# **Detailed Documentation for Video Processing and Keypoint Visualization using Detectron2 with the COCO Dataset**

## **Data Acquisition**

The script downloads and prepares datasets from the COCO dataset, specifically the validation images and annotations for 2017.

!wget http://images.cocodataset.org/zips/val2017.zip

!wget http://images.cocodataset.org/annotations/annotations\_trainval2017.zip

!unzip -q val2017.zip

!unzip -q annotations\_trainval2017.zip

## **Directory Setup**

Create necessary directories for storing processed data, intermediate results, and final outputs.

!mkdir -p /content/{inputs,intermediate,results}

## **Annotation Loading and Filtering**

Load annotations and filter out images based on specific criteria, such as the presence of a complete set of keypoints for the 'person' category.

from pycocotools.coco import COCO

coco = COCO(annFile)

catIds = coco.getCatIds(catNms=['person'])

imgIds = coco.getImgIds(catIds=catIds)

## **Image Processing**

Convert images to videos and resize them to maintain consistency for the pose estimation model.

import subprocess

for image\_file\_name in os.listdir('/content/images'):

# Image processing commands using ffmpeg

## **Detectron2 Integration**

Install Detectron2 and replace the infer\_video\_d2.py script which the code present in <https://github.com/Aadharsh1/ML-Deep-Learning/blob/main/Pose_Estimation_Experiments/VideoPose3D_Experiments/1_video_pose_3d(videos).ipynb>

This is crucial when using Detectron2's models for improved pose estimation.

!python -m pip install 'git+https://github.com/facebookresearch/detectron2.git'

# Instructions to replace infer\_video\_d2.py go here

## **VideoPose3D Setup**

Download VideoPose3D, set up checkpoints, and prepare for running pose estimation on the processed videos.

!git clone https://github.com/facebookresearch/VideoPose3D

cd VideoPose3D

mkdir checkpoint

!wget https://dl.fbaipublicfiles.com/video-pose-3d/pretrained\_h36m\_detectron\_coco.bin

## **Pose Estimation Execution**

Run detectron2 on the videos and extract keypoints.

cd /content/VideoPose3D/inference

!python infer\_video\_d2.py --cfg COCO-Keypoints/keypoint\_rcnn\_R\_101\_FPN\_3x.yaml --output-dir /content/output/ --image-ext mp4 /content/inputs

## **Save detected keypoints to a custom dataset**

cd /content/VideoPose3D/data

!python prepare\_data\_2d\_custom.py -i /content/output -o myvideos

### **Directory Setup**

1. **Define Directory Paths**:
   * input\_dir: Directory containing input videos.
   * intermediate\_dir: Directory to store intermediate processed files.
   * results\_dir: Directory to save the final results.
   * video\_pose3d\_dir: Path where the VideoPose3D repository is located.

import os

input\_dir = '/content/inputs'

intermediate\_dir = '/content/intermediate'

results\_dir = '/content/results'

video\_pose3d\_dir = '/content/VideoPose3D'

**Ensure Directories Exist**:

Create intermediate\_dir and results\_dir if they do not exist.

os.makedirs(intermediate\_dir, exist\_ok=True)

os.makedirs(results\_dir, exist\_ok=True)

### **Processing Videos**

**Iterate Through Videos**:

For each video file in input\_dir, perform pose estimation and generate a processed video file and an image capturing the first frame.

from tqdm import tqdm

import subprocess

for video\_name in tqdm(os.listdir(input\_dir)):

if video\_name.endswith('.mp4'):

name = video\_name.split('.')[0]

processed\_vid\_name = f'{name}\_processed.mp4'

final\_image\_name = f'{name}\_final.jpg'

# Run VideoPose3D pose estimation

subprocess.run([

'python', 'run.py', '-d', 'custom', '-k', 'myvideos', '-arc', '3,3,3,3,3', '-c', 'checkpoint',

'--evaluate', 'pretrained\_h36m\_detectron\_coco.bin', '--render',

'--viz-subject', video\_name, '--viz-action', 'custom', '--viz-camera', '0',

'--viz-video', os.path.join(input\_dir, video\_name),

'--viz-output', os.path.join(intermediate\_dir, processed\_vid\_name),

'--viz-size', '6'

], cwd=video\_pose3d\_dir)

# Extract the first frame from the processed video

subprocess.run([

'ffmpeg', '-i', os.path.join(intermediate\_dir, processed\_vid\_name),

'-vf', 'select=eq(n\,0)', '-q:v', '3', os.path.join(results\_dir, final\_image\_name)

])

### **Data Analysis**

**Load Predicted Keypoints**:

Load the 2D keypoints predicted by VideoPose3D from the processed data.

import numpy as np

data = np.load(os.path.join(video\_pose3d\_dir, 'data/data\_2d\_custom\_myvideos.npz'), allow\_pickle=True)

pred\_keypoints = data['positions\_2d'].item()

**Load Ground Truth Keypoints**:

Load keypoints and bounding boxes from a JSON file containing annotations.

def load\_keypoints(keypoints\_file):

with open(keypoints\_file, 'r') as file:

data = json.load(file)

keypoints\_dict = {}

for item in data:

image\_name = item['file\_name'].split('.')[0] + '.mp4'

keypoints = np.array(item['keypoints']).reshape(-1, 3)[:, :2]

bbox = item['bbox']

keypoints\_dict[image\_name] = {'keypoints': keypoints, 'bbox': bbox}

return keypoints\_dict

ground\_truth\_keypoints\_bbox = load\_keypoints('/content/keypoints\_data.json')

**Compute Keypoint Distances**:

Calculate the Euclidean distance between predicted and ground truth keypoints for analysis.

def compute\_keypoint\_distances(predicted\_keypoints, ground\_truth\_keypoints):

distances = np.sqrt(np.sum((predicted\_keypoints - ground\_truth\_keypoints)\*\*2, axis=1))

return distances

keypoint\_distances = {i: [] for i in range(17)}

for video\_name, data in pred\_keypoints.items():

if video\_name in ground\_truth\_keypoints\_bbox:

predicted\_kps = np.array(data['custom'][0][0])[:, :2]

ground\_truth\_kps = np.array(ground\_truth\_keypoints\_bbox[video\_name]['keypoints'])[:, :2]

distances = compute\_keypoint\_distances(predicted\_kps, ground\_truth\_kps)

for idx, dist in enumerate(distances):

keypoint\_distances[idx].append(dist)

**Average Distances**:

Calculate the average distances for each keypoint across all videos.

average\_distances = {k: np.mean(v) if v else None for k, v in keypoint\_distances.items()}

for keypoint\_idx, avg\_dist in average\_distances.items():

print(f"Average distance for keypoint {keypoint\_idx}: {avg\_dist:.2f}")

### **Visualization**

* **Plot Keypoints on Images**:
  + Visualize keypoints and their accuracy on original images and stickman plots, showcasing the prediction quality.