Custom caffe layers

Qingfu Wan

strawberryfgalois@gmail.com

1. DeepHumanModel

1.1. DeepHumanModelArgmaxHMLayer

Source file deep_human_model_argmax_2d_hm_layer.cpp
Input N×J×H×W 2d heatmap. J is number of joints
Output N× (J×2). predicted 2d joints.
Functionality Argmax on 2d heatmap.

1.2. DeepHumanModelConvert2D

Souce file deep_human_model_convert_2d_layer.cpp
Input N× (16 × 2) or N× (32 × 2) 2D joints
Output N× (32 × 2) or N× (16 × 2) 2D joints
Functionality Converts full 32 joints ⇔ usable 16 joints. (2D)

1.3. DeepHumanModelConvert3D

Souce file deep_human_model_convert_3d_layer.cpp
Input N× (16 × 3) or N× (32 × 3) 3D joints
Output N× (32 × 3) or N× (16 × 3) 3D joints
Functionality Converts full 32 joints ⇔ usable 16 joints. (3D)

1.4. DeepHumanModelConvertDepth

Source file deep_human_model_convert_depth_layer.cpp
Input
1. N× (J × 3) 3D joints or N× J normalized depth of joints
2. N× (J × 3) 3D gt.

Output N× J Normalized depth of joints or original depth of joints (denormalize).

Functionality Conversion of normalized depth ⇔ depth of camera frame coordinates.

Param
1. joint_num
2. depth_lb: depth lower bound of voxelized space
3. depth_ub: depth upper bound
4. root_joint_id

Disclaimer Lower/upper bound follows definition in c2f MatLab code **"limits", "Zcen"** field.

1.5. DeepHumanModelGen3DHeatmapInMoreDetailV3

Source file deep_human_model_gen_3d_heatmap_in_more_detail_v3_layer.cpp
Input
1. N× (J×3) camera frame 3d gt
2. N× (J×2) gt 2d in cropped bbx

Output N× (J×D) × H × W where D is depth dimension (16/32/64)

Functionality Render 3D gaussian ground truth. Follows closely with c2f Torch code & Yichen Wei simple baseline PyTorch code.

Param
1. depth_dims: depth dimension
2. map_size: H (W), default 64
3. crop_size: default 256 (See baseline PyTorch lib/dataset/JointsDataset.py generate_target function *self.image_size*)
4. render_sigma: default = 2, See lib/core/config.py *POSE_RESNET.SIGMA = 2*
5. stride: default = 4, See above 4.
6. x_lower_bound:
7. x_upper_bound:
8. y_lower_bound:
9. y_upper_bound:
10. z_lower_bound:
11. z_upper_bound:
12. output_res: Final depth resolution. Default = 64. See c2f PyTorch drawGaussian3D function.

1.6. DeepHumanModelH36MChaGenJointFrXYZHeatmap

Source file deep_human_model_h36m_cha_gen_joint_fr_xyz_heatmap_layer.cpp
Input N× (J×D) × H × W 3D heatmap
Output N× (J×3) predicted 3d joints
Functionality argmax operation on 3d heatmap
Param
1. depth_dims
2. map_size
3. x_lb

4. x_ub
5. y_lb
6. y_ub
7. z_lb
8. z_ub
9. joint_num

1.7. DeepHumanModelH36MGenAug3D

Source file deep_human_model_h36m_gen_aug_3d_layer.cpp
Input
1. N× (J×2) augmented 2d label
2. camera frame gt 3d (for getting root depth gt)
3. bbx_x1
4. bbx_y1
5. bbx_x2
6. bbx_y2
7. image_index (for indexing camera parameter file)

Output N× (J×3) augmented 3d label
Functionality Get augmented 3d label from augmented 2d label & intrinsic camera parameters
Param
1. joint_num
2. camera_parameters_prefix: the prefix to camera param file
3. crop_bbx_size

1.8. DeepHumanModelH36MGenPredMono3D

Source file deep_human_model_h36m_gen_pred_mono_3d_layer.cpp
Input
1. N× (J×2) predicted 2d in cropped bounding box [0, 1]
2. N× J predicted depth(in camera frame)
3. bbx_x1
4. bbx_y1
5. bbx_x2
6. bbx_y2
7. image_index

Output N× (J×3) predicted 3d joints in camera frame from 2.5D $\rightarrow$ 3D
Functionality Local $\rightarrow$ global coordinate
Param
1. joint_num
2. camera_parameters_prefix: the prefix to camera param file

1.9. DeepHumanModelIntegralVector

Source file deep_human_model_integral_vector_layer.cpp
Input N×C (C depends on the axis integral is performed along, for Z. C=depth_dims; for X or Y, C=H(W))
Output N×1
Functionality returns integral position for X (or Y or Z) $\sum_{i=0}^{C-1} prob(i) \times position(i)$
Param
1. dim_lb
2. dim_ub

1.10. DeepHumanModelIntegralX

Source file deep_human_model_integral_x_layer.cpp
Input N×C×H×W (C is depth_dims) 3D heatmap
Output N×W (a row vector)
Functionality Integral to get X

1.11. DeepHumanModelIntegralY

Source file deep_human_model_integral_y_layer.cpp
Input N×C×H×W (C is depth_dims) 3D heatmap
Output N×H (a column vector)
Functionality Integral to get Y

1.12. DeepHumanModelIntegralZ

Source file deep_human_model_integral_z_layer.cpp
Input N×C×H×W (C is depth_dims) 3D heatmap
Output N×C (a vector along depth axis)
Functionality Integral to get Z

1.13. DeepHumanModelNorm3DHM

Source file deep_human_model_norm_3d_hm_layer.cpp
Input N×(J×D)×H×W (D=depth_dims) 3D heatmap
Output N× (J×D)×H×W. Normalized 3D heatmap
Functionality Normalize 3d heatmap to make sure probability sums up to 1.0
Formula: $\frac{prob(i)}{\sum_{i \in VoxelSpace} prob(i)}$
Misc See the operation with outliers.
Param
1. joint_num
2. depth_dims
3. hm_threshold: only normalize responses larger than a given threshold

1.14. DeepHumanModelNormalizationResponseV0

Source file deep_human_model_normalization_response_v0_layer.cpp
Input N×J×H×W (J: number of joints) 2D heatmap
Output N×J×H×W normalized 2D heatmap
Functionality Normalize 2d heatmap to make response values sum up to 1.0
Formula: $\frac{prob(i)}{\sum_{i \in ImageSpace} prob(i)}$
Param
1. hm_threshold: see last section

1.15. DeepHumanModelNumericalCoordinateRegression

Source file deep_human_model_numerical_coordinate_regression_layer.cpp
Input N×J×H×W normalized 2d heatmap

Output N×(J×2) predicted 2d joints

Functionality integral on 2d heatmap → 2d joints

Formula: $\sum_{i \in ImageSpace} prob(i) \times position(i)$

1.16. DeepHumanModelOutputHeatmapSepChannel

Source file deep_human_model_output_heatmap_sep_channel_layer.cpp

Input

1. N×J×H×W 2d heatmap
2. image index

Output Nothing

Functionality Output heatmap of all joints to separate folders *e.g*. joint 3 to folder "3/"

Param

1. save_size: resolution of saved heatmap
2. heatmap_size: bottom blob heatmap resolution
3. save_path: prefix of saved heatmap
4. joint_num
5. output_joint_X: whether to output joint X (a more convenient way is to use repeated blob, like DeepLab v2 caffe **ImageSegData** repeatead field **scale_factors** in *TransformationParameter*

1.17. DeepHumanModelOutputJointOnSkeletonMapH36M

Source file deep_human_model_output_joint_on_skeleton_map_h36m_layer.cpp

Input

1. N×3×H×W image to be overlaid on
2. image index
3. N×(J×2) predicted 2d joints
4. N×(J×2) gt 2d joints

Output Nothing

Functionality Overlaid predicted 2d joints on raw image and save to file

Param

1. use_raw_rgb_image: if true load image from disk. Default = false
2. show_gt: whether to show gt joints simultaneously using another color encoding. For detailed color encoding, see h36m.h
3. save_path: prefix to save overlaid image
4. save_size: resolution of overlaid image
5. image_source: prefix of raw image (if load from disk)
6. skeleton_size: bottom image resolution
7. show_skeleton: whether to load bottom image. Default : true
8. circle_radius: OpenCV **circle** radiums argument
9. line_width: OpenCV **line** line width argument
10. is_c2f: whether c2f definition (17 joints, different joint/bone, supported by my code)

1.18. DeepHumanModelSoftmax3DHM

Source file deep_human_model_softmax_3d_hm_layer.cpp

Input N×(J×D)×H×W (D=depth_dims) 3D heatmap

Output N× (J×D)×H×W. Normalized 3D heatmap

Functionality Normalize 3d heatmap to make sure probability sums up to 1.0 (softmax)

Formula: $\frac{e^{prob(i)}}{\sum_{i \in VoxelSpace} e^{prob(i)}}$

Small tricks You have to multiply heatmap by a factor *e.g*. 30, *e.g*. 50 before softmax otherwise the contribution of each pixel is almost the same. Plot softmax in Google for more details. If you only use integral loss to supervise the training w/o heatmap loss, it's not necessary, as done in **Integral Human Pose Regression**. The reason is due, in large part to that learnt feature map does not necessarily have the semantic meaning of heatmap. However, if you want to use both heatmap loss and integral loss, learnt map is heatmap that lies within range [0, 1]. That said, scaling heatmap values before softmax is **THE CRUX!!!**. This is a crucial trick not revealed in the paper or code of **Integral Human Pose Regression**.

Param

1. joint_num
2. depth_dims

1.19. DeepHumanModelSoftmaxHM

Source deep_human_model_softmax_hm_layer.cpp

Input N×J×H×W unnormalized 2d heatmap

Output N×J×H×W normalized 2d heatmap

Functionality Softmax normalization on 2d heatmap

Misc The scale factor you need to use here is slightly different from 3D counterpart. You'll have to find the exact α such that after multiply 2d heatmap by α, softmax normalization results in propitious weight of each 2d pixel.

2. Operations

2.1. AdaptiveWeightEucLoss

Source file adaptive_weight_euc_loss_layer.cpp

input D is dimension of flattened pred/gt vector, M is number of losses

1. Loss 0 prediction blob N×D
2. Loss 0 ground truth blob N×D

...

2 * M - 1: Loss M - 1 prediction blob N×D

2 * M : Loss M -1 ground truth blob N×D

Output: total euclidean loss of all M losses

Functionality Balance average magnitude of each loss.

Disclaimer Designed to ease tuning weights between 2d heatmap and 3d heatmap. Not used anywhere during the entire training process. Left as future work. Similar idea was rejected recently. Trivial.

2.2. AddVectorByConstant

Source file add_vector_by_constant_layer.cpp
Input N×D flattened vector
Output N×D input vector added by a constant scalar
Functionality Add vector by a constant value
Param add_value: the constant value to be added

2.3. AddVectorBySingleVector

Source file add_vector_by_single_vector_layer.cpp
Input
1. N×D vector **A**
2. N×D vector **B**
Output: vector **C=A+B** element-wisely
Functionality element-wisely add two vectors

2.4. CrossValidationRandomChooseIndex

Source file cross_validation_random_choose_index_layer.cpp
Input M is number of different training split sources.
1. N×1: image index of source 1
2. N×1: image index of source 2
...
M: N×1: image index of source M
Output N×1: randomly selected image index
Functionality For each sample in the mini-batch, select a index from M different training sources. For instance, you want to fuse **H36M**, **MPII**, **LSP** and **Surreal** for pretraining 2D heatmap, all you have to do is to generate 4 index arrays, and then randomly select from these four arrays.

2.5. GenHeatmapAllChannels

Source file gen_heatmap_all_channels_layer.cpp
Input 2d gt in bbx [0, 1]
Output rendered gaussian 2d heatmap ground truth
Functionality Implement ground truth heatmap render of
1. c2f
2. simple baseline for human pose estimation and tracking
See their code respectively for details
Param
1. gen_size: heatmap resolution. Default 64
2. render_sigma: same as that in **DeepHumanModel-Gen3DHeatmapInMoreDetailV3**
3. all_one: whether to use binary heatmap classification ground truth as in

- G-RMI
- Integral human pose regression **H2** and **I2**

4. use_cpm_render: whether to use CPM caffe render in their data_transformer.cpp
5. use_baseline_render: whether to use simple baseline renderer in lib/datasets/JointsDataset.py
6. crop_size: See **DeepHumanModel-Gen3DHeatmapInMoreDetailV3**
7. stride: See **DeepHumanModel-Gen3DHeatmapInMoreDetailV3**

2.6. GenRandIndex

Source file gen_rand_index_layer.cpp
Input None
Output N×1: randomly generated index
Functionality Generate random index between valid range for training/testing.
Param
1. index_lower_bound
2. index_upper_bound
3. batch_size: This can be parsed from some specific blobs, say, gt 2d blob
4. missing_index_file: The path of file that stores all the invalid index. Invalid index should be deprecated for any use.
5. rand_generator_option: integer in range [0, 2] Three different random number generation methods. See code for details.

2.7. GenSequentialIndex

Source file gen_sequential_index_layer.cpp
Input None
Output N×1 sequential index for training/testing
Functionality Read current index from file, and then add one → mod sample_num to get new index → store to file.
Param
1. batch_size
2. current_index_file_path: the file that indicates current index. I usually use cur_train_id.txt in **train phase**, and cur_test_id.txt in **test phase**
3. num_of_samples: total number of samples
4. start_index: starting index *e.g.* MPII train is 0-25924, val is 25924-28881, start_index for test phase should therefore be 25924, num_of_samples should be 2958.

2.8. GenUnifiedDataAndLabel

Source file gen_unified_data_and_label_layer.cpp
Input
1. image index
2. center_x: center of person (x) on raw image (not bbx)
3. center_y: center of person (y) on raw image
4. scale_provided: provided scale of person divided by 200.0 (/200.0: for historical reasons MPII)
5. N×(J×2) gt_joint_2d_raw: ground truth 2d joint on raw image
Output

1. N×3×H×W: transformed data (augmented followed by mean subtraction and division by 256.0)

2. N×(J×2): transformed 2d joint (location in bbx) label

Functionality Random scale/rotation/flip augmentation of image & 2d gt.

Misc Original **CPMDataLayer** or the one used in **GNet** loads an offline generated LMDB file, and tries to parse annotation & image from this LMDB. All my layer does is to get rid of time-consuming LMDB generation part.

Key online data augmentation layer

Param

1. crop_size_x: crop bbx width

2. crop_size_y: crop bbx height

3. file_name_file_prefix: prefix of the file that stores image path

4. minus_pixel_value: Mean value to be subtracted for each channel. Default 128.0

5. stride: $\frac{image\_size}{heatmap\_size}$

6. max_rotate_degree: can be set to 0.0 during testing

7. scale_prob: probability of performing scale aug. Can be set to $\leq$ 0.0 during testing.

8. scale_min

9. scale_max

10. target_dist: See CPM caffe repo for details.

11. center_perterb_max: Default 0.0. Translation augmentation is prohibited as for datasets *e.g*. MPII, center of person is important for isolating between person of interest and other people.

12. do_clahe

13. put_gaussian: CPM residual. Set to false in my experiments.

14. transform_body_joint: whether to swap 2d gt of symmetric joints during flip augmentation

15. num_parts: number of joints (definition slightly different from **CPM** and **GNet**)

16. flip_prob: Set to $\leq$ during inference to prevent flipping.

2.9. Joint3DSquareRootLoss

Source file joint_3d_square_root_loss_layer.cpp

Input

1. predicted 3d joints

2. ground truth 3d joints

Output Average joint error (MPJPE) unit: *mm*

Functionality: Display average joint error caz caffe euclidean loss is nowhere close to lucid.

Param joint_num

2.10. JSRegularizationLoss

Source file js_regularization_loss_layer.cpp

Input

1. N×C×H×W predicted map

2. N×C×H×W ground truth map

Output Jenson Shannon regularization loss

Functionality Compute JS entropy loss

Note only pixel whose value $\geq$ 0 on both pred and gt map is accounted for final loss.

Param min_eps: threshold of heatmap value $\neq$ 0

2.11. MulRGB

Source file mul_rgb_layer.cpp

Input: N×3×H×W original RGB blob

output: N×3×H×W multiplied RGB

Functionality Scale the RGB image by a constant value

Param mul_factor: the constant multiplier factor

Usage Online aug layer or CPM data load layer **CPMDataLayer** generates a blob by mean subtraction & division by 256.0. Sometimes it is required to × 256 or / 256 before visualization.

Note This layer can be totally replaced by **ScaleVectorLayer**, **ReshapeLayer** and **FlattenLayer**.

2.12. OutputBlob

Source file output_blob_layer.cpp

Input: N×D, D is dimension of flattened vector

Output: Nothing

Functionality: Output values of a blob to disk file.

It is especially important while debugging to see if the ranges of some blobs fall onto the valid range or not.

Param

1. save_path: prefix to save the blob

2. blob_name: name of blob followed by save_path

3. if_per_section_output: if per section output (Rows×Cols) *e.g*. Say you want to output a 3D heatmap that is of size (J×D)×H×W

4. per_section_row_num: start a new line per this number rows.

5. per_section_col_num

2.13. OutputHeatmapOneChannel

Source file output_heatmap_one_channel_layer.cpp

Input:

1. N×1×H×W heatmap of one specific joint

2. image index **Output**: Nothing

Functionality: Output heatmap of one joint to disk file

Param

1. save_path

2. save_size

3. heatmap_size

2.14. ReadBlobFromFileIndexing

Source file read_blob_from_file_indexing_layer.cpp

Input image index

Output N×D: flattened vector of fetched blob

Functionality Read files corresponding to a sample index

Param
1. file_prefix: prefix of file to be fetched
2. num_to_read: dimension D of the fetched data blob

2.15. ReadBlobFromFile

Source file read_blob_from_file_layer.cpp
Input Nothing
Output N×D: flattened vector of fetched blob
Functionality Read file from a specific file path **Param**
1. file_prefix: prefix of file to be fetched
2. num_to_read: dimension D of the fetched data blob
3. batch_size

2.16. ReadImageFromFileName

Source file read_image_from_file_name_layer.cpp
Input Image index
Output N×3×H×W
Functionality Read image from a file. **Param**
1. resize_size
2. pad_square: whether to pad the fetched image to a square
3. channel_num: 1 or 3 (gray-scale or RGB)
4. file_name_file_prefix: image path prefix
5. pad_to_a_constant_size_before_resize
6. pad_to_constant_size

2.17. ReadImageFromImagePathFile

Source file read_image_from_image_path_file_layer.cpp
Input Nothing
Output N×3×H×W
Functionality Read image from a file that contains image path of all samples.
Param
1. image_path_file_path: the file that stores image path of all samples
2. batch_size
3. current_index_file_path: file that stores current sample index
4. num_of_samples
5. resize_image_size: resize fetched image to this resolution and feed to top blob

2.18. ReadImage

Source file read_image_layer.cpp
Input image index
Output N×3×H×W
Functionality Read image.
Param
1. read_path: the path for reading image
2. resize_size:
3. zero_pad: zero padding for image name
4. image_suffix: file extension .jpg or .png

2.19. ReadIndexFromFile

Source file read_index_from_file_layer.cpp
Input None
Output Image index
Functionality Read sample index from a file. For instance, you only want to train a subset or test a subset.
Param
1. index_file_path: the file that stores the index
2. batch_size
3. current_index_file_path: current sample index
4. num_of_samples

2.20. ScaleVector

Source file scale_vector_layer.cpp
Input N×D flattened vector
Output N×D scaled vector added by a constant scalar
Functionality Scale vector by a constant value
Param scale_factor: the constant value to be multiplied

3. Misc

If you are skilled in C++, congrats! You are among the few who write C++ faster than most people write Python. Most importantly, your code runs 100 times faster. I have used Keras for six months, PyTorch/Tensorflow for a few days. None of them reaches the speed of caffe.

Anyway, it takes time to master C++. That's the only cost.

4. Limitation

- Ambiguity in param field.
- A more convenient way to organize data is to generate one **single** annotation file covering all possible annotation *e.g*. ground truth 2d, ground truth 3d, bbx *etc*. per sample, and store them to a **single** folder on SSD.
- A layer that fetches all annotation for each sample instead of reading seperating files containing different annotation of the same sample.
- The current integral implementation deals with only one joint. Yet it is very easy to modify it to cope with all joints.