1s/step - loss: 0.3100 - accuracy: 0.8926
Validation Accuracy: 89.26%
1/1 [==============================] - 1s 1s/step
Result is Normal
```

```
[ ] # Save the trained model in .h5 format
    model.save('chest_xray_mobilenet_optimized.h5')
```

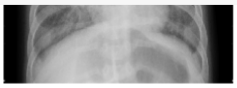
```
[ ] from google.colab import files

    # Download the model file
    files.download('chest_xray_mobilenet_optimized.h5')
```

Pneumonia Classification VGG16.ipynb ☆

File Edit View Insert Runtime Tools Help All changes saved

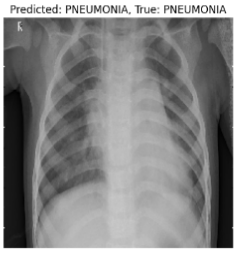
+ Code + Text

```
# Load the trained model
model = keras.models.load_model('chest_xray_mobilenet_optimized.h5')

# Function to display images with predictions
def display_predictions(test_set, model, num_images=5):
    # Get the class labels
    class_labels = list(test_set.class_indices.keys())

    # Get a batch of images and labels
    images, labels = next(test_set)

    # Loop through the first num_images images
    for i in range(num_images):
        img = images[i]
        true_label = labels[i]

        # Get the prediction
        prediction = model.predict(np.expand_dims(img, axis=0))
        predicted_label = np.argmax(prediction, axis=1)[0]

        # Display the image
        plt.imshow(img)
        plt.title(f"Predicted: {class_labels[predicted_label]}, True: {class_labels[np.argmax(true_label)]}")
        plt.axis('off')
        plt.show()

# Display predictions for a few validation images
display_predictions(test_set, model, num_images=5)
```



```
1/1 [==============================] - 0s 130ms/step
```

Predicted: PNEUMONIA, True: PNEUMONIA



```
1/1 [==============================] - 0s 53ms/step
```

Predicted: NORMAL, True: NORMAL

Pneumonia Classification VGG16.ipynb ☆

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

Predicted: PNEUMONIA, True: PNEUMONIA

```
1/1 [==============================] - 0s 130ms/step
```

Predicted: PNEUMONIA, True: PNEUMONIA

```
1/1 [==============================] - 0s 53ms/step
```

Predicted: NORMAL, True: NORMAL

Pneumonia Classification VGG16.ipynb ☆

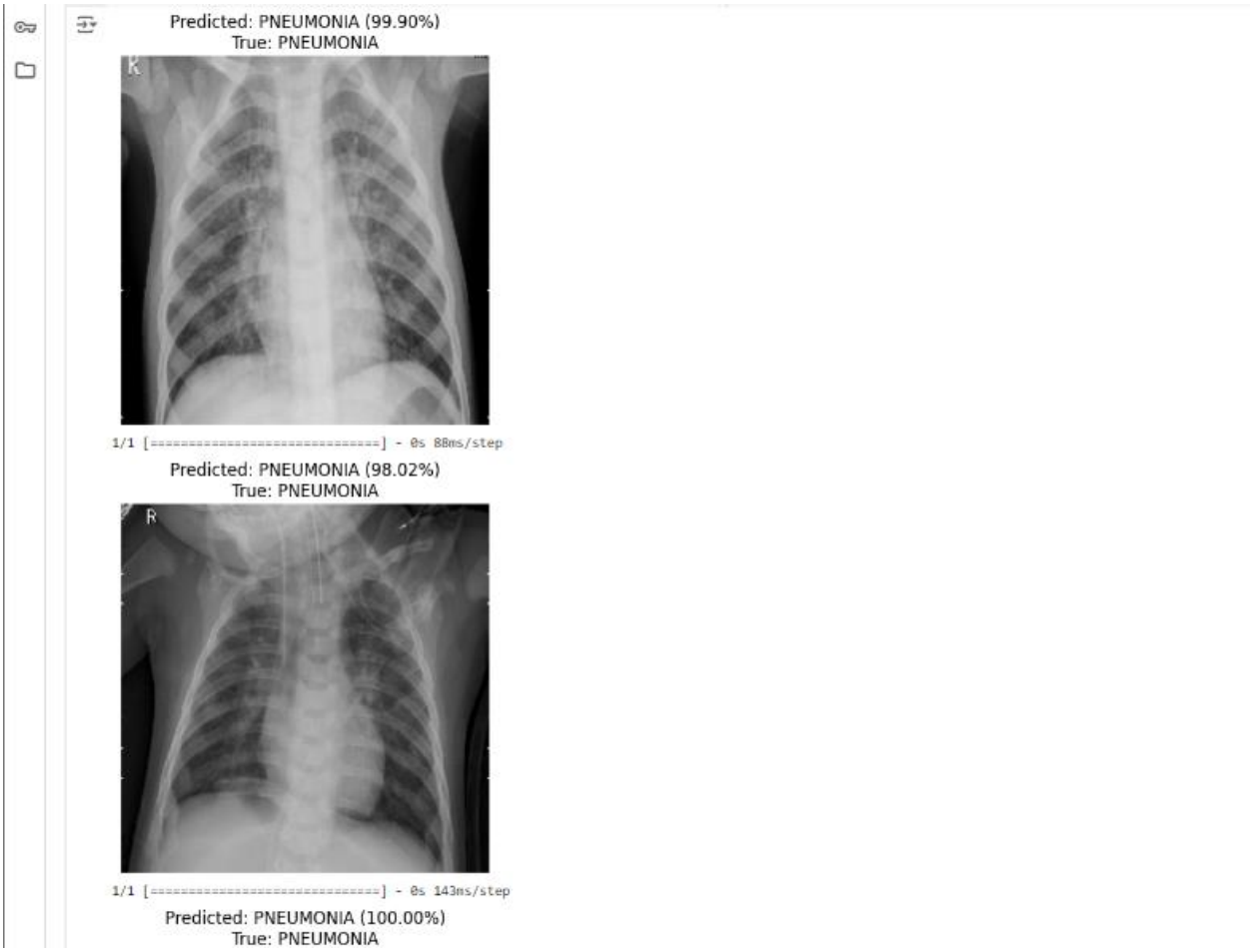
File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
[9] # Load the trained model
    model = keras.models.load_model('chest_xray_mobilenet_optimized.h5')

    # Function to display images with predictions and confidence levels
    def display_predictions_with_confidence(test_set, model, num_images=5):
        # Get the class labels
        class_labels = list(test_set.class_indices.keys())

        # Get a batch of images and labels
        images, labels = next(test_set)

        # Loop through the first num_images images
        for i in range(num_images):
            img = images[1]
            true_label = labels[1]

            # Get the prediction
            prediction = model.predict(np.expand_dims(img, axis=0))
            predicted_label = np.argmax(prediction, axis=1)[0]
            confidence = prediction[0][predicted_label]

            # Display the image
            plt.imshow(img)
            plt.title(f"Predicted: {class_labels[predicted_label]} ({confidence * 100:.2f}%)\nTrue: {class_labels[np.argmax(true_label)]}")
            plt.axis('off')
            plt.show()

    # Display predictions with confidence levels for a few validation images
    display_predictions_with_confidence(test_set, model, num_images=5)
```

```
1/1 [==============================] - 2s 2s/step
```


```
1/1 [==============================] - 0s 94ms/step
```



Justification for the Superiority of the MobileNetV2 Code:

1. Model Efficiency:

MobileNetV2: Known for being lightweight and faster to train and infer due to its efficient architecture designed for mobile and edge devices.

VGG16: Heavier and computationally more expensive due to its deeper architecture.

2. Advanced Features:

Global Average Pooling: Used in MobileNetV2 to reduce the number of parameters and prevent overfitting.

Batch Normalization and Dropout: Added to the MobileNetV2 model to enhance training stability and regularization, respectively.

3. Data Handling:

Automated Data Download and Extraction: MobileNetV2 code automatically downloads and unzips the dataset from Kaggle, making the code more streamlined and user-friendly.

4. Training Configuration:

Batch Size and Epochs: MobileNetV2 code uses a larger batch size (32) which can lead to faster training on suitable hardware. It also sets the number of epochs to 5, providing more training iterations for better model convergence.

Steps per Epoch: The steps per epoch and validation steps are dynamically set based on the dataset size, optimizing the training process.

5. Prediction and Visualization:

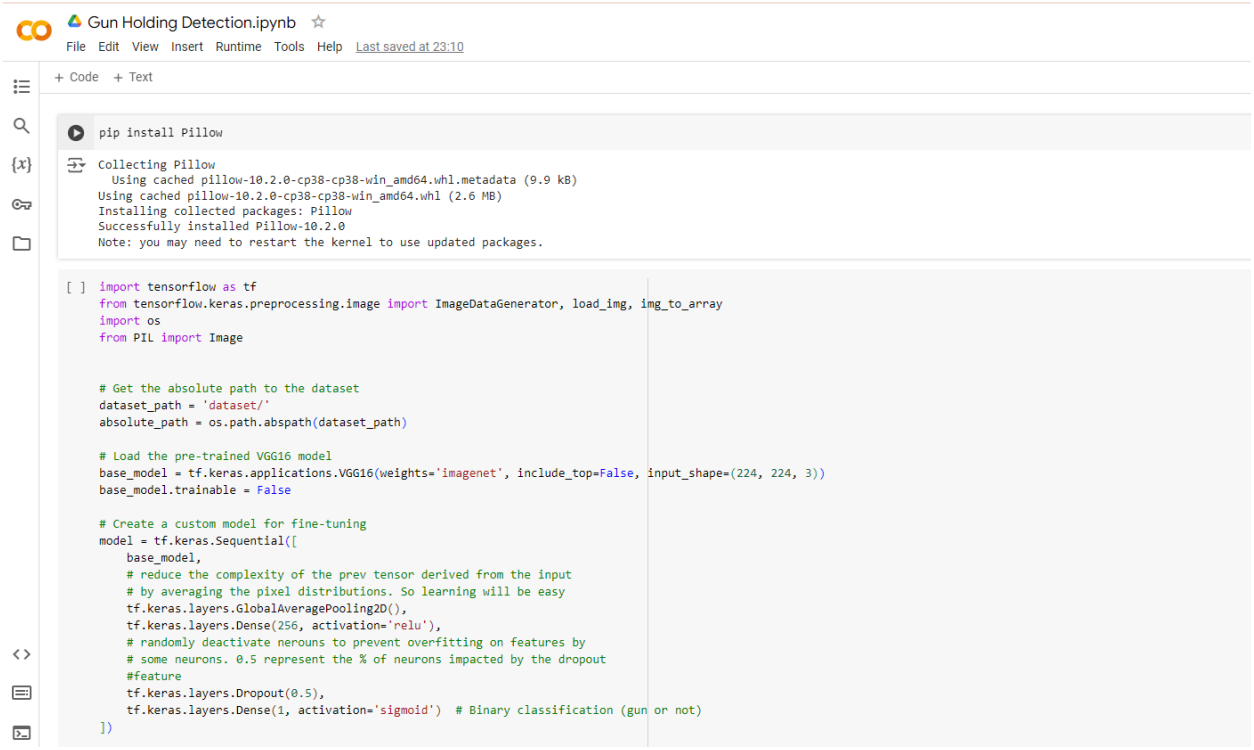
Confidence Levels: MobileNetV2 code provides a function to display predictions along with confidence levels, which is crucial for understanding the model's certainty in its predictions.

Flexibility: Functions are provided for loading the model, predicting new images, and displaying predictions with and without confidence levels.

6. Model Deployment:

Download Option: MobileNetV2 code includes an option to download the trained model file directly, facilitating easy deployment.

Practical -Gun Detection



Gun Holding Detection.ipynb

File Edit View Insert Runtime Tools Help Last saved at 23:10

+ Code + Text

```
[ ] # Compile the model
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

    # Data preparation with data augmentation. Larger batch sizes are good to
    # make the models to be more mature. However, it could lead to stuck due to
    # memory shortages. Hence, trial and error expertimentations are required.
    batch_size = 32
    # augment the images on the go as per the specs given during the training
    train_datagen = ImageDataGenerator(
        rescale=1./255,
        validation_split=0.2,
        rotation_range=20,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode='nearest'
    )
    # configured pipeline of datagenerator is directly linked to the folder
    train_generator = train_datagen.flow_from_directory(
        absolute_path,
        target_size=(224, 224),
        batch_size=batch_size,
        # augment the images considering correct and wrong representations.
        # this allow the model to learn with a comparative assesment.
        class_mode='binary',
        # can be either training or validation. In validation mode, adjust the
        # images to support the inferencing purposes.
        subset='training'
    )
```

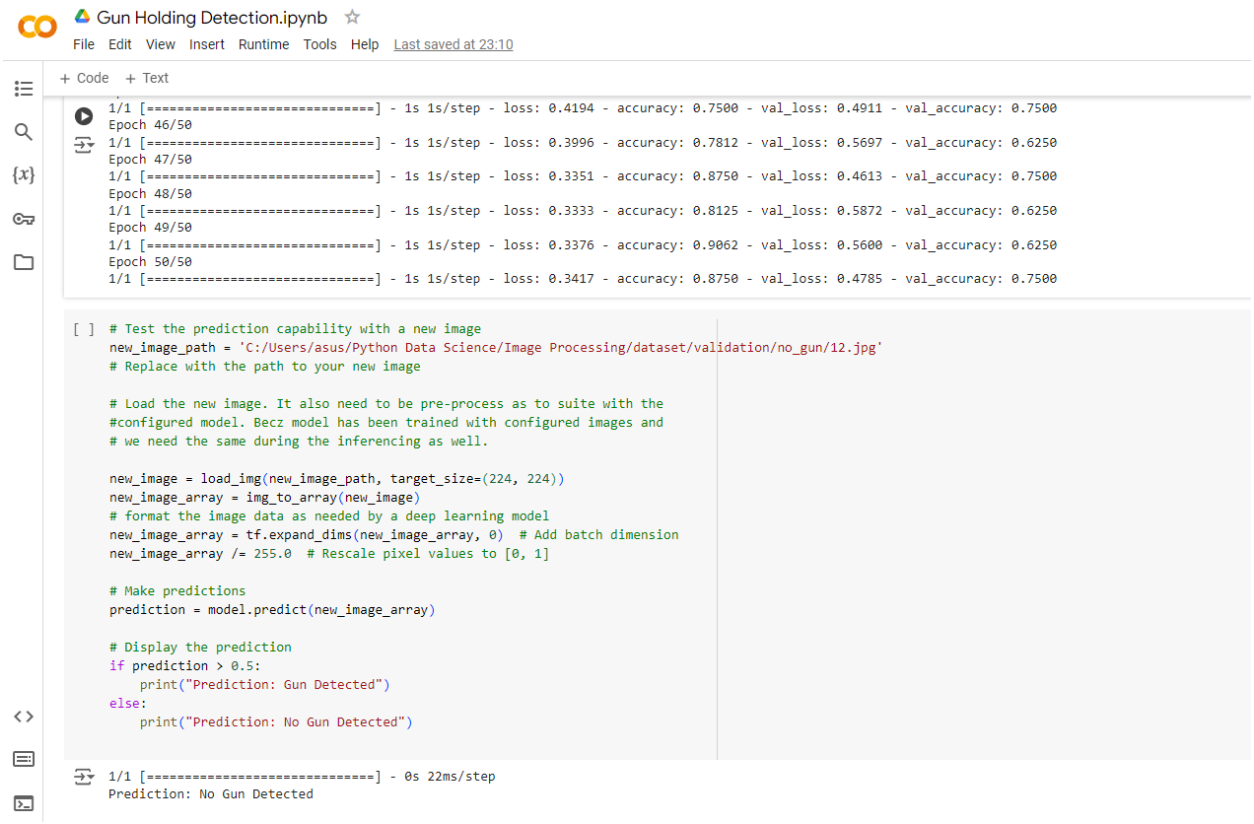
Gun Holding Detection.ipynb

File Edit View Insert Runtime Tools Help Last saved at 23:10

+ Code + Text

```
)

# arrangement of the validation dataset.
validation_generator = train_datagen.flow_from_directory(
    absolute_path,
    target_size=(224, 224),
    batch_size=batch_size,
    class_mode='binary',
    subset='validation'
)

# Train the model with augmented data
model.fit(
    train_generator,
    epochs=50,
    validation_data=validation_generator
)

# Save the model
model.save('custom_gun_model.h5')
```

```
Found 32 images belonging to 2 classes.
Found 8 images belonging to 2 classes.
Epoch 1/50
1/1 [==============================] - 2s 2s/step - loss: 0.6439 - accuracy: 0.7500 - val_loss: 0.5403 - val_accuracy: 0.7500
Epoch 2/50
1/1 [==============================] - 1s 1s/step - loss: 0.6355 - accuracy: 0.7188 - val_loss: 0.5610 - val_accuracy: 0.7500
Epoch 3/50
1/1 [==============================] - 1s 1s/step - loss: 0.5975 - accuracy: 0.7500 - val_loss: 0.5151 - val_accuracy: 0.7500
Epoch 4/50
1/1 [==============================] - 1s 1s/step - loss: 0.5597 - accuracy: 0.7500 - val_loss: 0.5306 - val_accuracy: 0.7500
Epoch 5/50
1/1 [==============================] - 1s 1s/step - loss: 0.5717 - accuracy: 0.7500 - val_loss: 0.5532 - val_accuracy: 0.7500
Epoch 6/50
1/1 [==============================] - 1s 1s/step - loss: 0.5178 - accuracy: 0.7812 - val_loss: 0.5686 - val_accuracy: 0.7500
Epoch 7/50
1/1 [==============================] - 1s 1s/step - loss: 0.4787 - accuracy: 0.7812 - val_loss: 0.4993 - val_accuracy: 0.7500
```



Install and Import Necessary Libraries:

Pillow: A Python Imaging Library used for image manipulation.

TensorFlow and Keras: Libraries used for building and training deep learning models.

os: A module that provides functions to interact with the operating system.

ImageDataGenerator, load_img, img_to_array: Functions from Keras for image preprocessing and augmentation.

Dataset Path and Model Loading:

dataset_path: Path to the dataset.

absolute_path: Converts the relative path to an absolute path.

VGG16: Loads the VGG16 model pre-trained on ImageNet, excluding the top (fully connected) layers.

base_model.trainable = False: Freezes the layers of the pre-trained model to prevent them from being updated during training.

Custom Model Creation:

GlobalAveragePooling2D: Reduces the spatial dimensions of the feature maps to a single vector by averaging the pixel values, which helps in reducing the model complexity and overfitting.

Dense(256): Adds a fully connected layer with 256 neurons and ReLU activation function.

Dropout(0.5): Randomly deactivates 50% of the neurons during training to prevent overfitting.

Dense(1, activation='sigmoid'): Adds an output layer with a single neuron and sigmoid activation function for binary classification.

Model Compilation:

optimizer='adam': Uses the Adam optimizer for training.

loss='binary_crossentropy': Uses binary cross-entropy loss for binary classification tasks.

metrics=['accuracy']: Monitors the accuracy metric during training.

Data Augmentation and Preparation:

batch_size: Sets the batch size for training.

ImageDataGenerator: Augments the training data with various transformations to improve the model's generalization.

rescale=1./255: Normalizes the pixel values to the range [0, 1].

validation_split=0.2: Reserves 20% of the data for validation.

rotation_range, width_shift_range, height_shift_range, shear_range, zoom_range, horizontal_flip, fill_mode: Parameters for data augmentation to create variations in the training images.

Data Generators:

train_generator: Generates augmented training data.

validation_generator: Generates validation data without augmentation transformations.

Model Training:

epochs=50: Sets the number of training epochs.

validation_data=validation_generator: Uses the validation generator for validation during training.

Model Saving:

model.save: Saves the trained model to a file.

y

Prediction on a New Image:

load_img: Loads the new image and resizes it to (224, 224).

img_to_array: Converts the image to a numpy array.

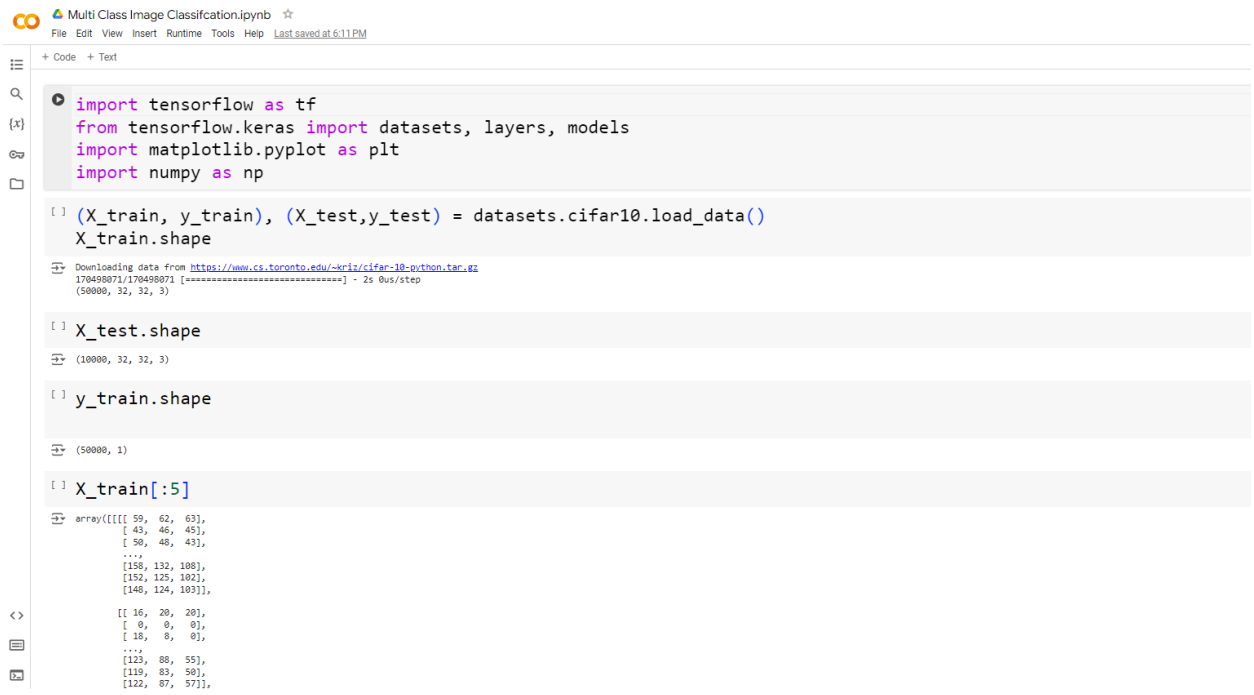
tf.expand_dims: Adds a batch dimension to the array.

new_image_array /= 255.0: Normalizes the pixel values.

model.predict: Makes a prediction on the new image.

if prediction > 0.5: Interprets the prediction result and prints whether a gun is detected or not.

Practical – Multi Class Image Classification



Multi Class Image Classifcation.ipynb

File Edit View Insert Runtime Tools Help Last saved at 6:11 PM

+ Code + Text

```
        ...,
        [ 76,  80,  83],
        [ 71,  75,  78],
        [ 71,  75,  78]],

       [[ 67,  75,  78],
        [ 68,  76,  79],
        [ 69,  75,  76],
        ...,
        [ 75,  79,  82],
        [ 71,  75,  78],
        [ 73,  77,  80]]], dtype=uint8)
```

```
y_train[:5]
```

```
array([[6],
       [9],
       [9],
       [4],
       [1]], dtype=uint8)
```

```
y_train.ndim # this denotes it`s a 2D array
```

```
2
```

```
y_train = y_train.reshape(-1,) # convert to 1d
y_train[:5]
```

```
array([6, 9, 9, 4, 1], dtype=uint8)
```

```
y_train.ndim # now it has become 1d
```

```
1
```

```
y_test = y_test.reshape(-1,) # re-shape the y axis as well
```

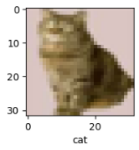
```
classes = ["airplane","automobile","bird","cat","deer","dog","frog","horse","ship","truck"]
```


```
classes = ["airplane","automobile","bird","cat","deer","dog","frog","horse","ship","truck"]
```

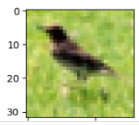

+ Code + Text

```
# for image plotting
def plot_sample(X, y, index):
    plt.figure(figsize = (15,2))
    plt.imshow(X[index])
    plt.xlabel(classes[y[index]])
```

```
plot_sample(X_train, y_train, 12225)
```




```
plot_sample(X_train, y_train, 45021)
```



```
# normalize training data
X_train = X_train / 255.0
X_test = X_test / 255.0
```

```
# simple ANN - accuracy is low compared to CNN
ann = models.Sequential([
        layers.Flatten(input_shape=(32,32,3)),
        layers.Dense(3000, activation='relu'),
        layers.Dense(1000, activation='relu'),
        layers.Dense(10, activation='softmax')  # 10 classes to be outputed
    ])

ann.compile(optimizer='SGD',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

ann.fit(X_train, y_train, epochs=5)
```

```
Epoch 1/5
1563/1563 [==============================] - 126s 80ms/step - loss: nan - accuracy: 0.1002
Epoch 2/5
1563/1563 [==============================] - 125s 80ms/step - loss: nan - accuracy: 0.1000
Epoch 3/5
1563/1563 [==============================] - 125s 80ms/step - loss: nan - accuracy: 0.1000
Epoch 4/5
1563/1563 [==============================] - 125s 80ms/step - loss: nan - accuracy: 0.1000
Epoch 5/5
1563/1563 [==============================] - 125s 80ms/step - loss: nan - accuracy: 0.1000
<keras.src.callbacks.History at 0x7a0fe3433b80>
```

Multi Class Image Classifcation.ipynb ☆

File Edit View Insert Runtime Tools Help

+ Code + Text

```
<keras.src.callbacks.History at 0x7a0fe3433b80>
```

```
# derive a clasification report for the 10 classes
from sklearn.metrics import confusion_matrix , classification_report
import numpy as np
y_pred = ann.predict(X_test)
y_pred_classes = [np.argmax(element) for element in y_pred]

print("Classification Report: \n", classification_report(y_test, y_pred_classes))
```

```
313/313 [==============================] - 1s 2ms/step
Classification Report:
              precision    recall  f1-score   support

           0       0.75      0.32      0.45      1000
           1       0.50      0.66      0.57      1000
           2       0.46      0.19      0.27      1000
           3       0.36      0.25      0.30      1000
           4       0.58      0.23      0.33      1000
           5       0.32      0.49      0.39      1000
           6       0.45      0.66      0.53      1000
           7       0.59      0.45      0.51      1000
           8       0.58      0.62      0.60      1000
           9       0.38      0.71      0.49      1000

    accuracy                           0.46     10000
   macro avg       0.50      0.46      0.44     10000
weighted avg       0.50      0.46      0.44     10000
```

```
# define high accurate CNN model
cnn = models.Sequential([
    layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
```

Multi Class Image Classifcation.ipynb

File Edit View Insert Runtime Tools Help

+ Code + Text

```
      layers.Flatten(),
      layers.Dense(64, activation='relu'),
      layers.Dense(10, activation='softmax')
  ])
```

```
cnn.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
cnn.fit(X_train, y_train, epochs=5)
```

```
Epoch 1/5
1563/1563 [==============================] - 52s 32ms/step - loss: 2.2679 - accuracy: 0.2644
Epoch 2/5
1563/1563 [==============================] - 51s 32ms/step - loss: 1.4269 - accuracy: 0.4875
Epoch 3/5
1563/1563 [==============================] - 50s 32ms/step - loss: 1.2083 - accuracy: 0.5752
Epoch 4/5
1563/1563 [==============================] - 50s 32ms/step - loss: 1.0817 - accuracy: 0.6234
Epoch 5/5
1563/1563 [==============================] - 50s 32ms/step - loss: 0.9893 - accuracy: 0.6572
<keras.src.callbacks.History at 0x7a0fd37e1c00>
```

```
cnn.evaluate(X_test,y_test)
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.9377 - accuracy: 0.6922
[0.9377458095550537, 0.6922000050544739]
```

```
y_pred = cnn.predict(X_test)
y_pred[:5]
```

```
313/313 [==============================] - 1s 2ms/step
array([[5.4908674e-03, 1.0147246e-05, 6.1776683e-02, 7.0458162e-01,
        3.5941335e-03, 2.1253419e-01, 3.6008940e-03, 3.1905137e-03,
        5.1300577e-03, 9.0841648e-05],
       [4.2236099e-04, 4.6995152e-03, 1.3979718e-07, 1.7947327e-08,
        2.0777520e-09, 1.3002817e-08, 1.0521770e-08, 9.8189867e-10,
```

Multi Class Image Classifcation.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
       [5.3417971e-05, 3.1073292e-04, 5.9530996e-02, 2.4605791e-01,
        6.0111934e-01, 1.5732184e-02, 7.6759972e-02, 8.5803717e-05,
        3.4425533e-04, 5.3870340e-06]], dtype=float32)
```

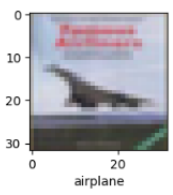
```
y_classes = [np.argmax(element) for element in y_pred]
y_classes[:5]
```

```
[3, 8, 8, 0, 4]
```

```
y_test[:5]
```

```
array([3, 8, 8, 0, 6], dtype=uint8)
```

```
plot_sample(X_test, y_test,3)
```




```
classes[y_classes[3]]
```

```
'airplane'
```

Loading and Inspecting the Dataset

Imports: Importing TensorFlow, matplotlib for visualization, and numpy for numerical operations.

Data Loading: Loading the CIFAR-10 dataset using datasets.cifar10.load_data(), which returns two tuples: one for training and one for testing.

Shape Inspection: Printing the shapes of the training and testing datasets to understand their dimensions.

Data Inspection and Reshaping

Data Inspection: Inspecting the first few samples of the training data and labels.

Reshaping Labels: The y_train and y_test arrays are reshaped from 2D to 1D. This is necessary because the labels initially have an extra dimension, making them 2D arrays. For many TensorFlow functions, 1D labels are required.

Class Definitions and Visualization

Class Labels: Defining the class labels corresponding to the CIFAR-10 dataset.

Visualization Function: A helper function to visualize samples from the dataset. It takes the dataset X, labels y, and an index index to plot the image and display the corresponding label.

Sample Visualization: Plotting specific samples to visually inspect the dataset.

Data Normalization

Normalization: Normalizing the image data to a range of [0, 1] by dividing by 255. This is crucial for neural networks as it helps in faster convergence during training by standardizing the input range.

Building and Training a Simple ANN

Model Architecture (ANN):

Flatten Layer: Converts the 3D input (32x32x3) into a 1D array (3072).

Dense Layer: A fully connected layer with 1000 neurons and ReLU activation.

Output Layer: A dense layer with 10 neurons (one for each class) and softmax activation for classification.

Compilation: Compiling the model with SGD optimizer and sparse categorical cross-entropy loss.

Training: Training the ANN for 5 epochs on the normalized training data.

Model Evaluation and Classification Report

Prediction: Making predictions on the test set.

Class Conversion: Converting the prediction probabilities to class labels using np.argmax.

Classification Report: Generating and printing a classification report to evaluate the model's performance across different classes using precision, recall, and F1-score.

Building and Training a CNN

Model Architecture (CNN):

Convolutional Layers: Two convolutional layers with 32 and 64 filters respectively, each followed by ReLU activation and max pooling.

Flatten Layer: Converts the 3D output from the convolutional layers to a 1D array.

Dense Layers: A fully connected layer with 64 neurons and ReLU activation, followed by the output layer with 10 neurons and softmax activation.

Compilation: Compiling the model with the Adam optimizer and sparse categorical cross-entropy loss.

Training: Training the CNN for 5 epochs on the normalized training data.

CNN Evaluation and Prediction

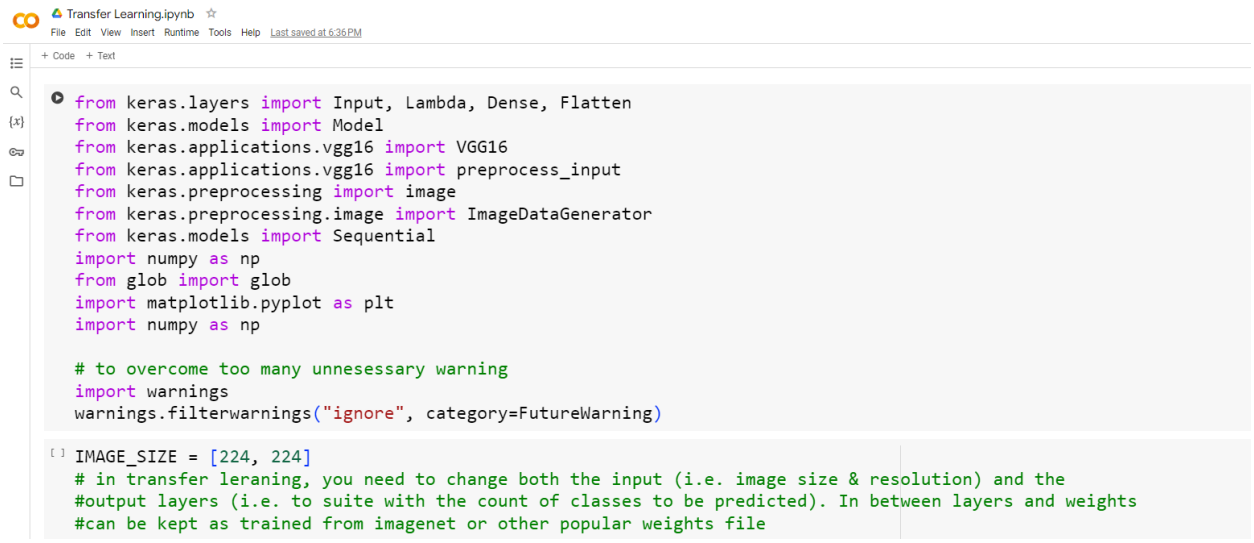
Evaluation: Evaluating the trained CNN on the test set.

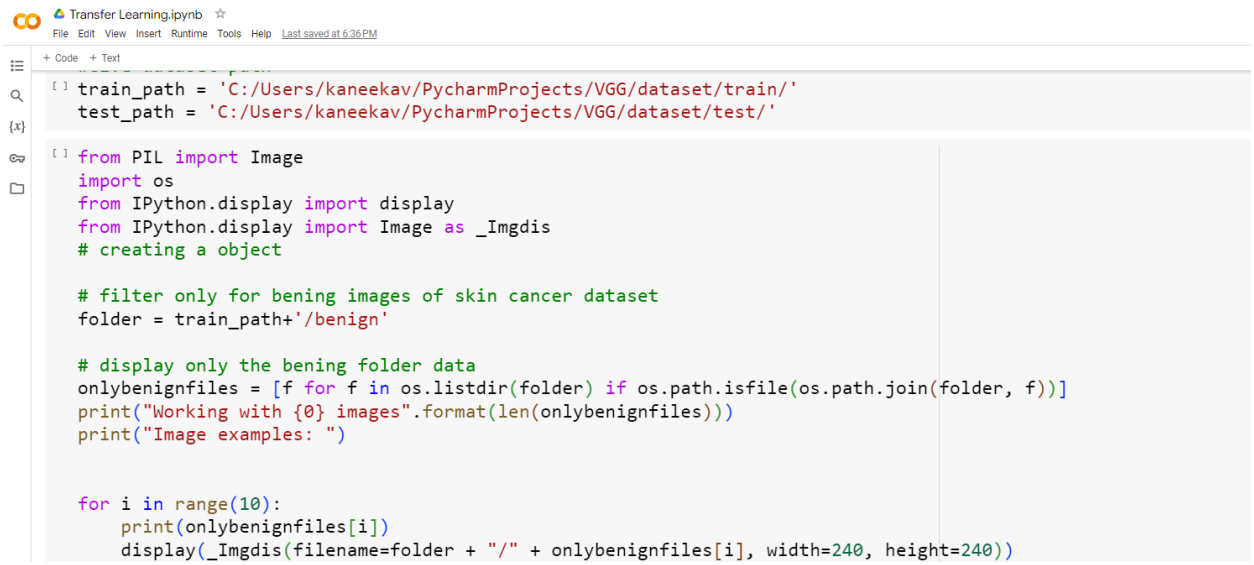
Prediction: Making predictions on the test set.

Class Conversion: Converting the prediction probabilities to class labels using np.argmax.

Sample Visualization: Plotting a sample image from the test set and displaying the predicted label.

Practical – Transfer learning





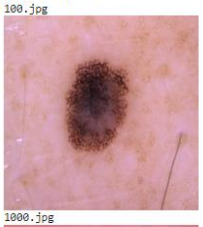
Transfer Learning.ipynb

File Edit View Insert Runtime Tools Help Last saved at 6:36 PM

+ Code + Text

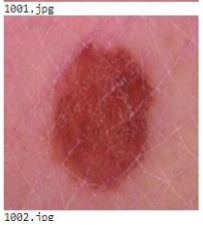
```
Working with 1440 images
Image examples:
```

100.jpg



1000.jpg

1001.jpg



1002.jpg

```
vgg = VGG16(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=False)
# in above we have defined the image size as 224,224 and it`s rgb there for we pass 3 channels, we re-use imagenet training
# weights. Include_top =false depicts, that the original fully connected layer of the VGG-16 is not going to be loaded.
# instead, we plan to load our own fully connected layer to recognize maignent and bening classes.
```

```
vgg.input
```

```
<KerasTensor: shape=(None, 224, 224, 3) dtype=float32 (created by layer 'input_2')>
```

```
for layer in vgg.layers:
  layer.trainable = False
# do not train the vgg layers with new dataset. Keep the imgenet weights as it is. Only, input and ouput layers are altered.
```

```
folders = glob('C:/Users/kaneekav/PycharmProjects/VGG/dataset/train/*')
print(len(folders))
```

```
2
```

```
x = Flatten()(vgg.output) # this is the custom fully connected layer we are defining. we say, condense it,
#just to 2 classes only, as the folders vaiable contains the required class counts
prediction = Dense(len(folders), activation='softmax')(x)
model = Model(inputs=vgg.input, outputs=prediction) # add the newly created output layer to the VGG model
model.summary()
```

Transfer Learning.ipynb

File Edit View Insert Runtime Tools Help Last saved at 6:36 PM

+ Code + Text

```
Model: "model_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 224, 224, 3)]     0

block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792

block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928

block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0

block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856

block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584

block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0

block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168

block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080

block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080

block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0

block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160

block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808

block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808

block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0

block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808

block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808

block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808

block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0

flatten_4 (Flatten)          (None, 25088)             0

dense_4 (Dense)              (None, 2)                 50178

=================================================================
Total params: 14,764,866
Trainable params: 50,178
Non-trainable params: 14,714,688
_________________________________________________________________
```

Transfer Learning.ipynb

File Edit View Insert Runtime Tools Help Last saved at 6:36 PM

+ Code + Text

```
[ ] import tensorflow as tf
    from tensorflow import keras
```

```
[ ] # keras.optimizers.Adam
    optimizer = tf.keras.optimizers.Adam()
```

```
[ ] # define accuracy matrices
    adam = optimizer
    model.compile(loss='binary_crossentropy',
                  optimizer=adam,
                  metrics=['accuracy'])
```

```
[ ] # data augmentation of the skin cancer dataset. will prevent overfitting
    train_datagen = ImageDataGenerator(
        preprocessing_function=preprocess_input,
        rotation_range=40,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode='nearest')
```

Transfer Learning.ipynb

File Edit View Insert Runtime Tools Help Last saved at 6:36 PM

+ Code + Text

```
[ ] test_datagen = ImageDataGenerator(
        preprocessing_function=preprocess_input,
        rotation_range=40,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode='nearest')
```

```
[ ] train_set = train_datagen.flow_from_directory(train_path,
                                                  target_size = (224, 224),
                                                  batch_size = 32,
                                                  class_mode = 'categorical')
```

```
Found 2637 images belonging to 2 classes.
```

```
[ ] test_set = test_datagen.flow_from_directory(test_path,
                                                target_size = (224, 224),
                                                batch_size = 32,
                                                class_mode = 'categorical')
```

```
Found 660 images belonging to 2 classes.
```

```
[ ] from datetime import datetime
    from keras.callbacks import ModelCheckpoint

    # model check points allows to select and save the best model resulted during the epocs
```

Transfer Learning.ipynb

File Edit View Insert Runtime Tools Help Last saved at 6:36 PM

Comment Share

+ Code + Text

Connect + Gemini

```
checkpoint = ModelCheckpoint(filepath='mymodel.h5',
                             verbose=2, save_best_only=True)

callbacks = [checkpoint]

start = datetime.now()

# model training commences
model_history=model.fit_generator(
  train_set, # augmented training dataset
  validation_data=test_set,
  epochs=10,
  steps_per_epoch=5,
  validation_steps=32,
    callbacks=callbacks ,verbose=2)


duration = datetime.now() - start
print("Training completed in time: ", duration)
```

```
C:\Users\kaneekav\AppData\Local\Temp/ipykernel_13696/3905777936.py:14: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.
  model_history=model.fit_generator(
Epoch 1/10
WARNING:tensorflow:Your input ran out of data; interrupting training. Make sure that your dataset or generator can generate at least `steps_per_epoch * epochs` batches (in this case, 32 batches). You may need to use the repeat() function when building your dataset.

Epoch 1: val_loss improved from inf to 2.59102, saving model to mymodel.h5
5/5 - 89s - loss: 3.9860 - accuracy: 0.6125 - val_loss: 2.5910 - val_accuracy: 0.7803 - 89s/epoch - 18s/step
Epoch 2/10
WARNING:tensorflow:Can save best model only with val_loss available, skipping.
5/5 - 20s - loss: 2.6004 - accuracy: 0.7437 - 20s/epoch - 4s/step
Epoch 3/10
WARNING:tensorflow:Can save best model only with val_loss available, skipping.
5/5 - 20s - loss: 1.6282 - accuracy: 0.8250 - 20s/epoch - 4s/step
Epoch 4/10
WARNING:tensorflow:Can save best model only with val_loss available, skipping.
```

Transfer Learning.ipynb

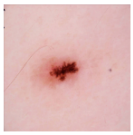
File Edit View Insert Runtime Tools Help Last saved at 6:36 PM

+ Code + Text

```
duration = datetime.now() - start
print("Training completed in time: ", duration)
```

```
C:\Users\kaneekav\AppData\Local\Temp/ipykernel_13696/3905777936.py:14: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.
  model_history=model.fit_generator(
Epoch 1/10
WARNING:tensorflow:Your input ran out of data; interrupting training. Make sure that your dataset or generator can generate at least `steps_per_epoch * epochs` batches (in this case, 32 batches). You may need to use the repeat() function when building your dataset.

Epoch 1: val_loss improved from inf to 2.59102, saving model to mymodel.h5
5/5 - 89s - loss: 3.9860 - accuracy: 0.6125 - val_loss: 2.5910 - val_accuracy: 0.7803 - 89s/epoch - 18s/step
Epoch 2/10
WARNING:tensorflow:Can save best model only with val_loss available, skipping.
5/5 - 20s - loss: 2.6004 - accuracy: 0.7437 - 20s/epoch - 4s/step
Epoch 3/10
WARNING:tensorflow:Can save best model only with val_loss available, skipping.
5/5 - 20s - loss: 1.6282 - accuracy: 0.8250 - 20s/epoch - 4s/step
Epoch 4/10
WARNING:tensorflow:Can save best model only with val_loss available, skipping.
5/5 - 19s - loss: 1.6017 - accuracy: 0.7660 - 19s/epoch - 4s/step
Epoch 5/10
WARNING:tensorflow:Can save best model only with val_loss available, skipping.
5/5 - 22s - loss: 1.2480 - accuracy: 0.8500 - 22s/epoch - 4s/step
Epoch 6/10
WARNING:tensorflow:Can save best model only with val_loss available, skipping.
5/5 - 22s - loss: 1.3427 - accuracy: 0.7937 - 22s/epoch - 4s/step
Epoch 7/10
WARNING:tensorflow:Can save best model only with val_loss available, skipping.
5/5 - 22s - loss: 2.3594 - accuracy: 0.7375 - 22s/epoch - 4s/step
Epoch 8/10
WARNING:tensorflow:Can save best model only with val_loss available, skipping.
5/5 - 21s - loss: 2.5879 - accuracy: 0.7000 - 21s/epoch - 4s/step
Epoch 9/10
WARNING:tensorflow:Can save best model only with val_loss available, skipping.
5/5 - 22s - loss: 2.0544 - accuracy: 0.7563 - 22s/epoch - 4s/step
Epoch 10/10
WARNING:tensorflow:Can save best model only with val_loss available, skipping.
5/5 - 22s - loss: 2.6998 - accuracy: 0.7625 - 22s/epoch - 4s/step
Training completed in time:  0:04:40.282912
```

```
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.vgg16 import decode_predictions
import numpy as np
img_pth="C:/Users/kaneekav/PycharmProjects/VGG/benign.jpg"
img=image.load_img(img_pth,color_mode='rgb', target_size=(224,224))
display(img)
```



```
x=image.img_to_array(img)
```

Transfer Learning.ipynb

File Edit View Insert Runtime Tools Help Last saved at 6:36 PM

+ Code + Text

```
x=image.img_to_array(img)
x.shape
```

```
(224, 224, 3)
```

+ Code + Text

```
x=np.expand_dims(x,axis=0)
```

+ Code + Text

```
x.shape
```

```
(1, 224, 224, 3)
```

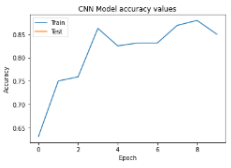
```
x=preprocess_input(x)
# features=model.predict(x)
# p=decode_predictions(features)
print(x.shape)
```

```
(1, 224, 224, 3)
```

```
features=model.predict(x)
print(features)
```

```
[[1.16204726e-10 1.00000000e+00]]
```

```
# Plot training & validation loss values
plt.plot(model_history.history['accuracy'])
plt.plot(model_history.history['val_accuracy'])
plt.title('CNN Model accuracy values')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```



1. Importing Required Libraries

Keras Layers and Models: Importing necessary modules from Keras for building and modifying neural network layers and models.

VGG16 Model: Importing VGG16 and its preprocessing function from Keras applications for transfer learning.

Image Processing: Modules for image loading and preprocessing.

Miscellaneous Imports: Importing numpy for numerical operations, glob for file pattern matching, matplotlib for visualization, and warnings to suppress unnecessary warnings.

2. Defining Image Size and Dataset Paths

Image Size: Setting the image size to 224x224 pixels, as required by VGG16.

Dataset Paths: Specifying paths to the training and testing datasets.

3. Displaying Sample Images from Dataset

PIL and IPython: Importing PIL for image manipulation and IPython for displaying images within a notebook environment.

Display Sample Images: Listing and displaying sample images from the benign category to visually inspect the dataset.

4. Loading Pre-trained VGG16 Model

VGG16 Model: Loading the VGG16 model pre-trained on ImageNet, excluding the top fully connected layers. This allows us to add our own classification layers suitable for our specific dataset.

5. Freezing Pre-trained Layers

Freezing Layers: Setting the trainable attribute of each layer to False to retain the pre-trained weights and avoid retraining these layers. Only the newly added layers will be trained.

6. Adding Custom Classification Layers

Class Count: Using glob to determine the number of classes by counting the subdirectories in the training path.

Adding Layers: Adding a Flatten layer to convert the 3D tensor outputs of VGG16 to 1D. Adding a Dense layer with softmax activation for classification, with the number of output neurons equal to the number of classes.

7. Compiling the Model

Optimizer: Using the Adam optimizer for training the model.

Compilation: Compiling the model with binary cross-entropy loss and accuracy as the evaluation metric.

8. Data Augmentation

Train Data Augmentation: Creating an ImageDataGenerator for the training set with various augmentation techniques such as rotation, shifting, shearing, zooming, and flipping to prevent overfitting and improve generalization.

Test Data Augmentation: Similarly creating an ImageDataGenerator for the test set with the same augmentation techniques.

9. Preparing Data Generators

Data Generators: Creating data generators for the training and testing datasets with the specified target size, batch size, and class mode.

10. Model Checkpointing

Model Checkpointing: Setting up a checkpoint callback to save the best model during training based on validation performance.

11. Training the Model

duration = datetime.now() - start

print("Training completed in time: ", duration)

Model Training: Training the model using the augmented training set and validating on the augmented test set. The training runs for 10 epochs, with checkpoints and duration tracking.

12. Making Predictions

Loading Image: Loading and displaying an image for prediction.

Preprocessing Image: Converting the image to an array, expanding its dimensions to match the model's input, and preprocessing it.

Prediction: Making a prediction on the preprocessed image and printing the results.

13. Plotting Training History

Plotting Accuracy: Plotting the training and validation accuracy over the epochs to visualize the model's performance during training.