Prototxt

Qingfu Wan

strawberryfgalois@gmail.com

1. Introduction

This is a sketch about train/test prototxt.

2. Training pipeline

The basic structure is as follows:

1 Generate a random index for training

2 Read annotation including

- bbx
- center of person center_x, center_y
- person scale (crop bbx size)
- 2d ground truth on raw image
- camera frame 3d ground truth (gt_3d_mono of H36M matlab)

3 Convert 32 joints to usable 16 joints

4 Generate random augmentation of image and corresponding augmented 2d joints label (2d in cropped bbx)

5 Generate augmented 3d based on augmented 2d and fetched camera intrinsic parameters.

6 Convert augmented 3d to normalized root-relative joint depth. Normalized to [-1, 1]

7 Generate 2d heatmap ground truth using C2F or simple baseline render strategy.

8 Generate 3d heatmap ground truth based on 2d in bbx and normalized depth. 2d in bbx is in range [0, 1], and normalized depth is in range [-1, 1]

9 2-stage Hourglass

10 Enforce euclidean loss on 2d and 3d heatmap

11 Integral operation on 3D heatmap

- ▶ Normalize 3d heatmap response of each joint and make it sum up to 1.0.
  - If you use **DeepHumanModelNorm3DHM"**, there is no need to scale the 3d heatmap before hand.
  - If you use softmax normalization *i.e*. **DeepHumanModelSoftmax3DHM"**, you need to first scale the 3d heatmap by a constant value *e.g*. **30.0**, or **50.0**. This is discussed in another pdf **code.pdf**. It is easy to find this little trick by plotting softmax function in Google. However, if you only use integral loss w/o heatmap supervision, it is totally a different story caz the semantic meaning of 3D heatmap does not hold.
- ▶ Flat → (optionally) scale → reshape → slice to cubes for each joint individually
- ▶ For each joint, perform integral operation along X, Y, Z axis on this joint's cube. (Call **DeepHumanModelIntegralX**→ **DeepHumanModelIntegralY**→ **DeepHumanModelIntegralZ**→ **DeepHumanModelIntegralVector**)
- ▶ Concat integral X, Y of all joints to get predicted 2d joints (image space)
- ▶ Concat integral Z of all joints to get predicted depth joints

12 Convert predicted normalized depth to depth in global space by calling **DeepHumanModelConvertDepth**

13 Use camera intrinsic parameter, predicted 2d joints and predicted unnormalized depth in camera frame to get predicted 3d joints in global camera frame (w/ the help of global root depth as known constant) by calling **DeepHumanModelH36MGenPredMono3D**

14 For 3d heatmap argmax operation, repeat step 12 and 13.

15 (*optional*) Possibly enforce other losses.

Note that other losses e.g. *JS regularization loss, smooth L1 loss, adaptive weight euclidean loss, integral loss are included in prototxt with loss_weight set to 0.0.*

16 Output blobs *e.g*.

- 2d heatmap *heatmap*
- 3d heatmap *heatmap2*
- integral 2d joints *pred_joint_2d_s2_int* (stage 2), *pred_joint_2d_s1_int* (stage 1)
- argmax 2d joints *pred_joint_2d_s2_max* (argmax on 3d heatmap), *pred_joint_2d_s2_int* (stage 1)
- integral depth of joints *pred_depth_s2_int*
- argmax depth of joints *pred_depth_s2_max*
- augmented 3d *aug_3d*
- predicted camera frame coordiante *pred_joint_global_s2_int* of integral operation
- predicted camera frame coordiante *pred_joint_global_s2_max* of argmax operation

for debugging?

17 Overlay predicted 2d joints on image and save visualized image to folder

3. Testing pipeline

1 Generate sequential testing index by calling **GenSequentialIndex** or read from file that contains all the index to be tested **ReadIndexFromFile**

2 Same as train phase.

3 Same as train phase.

4 Set augmentation factors to 0.0 (no aug) and call the same function during train phase

5 Deprecated.

6-17 Same as train phase.

4. Losses

About usage of losses and what they really mean, see prototxt and **code.pdf**.

5. MPJPE

We care about MPJPE (mean per joint position error) unit: *mm*. As integral loss is only implemented but not used during training, it is suggested to watch *error(mm)_3d_s2_max* as it reflects average joint error using argmax on 3d heatmap. During inference, the integral MPJPE error is *error(mm)_3d_s2_int*. I should stress again integral here is only used when testing, not when training,

6. Misc

Back in 2015, Caffe was a pop star. But it has turned out that not too many people have the patience and vision to use Caffe efficiently and conveniently. It is actually super easy to register a new layer, just like when you create a visual studio project and write header / source files. About backward computation, chain rule is simple and easy. The only downside is that on windows, it takes some time (sometimes half an hour) to compile. Nonetheless with ubuntu sudo make all -j128, it takes less than a minute to compile. Compile is once and for all, caz you will have super fast running speed during training or inference.

I know it's a bit digressive. Anyway, I guess this repository is just for personal backup.