PMP596 Homework 4

Due November 28, Sunday, 11:59 PM

Problem 1: Human Pose Estimation (25%)

Human pose estimation (HPE) is an important task in the computer vision community. For the deep learning based HPE methods, they are usually divided into two categories, i.e., **top-down** and **bottom-up**. The top-down approach starts by identifying and localizing individual person instances using a bounding box object detector, such as a Faster R-CNN. This is then followed by estimating the pose of a single person. The bottom-up approach starts by localizing identity-free semantic entities, then grouping them into person instances. OpenPose introduced in the lecture is a bottom-up method that predicts the keypoints directly from the global image and then groups the keypoints from the same identity.

In this problem, we would like to discuss a top-down HPE method, named Stacked Hourglass Network and work with an MPII human pose dataset (http://human-pose.mpi-inf.mpg.de/).

(a) MPII Dataset Introduction

MPII Human Pose dataset, provided by the Max Planck Institute for Informatics, is a comprehensive benchmark for evaluation of articulated human pose estimation. The dataset includes around 25K images containing over 40K people with annotated body joints. The images are systematically collected using an established taxonomy of everyday human activities.

Overall the dataset covers 410 human activities and each image is provided with an activity label. Each image is extracted from a YouTube video and provided with preceding and following un-annotated frames. In addition, for the test set richer annotations are available, including body part occlusions and 3D torso and head orientations.

Fig. 1: Some examples of human poses in MPII dataset.

(b) Stacked Hourglass Network

The stacked hourglass network (HG) for human pose estimation was proposed in ECCV'16 (The paper can be found at http://arxiv.org/abs/1603.06937). Unlike OpenPose, HG is a top-down method for human pose estimation, which predicts a single human pose from each input image.

HG is a stack of hourglass modules. It got this name because the shape of each hourglass module closely resembles an hourglass (shown in Fig. 2), which is similar to the fully convolution network (FCN) discussed in object segmentation. The idea behind stacking multiple HG modules instead of forming a giant encoder and decoder network is that each HG stack module will produce a full heat-map for joint prediction. Intermediate supervision is applied to the predictions of each hourglass stage, i.e, the predictions of each hourglass in the stack are supervised, and not only the final hourglass predictions. Thus, the latter HG module can learn from the joint predictions of the previous HG module.

Similar with the definition of heatmaps to represent joint locations mentioned in OpenPose, the authors of this work also adopt the idea and find the peaks of the heatmaps and use that as the joint locations.

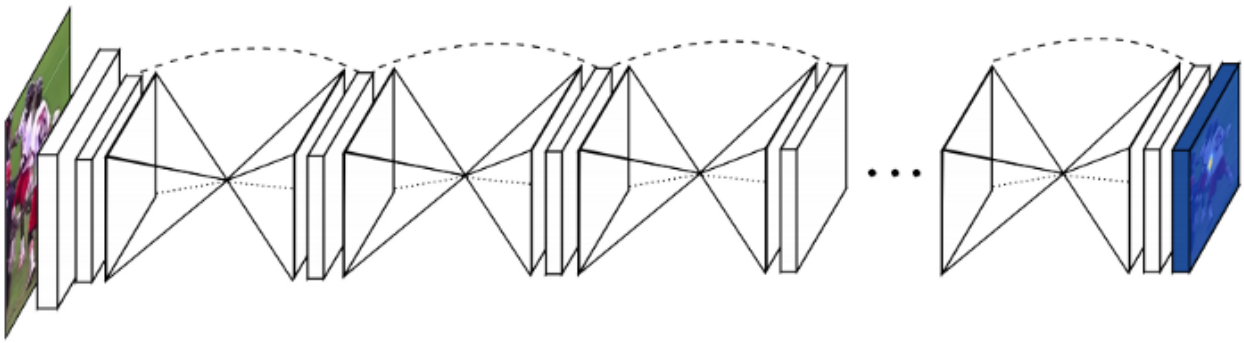

Fig. 2: Diagram of Stacked Hourglass Networks for Human Pose Estimation.

As for each hourglass module inside the diagram, the architecture is shown in Fig. 3. In the figure, each box is a residual block, proposed in the ResNet paper (https://arxiv.org/abs/1512.03385). In general, an HG module is an encoder and decoder architecture, where we downsample the features first, and then upsample the features to recover the info and form a heat-map. Each encoder layer would have a connection to its decoder counterpart, and we could stack as many as layers we want.

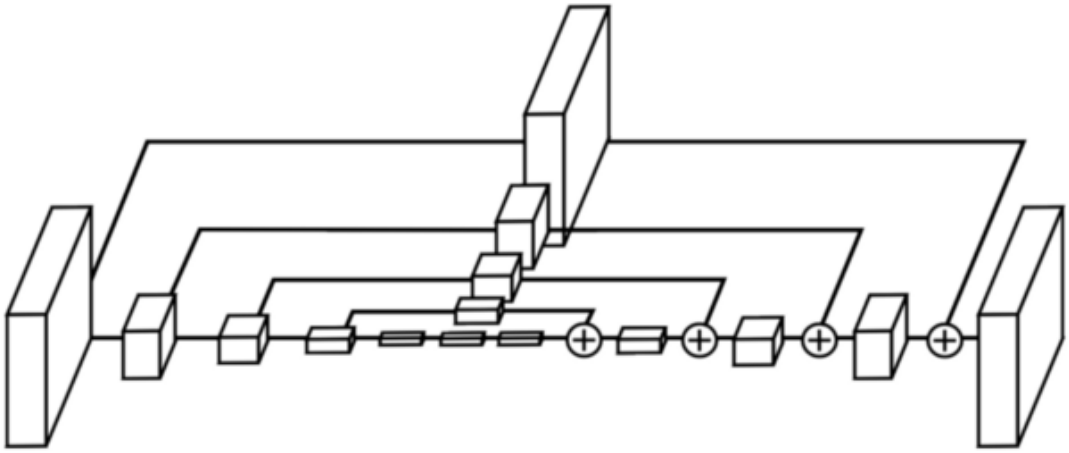

Fig. 3: Architecture of the hourglass module.

Some resources above credit to:
https://towardsdatascience.com/human-pose-estimation-with-stacked-hourglass-network-and-tensorflow-c4e9f84fd3ce.

(c) Train the Network using MPII Dataset (5%)

Here, we use an open source package of the stacked HG:
https://github.com/princeton-vl/pytorch_stacked_hourglass, for this problem.

1) Download the source code from GitHub using the provided sample code.
2) Download the MPII dataset and unzip the data to the directory (may take ~20 min).
3) Visualize some images in the MPII dataset.
4) Change the following parameters in the configurations:
   - `'nstack': 2` (number of stacks)
   - `'train_iters': 100` (number of iterations in each epoch)

   Train the network for 4 epochs. Draw the plot of the loss values from your training log file. (No need to be well-trained if you do not have enough time.)
5) Evaluate your trained models on the MPII validation set using the tool provided. Print out your evaluation scores. (Your scores are not necessarily good since it requires a very long time for the training)

(d) Inference and Visualization (10%)

1) Download 2HG and 8HG pretrained model from https://github.com/princeton-vl/pytorch_stacked_hourglass#pretrained-models (8HG means the number of stacks is 8).
2) Infer the HPE results on MPII validation set and evaluate the performance for both 2HG and 8HG models.
3) Write visualization code and visualize some human pose inference results.
4) Find some images from the Internet (or your own images) with humans and try to infer human pose using the pretrained model. Visualize the results.

In the previous question, we have been familiar with the 2D human pose estimation for single image input using **top-down** method, i.e., Stacked Hourglass Network. However, introducing the video-based human pose estimation can further improve the accuracy by taking advantage of the "smoothness" in the temporal domain. It is also highly desired that we can generate 3D human poses from 2D pose keypoints. The recent innovation of VideoPose3D (by Facebook) shows that a convolutional architecture can also offer precise control over the temporal receptive field, which they also found beneficial to model temporal dependencies for the task of 3D pose estimation.

In this problem, we will play around with this VideoPose3D. You can first take a look at how VideoPose3D performs in their demo.

(e) Install VideoPose3D

The official VideoPose3D is based on Detectron (Caffe) implementation.

1) Install VideoPose3D using the provided sample codes. (This may take some time ~10 mins)
2) Download the pre-trained Detectron Mask R-CNN 2D keypoint model and extract it to the right repository using the provided sample codes.
3) Download the pre-trained VideoPose model (receptive fields: 243 frames) from COCO & Human3.6M (http://vision.imar.ro/human3.6m/description.php) and extract it to the right repository using the provided sample codes.

(f) Inference and Visualization using the pre-trained Model (10%)

1) Use the provided video and the pre-trained model of **BOTH Stacked Hourglass and Detectron** to do the inference and visualize the 2D human pose estimation results.
2) Infer the 3D keypoints based on the 2D keypoints results from the two methods in 1). (needs to be displayed in Colab with mp4 format, sample images are given below.)

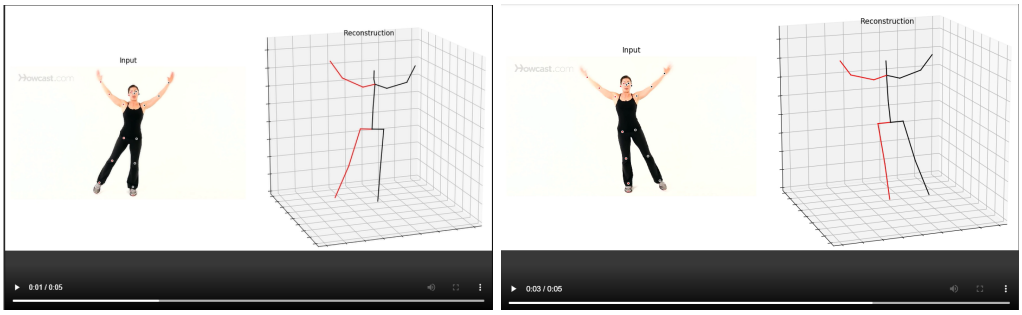

Fig 8: Visualization of VideoPose3D results (left: 2D keypoints, right: 3D keypoints)

3) Besides the quality of the 2D keypoint detector, what other parameters or model architecture do you think that may affect the 3D performance? Explain why. (e.g., receptive fields, numbers of residual blocks, etc.)

Problem 2: Tracktor for Pedestrian Multi-Object Tracking (30%)

Modern multiple object tracking (MOT) systems usually follow the **tracking-by-detection** paradigm. It has (1) a detection model for target localization and (2) an appearance embedding model for data association (3) frame-by-frame association rules. As discussed in the class, Tracktor (https://arxiv.org/pdf/1903.05625.pdf) is a recent MOT method that accomplishes tracking without specifically targeting any of these tasks, in particular, this method performs no training or optimization by exploiting the bounding box regression of an object detector to predict the position of an object in the next frame, thereby converting a detector into a Tracktor. (Tracktor is also the paper reading task in this HW.)

As shown in Fig. 6, first, the regression of the object detector aligns already existing track bounding boxes b_{t-1}^{k} of frame $t-1$ to the object's new position at frame t. The corresponding object classification scores s_{t}^{k} of the new bounding boxes positions are then used to kill potentially occluded tracks. Second, the object detector (or a given set of public detections) provides a set of detections D_{t} of frame t. Finally, a new track is initialized if a detection has no substantial IoU with any bounding box of the set of active tracks $B_{t} = \{b_{t}^{k1}, b_{t}^{k2}, \ldots\}$.

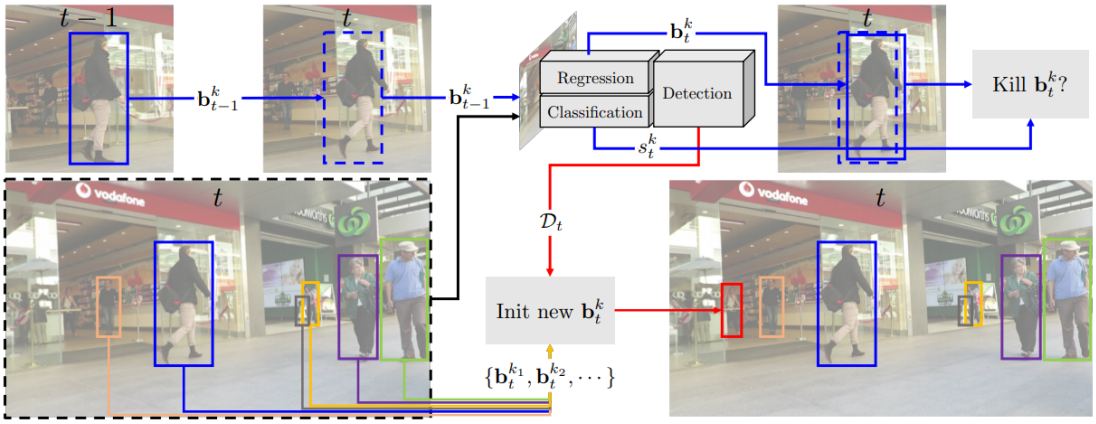

Fig 6: The presented Tracktor accomplishes multi-object tracking only with an object detector and consists of two primary processing steps, indicated in blue and red.

(a) Prepare Codes and MOT-17 Datasets

In this problem, we will use the MOT-17 dataset to train an object detector. The multi-object tracking benchmark MOT-17 (https://motchallenge.net/) consists of several challenging

pedestrian tracking sequences, with frequent occlusions and crowded scenes. Sequences vary in their angle of view, size of objects, camera motion and frame rate.

In order to use the MOT-17 dataset to train both models, we first download and prepare this dataset:

1) Download the source codes from GitHub using the provided sample codes.
2) Download the MOT-17 dataset (~1.8G) and unzip the data to the directory.
3) Visualize some images in the MOT-17 dataset.

Fig. 7: Examples in the MOT-17 dataset

(b) Train and Test the Object Detector (10%)

Remember we've learned multiple object detectors, which are mainly sorted into (1) two-stage detectors, i.e., Faster R-CNN (2) one-stage detectors, i.e, RetinaNet. Successful application of multi-object tracking requires not only a reliable but a real-time detector as well. Here, we adopt the two-stage Faster R-CNN with the backbone of ResNet-50 and Feature Pyramid Network (FPN), which can detect objects of different scales, with some modifications.

1) **(5 points)** Use the provided pretrained Faster R-CNN to further train the model for 27 epochs on MOT-17 dataset. Use the sample codes to evaluate and report the accuracy of Average Precision (AP) on both train set (**val test**).
2) **(5 points)** Randomly select some images and visualize their detection results.

(c) Inference and Visualization (20%)

Based on the (b), the model for object detection is saved in `outputs/` directory. Now we can run the Tracktor to perform the multi-person tracking. The Tracktor can be configured by changing the corresponding `experiments/cfgs/tracktor.yaml` config file. The default configurations runs Tracktor with the FPN object detector are almost same as described in the paper except the Re-identification model is turned off (do_reid=False, **load_results=True**).

1) **(5 points)** Run the inference `experiments/scripts/test_tracktor.py` using the MOT-17 train set input. The tracking results are logged in the corresponding `outputs/` folder. Open one of the generated results, explain what are the first six

values generated in each line? (i.e., frame_id, bounding box (xywh/xyxy?), confidence, track_id, etc.). Plot the values on the corresponding images and show the video results.

Fig. 7: Visualization of Tracktor tracking results

2) **(5 points)** Evaluate the performance using `test_tracktor.py` and report the following metrics: MOTA, MOTP, IDF1, FP. (****Hints: These metrics have already been logged in Colab outputs from the previous problems. You can just copy down here.****)
3) **(10 points)** Run the inference `test_tracktor.py` with the changed configurations in `experiments/cfgs/tracktor.yaml` and evaluate the performance:
   tracktor/tracker:
   - detection_person_thresh (FRCNN score threshold for detections): 0.5
   - detection_nms_thresh (NMS threshold for detection): 0.3
   - number_of_iterations (maximal number of iterations): 100
   - max_features_num (How much last appearance features are to keep): 10
   - motion_model (motion model settings, mentioned in 2.3): disabled

   Feel free to change at least **three** hyperparameters (can be from detection or tracking). **Discuss** how these changes may affect the tracking performance based on MOTA.

Problem 3: Radar Object Detection (20%)

Image-based object detection that has been explored in the last homework is an important task for computer vision related areas. In the autonomous driving community, however, radar is another common sensor that can provide accurate ranging information and is robust to different driving scenarios.

The sensor platform for the CRUW dataset contains a pair of stereo cameras and two 77GHz FMCW radar antenna arrays. The sensors, assembled and mounted together as shown in Fig. 4, are well-calibrated and synchronized.

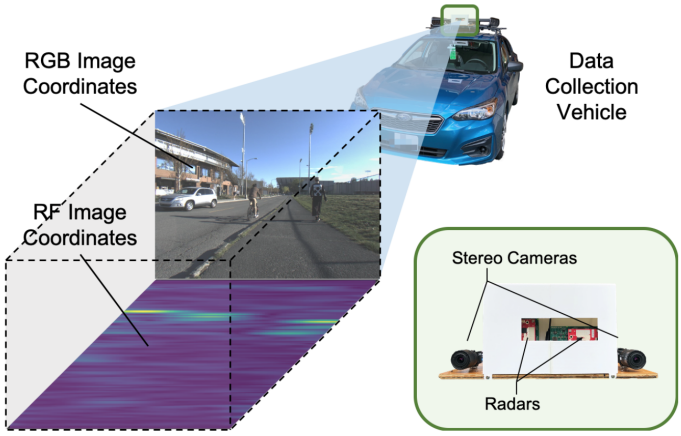

Fig. 4: Sensor platform for the CRUW dataset.

The CRUW dataset contains more than 3 hours with 30 FPS (about 400K frames) of synchronized camera images and radar-frequency (RF) images under different driving scenarios, including campus road, city street, highway, parking lot, etc. Some sample data are shown in Fig. 5.

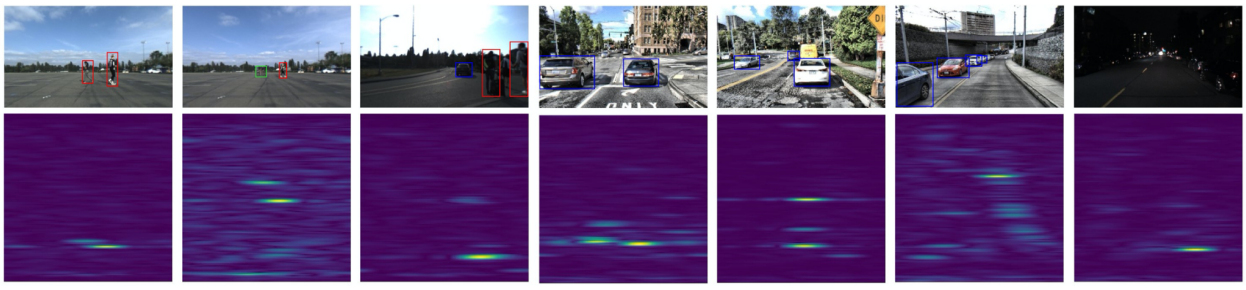

Fig. 5: Some example data in the CRUW dataset.

(a) Prepare CRUW_MINI Dataset

In this problem, we will use a subset of CRUW called CRUW_MINI for the experiments. The CRUW_MINI includes 10 sequences, with about 12K frames in total. The scenarios include parking lot (PL), campus road (CR), city street (CS), and highway (HW). The visibility conditions include NORMAL, BLUR, and NIGHT.

In order to use CRUW_MINI dataset for radar object detection, we first download and prepare the dataset:

1) Download CRUW_MINI dataset and devkit using the sample code.
2) Install cruw-devkit and visualize some sample data.

(b) Train RODNet using CRUW_MINI (10%)

To train the RODNet, we have the definition of ConfMap as the supervision. So, we first prepare the dataset and generate ConfMaps according to the annotations. After the data and supervision are prepared, RODNet can be trained afterwards.

1) Based on the provided RF images, can you think of an object detection scheme (without using machine learning training, to identify different types of objects (pedestrian, cyclist and car)?
2) Run `prepare_data.py` for generating ConfMaps. Visualize some ConfMaps with the corresponding RGB images and radar images.
3) Train the RODNet using the configuration named `config_rodnet_cdc_win16.py` for at least 5 epochs. Change the `win_size` from 16 to 1. Train the RODNet again for at least 5 epochs.

(c) Inference and Visualization (5%)

1) Infer radar object detection results using the trained model with `win_size` = 1 and 16. Visualize your prediction results.
2) Download the pretrained weights (`n_epoch`=50) and infer the radar object detection. Visualize the prediction results.

(d) Discussions (5%)

1) What's the detection result difference between `win_size` = 1 and 16? Why do we need to input clips of frames instead of input single frames like image-based object detection?
2) During your experiments, list some failure cases. Explain the probable reason for the failures and give some ideas for improvement.

Problem 4: Paper Reading (25%)

Title: Tracking without bells and whistles
Authors: Philipp Bergmann, Tim Meinhardt, Laura Leal-Taixe
Abstract: The problem of tracking multiple objects in a video sequence poses several challenging tasks. For tracking-by-detection, these include object re-identification, motion prediction and dealing with occlusions. We present a tracker (without bells and whistles) that accomplishes tracking without specifically targeting any of these tasks, in particular, we perform no training or optimization on tracking data. To this end, we exploit the bounding box regression of an object detector to predict the position of an object in the next frame, thereby converting a detector into a Tracktor. We demonstrate the potential of Tracktor and provide a new state-of-the-art on three multi-object tracking benchmarks by extending it with a straightforward re-identification and camera motion compensation. We then perform an analysis on the performance and failure cases of several state-of-the-art tracking methods in comparison to our Tracktor. Surprisingly, none of the dedicated tracking methods are considerably better in dealing with complex tracking scenarios, namely, small and occluded objects or missing detections. However, our approach tackles most of the easy tracking scenarios. Therefore, we motivate our approach as a new tracking paradigm and point out promising future research directions. Overall, Tracktor yields superior tracking performance than any current tracking method and our analysis exposes remaining and unsolved tracking challenges to inspire future research directions.