Contents

Activity 1 – Pose Estimation with OpenPose
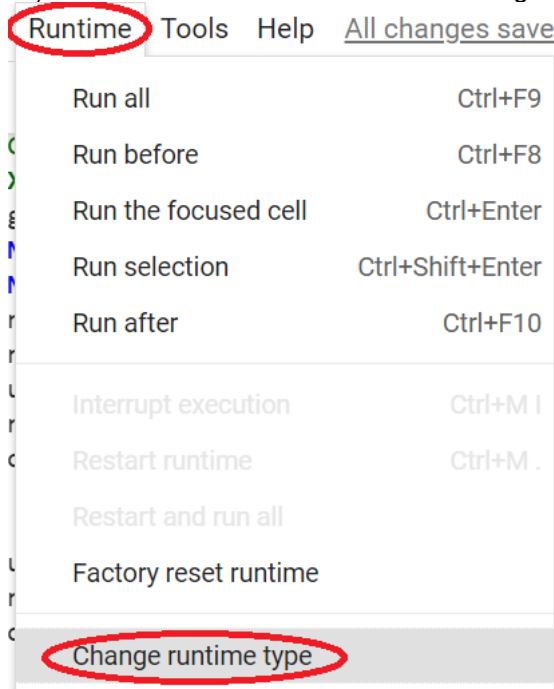
In this activity, we will learn:
❑ How to setup Google Colab
❑ How to install OpenPose
❑ How to use the OpenPose to estimate pose on a test video
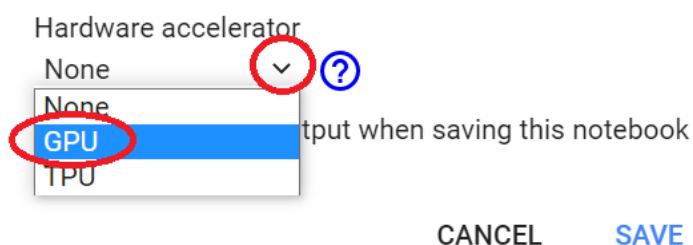❑ Visualise the result of the pose estimation on Google Colab

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":

Runtime	Tools	Help	All changes save

Run all	Ctrl+F9
Run before	Ctrl+F8
Run the focused cell	Ctrl+Enter
Run selection	Ctrl+Shift+Enter
Run after	Ctrl+F10
Interrupt execution	Ctrl+M I
Restart runtime	Ctrl+M .
Restart and run all	
Factory reset runtime	
Change runtime type	

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

    ### Notebook settings

    Hardware accelerator

    None   ⌄   ⑦

    None
    GPU   tput when saving this notebook
    TPU

    CANCEL   SAVE

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.

```
01  if not exists(project_name):
02    # see: https://github.com/CMU-Perceptual-Computing-Lab/openpose/issues/949
03    # install new CMake becaue of CUDA10
04    !wget -q https://cmake.org/files/v3.13/cmake-3.13.0-Linux-x86_64.tar.gz
05    !tar xfz cmake-3.13.0-Linux-x86_64.tar.gz --strip-components=1 -C /usr/local
06    # clone openpose
07    !git clone -q --depth 1 $git_repo_url
08    !sed -
09  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.

```
01    # install system dependencies
02    !apt-get -qq install -y libatlas-base-dev libprotobuf-dev libleveldb-dev libsnappy-
03  dev libhdf5-serial-dev protobuf-compiler libgflags-dev libgoogle-glog-dev liblmdb-
04  dev opencl-headers ocl-icd-opencl-dev libviennacl-dev
05    # install python dependencies
06    !pip install -q youtube-dl
07    # build openpose
08    !cd openpose && rm -rf build || true && mkdir build && cd build && cmake .. && make -
09  j`nproc`
10
11  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.

```
01  YOUTUBE_ID = '23QOy9Q2qNI'
02  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.

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

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

```
01  def show_local_mp4_video(file_name, width=640, height=480):
02    import io
03    import base64
04    from IPython.display import HTML
05    video_encoded = base64.b64encode(io.open(file_name, 'rb').read())
06    return HTML(data='''<video width="{0}" height="{1}" alt="test" controls>
07                        <source src="data:video/mp4;base64,{2}" type="video/mp4" />
08                      </video>'''.format(width, height, video_encoded.decode('ascii')))
09
10  show_local_mp4_video('output.mp4', width=960, height=720)
```

OFFICIAL (CLOSED) \ NON-SENSITIVE

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')
```

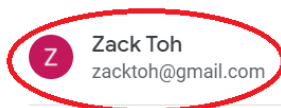   b) Click on the URL link when it appears below the cell.

   Go to this URL in a browser: https://accounts.google.com/

   c) Click on the desired Google Account.

   Choose an account

   to continue to **Google Drive**

   Z Zack Toh
      zacktoh@gmail.com

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

   Allow

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

   Please copy this code, switch to your application and paste it there:

   4/1AY0e-

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

   Enter your authorization code:
   4/1AY0e-g7aTl6qnxbQS9kF

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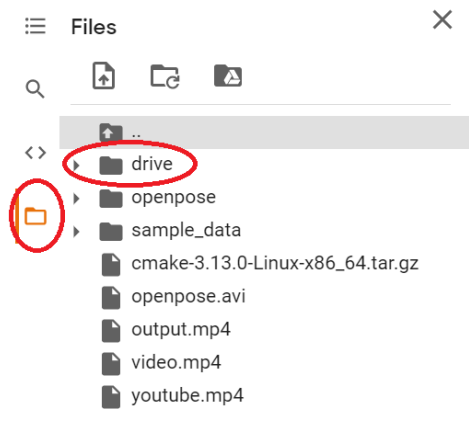
i) Change the current working directory of the Google Colab to that of Google Drive and clone the PoseNet library into Google Drive. After cloning of the git repository, to change the current working directory into the folder.

```
01   # Change director listing to your google drive.
02   % cd /content/drive/My Drive
03
04   # Execute this only once so that the repository is cloned into your "Workspace" folder.
05   ! git clone https://github.com/zacktohsh/Workspace_Pose
06
07   # Change director listing to "Workspace_Pose"
08   % cd /content/drive/My Drive/Workspace_Pose/PoseExercise
09
10   # Verify correct path and content downloaded
11   ! pwd
12   ! ls -l
```

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

```
01   #Installng Libraries
02   !pip3 install tensorflow==1.15
03   !pip3 install tensorflow-gpu==1.15
04   !pip3 install scipy
05   !pip3 install pyyaml
06   !pip3 install opencv-python==3.4.5.20
07   !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.

```
01   #For any change of code, a restart of runtime session is required.
02   #To restart runtime session: click on "Runtime" -> "Restart runtime"
03
04   #Run this cell after any Run time restart.
05
06   # Change director listing to "Workspace_Pose"
07   % cd /content/drive/My Drive/Workspace_Pose/PoseExercise
08   !pwd
09   !ls -l
```

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

```
01   # PoseNet python sample program
02   import tensorflow as tf
03   import cv2
04   import time
05   import argparse
06   import os
07   import posenet
08
09   print('INIT:')
10   drivepath="../videos/"
11   driveinfile = drivepath + 'dancing.mp4'
12   driveoutfile = drivepath + 'output.mp4'
```

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

```
01   # VideoReaderWriter
02   cap = cv2.VideoCapture(driveinfile)
03   width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
04   height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
05   fps = cap.get(cv2.CAP_PROP_FPS)
06   fourcc = cv2.VideoWriter_fourcc('m','p','4', '2')
07   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

```
01   model = 101
02   scale_factor = 0.2
03
04   POSE_THRESHOLD = 0.15
05   PART_THRESHOLD = 0.15
06   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.

```
01   with tf.Session() as sess:
02       print('MODEL-INIT:')
03       ####model_cfg, model_outputs = posenet.load_model(args.model, sess)
04       model_cfg, model_outputs = posenet.load_model(model, sess)
05       output_stride = model_cfg['output_stride']
06       start = time.time()
07       print('START:')
08
09       incnt = 0
10       while True:
11           incnt = incnt + 1
12           try: input_image, draw_image, output_scale = posenet.read_cap(
13                   cap, scale_factor=scale_factor, output_stride=output_stride)
14           except:break
15           heatmaps_result, offsets_result, displacement_fwd_result, displacement_bwd_result
16   = sess.run(
17                   model_outputs,
18                   feed_dict={'image:0': input_image}
19               )
```

OFFICIAL (CLOSED) \ NON-SENSITIVE

```
20          pose_scores, keypoint_scores, keypoint_coords = posenet.decode_multiple_poses(
21              heatmaps_result.squeeze(axis=0),
22              offsets_result.squeeze(axis=0),
23              displacement_fwd_result.squeeze(axis=0),
24              displacement_bwd_result.squeeze(axis=0),
25              output_stride=output_stride,
26              max_pose_detections=MAX_PERSON,
27              min_pose_score=POSE_THRESHOLD)
```

   p) Setting up and drawing of the keypoint coordinates (Note: This set codes needs to be in the same code sell as part 'o').

```
01          keypoint_coords *= output_scale
02
03          draw_image = posenet.draw_skel_and_kp(
04              draw_image, pose_scores, keypoint_scores, keypoint_coords,
05              min_pose_score=POSE_THRESHOLD, min_part_score=PART_THRESHOLD)
06
07          video.write(draw_image)
08          if incnt % 100 == 0:
09              print("cnt=", incnt, "fps=", incnt / (time.time() - start) )
10
11          if False:
12              #debug
13              for pi in range(len(pose_scores)):
14                  if pose_scores[pi] == 0.:
15                      break
16                  print('Pose #%d, score = %f' % (pi, pose_scores[pi]))
17                  for ki, (s, c) in enumerate(zip(keypoint_scores[pi, :], keypoint_coords[p
18 i, :, :])):
19                      print('Keypoint %s, score = %f, coord = %s' % (posenet.PART_NAMES[ki]
20 , s, c))
21 video.release()
22 cap.release()
23 print('END:')
```

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

```
01  # The output video should be in the video directory:
02  ! ls -l "/content/drive/My Drive/Workspace_Pose/videos/output.mp4"
03
04  # 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