Contents

Activity 1 – Action Recognition with Two-Stream Inflated 3D ConvNet (I3D)

In this activity, we will learn:

- ❑ How to setup and install ID3 for action inference
- ❑ How to interact with UCF101 dataset to retrieve useful information
- ❑ How to retrieve labels from kinetics-400
- ❑ How to retrieve the videos from UCF101 dataset for testing purpose
- ❑ Visualise the result of the action detection on Google Colab

Reference: https://github.com/deepmind/kinetics-i3d
Model reference (i3d-kinetics-400): https://tfhub.dev/deepmind/i3d-kinetics-400/1
Action reference (i3d-kinetics-400): https://github.com/deepmind/kinetics-i3d/blob/master/data/label_map.txt

1. Installation and execution of I3D for action inference

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

```
!pip install -q imageio
!pip install -q opencv-python
!pip install -q git+https://github.com/tensorflow/docs
```

b) Import the necessary libraries.

```
# TensorFlow and TF-Hub modules.
from absl import logging

import tensorflow as tf
import tensorflow_hub as hub
from tensorflow_docs.vis import embed

logging.set_verbosity(logging.ERROR)

# Some modules to help with reading the UCF101 dataset.
import random
import re
import os
import tempfile
import ssl
import cv2
import numpy as np

# Some modules to display an animation using imageio.
import imageio
from IPython import display

from urllib import request  # requires python3
```

c) Functions necessary to interact with the UCF101 dataset.

```
# Utilities to fetch videos from UCF101 dataset
UCF_ROOT = "https://www.crcv.ucf.edu/THUMOS14/UCF101/UCF101/"
_VIDEO_LIST = None
_CACHE_DIR = tempfile.mkdtemp()
unverified_context = ssl._create_unverified_context()

# Function to list the ucf video
def list_ucf_videos():
  """Lists videos available in UCF101 dataset."""
  global _VIDEO_LIST
  if not _VIDEO_LIST:
    index = request.urlopen(UCF_ROOT, context=unverified_context).read().decode("utf-8")
    videos = re.findall("(v_[\w_]+\.avi)", index)
    _VIDEO_LIST = sorted(set(videos))
  return list(_VIDEO_LIST)

# Function to retrieve the ucf video
def fetch_ucf_video(video):
  """Fetchs a video and cache into local filesystem."""
```

```
  cache_path = os.path.join(_CACHE_DIR, video)
  if not os.path.exists(cache_path):
    urlpath = request.urljoin(UCF_ROOT, video)
    print("Fetching %s => %s" % (urlpath, cache_path))
    data = request.urlopen(urlpath, context=unverified_context).read()
    open(cache_path, "wb").write(data)
  return cache_path

# Function to open video files using CV2
def crop_center_square(frame):
  y, x = frame.shape[0:2]
  min_dim = min(y, x)
  start_x = (x // 2) - (min_dim // 2)
  start_y = (y // 2) - (min_dim // 2)
  return frame[start_y:start_y+min_dim,start_x:start_x+min_dim]

#Function to load video
def load_video(path, max_frames=0, resize=(224, 224)):
  cap = cv2.VideoCapture(path)
  frames = []
  try:
    while True:
      ret, frame = cap.read()
      if not ret:
        break
      frame = crop_center_square(frame)
      frame = cv2.resize(frame, resize)
      frame = frame[:, :, [2, 1, 0]]
      frames.append(frame)

      if len(frames) == max_frames:
        break
  finally:
    cap.release()
  return np.array(frames) / 255.0

#function to convert from video to animated gif
def to_gif(images):
  converted_images = np.clip(images * 255, 0, 255).astype(np.uint8)
  imageio.mimsave('./animation.gif', converted_images, fps=25)
  return embed.embed_file('./animation.gif')
```

d) Retrieve the labels from kinetics-400

```
# Get the kinetics-400 action labels from the GitHub repository.
KINETICS_URL = "https://raw.githubusercontent.com/deepmind/kinetics-
i3d/master/data/label_map.txt"
with request.urlopen(KINETICS_URL) as obj:
  labels = [line.decode("utf-8").strip() for line in obj.readlines()]
print("Found %d labels." % len(labels))
```

e) Retrieving the list of videos from the UCF dataset.

```
# Get the list of videos in the dataset.
ucf_videos = list_ucf_videos()

categories = {}
for video in ucf_videos:
  category = video[2:-12]
  if category not in categories:
    categories[category] = []
  categories[category].append(video)
print("Found %d videos in %d categories." % (len(ucf_videos), len(categories)))

for category, sequences in categories.items():
  summary = ", ".join(sequences[:2])
  print("%-20s %4d videos (%s, ...)" % (category, len(sequences), summary))
```

f) Using one of the videos from the list of videos from the UCF dataset. You may change this video to another video of your preference.

```
# Get a sample cricket video.
video_path = fetch_ucf_video("v_CricketShot_g04_c02.avi")
sample_video = load_video(video_path)
```

g) Show the dimension of the loaded video.

```
sample_video.shape
```

h) Load the I3D model.

```
i3d = hub.load("https://tfhub.dev/deepmind/i3d-kinetics-400/1").signatures['default']
```

i) Create and call the function that uses the ID3 to perform the action prediction and display the top 5 predicted categories.

```
def predict(sample_video):
  # Add a batch axis to the to the sample video.
  model_input = tf.constant(sample_video, dtype=tf.float32)[tf.newaxis, ...]

  logits = i3d(model_input)['default'][0]
  probabilities = tf.nn.softmax(logits)

  print("Top 5 actions:")
  for i in np.argsort(probabilities)[::-1][:5]:
    print(f"  {labels[i]:22}: {probabilities[i] * 100:5.2f}%")

predict(sample_video)
```

j) Retrieve another video from the internet for testing. You may also choose an alternate video from the following: https://commons.wikimedia.org/wiki/Category:Videos_of_sports

```
!curl -O https://upload.wikimedia.org/wikipedia/commons/8/86/End_of_a_jam.ogv

video_path = "End_of_a_jam.ogv"

sample_video = load_video(video_path)[:100]
sample_video.shape
```

k) Converting the video to an animated GIF for viewing through colab.

```
to_gif(sample_video)
```

l) Performing prediction on the latest video.

```
predict(sample_video)
```

Activity wrap-up:
We learn
- ❑ How to setup and install ID3 for action inference
- ❑ How to interact with UCF101 dataset to retrieve useful information
- ❑ How to retrieve labels from kinetics-400
- ❑ How to retrieve the videos from UCF101 dataset for testing purpose
- ❑ Visualise the result of the action detection on Google Colab

Activity 2 – Action Recognition with MMAction2

<u>In this activity, we will learn:</u>

- ❑ How to mounting Google Drive to Google Colab
- ❑ How to install and configure MMAction 2 for action inference
- ❑ How to train a new action recognition model
- ❑ How to validate the accuracy of the newly created model

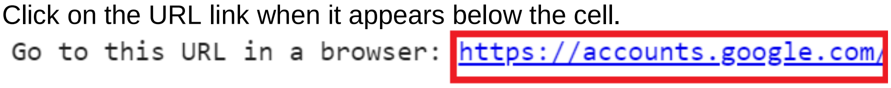
Reference: https://github.com/open-mmlab/mmaction2

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

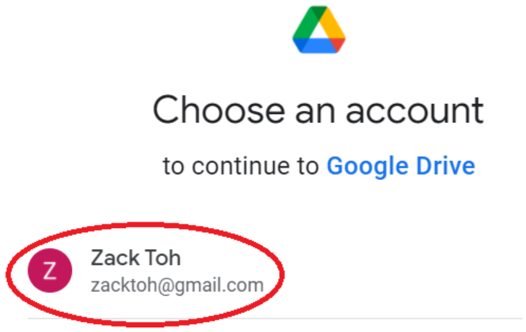
a) Import the Google Colab's drive library and mounting it to your Google Drive.

```
# Load the Drive helper and mount
from google.colab import drive

# This will prompt for authorization.
drive.mount('/content/drive')

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

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

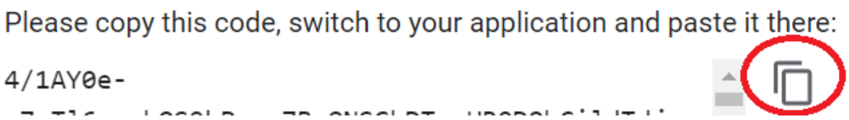


c) Click on the desired Google Account.

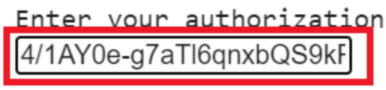


d) Scroll down and click on the "Allow" button.

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

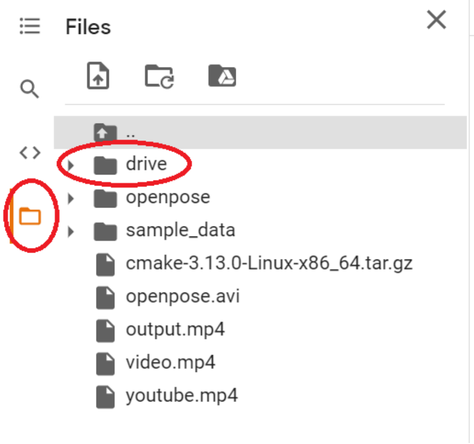


f) Go back to your Google Colab tab and paste the code into the "authorization code" field and press enter.



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) Clone the workspace from github.

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

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

#https://drive.google.com/file/d/1R3Be-PIy3ZeXNpKsGaQa7gR82wOLhHlZ/view?usp=sharing
!gdown --id 1R3Be-PIy3ZeXNpKsGaQa7gR82wOLhHlZ

!unzip -a Workspace_Action.zip
!rm Workspace_Action.zip
# Verify correct path and content downloaded
! pwd
! ls -l
```

j) Checking the current version of libraries.

```
# Check nvcc version
!nvcc -V
# Check GCC version
!gcc --version

# Check python version
!python --version
```

k) Installing the necessary libraries.

```
#Installation of required libraries and files.
!pip install torch-1.5.1+cu101-cp36-cp36m-linux_x86_64.whl
!pip install torchvision-0.6.1+cu101-cp36-cp36m-linux_x86_64.whl
!pip install mmcv_full-latest+torch1.5.0+cu101-cp36-cp36m-manylinux1_x86_64.whl

%cd mmaction2
!pip install -e .

# Install some optional requirements
!pip install -r requirements/optional.txt
```

```
Ignore the following error:
ERROR: torchvision 0.8.1+cu101 has requirement torch==1.7.0, but you'll have torch 1.5.1+cu101 which is incompatible.
```

l) Import required libraries.

```
# Check Pytorch installation
import torch, torchvision
print(torch.__version__, torch.cuda.is_available())

# Check MMAction2 installation
import mmaction
print(mmaction.__version__)

# Check MMCV installation
from mmcv.ops import get_compiling_cuda_version, get_compiler_version
print(get_compiling_cuda_version())
print(get_compiler_version())

from mmaction.apis import inference_recognizer, init_recognizer
```

m) Define the location of both the configuration file along with the checkpoint file to be used.

```
# Choose to use a config and initialize the recognizer
config = 'configs/recognition/tsn/tsn_r50_video_inference_1x1x3_100e_kinetics400_rgb.py'
# Setup a checkpoint file to load
checkpoint = 'checkpoints/tsn_r50_1x1x3_100e_kinetics400_rgb_20200614-e508be42.pth'
# Initialize the recognizer
model = init_recognizer(config, checkpoint, device='cuda:0')
```

n) Setting the video file and label to use followed by initiating the recognizer function.

```
# Use the recognizer to do inference
video = 'demo/demo.mp4'
label = 'demo/label_map.txt'
results = inference_recognizer(model, video, label)
```

o) Show the results from the recognizer function.

```
# Let's show the results
for result in results:
    print(f'{result[0]}: ', result[1])
```

2. Training an Action Recognition Model

a) Install and view the tree structure of kinetics400_tiny, its folder directory and the file content in it.

```
# Install tree first
!apt-get -q install tree
!tree kinetics400_tiny
```

b) Displaying the content of the video dataset, displaying the path, file name together with the label.

```
# After downloading the data, we need to check the annotation format
!cat kinetics400_tiny/kinetics_tiny_train_video.txt
```

c) Load the configuration file.

```
from mmcv import Config
cfg = Config.fromfile('./configs/recognition/tsn/tsn_r50_video_1x1x8_100e_kinetics400_rgb
.py')
```

d) Modifying the configuration so that it is compatible with the Kinetics400-tiny dataset.

```
from mmcv.runner import set_random_seed

# Modify dataset type and path
cfg.dataset_type = 'VideoDataset'
cfg.data_root = 'kinetics400_tiny/train/'
cfg.data_root_val = 'kinetics400_tiny/val/'
cfg.ann_file_train = 'kinetics400_tiny/kinetics_tiny_train_video.txt'
```

```
cfg.ann_file_val = 'kinetics400_tiny/kinetics_tiny_val_video.txt'
cfg.ann_file_test = 'kinetics400_tiny/kinetics_tiny_val_video.txt'

cfg.data.test.type = 'VideoDataset'
cfg.data.test.ann_file = 'kinetics400_tiny/kinetics_tiny_val_video.txt'
cfg.data.test.data_prefix = 'kinetics400_tiny/val/'

cfg.data.train.type = 'VideoDataset'
cfg.data.train.ann_file = 'kinetics400_tiny/kinetics_tiny_train_video.txt'
cfg.data.train.data_prefix = 'kinetics400_tiny/train/'

cfg.data.val.type = 'VideoDataset'
cfg.data.val.ann_file = 'kinetics400_tiny/kinetics_tiny_val_video.txt'
cfg.data.val.data_prefix = 'kinetics400_tiny/val/'

# The flag is used to determine whether it is omnisource training
cfg.setdefault('omnisource', False)
# Modify num classes of the model in cls_head
cfg.model.cls_head.num_classes = 2
# We can use the pre-trained TSN model
cfg.load_from = './checkpoints/tsn_r50_1x1x3_100e_kinetics400_rgb_20200614-e508be42.pth'

# Set up working dir to save files and logs.
cfg.work_dir = './tutorial_exps'

# The original learning rate (LR) is set for 8-GPU training.
# We divide it by 8 since we only use one GPU.
cfg.data.videos_per_gpu = cfg.data.videos_per_gpu // 16
cfg.optimizer.lr = cfg.optimizer.lr / 8 / 16
cfg.total_epochs = 30

# We can set the checkpoint saving interval to reduce the storage cost
cfg.checkpoint_config.interval = 10
# We can set the log print interval to reduce the the times of printing log
cfg.log_config.interval = 5

# Set seed thus the results are more reproducible
cfg.seed = 0
set_random_seed(0, deterministic=False)
cfg.gpu_ids = range(1)


# We can initialize the logger for training and have a look
# at the final config used for training
print(f'Config:\n{cfg.pretty_text}')
```

e) Initialize both the dataset and the recognizer, followed by training the recognition model.

```
import os.path as osp

from mmaction.datasets import build_dataset
from mmaction.models import build_model
from mmaction.apis import train_model

import mmcv

# Build the dataset
datasets = [build_dataset(cfg.data.train)]

# Build the recognizer
model = build_model(cfg.model, train_cfg=cfg.train_cfg, test_cfg=cfg.test_cfg)

# Create work_dir
mmcv.mkdir_or_exist(osp.abspath(cfg.work_dir))
train_model(model, datasets, cfg, distributed=False, validate=True)
```

f) Validation of the newly trained action recognition model's prediction accuracy.

```
from mmaction.apis import single_gpu_test
from mmaction.datasets import build_dataloader
from mmcv.parallel import MMDataParallel

# Build a test dataloader
dataset = build_dataset(cfg.data.test, dict(test_mode=True))
data_loader = build_dataloader(
        dataset,
        videos_per_gpu=1,
        workers_per_gpu=cfg.data.workers_per_gpu,
        dist=False,
        shuffle=False)
model = MMDataParallel(model, device_ids=[0])
outputs = single_gpu_test(model, data_loader)

eval_config = cfg.evaluation
eval_config.pop('interval')
eval_res = dataset.evaluate(outputs, **eval_config)
for name, val in eval_res.items():
    print(f'{name}: {val:.04f}')
```

g) Setup the new model to be tested with a new video.

```
from mmaction.apis import inference_recognizer, init_recognizer

# Choose to use a config and initialize the recognizer
config = 'configs/recognition/tsn/tsn_r50_video_inference_1x1x3_100e_kinetics400_rgb.py'
# Setup a checkpoint file to load
checkpoint = 'tutorial_exps/epoch_30.pth'
# Initialize the recognizer
model = init_recognizer(cfg, checkpoint, device='cuda:0')
```

h) Copy both the label map file and the test video into the working director for testing purpose. You may also use other video files that are present in the validation (val) folder.
(Note: Google Drive may take a while to reflect the presence of the file so refer to the folder structure from Google Colab.)

```
from shutil import copyfile
copyfile('./kinetics400_tiny/val/0pVGiAU6XEA.mp4', './tutorial_exps/0pVGiAU6XEA.mp4')
copyfile('./demo/label_map.txt', './tutorial_exps/label_map.txt')
```

i) Setting the video file and label to use followed by initiating the recognizer function.

```
# Use the recognizer to do inference
video = 'tutorial_exps/0pVGiAU6XEA.mp4'
label = 'tutorial_exps/label_map.txt'
results = inference_recognizer(model, video, label)
```

j) Show the result of the prediction.

```
# Let's show the results
for result in results:
    print(f'{result[0]}: ', result[1])
```

Activity wrap-up:
We learn
- ❑ How to mounting Google Drive to Google Colab
- ❑ How to install and configure MMAction 2 for action inference
- ❑ How to train a new action recognition model
- ❑ How to validate the accuracy of the newly created model

Activity 3 – Creating your own action recognition model

In this activity, we will learn:

- ❑ How to use MMAction2 to create a new action recognition model

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

a) In this activity, you are required to train your own action recognition model.

b) You are to find a partner and choose between two to three action of interest.

c) Perform a video recording of the actions of choice using your mobile phone and upload the into the “tutorial_exps/video” folder. (the mode videos you use for your training, the more accurate your model will be, but do be mindful of the storage limit on your Google Drive.

d) Create a file “label_map.txt” and key in the actions.

*label_map - Notepad
File Edit Format View Help

```
running
jogging
```

e) Create three files (train_video, test_video, val_video).

f) Inside each file, include the files that you wish to include for each phase of the training process followed by the respective label (0,1,2).

train_video - Notepad
File Edit Format View Help

```
chad_front_elevated_v3_avc_360p_cropped_1_37_448.mp4    0
chad_front_elevated_v3_avc_360p_cropped_10_0_38_265.mp4 0
chad_front_elevated_v3_avc_360p_cropped_2_64_277.mp4    0
chad_front_elevated_v3_avc_360p_cropped_3_68_242.mp4    0
```

g) Update the codes used in Activity 2, 2d as follow:

```
# Modify dataset type and path
cfg.dataset_type = 'VideoDataset'
cfg.data_root = 'tutorial_exps/videos/'
cfg.data_root_val = 'tutorial_exps/videos/'
cfg.ann_file_train = 'tutorial_exps/train_video.txt'
cfg.ann_file_val = 'tutorial_exps/val_video.txt'
cfg.ann_file_test = 'tutorial_exps/test_video.txt'

cfg.data.test.type = 'VideoDataset'
cfg.data.test.ann_file = 'tutorial_exps/test_video.txt'
cfg.data.test.data_prefix = 'tutorial_exps/videos/'

cfg.data.train.type = 'VideoDataset'
cfg.data.train.ann_file = 'tutorial_exps/train_video.txt'
cfg.data.train.data_prefix = 'tutorial_exps/videos/'

cfg.data.val.type = 'VideoDataset'
cfg.data.val.ann_file = 'tutorial_exps/val_video.txt'
cfg.data.val.data_prefix = 'tutorial_exps/videos/'
```

h) Do not modify the number of class to the correct integer representation.

```
cfg.model.cls_head.num_classes = X
```

i) Feel free to play around with the rest of the parameters such as:

```
cfg.total_epochs = 30
cfg.checkpoint_config.interval = 10
cfg.log_config.interval = 5
```

j) After you are satisfied with the configurations, rerun the configurations of Activity 2, steps 2d – 2j in sequence.

k) If you find that the validation results in Activity 2 step 2f does not produce a high accuracy for your new model, you may modify the code in Activity 2, 2d to continue the training from where you had left off by changing the configuration to load from your last recorded epoch.

```
cfg.load_from = epoch_XX.pth
```

l) After you are satisfied with the configurations, be sure to rerun the configurations of Activity 2, steps 2d – 2j in sequence.

Activity wrap-up:

We learn how to

- ❑ How to use MMAction2 to create a new action recognition model