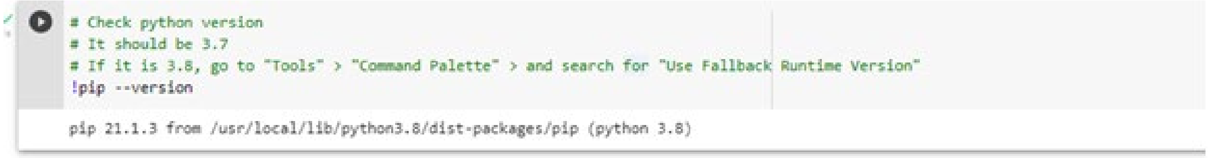
Create Lips Segmentation Dataset

Tensorflow Deeplab

1. Check python version
   - Method 1 (Deprecated as of December 17, 2022)



```
# Check python version
# It should be 3.7
# If it is 3.8, go to "Tools" > "Command Palette" > and search for "Use Fallback Runtime Version"
!pip --version
```

```
pip 21.1.3 from /usr/local/lib/python3.8/dist-packages/pip (python 3.8)
```

Figure 1.1: Wrong Python Version

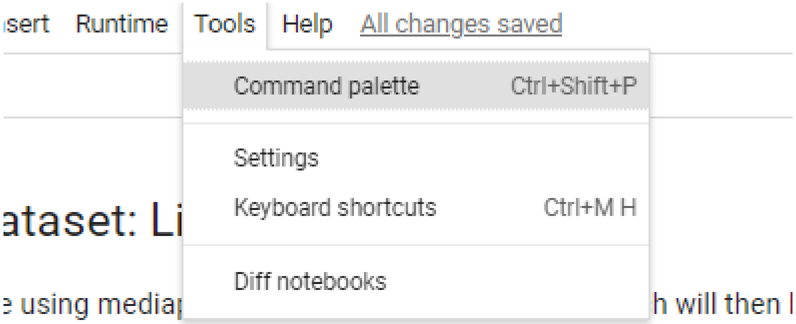


Figure 1.2: Go to Tools > Command palette

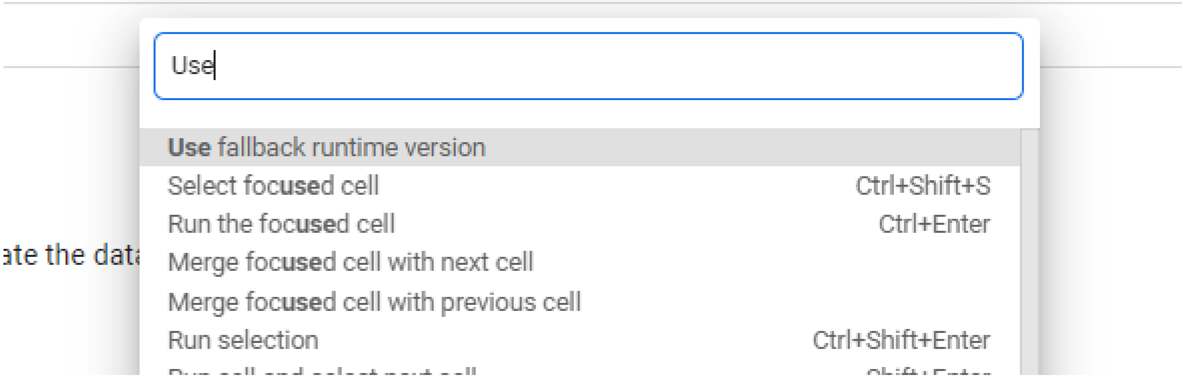
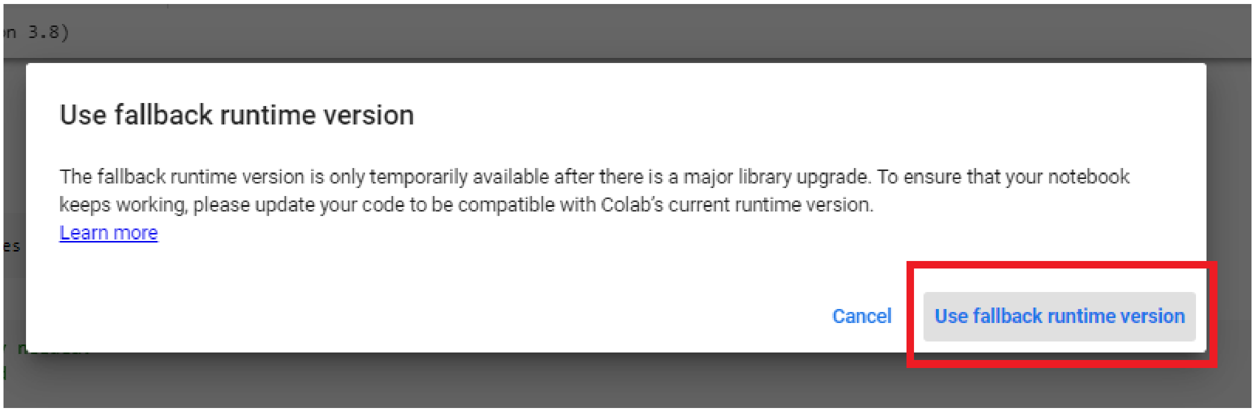


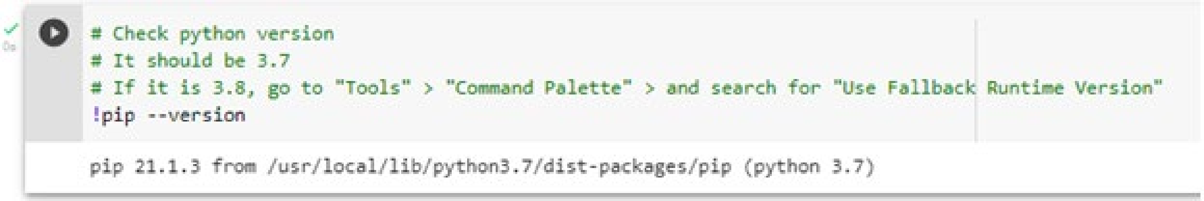
Figure 1.3: Search for “Use Fallback Runtime Version”



Use fallback runtime version

The fallback runtime version is only temporarily available after there is a major library upgrade. To ensure that your notebook keeps working, please update your code to be compatible with Colab's current runtime version.
Learn more

Figure 1.4: Use Fallback Runtime Version

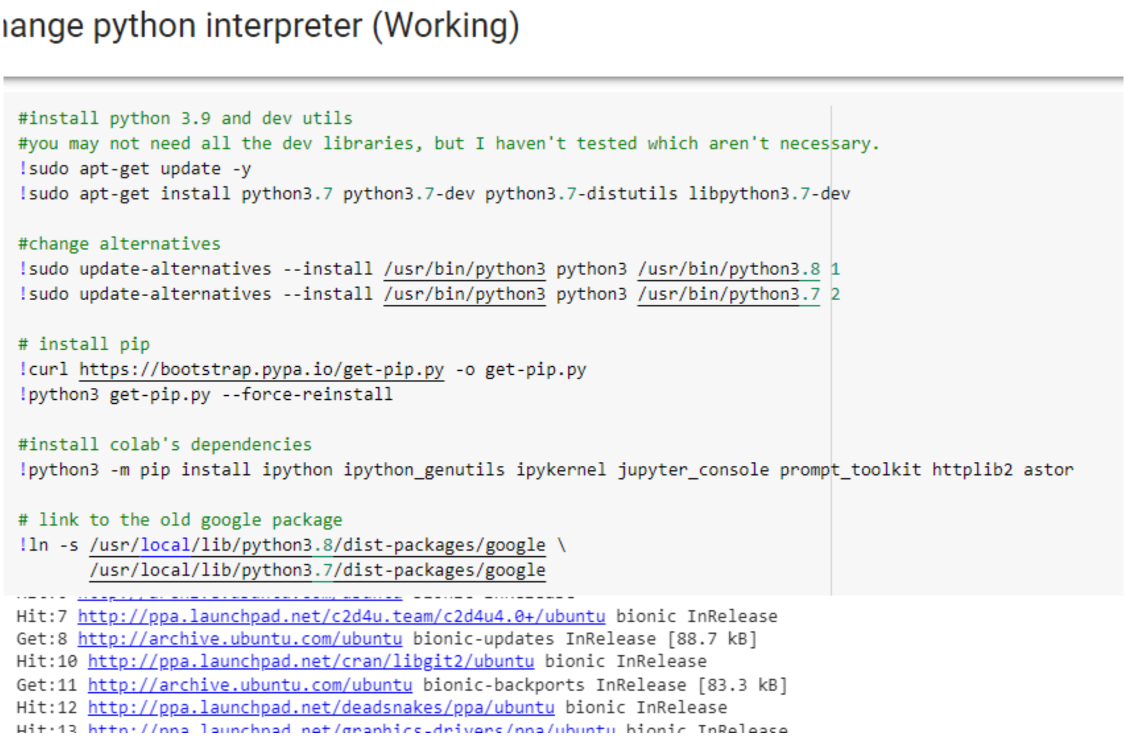


```
# Check python version
# It should be 3.7
# If it is 3.8, go to "Tools" > "Command Palette" > and search for "Use Fallback Runtime Version"
!pip --version
```

```
pip 21.1.3 from /usr/local/lib/python3.7/dist-packages/pip (python 3.7)
```

Figure 1.5: Check the python version again

- Method 2 (Made on December 17, 2022)

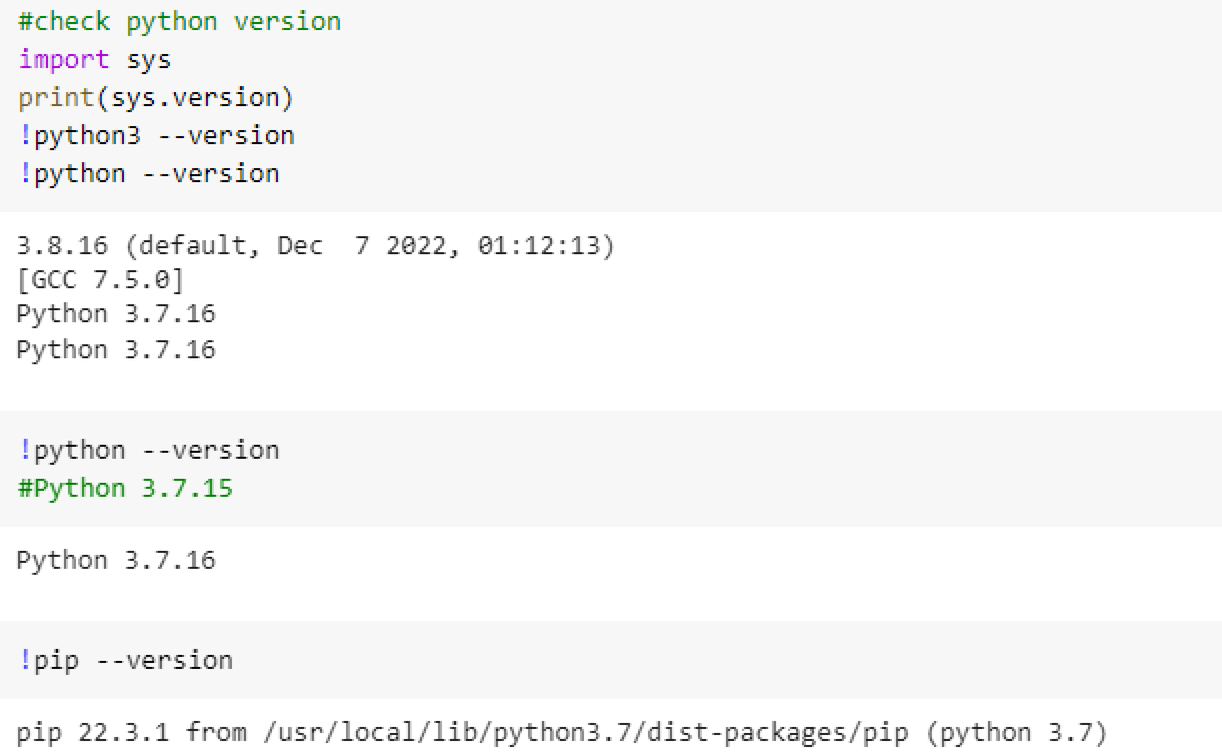


ıange python interpreter (Working)

```
#install python 3.9 and dev utils
#you may not need all the dev libraries, but I haven't tested which aren't necessary.
!sudo apt-get update -y
!sudo apt-get install python3.7 python3.7-dev python3.7-distutils libpython3.7-dev

#change alternatives
!sudo update-alternatives --install /usr/bin/python3 python3 /usr/bin/python3.8 1
!sudo update-alternatives --install /usr/bin/python3 python3 /usr/bin/python3.7 2

# install pip
!curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
!python3 get-pip.py --force-reinstall

#install colab's dependencies
!python3 -m pip install ipython ipython_genutils ipykernel jupyter_console prompt_toolkit httplib2 astor

# link to the old google package
!ln -s /usr/local/lib/python3.8/dist-packages/google \
       /usr/local/lib/python3.7/dist-packages/google
```

```
Hit:7 http://ppa.launchpad.net/c2d4u.team/c2d4u4.0+/ubuntu bionic InRelease
Get:8 http://archive.ubuntu.com/ubuntu bionic-updates InRelease [88.7 kB]
Hit:10 http://ppa.launchpad.net/cran/libgit2/ubuntu bionic InRelease
Get:11 http://archive.ubuntu.com/ubuntu bionic-backports InRelease [83.3 kB]
Hit:12 http://ppa.launchpad.net/deadsnakes/ppa/ubuntu bionic InRelease
Hit:13 http://ppa.launchpad.net/graphics-drivers/ppa/ubuntu bionic InRelease
```

Figure 1.6: Install python 3.7 and its dependencies on google colab. It will also copy google dependencies from the default runtime python 3.8



```
#check python version
import sys
print(sys.version)
!python3 --version
!python --version
```

```
3.8.16 (default, Dec  7 2022, 01:12:13)
[GCC 7.5.0]
Python 3.7.16
Python 3.7.16
```

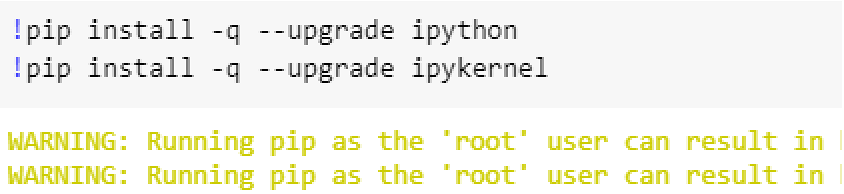
```
!python --version
#Python 3.7.15
```

```
Python 3.7.16
```

```
!pip --version
```

```
pip 22.3.1 from /usr/local/lib/python3.7/dist-packages/pip (python 3.7)
```

Figure 1.7: Check result

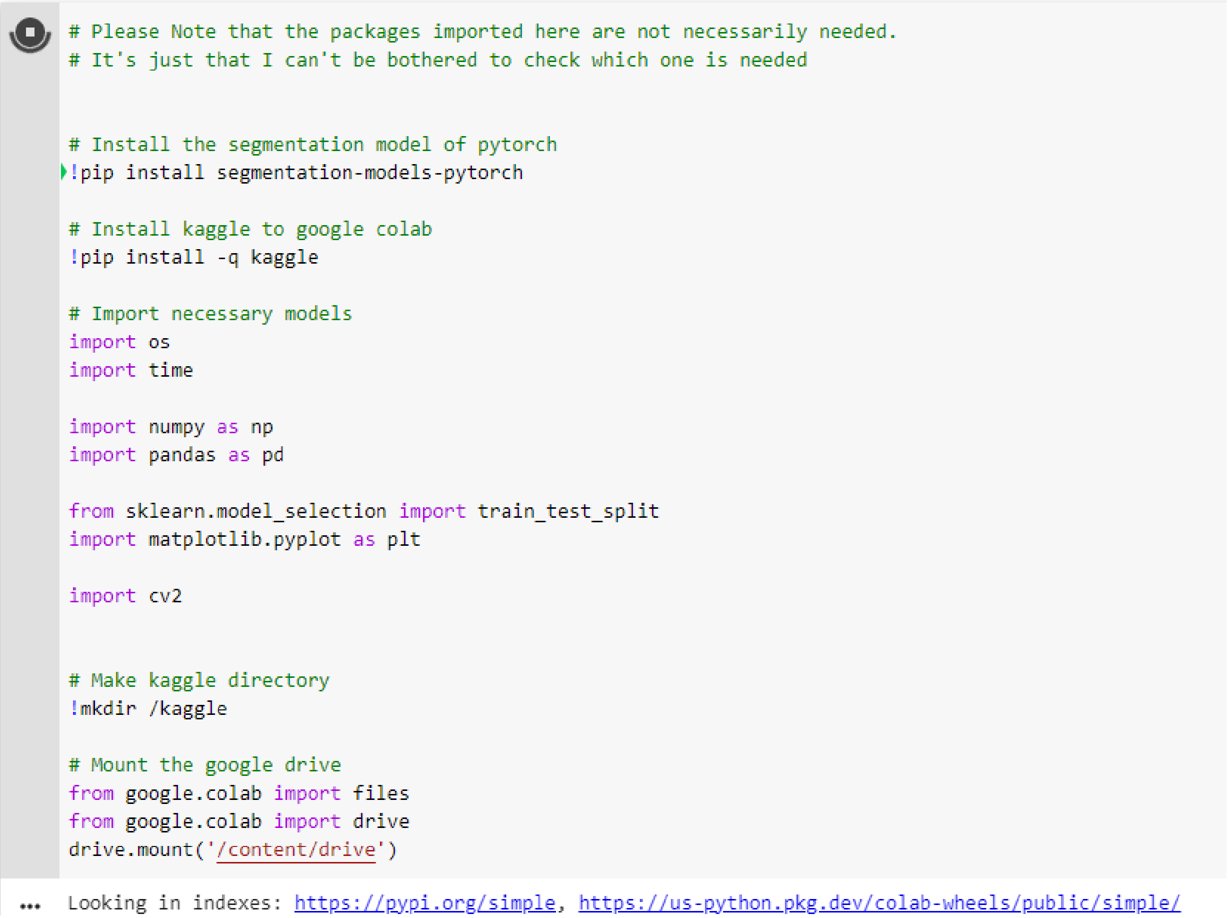


```
!pip install -q --upgrade ipython
!pip install -q --upgrade ipykernel
```

```
WARNING: Running pip as the 'root' user can result in
WARNING: Running pip as the 'root' user can result in
```

Figure 1.8: Install related packages (After running this cell, you should restart the runtime)

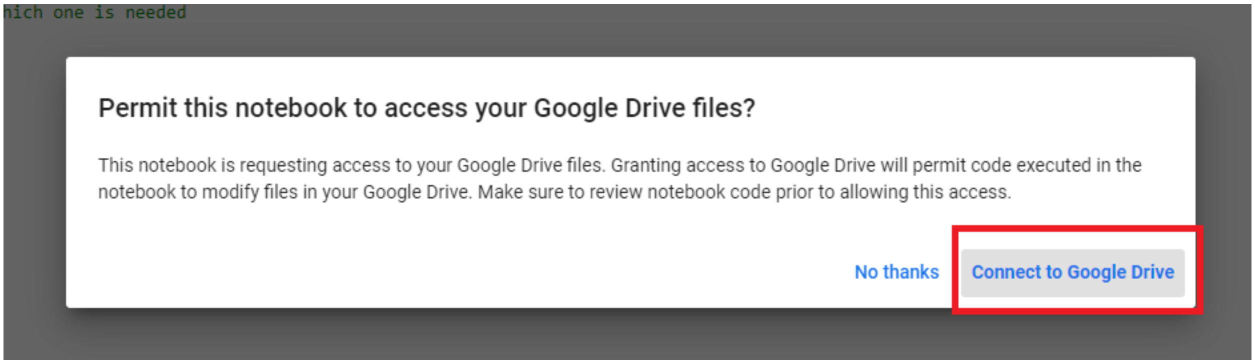
2. Our Lips Segmentation Dataset



```
# Please Note that the packages imported here are not necessarily needed.
# It's just that I can't be bothered to check which one is needed


# Install the segmentation model of pytorch
!pip install segmentation-models-pytorch

# Install kaggle to google colab
!pip install -q kaggle

# Import necessary models
import os
import time

import numpy as np
import pandas as pd

from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

import cv2


# Make kaggle directory
!mkdir /kaggle

# Mount the google drive
from google.colab import files
from google.colab import drive
drive.mount('/content/drive')
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
```

Figure 2.1: Run the First Cell

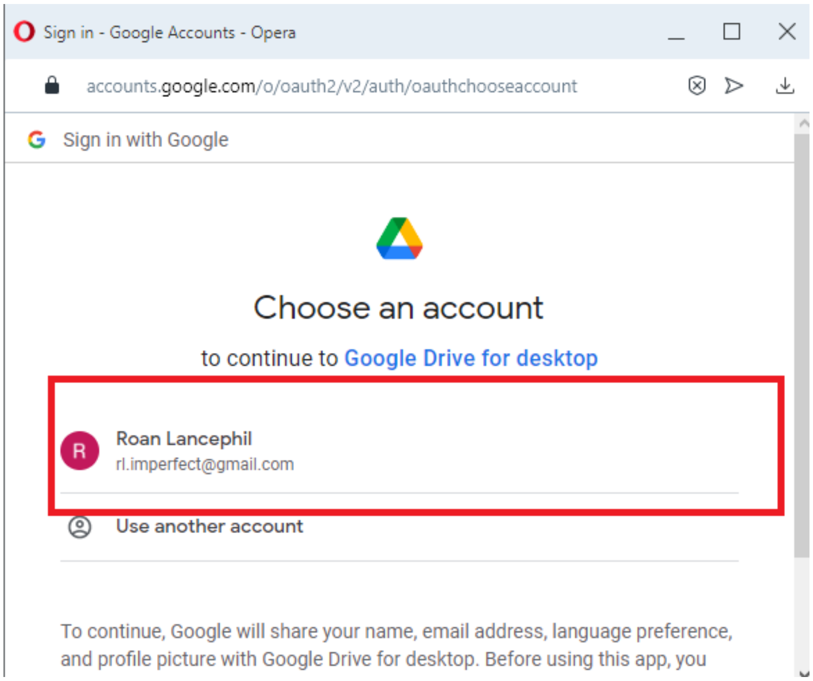


hich one is needed

Permit this notebook to access your Google Drive files?

This notebook is requesting access to your Google Drive files. Granting access to Google Drive will permit code executed in the notebook to modify files in your Google Drive. Make sure to review notebook code prior to allowing this access.

Figure 2.2: Connect to Google Drive


Sign in - Google Accounts - Opera
accounts.google.com/o/oauth2/v2/auth/oauthchooseaccount
Sign in with Google
R
Use another account

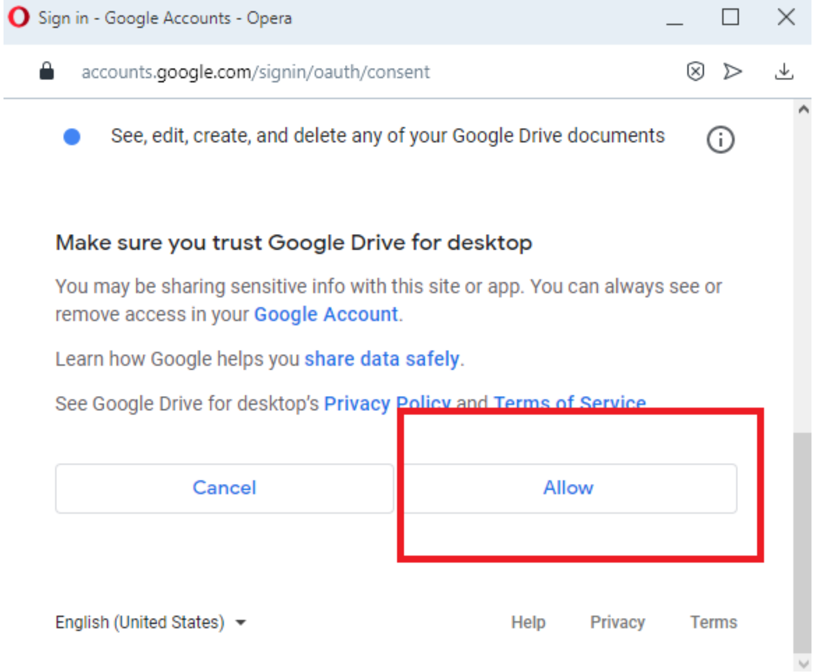
Choose an account

to continue to Google Drive for desktop

Roan Lancephil
rl.imperfect@gmail.com

To continue, Google will share your name, email address, language preference, and profile picture with Google Drive for desktop. Before using this app, you

Figure 2.3: Choose a Google Account


Sign in - Google Accounts - Opera
accounts.google.com/signin/oauth/consent
Cancel
Allow
English (United States)
Help
Privacy
Terms

See, edit, create, and delete any of your Google Drive documents

Make sure you trust Google Drive for desktop

You may be sharing sensitive info with this site or app. You can always see or remove access in your Google Account.

Learn how Google helps you share data safely.

See Google Drive for desktop's Privacy Policy and Terms of Service.

Figure 2.4: Grant Access

3. If you don’t have the Kaggle.json (Optional). Note: You should first create your own Kaggle Account.

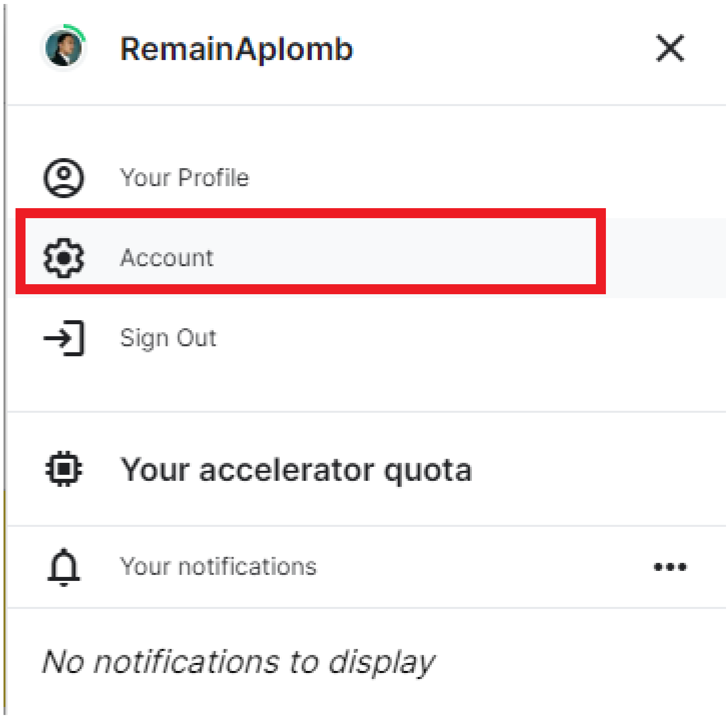
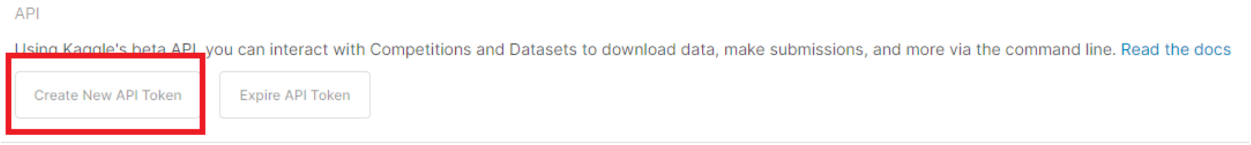


Figure 3.1: Go to Account


API

Using Kaggle's beta API, you can interact with Competitions and Datasets to download data, make submissions, and more via the command line. Read the docs

Figure 3.2: Scroll down and choose Create New API Token

4. Download Datasets from Kaggle

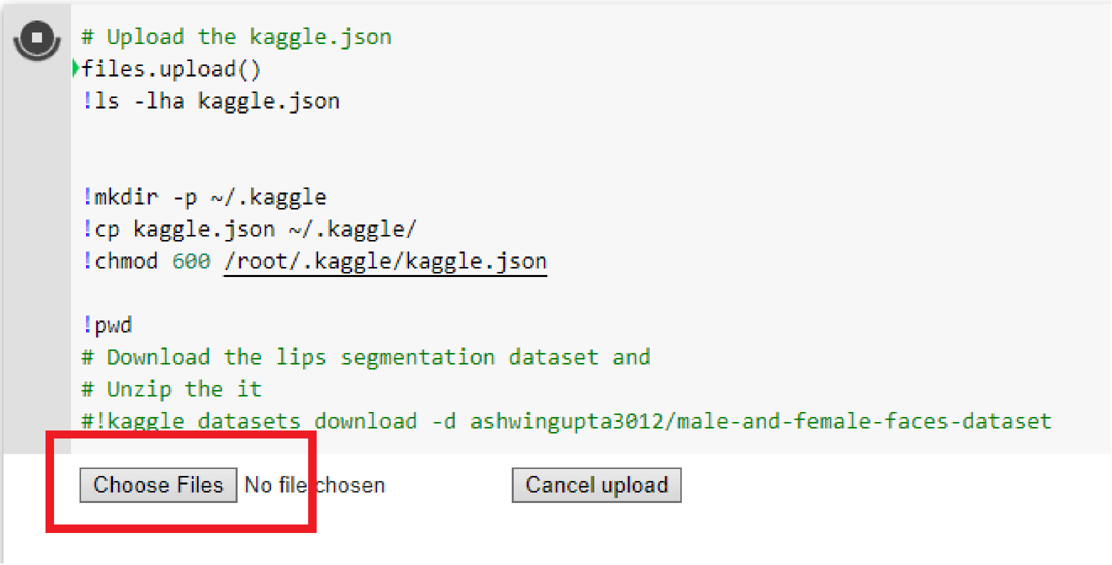
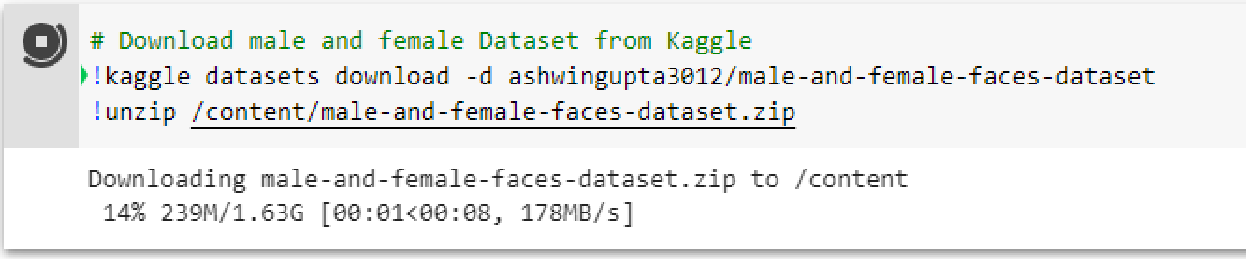

```
# Upload the kaggle.json
files.upload()
!ls -lha kaggle.json


!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 /root/.kaggle/kaggle.json

!pwd
# Download the lips segmentation dataset and
# Unzip the it
#!kaggle datasets download -d ashwingupta3012/male-and-female-faces-dataset
```

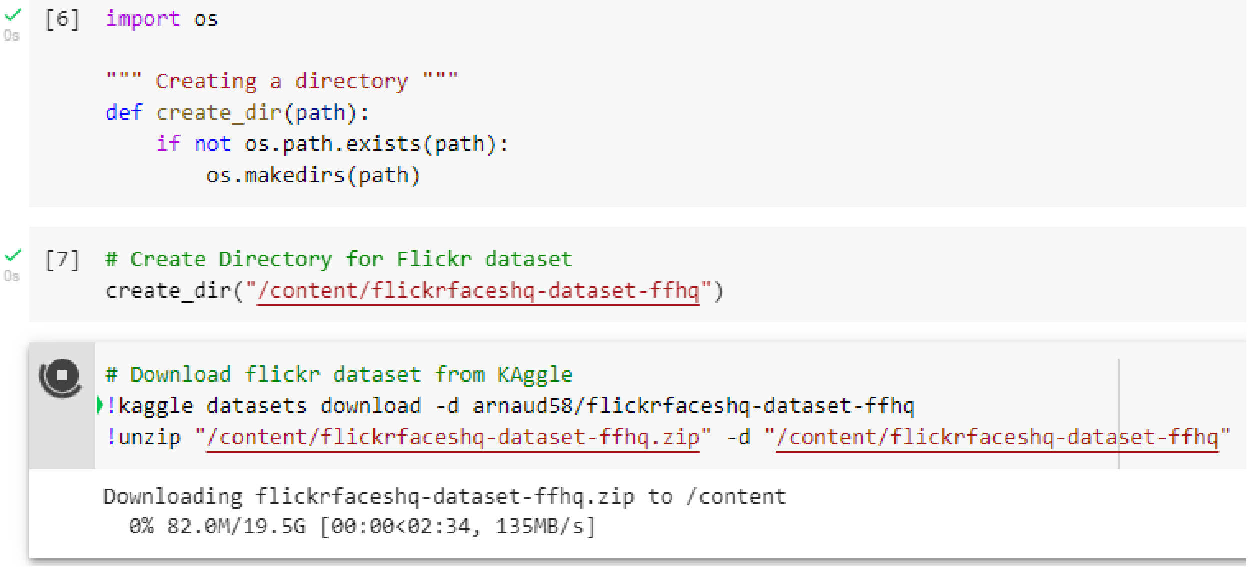
Figure 4.1: Upload your Kaggle.json



```
# Download male and female Dataset from Kaggle
!kaggle datasets download -d ashwingupta3012/male-and-female-faces-dataset
!unzip /content/male-and-female-faces-dataset.zip
```

```
Downloading male-and-female-faces-dataset.zip to /content
 14% 239M/1.63G [00:01<00:08, 178MB/s]
```

Figure 4.2: Download and extract male and female faces dataset



```
[6]  import os

     """ Creating a directory """
     def create_dir(path):
         if not os.path.exists(path):
             os.makedirs(path)
```

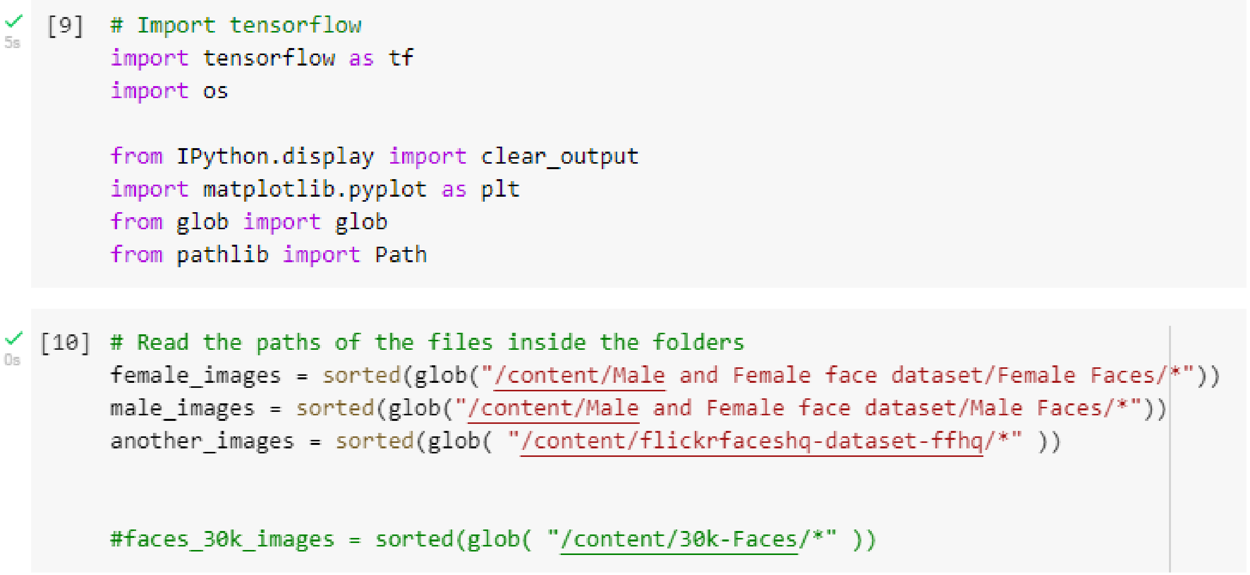
```
[7]  # Create Directory for Flickr dataset
     create_dir("/content/flickrfaceshq-dataset-ffhq")
```

```
# Download flickr dataset from KAggle
!kaggle datasets download -d arnaud58/flickrfaceshq-dataset-ffhq
!unzip "/content/flickrfaceshq-dataset-ffhq.zip" -d "/content/flickrfaceshq-dataset-ffhq"
```

```
Downloading flickrfaceshq-dataset-ffhq.zip to /content
  0% 82.0M/19.5G [00:00<02:34, 135MB/s]
```

Figure 4.3: Create directory, download, and extract flickr dataset

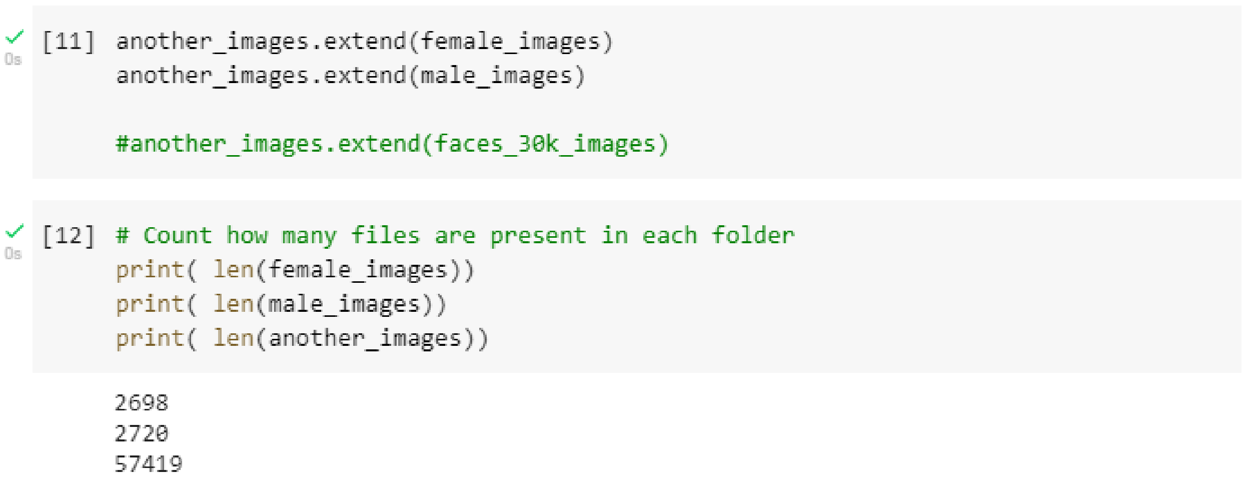
5. Initialize the image paths of our dataset



```
[9]  # Import tensorflow
     import tensorflow as tf
     import os

     from IPython.display import clear_output
     import matplotlib.pyplot as plt
     from glob import glob
     from pathlib import Path
```

```
[10] # Read the paths of the files inside the folders
     female_images = sorted(glob("/content/Male and Female face dataset/Female Faces/*"))
     male_images = sorted(glob("/content/Male and Female face dataset/Male Faces/*"))
     another_images = sorted(glob( "/content/flickrfaceshq-dataset-ffhq/*" ))


     #faces_30k_images = sorted(glob( "/content/30k-Faces/*" ))
```

Figure 5.1: Initialize paths

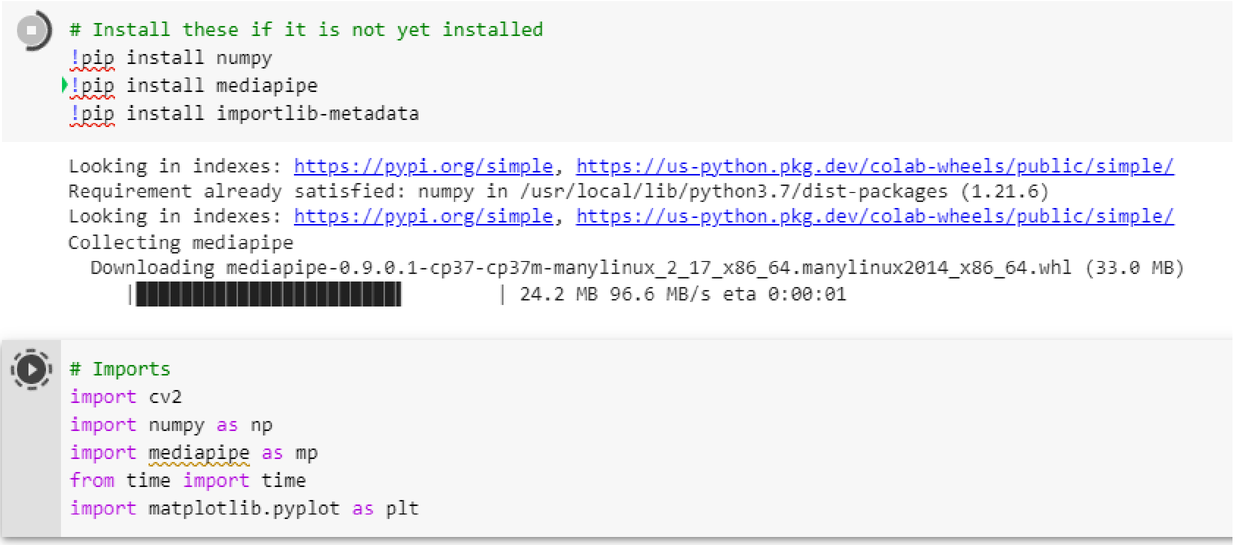


```
[11] another_images.extend(female_images)
     another_images.extend(male_images)

     #another_images.extend(faces_30k_images)
```

```
[12] # Count how many files are present in each folder
     print( len(female_images))
     print( len(male_images))
     print( len(another_images))

     2698
     2720
     57419
```

Figure 5.2: Add the image paths of female and male faces into the another_images list.

6. Install, Initialize, and Import necessary packages and functions

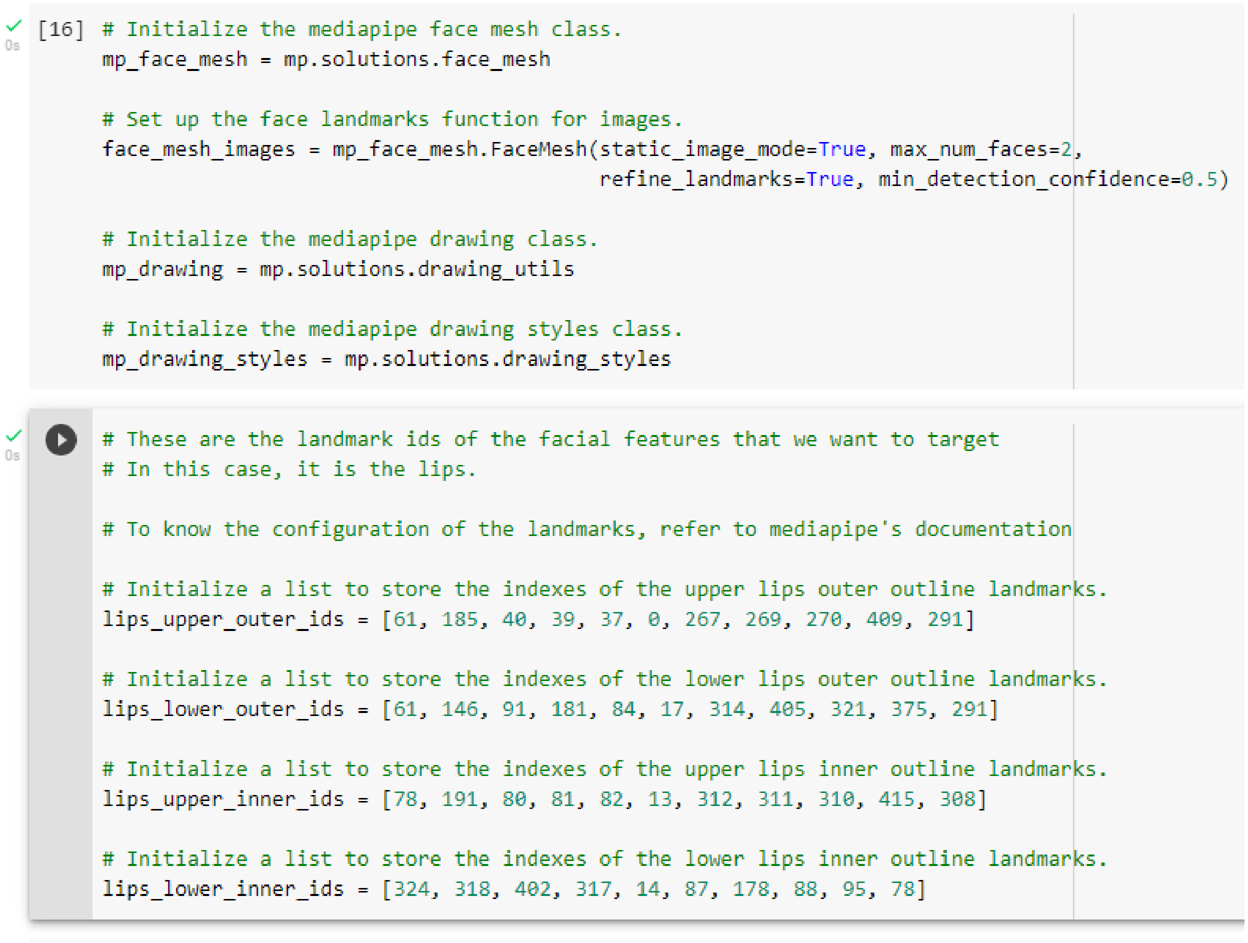


```
# Install these if it is not yet installed
!pip install numpy
!pip install mediapipe
!pip install importlib-metadata
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (1.21.6)
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting mediapipe
  Downloading mediapipe-0.9.0.1-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (33.0 MB)
     |████████████████████████        | 24.2 MB 96.6 MB/s eta 0:00:01
```

```
# Imports
import cv2
import numpy as np
import mediapipe as mp
from time import time
import matplotlib.pyplot as plt
```

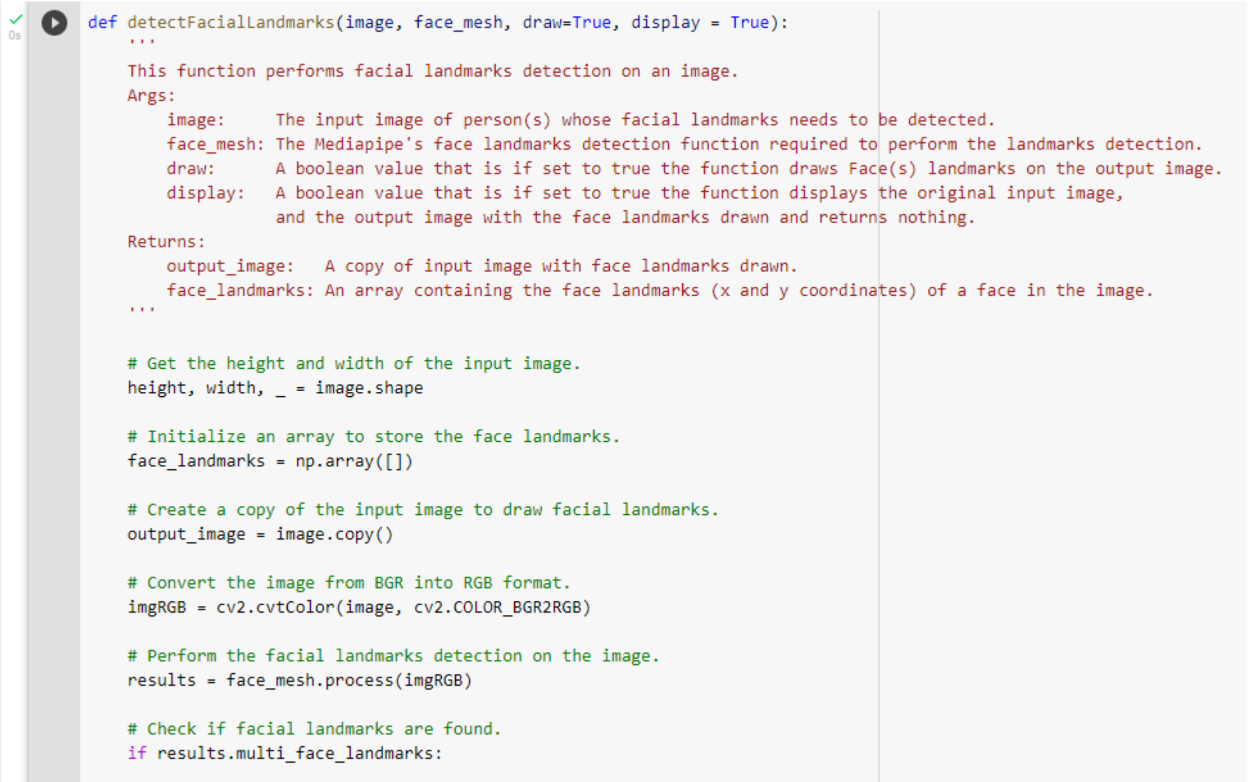
Figure 6.1: Install and Import necessary packages



```
# Initialize the mediapipe face mesh class.
mp_face_mesh = mp.solutions.face_mesh

# Set up the face landmarks function for images.
face_mesh_images = mp_face_mesh.FaceMesh(static_image_mode=True, max_num_faces=2,
                                         refine_landmarks=True, min_detection_confidence=0.5)

# Initialize the mediapipe drawing class.
mp_drawing = mp.solutions.drawing_utils

# Initialize the mediapipe drawing styles class.
mp_drawing_styles = mp.solutions.drawing_styles
```

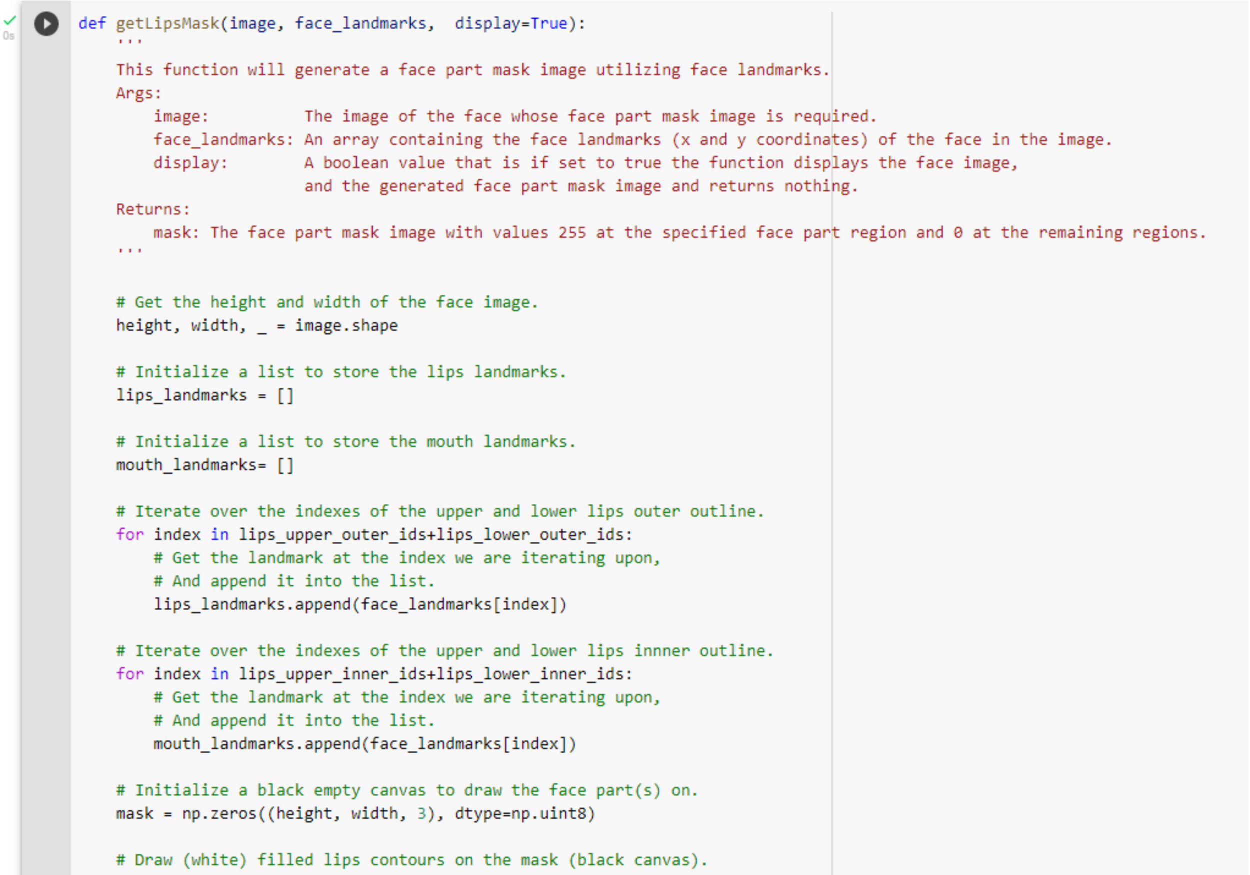
```
# These are the landmark ids of the facial features that we want to target
# In this case, it is the lips.

# To know the configuration of the landmarks, refer to mediapipe's documentation

# Initialize a list to store the indexes of the upper lips outer outline landmarks.
lips_upper_outer_ids = [61, 185, 40, 39, 37, 0, 267, 269, 270, 409, 291]

# Initialize a list to store the indexes of the lower lips outer outline landmarks.
lips_lower_outer_ids = [61, 146, 91, 181, 84, 17, 314, 405, 321, 375, 291]

# Initialize a list to store the indexes of the upper lips inner outline landmarks.
lips_upper_inner_ids = [78, 191, 80, 81, 82, 13, 312, 311, 310, 415, 308]

# Initialize a list to store the indexes of the lower lips inner outline landmarks.
lips_lower_inner_ids = [324, 318, 402, 317, 14, 87, 178, 88, 95, 78]
```

Figure 6.2: Initialize necessary functions and variables



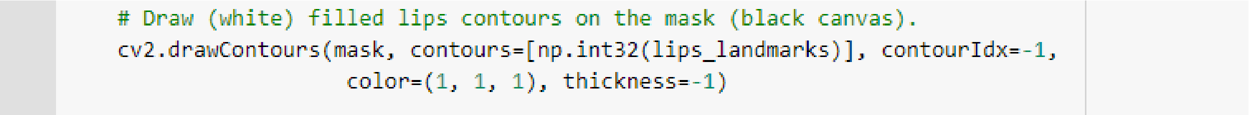
```
def detectFacialLandmarks(image, face_mesh, draw=True, display = True):
    '''
    This function performs facial landmarks detection on an image.
    Args:
        image:     The input image of person(s) whose facial landmarks needs to be detected.
        face_mesh: The Mediapipe's face landmarks detection function required to perform the landmarks detection.
        draw:      A boolean value that is if set to true the function draws Face(s) landmarks on the output image.
        display:   A boolean value that is if set to true the function displays the original input image,
                   and the output image with the face landmarks drawn and returns nothing.
    Returns:
        output_image:   A copy of input image with face landmarks drawn.
        face_landmarks: An array containing the face landmarks (x and y coordinates) of a face in the image.
    '''

    # Get the height and width of the input image.
    height, width, _ = image.shape

    # Initialize an array to store the face landmarks.
    face_landmarks = np.array([])

    # Create a copy of the input image to draw facial landmarks.
    output_image = image.copy()

    # Convert the image from BGR into RGB format.
    imgRGB = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    # Perform the facial landmarks detection on the image.
    results = face_mesh.process(imgRGB)

    # Check if facial landmarks are found.
    if results.multi_face_landmarks:
```

Figure 6.3: Initialize necessary functions



```
def getLipsMask(image, face_landmarks,  display=True):
    '''
    This function will generate a face part mask image utilizing face landmarks.
    Args:
        image:          The image of the face whose face part mask image is required.
        face_landmarks: An array containing the face landmarks (x and y coordinates) of the face in the image.
        display:        A boolean value that is if set to true the function displays the face image,
                        and the generated face part mask image and returns nothing.
    Returns:
        mask: The face part mask image with values 255 at the specified face part region and 0 at the remaining regions.
    '''

    # Get the height and width of the face image.
    height, width, _ = image.shape

    # Initialize a list to store the lips landmarks.
    lips_landmarks = []

    # Initialize a list to store the mouth landmarks.
    mouth_landmarks= []

    # Iterate over the indexes of the upper and lower lips outer outline.
    for index in lips_upper_outer_ids+lips_lower_outer_ids:
        # Get the landmark at the index we are iterating upon,
        # And append it into the list.
        lips_landmarks.append(face_landmarks[index])

    # Iterate over the indexes of the upper and lower lips innner outline.
    for index in lips_upper_inner_ids+lips_lower_inner_ids:
        # Get the landmark at the index we are iterating upon,
        # And append it into the list.
        mouth_landmarks.append(face_landmarks[index])

    # Initialize a black empty canvas to draw the face part(s) on.
    mask = np.zeros((height, width, 3), dtype=np.uint8)

    # Draw (white) filled lips contours on the mask (black canvas).
```

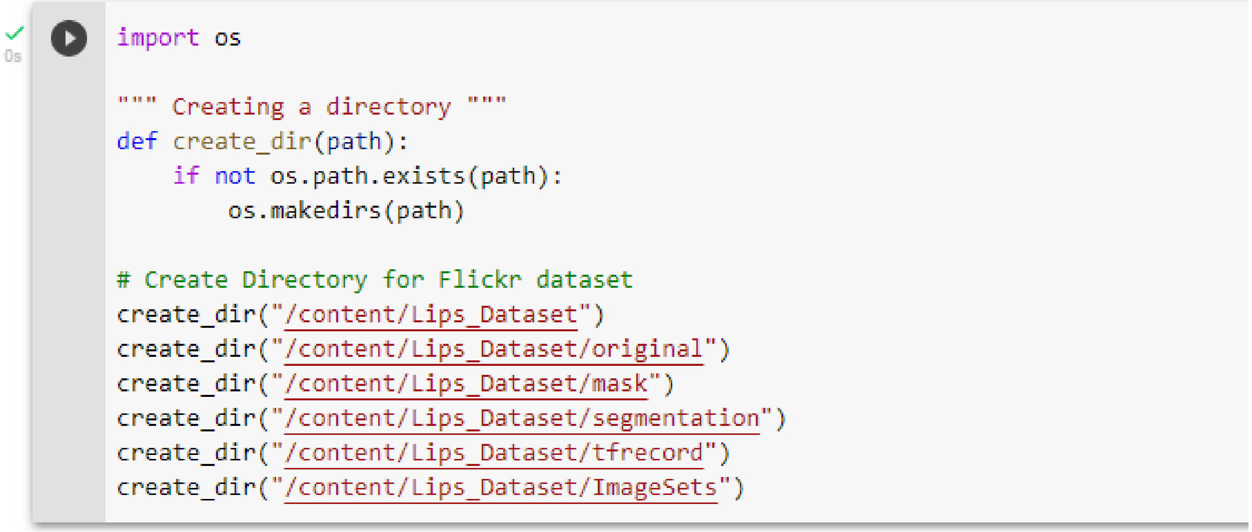
Figure 6.4: Initialize necessary functions



```
    # Draw (white) filled lips contours on the mask (black canvas).
    cv2.drawContours(mask, contours=[np.int32(lips_landmarks)], contourIdx=-1,
                     color=(1, 1, 1), thickness=-1)
```

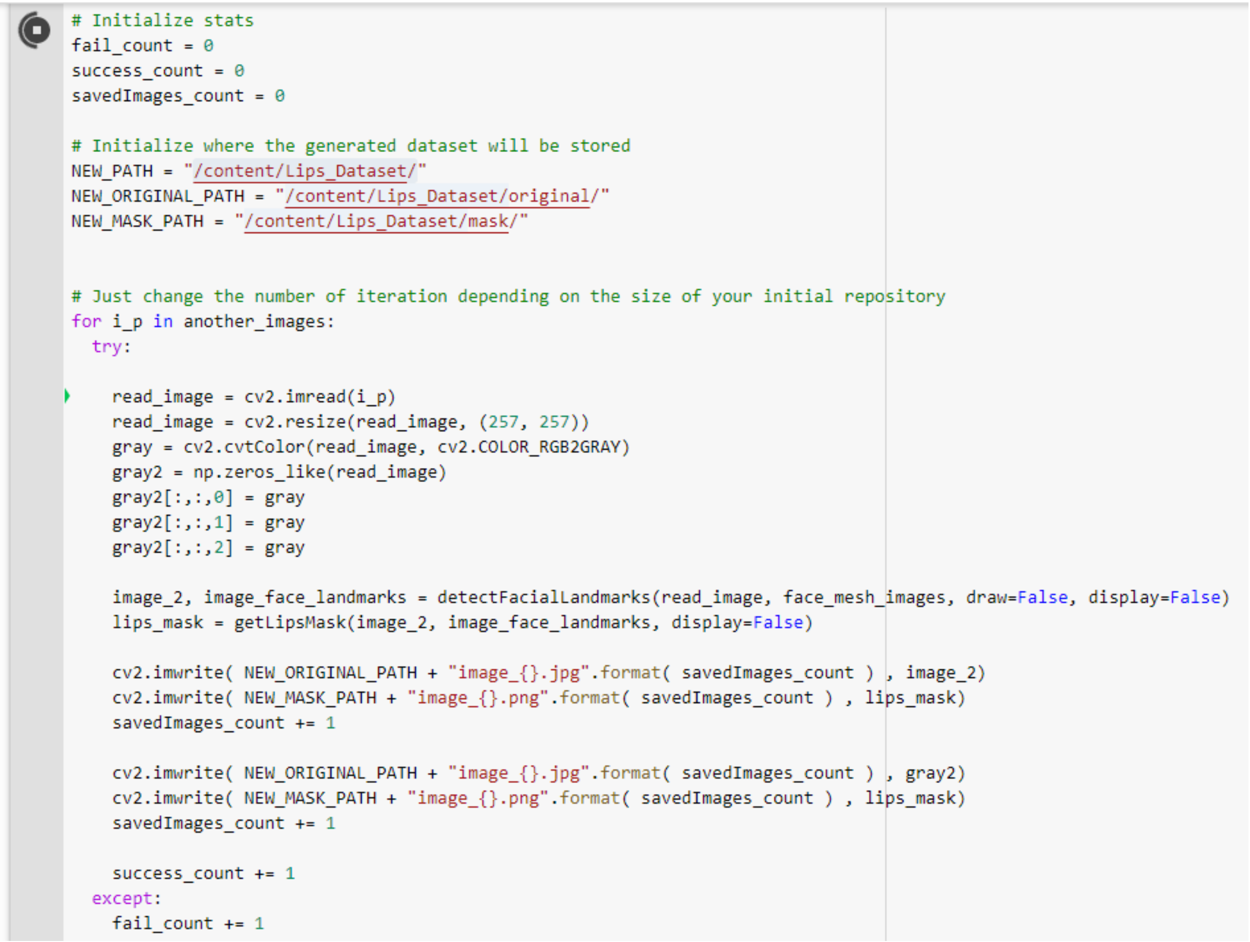
Figure 6.5: This can be changed if you want a different segmentation color (Optional/Not recommended)

7. Create Mask using Mediapipe



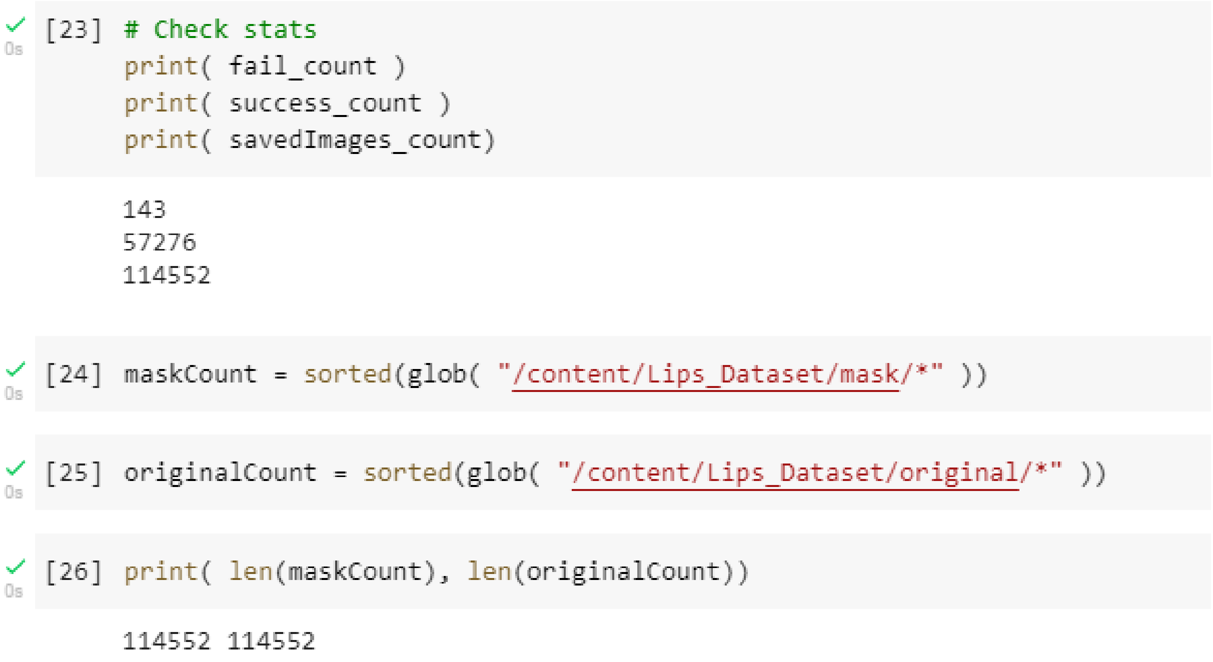
```
import os

""" Creating a directory """
def create_dir(path):
    if not os.path.exists(path):
        os.makedirs(path)

# Create Directory for Flickr dataset
create_dir("/content/Lips_Dataset")
create_dir("/content/Lips_Dataset/original")
create_dir("/content/Lips_Dataset/mask")
create_dir("/content/Lips_Dataset/segmentation")
create_dir("/content/Lips_Dataset/tfrecord")
create_dir("/content/Lips_Dataset/ImageSets")
```

Figure 7.1: Create directory for our Lips Dataset



```python
# Initialize stats
fail_count = 0
success_count = 0
savedImages_count = 0

# Initialize where the generated dataset will be stored
NEW_PATH = "/content/Lips_Dataset/"
NEW_ORIGINAL_PATH = "/content/Lips_Dataset/original/"
NEW_MASK_PATH = "/content/Lips_Dataset/mask/"


# Just change the number of iteration depending on the size of your initial repository
for i_p in another_images:
  try:

    read_image = cv2.imread(i_p)
    read_image = cv2.resize(read_image, (257, 257))
    gray = cv2.cvtColor(read_image, cv2.COLOR_RGB2GRAY)
    gray2 = np.zeros_like(read_image)
    gray2[:,:,0] = gray
    gray2[:,:,1] = gray
    gray2[:,:,2] = gray

    image_2, image_face_landmarks = detectFacialLandmarks(read_image, face_mesh_images, draw=False, display=False)
    lips_mask = getLipsMask(image_2, image_face_landmarks, display=False)

    cv2.imwrite( NEW_ORIGINAL_PATH + "image_{}.jpg".format( savedImages_count ) , image_2)
    cv2.imwrite( NEW_MASK_PATH + "image_{}.png".format( savedImages_count ) , lips_mask)
    savedImages_count += 1

    cv2.imwrite( NEW_ORIGINAL_PATH + "image_{}.jpg".format( savedImages_count ) , gray2)
    cv2.imwrite( NEW_MASK_PATH + "image_{}.png".format( savedImages_count ) , lips_mask)
    savedImages_count += 1

    success_count += 1
  except:
    fail_count += 1
```

Figure 7.2: Generate Mask



```python
[23] # Check stats
     print( fail_count )
     print( success_count )
     print( savedImages_count)
```

```
143
57276
114552
```

```python
[24] maskCount = sorted(glob( "/content/Lips_Dataset/mask/*" ))
```

```python
[25] originalCount = sorted(glob( "/content/Lips_Dataset/original/*" ))
```

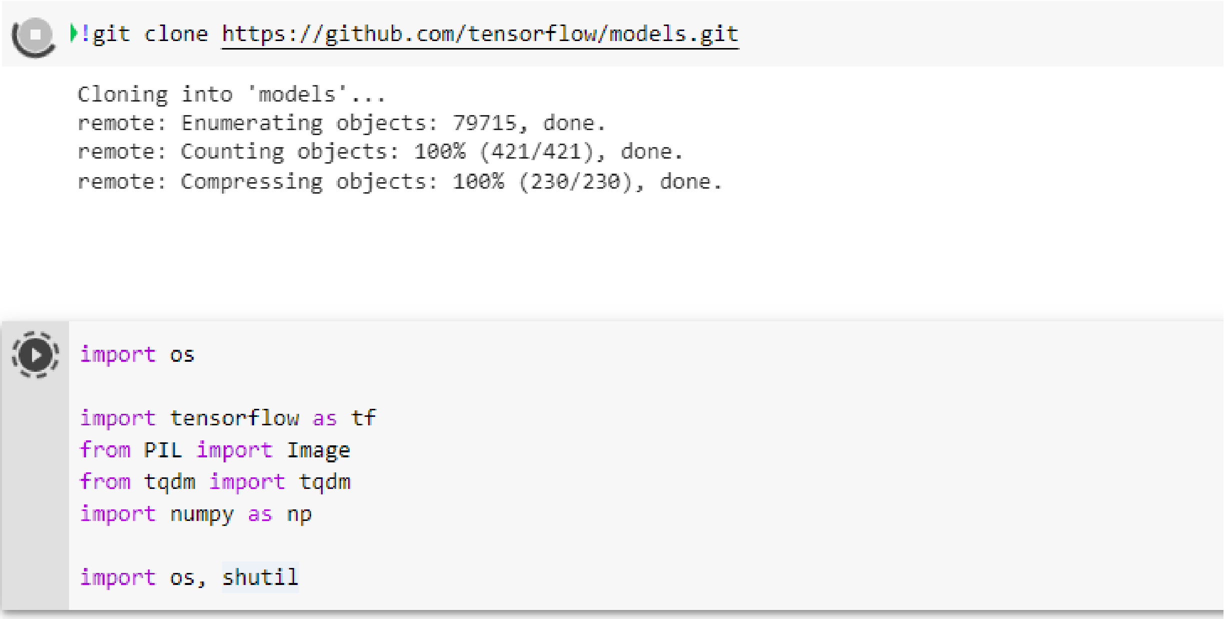
```python
[26] print( len(maskCount), len(originalCount))
```

```
114552 114552
```

Figure 7.3: Check Results. It needs to have matching number of images

8. Convert Color Segmentation

Create Tfrecord

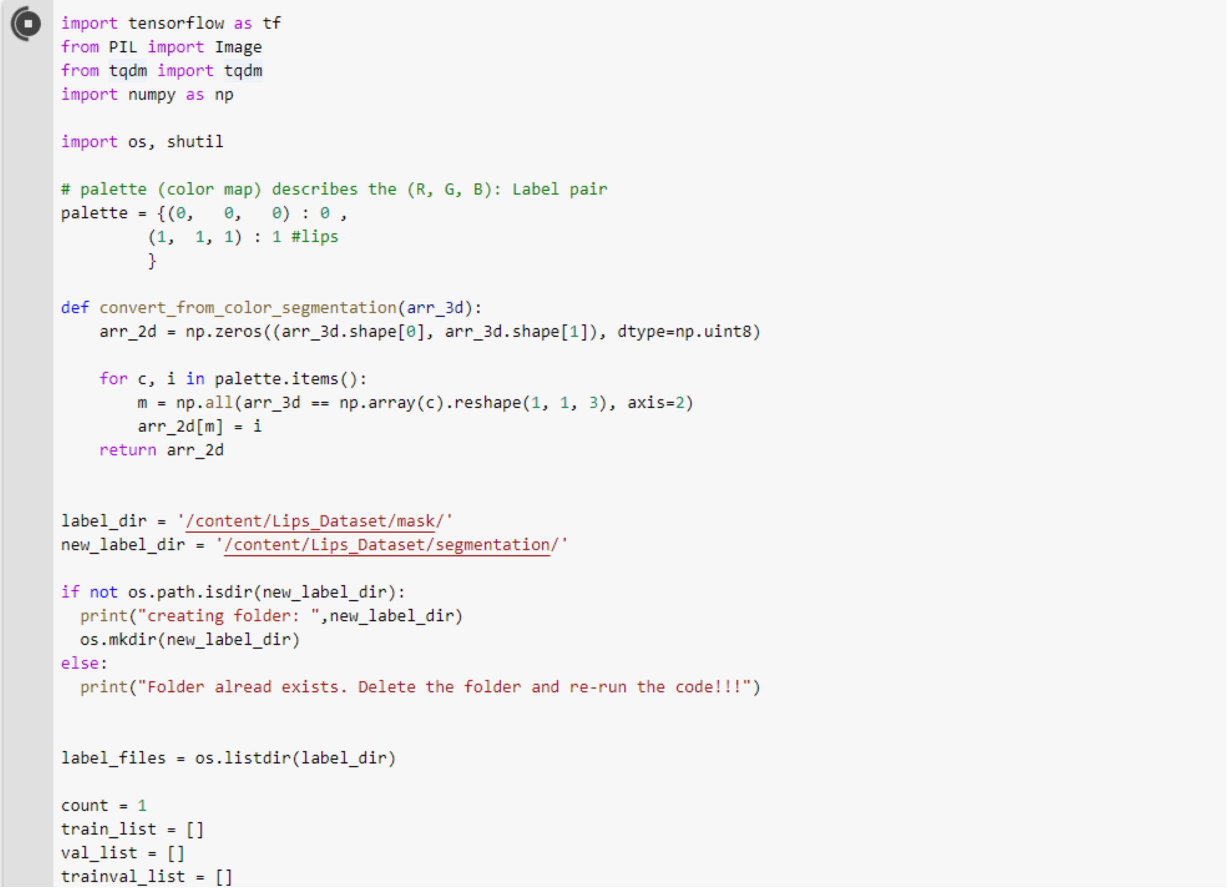


```
!git clone https://github.com/tensorflow/models.git
```

```
Cloning into 'models'...
remote: Enumerating objects: 79715, done.
remote: Counting objects: 100% (421/421), done.
remote: Compressing objects: 100% (230/230), done.
```

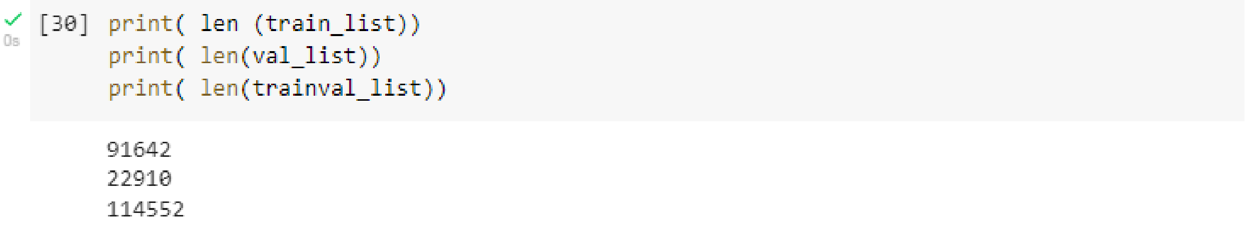
```
import os

import tensorflow as tf
from PIL import Image
from tqdm import tqdm
import numpy as np

import os, shutil
```

Figure 8.1: Clone Repository, and import necessary packages



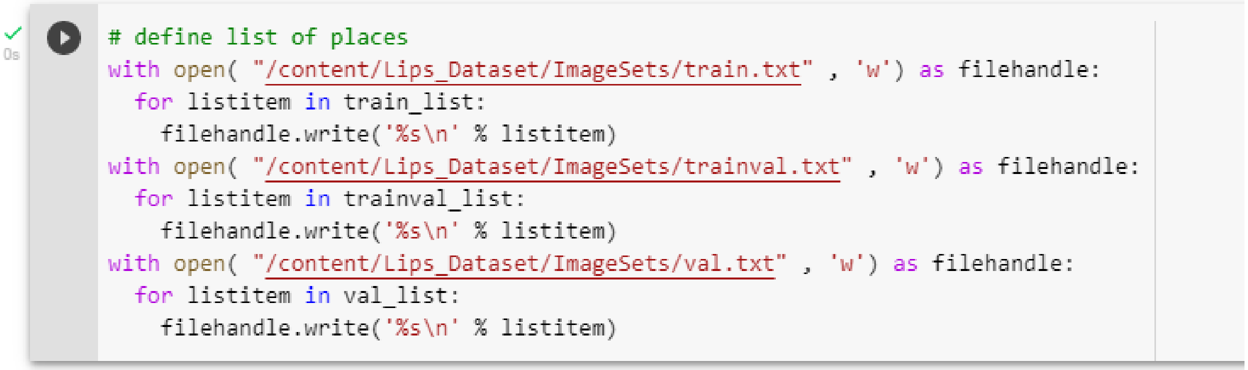
```
import tensorflow as tf
from PIL import Image
from tqdm import tqdm
import numpy as np

import os, shutil

# palette (color map) describes the (R, G, B): Label pair
palette = {(0,   0,   0) : 0 ,
         (1,  1, 1) : 1 #lips
         }

def convert_from_color_segmentation(arr_3d):
    arr_2d = np.zeros((arr_3d.shape[0], arr_3d.shape[1]), dtype=np.uint8)

    for c, i in palette.items():
        m = np.all(arr_3d == np.array(c).reshape(1, 1, 3), axis=2)
        arr_2d[m] = i
    return arr_2d


label_dir = '/content/Lips_Dataset/mask/'
new_label_dir = '/content/Lips_Dataset/segmentation/'

if not os.path.isdir(new_label_dir):
  print("creating folder: ",new_label_dir)
  os.mkdir(new_label_dir)
else:
  print("Folder alread exists. Delete the folder and re-run the code!!!")


label_files = os.listdir(label_dir)

count = 1
train_list = []
val_list = []
trainval_list = []
```

Figure 8.2: Convert color segmentation



```
[30] print( len (train_list))
     print( len(val_list))
     print( len(trainval_list))
```

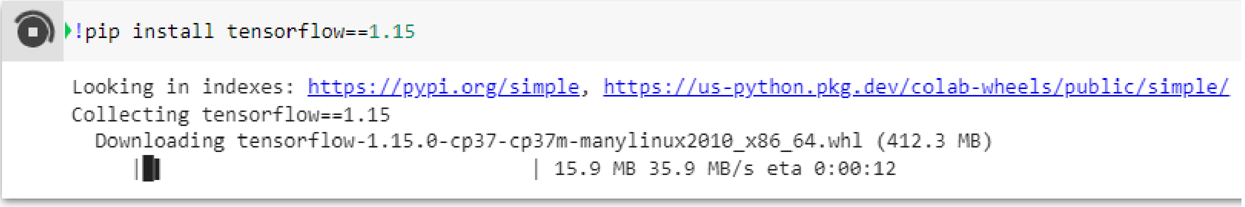
```
91642
22910
114552
```



```
# define list of places
with open( "/content/Lips_Dataset/ImageSets/train.txt" , 'w') as filehandle:
  for listitem in train_list:
    filehandle.write('%s\n' % listitem)
with open( "/content/Lips_Dataset/ImageSets/trainval.txt" , 'w') as filehandle:
  for listitem in trainval_list:
    filehandle.write('%s\n' % listitem)
with open( "/content/Lips_Dataset/ImageSets/val.txt" , 'w') as filehandle:
  for listitem in val_list:
    filehandle.write('%s\n' % listitem)
```

Figure 8.3: Export the lists containing the dataset split

9. Create Tfrecord



```
!pip install tensorflow==1.15
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting tensorflow==1.15
  Downloading tensorflow-1.15.0-cp37-cp37m-manylinux2010_x86_64.whl (412.3 MB)
     |█                               | 15.9 MB 35.9 MB/s eta 0:00:12
```

Figure 9.1: Install tensorflow 1.15

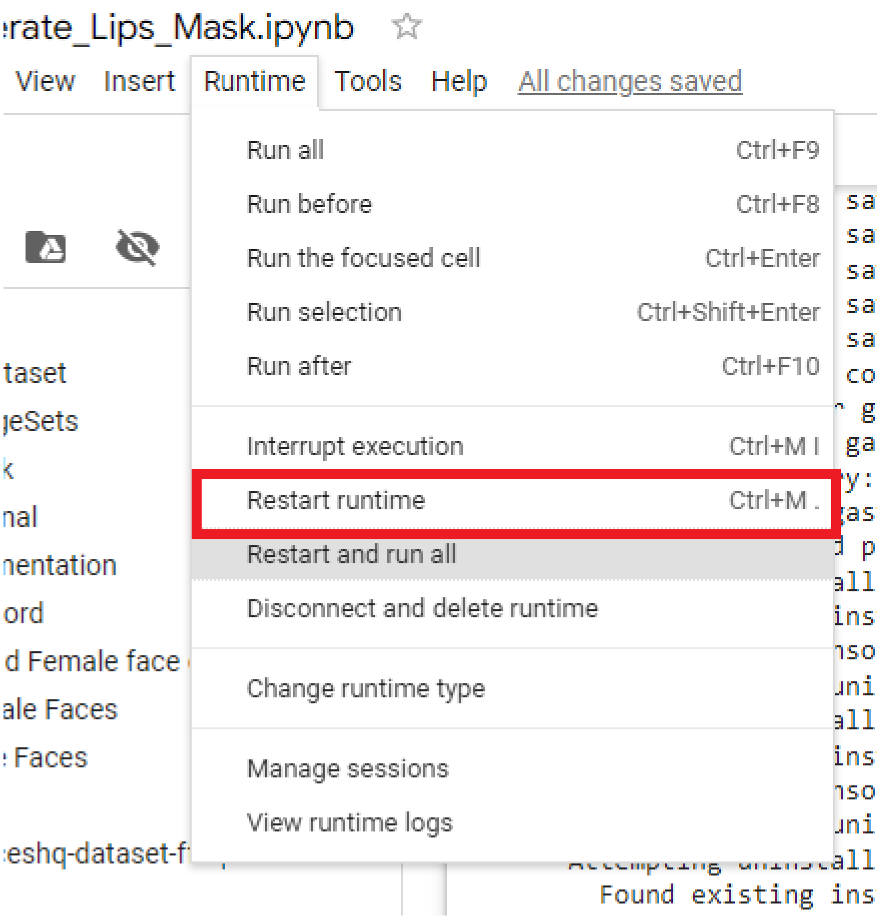
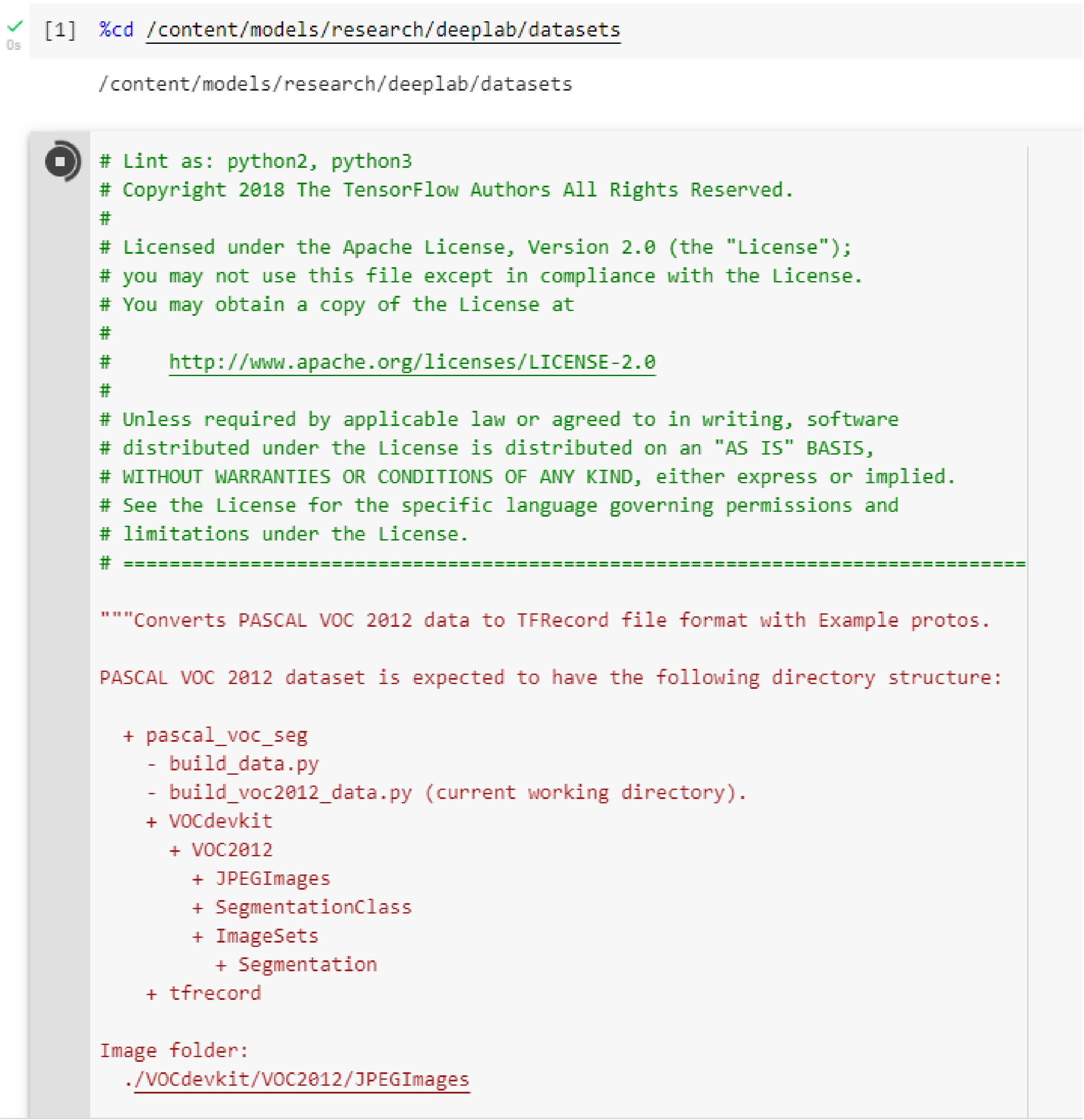


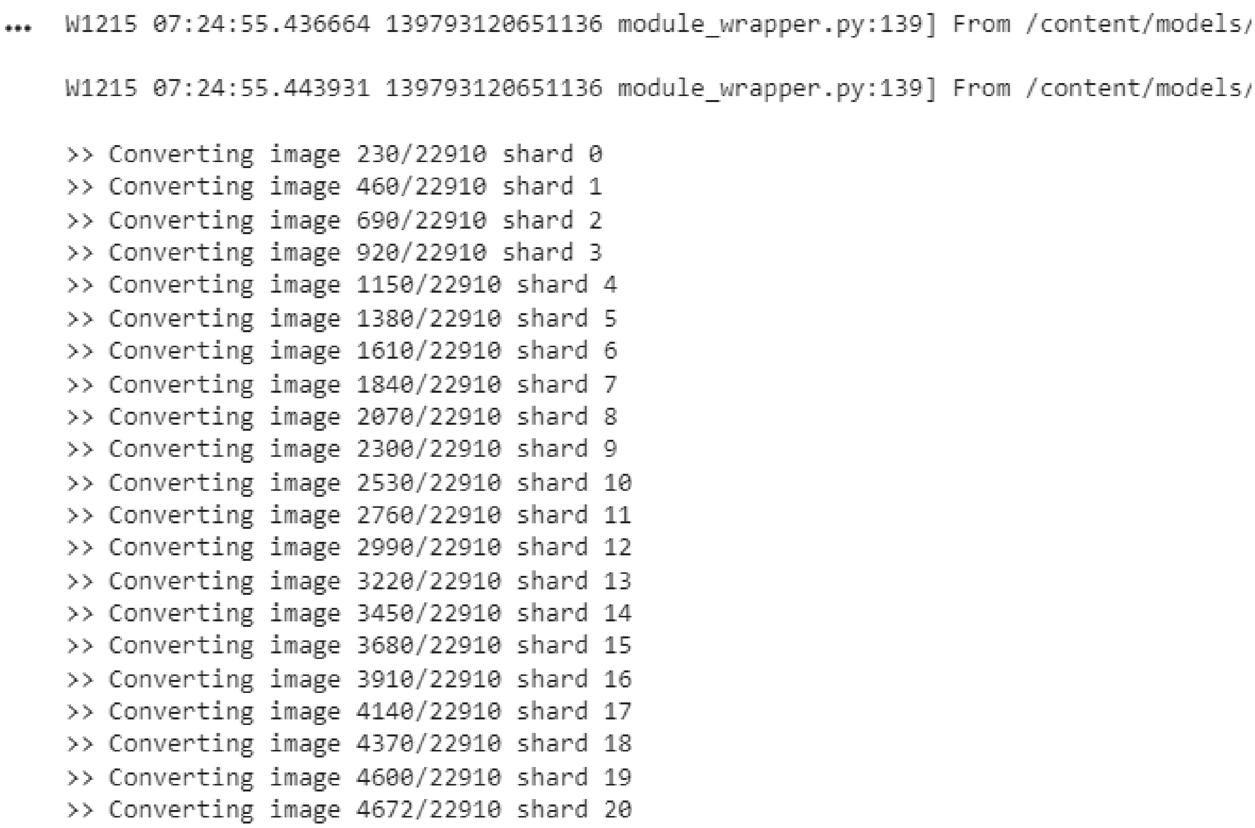
Figure 9.2: Restart runtime after installing tensorflow 1.15



```
[1] %cd /content/models/research/deeplab/datasets
```

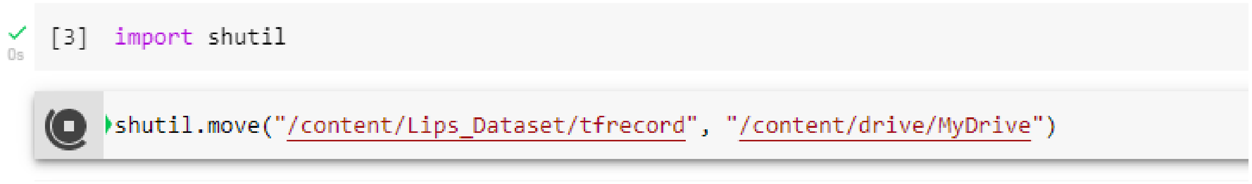
```
/content/models/research/deeplab/datasets
```

```
# Lint as: python2, python3
# Copyright 2018 The TensorFlow Authors All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
# ==============================================================================

"""Converts PASCAL VOC 2012 data to TFRecord file format with Example protos.

PASCAL VOC 2012 dataset is expected to have the following directory structure:

  + pascal_voc_seg
    - build_data.py
    - build_voc2012_data.py (current working directory).
    + VOCdevkit
      + VOC2012
        + JPEGImages
        + SegmentationClass
        + ImageSets
          + Segmentation
    + tfrecord

Image folder:
  ./VOCdevkit/VOC2012/JPEGImages
```



```
W1215 07:24:55.436664 139793120651136 module_wrapper.py:139] From /content/models/

W1215 07:24:55.443931 139793120651136 module_wrapper.py:139] From /content/models/

>> Converting image 230/22910 shard 0
>> Converting image 460/22910 shard 1
>> Converting image 690/22910 shard 2
>> Converting image 920/22910 shard 3
>> Converting image 1150/22910 shard 4
>> Converting image 1380/22910 shard 5
>> Converting image 1610/22910 shard 6
>> Converting image 1840/22910 shard 7
>> Converting image 2070/22910 shard 8
>> Converting image 2300/22910 shard 9
>> Converting image 2530/22910 shard 10
>> Converting image 2760/22910 shard 11
>> Converting image 2990/22910 shard 12
>> Converting image 3220/22910 shard 13
>> Converting image 3450/22910 shard 14
>> Converting image 3680/22910 shard 15
>> Converting image 3910/22910 shard 16
>> Converting image 4140/22910 shard 17
>> Converting image 4370/22910 shard 18
>> Converting image 4600/22910 shard 19
>> Converting image 4672/22910 shard 20
```

Figure 9.3: Create tfrecord shards



```
[3] import shutil
```

```
shutil.move("/content/Lips_Dataset/tfrecord", "/content/drive/MyDrive")
```

Figure 9.4: Move the tfrecords in your Google Drive

10. s