Face Detection

For Orin board users, directly open a web page and enter the IP address:8888 to access jupyter-lab and run directly. For Jetson-Nano board users, you need to first enter the docker container, then enter the following command in docker:

```
cd
jupyter-lab --allow-root
```

Then open a web page and enter the IP address:9999 to access jupyter-lab and run the following program.

1. Basic Theory

Face detection algorithms can be divided into two major categories based on methods: **feature-based algorithms and image-based algorithms**.

- **Feature-based Algorithms**

  Feature-based algorithms extract features from images and match them with face features. If they match, it indicates a face; otherwise, it's not. The extracted features are manually designed features, such as Haar, FHOG. After feature extraction, classifiers are used for judgment. Simply put, template matching is used - matching face template images with various positions in the image to be detected. The matching content is the extracted features, and then classifiers are used to determine whether there is a face.

- **Image-based Algorithms**

  Image-based algorithms divide the image into many small windows and then determine whether each small window contains a face. Usually, image-based methods rely on statistical analysis and machine learning, finding statistical relationships between faces and non-faces through statistical analysis or learning processes for face detection. The most representative is CNN, **CNN for face detection is currently the most effective and fastest**.

- **Haar Features**

  We use **machine learning** methods to complete **face detection**. First, **we need a large number of positive sample images (face images) and negative sample images (images without faces) to train the classifier**. We need to **extract features from them**. The **Haar features** shown in the figure below will be used, like our convolution kernels. **Each feature is a value**, which equals the sum of pixel values in the **black rectangle** minus the sum of pixel values in the **white rectangle**.

This tutorial uses Haar cascade classifiers, which is one of the most common and efficient object detection methods. This machine learning method trains cascade functions based on a large number of positive and negative images, then uses them to detect objects in other images. If you don't want to create your own classifier, OpenCV also includes many pre-trained classifiers that can be used for face, eye, smile detection, etc.

2. Main Code

Code path:

```
#Jetson-Nano users need to enter the docker container to view
~/dofbot_pro/src/dofbot_basic_visual/scripts/04.Face_Detection.ipynb
```

Import Haar cascade classifier xml file

```
 self.faceDetect = cv.CascadeClassifier("haarcascade_frontalface_default.xml")
```

Face filtering

```
def face_filter(self, faces):
        '''
        Filter the face
        '''
        if len(faces) == 0: return None
        # At present, we are looking for the face with the largest area in the
picture
        max_face = max(faces, key=lambda face: face[2] * face[3])
        (x, y, w, h) = max_face
        # Set the minimum threshold of face detection
        if w < 10 or h < 10: return None
        return max_face
```

Detect and frame the face

```
def find_face(image):
    # Copy the original image to avoid interference during processing
    img = image.copy()
    # Convert image to grayscale
    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    # Face detection
    faces = faceDetect.detectMultiScale(gray, scaleFactor=1.3, minNeighbors=5)
    pos = None
    if len(faces) != 0:
        # Face filtering
        face = face_filter(faces)
        if face is not None:
            (x, y, w, h) = face
            # Draw a rectangle on the original color map
            cv.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 4)
    return image
```

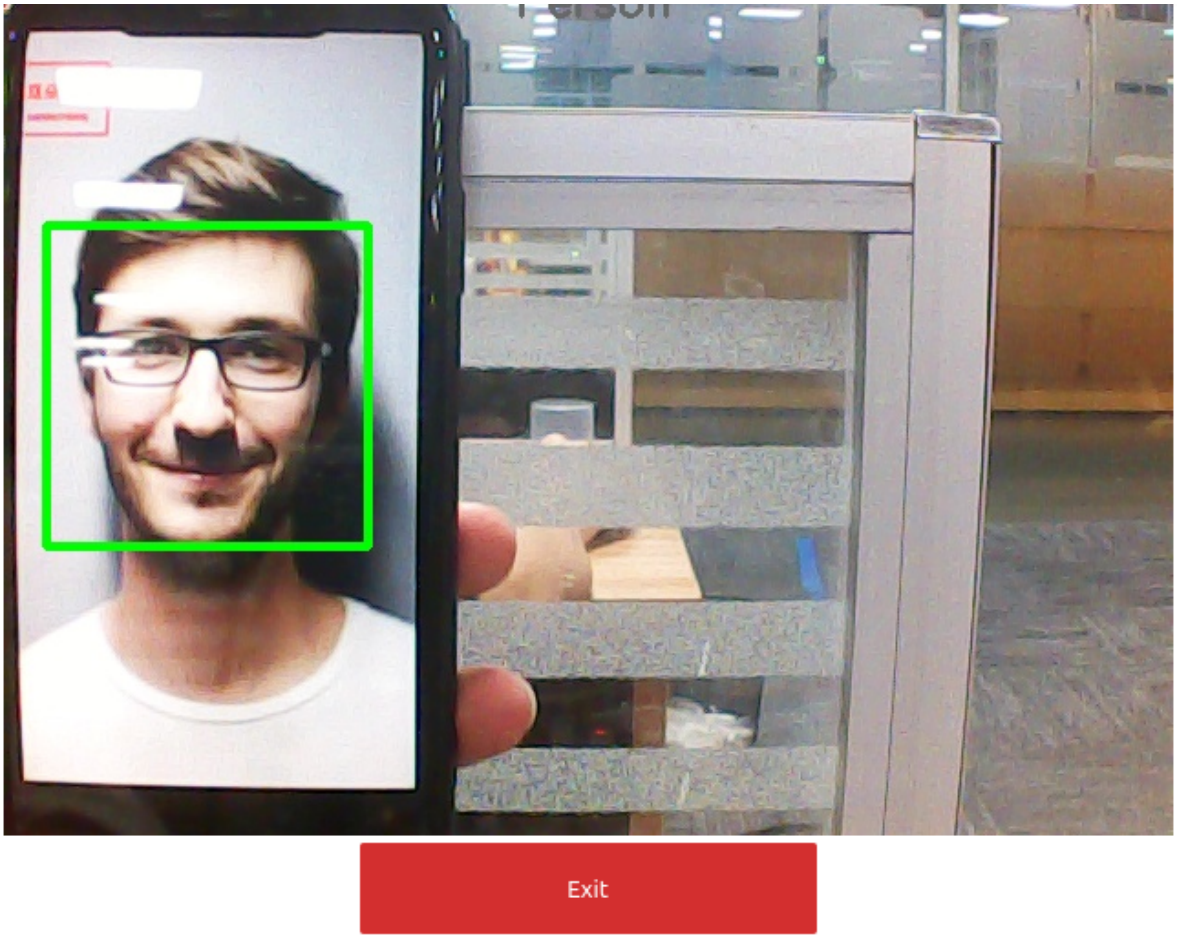
Main thread:

```
def camera():
    global model
    # Open camera
    capture = cv.VideoCapture(0)
    capture.set(cv.CAP_PROP_FRAME_WIDTH, 640)
    capture.set(cv.CAP_PROP_FRAME_HEIGHT, 480)
    while capture.isOpened():
```

```
        try:
            if model == 'Exit':
                capture.release()
                break
            _, img = capture.read()
            fps.update_fps()
            find_face(img)
            fps.show_fps(img)
            imgbox.value = cv.imencode('.jpg', img)[1].tobytes()
        except Exception as e:
            capture.release()
            print(e)
            break
```

Click the "Run entire program" button on the jupyterlab toolbar, then scroll to the bottom to see the camera component display.

If a face appears in the camera view at this time, the program will detect and frame the face.



If you need to exit the program, please click the [Exit] button.