In [ ]:
```
!pip install dlib opencv-contrib-python imutils
```

In [ ]:
```
%load_ext autoreload
%autoreload 2
```

In [ ]:
```
from google.colab import drive

drive.mount('/content/gdrive', force_remount=True)
```

In [ ]:
```
%cd /content/gdrive/MyDrive/Drowsiness_detection_using_dlib/
```

In [ ]:
```
!ls
```

In [ ]:
```
!wget http://dlib.net/files/data/ibug_300W_large_face_landmark_dataset.tar.gz
!tar -xvf ibug_300W_large_face_landmark_dataset.tar.gz
```

In [ ]:
```
source_path = '/content/ibug_300W_large_face_landmark_dataset'
destination_path = '/content/gdrive/MyDrive/Drowsiness_detection_using_dlib/dat

import shutil
shutil.copytree(source_path, destination_path)
```

In [ ]:
```
import re

input_path = "/content/gdrive/MyDrive/Drowsiness_detection_using_dlib/data/labe
output_path = "/content/gdrive/MyDrive/Drowsiness_detection_using_dlib/data/lab


LANDMARKS = set(list(range(36, 48)))

PART = re.compile("part name='[0-9]+'")

rows = open(input_path).read().strip().split("\n")
output = open(output_path, "w")

for row in rows:
    parts = re.findall(PART, row)

    if len(parts) == 0:
        output.write("{}\n".format(row))
    else:
        attr = "name='"
        i = row.find(attr)
        j = row.find("'", i + len(attr) + 1)
        name = int(row[i + len(attr):j])
        if name in LANDMARKS:
            output.write("{}\n".format(row))

output.close()
```

7 most important hyperparameters we can tune/set when training your own custom dlib shape predictor. These values are:

- tree_depth
- nu
- cascade_depth

In [ ]:
```
import multiprocessing
import dlib

training_path = "/content/gdrive/MyDrive/Drowsiness_detection_using_dlib/data/
model_path = "/content/gdrive/MyDrive/Drowsiness_detection_using_dlib/shape_pre

options = dlib.shape_predictor_training_options()
options.tree_depth = 4
options.nu = 0.1
options.cascade_depth = 15
options.feature_pool_size = 400
options.num_test_splits = 50
options.oversampling_amount = 5
options.oversampling_translation_jitter = 0.1
options.be_verbose = True
options.num_threads = multiprocessing.cpu_count()
options.num_test_splits = 25

print(options)
```

In [ ]:
```
dlib.train_shape_predictor(training_path, model_path, options)
print(f"Model saved to {model_path}")
```

In [ ]:
```
import dlib

predictor_path = "/content/gdrive/MyDrive/Drowsiness_detection_using_dlib/data/
xml_path = "/content/gdrive/MyDrive/Drowsiness_detection_using_dlib/data/labels
```

In [ ]:
```
print(predictor_path)
print(xml_path)
```

In [ ]:
```
error = dlib.test_shape_predictor("/content/gdrive/MyDrive/Drowsiness_detection
print("Training accuracy: {}".format(error))
```

In [ ]:
```
import re

input_path = "/content/gdrive/MyDrive/Drowsiness_detection_using_dlib/data/labe
output_path = "/content/gdrive/MyDrive/Drowsiness_detection_using_dlib/data/lab

LANDMARKS = set(list(range(36, 48)))

PART = re.compile("part name='[0-9]+'")

rows = open(input_path).read().strip().split("\n")
output = open(output_path, "w")

for row in rows:

    parts = re.findall(PART, row)

    if len(parts) == 0:
        output.write("{}\n".format(row))
    else:
        attr = "name='"
        i = row.find(attr)
        j = row.find("'", i + len(attr) + 1)
        name = int(row[i + len(attr):j])

        if name in LANDMARKS:
            output.write("{}\n".format(row))

output.close()
print(f"Finished writing to {output_path}")
```

In [ ]:
```
import dlib

predictor_path = "/content/gdrive/MyDrive/Drowsiness_detection_using_dlib/data/
xml_path = "/content/gdrive/MyDrive/Drowsiness_detection_using_dlib/data/labels

error_testing = dlib.test_shape_predictor("/content/gdrive/MyDrive/Drowsiness_d
print("Testing accuracy: {}".format(error_testing))
```

In [ ]:
```
import os
import dlib
import pandas as pd
import matplotlib.pyplot as plt

faces_folder = 'path/to/your/faces_folder'
training_xml_path = os.path.join(faces_folder, "training_with_face_landmarks.xm
testing_xml_path = os.path.join(faces_folder, "testing_with_face_landmarks.xml"

options = dlib.shape_predictor_training_options()
options.tree_depth = 5
options.nu = 0.1
options.cascade_depth = 10
options.num_trees_per_cascade_level = 500
options.oversampling_amount = 5
options.feature_pool_size = 400
options.feature_pool_region_padding = 0
options.lambda_param = 0.1
options.num_test_splits = 25
```

In [ ]:
```
metrics_df = pd.read_csv(metrics_csv_path)

def plot_metrics(metrics_df):
    epochs = metrics_df['epoch']

    fig, ax1 = plt.subplots()

    ax1.set_xlabel('Epoch')
    ax1.set_ylabel('Loss', color='tab:red')
    ax1.plot(epochs, metrics_df['loss'], color='tab:red', label='Loss')
    ax1.tick_params(axis='y', labelcolor='tab:red')

    ax2 = ax1.twinx()
    ax2.set_ylabel('Metrics', color='tab:blue')
    ax2.plot(epochs, metrics_df['precision'], color='tab:blue', linestyle='--'
    ax2.plot(epochs, metrics_df['accuracy'], color='tab:green', linestyle='-.'
    ax2.tick_params(axis='y', labelcolor='tab:blue')

    fig.tight_layout()
    ax1.legend(loc='upper left')
    ax2.legend(loc='upper right')

    plt.title('Training Metrics Over Epochs')
    plt.show()

plot_metrics(metrics_df)
```

In [ ]:
```
from IPython.display import display, Javascript
from google.colab.output import eval_js
from base64 import b64decode

def take_photo(filename='photo.jpg', quality=0.8):
  js = Javascript('''
    async function takePhoto(quality) {
      const div = document.createElement('div');
      const capture = document.createElement('button');
      capture.textContent = 'Capture';
      div.appendChild(capture);

      const video = document.createElement('video');
      video.style.display = 'block';
      const stream = await navigator.mediaDevices.getUserMedia({video: true});

      document.body.appendChild(div);
      div.appendChild(video);
      video.srcObject = stream;
      await video.play();

      // Resize the output to fit the video element.
      google.colab.output.setIframeHeight(document.documentElement.scrollHeight

      // Wait for Capture to be clicked.
      await new Promise((resolve) => capture.onclick = resolve);

      const canvas = document.createElement('canvas');
      canvas.width = video.videoWidth;
      canvas.height = video.videoHeight;
      canvas.getContext('2d').drawImage(video, 0, 0);
      stream.getVideoTracks()[0].stop();
      div.remove();
      return canvas.toDataURL('image/jpeg', quality);
    }
    ''')
  display(js)
  data = eval_js('takePhoto({})'.format(quality))
  binary = b64decode(data.split(',')[1])
  with open(filename, 'wb') as f:
    f.write(binary)
  return filename
```

In [ ]:
```
from IPython.display import Image
try:
  filename = take_photo()
  print('Saved to {}'.format(filename))

  display(Image(filename))
except Exception as err:

  print(str(err))
```

In [ ]:

```
import cv2
import dlib
from imutils import face_utils
import imutils
from google.colab.patches import cv2_imshow

shape_predictor_path = "/content/gdrive/MyDrive/Drowsiness_detection_using_dlib
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor(shape_predictor_path)

image_path = "/content/photo.jpg"
image = cv2.imread(image_path)

image = imutils.resize(image, width=400)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
rects = detector(gray, 0)

for rect in rects:

    (x, y, w, h) = face_utils.rect_to_bb(rect)
    cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 255), 2)
    shape = predictor(gray, rect)
    shape = face_utils.shape_to_np(shape)
    for (sX, sY) in shape:
        cv2.circle(image, (sX, sY), 1, (0, 0, 255), -1)

cv2_imshow(image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

In [ ]:
```
import re

input_path = "/content/gdrive/MyDrive/Drowsiness_detection_using_dlib/data/labe
output_path = "/content/gdrive/MyDrive/Drowsiness_detection_using_dlib/data/lab


LANDMARKS = set(list(range(36, 48)))

PART = re.compile("part name='[0-9]+'")

rows = open(input_path).read().strip().split("\n")
output = open(output_path, "w")

for row in rows:

    parts = re.findall(PART, row)

    if len(parts) == 0:
        output.write("{}\n".format(row))
    else:
        attr = "name='"
        i = row.find(attr)
        j = row.find("'", i + len(attr) + 1)
        name = int(row[i + len(attr):j])

        if name in LANDMARKS:
            output.write("{}\n".format(row))

output.close()
print(f"Finished writing to {output_path}")
```

In [ ]:
```
%cd /content/gdrive/MyDrive/Drowsiness_detection_using_dlib/
```

In [ ]:

```
import cv2
import dlib
from imutils import face_utils
import imutils
import os

shape_predictor_path = "/content/gdrive/MyDrive/Drowsiness_detection_using_dlib

detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor(shape_predictor_path)

image_dir = "/content/gdrive/MyDrive/Drowsiness_detection_using_dlib/testing"
output_dir = "/content/gdrive/MyDrive/Drowsiness_detection_using_dlib/testoutpu

os.makedirs(output_dir, exist_ok=True)

for filename in os.listdir(image_dir):
    if filename.endswith(".jpg") or filename.endswith(".png"):
        image_path = os.path.join(image_dir, filename)
        image = cv2.imread(image_path)

        image = imutils.resize(image, width=400)
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

        rects = detector(gray, 0)

        for rect in rects:

            (x, y, w, h) = face_utils.rect_to_bb(rect)
            cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 255), 2)
            shape = predictor(gray, rect)
            shape = face_utils.shape_to_np(shape)
            for (sX, sY) in shape:
                cv2.circle(image, (sX, sY), 1, (0, 0, 255), -1)

        output_path = os.path.join(output_dir, filename)
        cv2.imwrite(output_path, image)
        print(f"[INFO] Processed and saved {filename}")

print("All images have been processed and saved.")
```