Contents

Activity 1 – Pose Estimation with OpenPose

In this activity, we will learn:

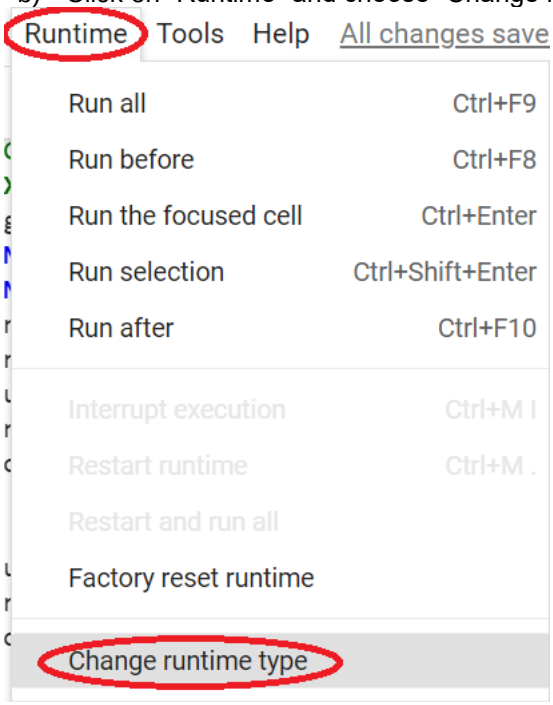
- ❑ How to setup Google Colab
- ❑ How to install OpenPose
- ❑ How to use the OpenPose to estimate pose on a test video
- ❑ Visualise the result of the pose estimation on Google Colab

Reference: https://github.com/CMU-Perceptual-Computing-Lab/openpose.git

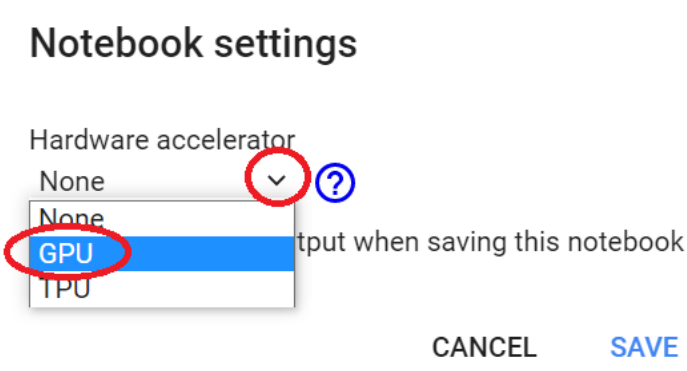
1. Setting up of Google Colab.

a) Go to https://colab.research.google.com/ and start a new notebook.

b) Click on “Runtime” and choose “Change runtime type”:



c) Click on the inverted carrot symbol and choose “GPU”:



2. Installation of OpenPose

a) Importing the os library and preparing the name and URL of the git repository to be downloaded.

```
01  import os
02  from os.path import exists, join, basename, splitext
03
04  git_repo_url = 'https://github.com/CMU-Perceptual-Computing-Lab/openpose.git'
05  project_name = splitext(basename(git_repo_url))[0]
```

b) Checking if the project exist and downloading, unzipping the CMake library for installation. We then clone the repository onto the temporary storage on google colab followed by changing the path of the libraries in the CMakeLists file to the local colab path for installation later.

```
if not exists(project_name):
  # see: https://github.com/CMU-Perceptual-Computing-Lab/openpose/issues/949
  # install new CMake becaue of CUDA10
  !wget -q https://cmake.org/files/v3.13/cmake-3.13.0-Linux-x86_64.tar.gz
  !tar xfz cmake-3.13.0-Linux-x86_64.tar.gz --strip-components=1 -C /usr/local
  # clone openpose
  !git clone -q --depth 1 $git_repo_url
  !sed -
i 's/execute_process(COMMAND git checkout master WORKING_DIRECTORY ${CMAKE_SOURCE_DIR}\/3
rdparty\/caffe)/execute_process(COMMAND git checkout f019d0dfe86f49d1140961f8c7dec22130c8
3154 WORKING_DIRECTORY ${CMAKE_SOURCE_DIR}\/3rdparty\/caffe)/g' openpose/CMakeLists.txt
```

c) Install all the necessary libraries that are required by OpenPose. The library youtube-dl is not being used by OpenPose but it is just a tool for us to grab a test video for testing purpose. You can also upload your own video for this part of the work.

```
  # install system dependencies
  !apt-get -qq install -y libatlas-base-dev libprotobuf-dev libleveldb-dev libsnappy-
dev libhdf5-serial-dev protobuf-compiler libgflags-dev libgoogle-glog-dev liblmdb-
dev opencl-headers ocl-icd-opencl-dev libviennacl-dev
  # install python dependencies
  !pip install -q youtube-dl
  # build openpose
  !cd openpose && rm -rf build || true && mkdir build && cd build && cmake .. && make -
j`nproc`

from IPython.display import YouTubeVideo
```

(*The installation may take about 15minutes so sit back and relax or now.)

d) Set the ID of a YouTube video and display using the library that we had imported earlier just for preview purpose.

```
YOUTUBE_ID = '23QOy9Q2qNI'
YouTubeVideo(YOUTUBE_ID)
```

e) Parse the first five seconds of the video from YouTube into OpenPose for pose detection before we convert it from avi video format to mp4 video format.

```
!rm -rf youtube.mp4
!youtube-dl -f 'bestvideo[ext=mp4]' --
output "youtube.%(ext)s" https://www.youtube.com/watch?v=$YOUTUBE_ID
!ffmpeg -y -loglevel info -i youtube.mp4 -t 5 video.mp4
# detect poses on the these 5 seconds
!rm openpose.avi
!cd openpose && ./build/examples/openpose/openpose.bin --video ../video.mp4 --
write_json ./output/ --display 0  --write_video ../openpose.avi
# convert the result into MP4
!ffmpeg -y -loglevel info -i openpose.avi output.mp4
```

f) Display the output mp4 video file with the help of IPython display library.

```
def show_local_mp4_video(file_name, width=640, height=480):
  import io
  import base64
  from IPython.display import HTML
  video_encoded = base64.b64encode(io.open(file_name, 'rb').read())
  return HTML(data='''<video width="{0}" height="{1}" alt="test" controls>
                        <source src="data:video/mp4;base64,{2}" type="video/mp4" />
                      </video>'''.format(width, height, video_encoded.decode('ascii')))

show_local_mp4_video('output.mp4', width=960, height=720)
```

Activity wrap-up:
We learn
- ❑ How to setup Google Colab
- ❑ How to install OpenPose
- ❑ How to use the OpenPose to estimate pose on a test video
- ❑ Visualise the result of the pose estimation on Google Colab

Activity 2 – Pose Estimation with PoseNet

In this activity, we will learn:

- ❑ How to mounting Google Drive to Google Colab
- ❑ How to install PoseNet
- ❑ How to use OpenCV to read in video file for processing
- ❑ How to define the parameters required by PoseNet for pose estimation
- ❑ How to run the PoseNet for pose estimation
- ❑ Where to locate the output of the file in Google Drive

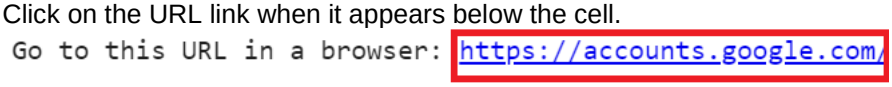
Reference: https://github.com/tensorflow/tfjs-models/tree/master/posenet

1. Mounting of Google Drive as a storage for Google Colab.

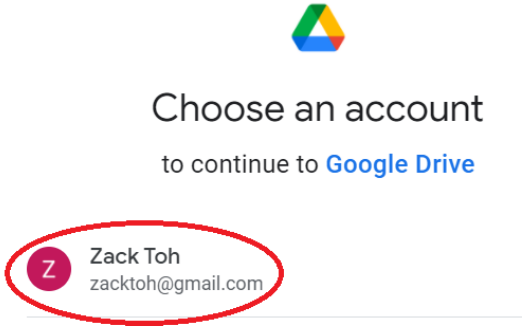
a) Import the Google Colab's drive library and mounting it to your Google Drive.

```
01  # Load the Drive helper and mount
02  from google.colab import drive
03  # This will prompt for authorization.
04  drive.mount('/content/drive')
```

b) Click on the URL link when it appears below the cell.

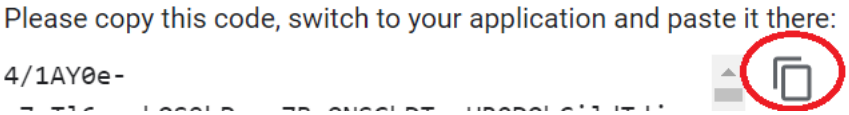


c) Click on the desired Google Account.

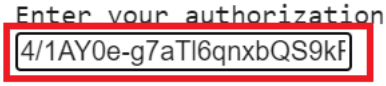


d) Scroll down and click on the "Allow" button.

e) Click on the "Copy" icon to copy the code.

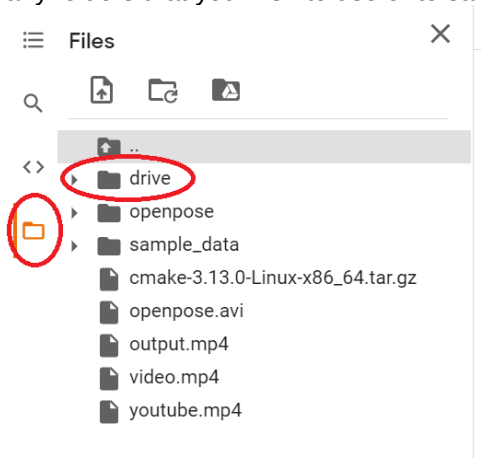


f) Go back to your Google Colab tab and paste the code into the "authorization code" field and press enter.



g) You will see "Mounted at /content/drive" where /content/drive is the mount point on Google Colab.

h) By clicking on the “Folder” icon, you will be able to view the current files create in the temporary space in Google Colab. By expanding on the folder “drive”, you will be able to view the content of your Google Drive and access any folders that you wish to use or to save contents directly into Google Drive.



i) Change the current working directory of the Google Colab to that of Google Drive and clone the PoseNet library into Google Drive. After cloning of the git repository, to change the current working directory into the folder.

```
# Change director listing to your google drive.
% cd /content/drive/My Drive

# Execute this only once so that the repository is cloned into your "Workspace" folder.
! git clone https://github.com/zacktohsh/Workspace_Pose

# Change director listing to "Workspace_Pose"
% cd /content/drive/My Drive/Workspace_Pose/PoseExercise

# Verify correct path and content downloaded
! pwd
! ls -l
```

j) Install all necessary libraries that are required for PoseNet.

```
#Installng Libraries
!pip3 install tensorflow==1.15
!pip3 install tensorflow-gpu==1.15
!pip3 install scipy
!pip3 install pyyaml
!pip3 install opencv-python==3.4.5.20
!pip3 install ipykernel
#Ignore the following errors:
ERROR: tensorflow-probability 0.12.1 has requirement gast>=0.3.2, but you'll have gast 0.2.2
which is incompatible.
ERROR: albumentations 0.1.12 has requirement imgaug<0.2.7,>=0.2.5, but you'll have imgaug 0.2.9
which is incompatible.
#Error is due to the difference in version as tensorflow-
gpu 2 still exist which needs an updated version of various libraries.
```

k) ATTENTION: For this set of codes, you will only need to run this if you have just done a runtime restart.

```
#For any change of code, a restart of runtime session is required.
#To restart runtime session: click on "Runtime" -> "Restart runtime"

#Run this cell after any Run time restart.

# Change director listing to "Workspace_Pose"
% cd /content/drive/My Drive/Workspace_Pose/PoseExercise
!pwd
!ls -l
```

l) Import the necessary libraries and set the path for both input and output files.

```
# PoseNet python sample program
import tensorflow as tf
import cv2
import time
import argparse
import os
import posenet

print('INIT:')
drivepath="../videos/"
driveinfile = drivepath + 'dancing.mp4'
driveoutfile = drivepath + 'output.mp4'
```

m) Setting up opencv2 parameters to read in the video file..

```
# VideoReaderWriter
cap = cv2.VideoCapture(driveinfile)
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
fps = cap.get(cv2.CAP_PROP_FPS)
fourcc = cv2.VideoWriter_fourcc('m','p','4', '2')
video = cv2.VideoWriter(driveoutfile, fourcc, fps, (width, height))
```

n) Define the mobilenet version 1 checkpoint, the scaling factor, pose threshold, part threshold and the maximum number of person for detection.

```
MOBILENET_V1_CHECKPOINTS = {
    50: 'mobilenet_v1_050',
    75: 'mobilenet_v1_075',
    100: 'mobilenet_v1_100',
    101: 'mobilenet_v1_101'
}
```

Image scale factor — A number between 0.2 and 1. Defaults to 0.50.
What to scale the image by before feeding it through the network.
Lower = scale down, speed = faster, accuracy = lower

```
model = 101
scale_factor = 0.2

POSE_THRESHOLD = 0.15
PART_THRESHOLD = 0.15
MAX_PERSON = 2
```

o) Initiating TensorFlow session and loading the PoseNet model. Define the mobilenet version 1 checkpoint, the scaling factor, pose threshold, part threshold and the maximum number of persons for detection.

```
with tf.Session() as sess:
    print('MODEL-INIT:')
    ####model_cfg, model_outputs = posenet.load_model(args.model, sess)
    model_cfg, model_outputs = posenet.load_model(model, sess)
    output_stride = model_cfg['output_stride']
    start = time.time()
    print('START:')

    incnt = 0
    while True:
        incnt = incnt + 1
        try: input_image, draw_image, output_scale = posenet.read_cap(
                cap, scale_factor=scale_factor, output_stride=output_stride)
        except:break
        heatmaps_result, offsets_result, displacement_fwd_result, displacement_bwd_result
 = sess.run(
            model_outputs,
            feed_dict={'image:0': input_image}
        )
```

```
            pose_scores, keypoint_scores, keypoint_coords = posenet.decode_multiple_poses(
                heatmaps_result.squeeze(axis=0),
                offsets_result.squeeze(axis=0),
                displacement_fwd_result.squeeze(axis=0),
                displacement_bwd_result.squeeze(axis=0),
                output_stride=output_stride,
                max_pose_detections=MAX_PERSON,
                min_pose_score=POSE_THRESHOLD)
```

p) Setting up and drawing of the keypoint coordinates.

```
            keypoint_coords *= output_scale

            draw_image = posenet.draw_skel_and_kp(
                    draw_image, pose_scores, keypoint_scores, keypoint_coords,
                    min_pose_score=POSE_THRESHOLD, min_part_score=PART_THRESHOLD)

            video.write(draw_image)
            if incnt % 100 == 0:
                print("cnt=", incnt, "fps=", incnt / (time.time() - start) )

            if False:
                #debug
                for pi in range(len(pose_scores)):
                    if pose_scores[pi] == 0.:
                        break
                    print('Pose #%d, score = %f' % (pi, pose_scores[pi]))
                    for ki, (s, c) in enumerate(zip(keypoint_scores[pi, :], keypoint_coords[p
i, :, :])):
                        print('Keypoint %s, score = %f, coord = %s' % (posenet.PART_NAMES[ki]
, s, c))
video.release()
cap.release()
print('END:')
```

q) The output of PoseNet can be seen in the director.

```
# The output video should be in the video directory:
! ls -l "/content/drive/My Drive/Workspace_Pose/videos/output.mp4"

# You may open your google drive to access this video.
```

Activity wrap-up:

We learn

- ❑ How to mounting Google Drive to Google Colab
- ❑ How to install PoseNet
- ❑ How to use OpenCV to read in video file for processing
- ❑ How to define the parameters required by PoseNet for pose estimation
- ❑ How to run the PoseNet for pose estimation
- ❑ Where to locate the output of the file in Google Drive

Activity 3 – Action Recognition with PoseNet?

In this activity, we will learn:
- ❑ How to use PoseNet to create an Action Recognition system

1. Modifying the current files in PoseNet library in an attempt to produce an action recognition system

a) In this activity, we will attempt to modify the existing set of codes to produce an action recognition software. (The motivation behind this approach is for the appreciation of how action recognition is achieved in the lesson tomorrow.)

b) Modify the file "Workspace_Pose\PoseExercise\posenet\utils.py", specifically the "draw_skel_and_kp" function in order to achieve this.

c) The variable "body_coords[X]" contains the information on the respective keypoints that are processed from PoseNet.

X can be denoted from the following:
NOSE = 0
LEFT_EYE = 1
RIGHT_EYE = 2
LEFT_EAR = 3
RIGHT_EAR = 4
LEFT_SHOULDER = 5
RIGHT_SHOULDER = 6
LEFT_ELBOW = 7
RIGHT_ELBOW = 8
LEFT_HAND = 9
RIGHT_HAND = 10
LEFT_HIP = 11
RIGHT_HIP = 12
LEFT_KNEE = 13
RIGHT_KNEE = 14
LEFT_ANKLE = 15
RIGHT_ANKLE = 16

*Note: to prevent the program from crashing, you will need to check if "body_coords[X]" exist or not.

Activity wrap-up:
We learn how to
- ❑ How to use PoseNet to create an Action Recognition system