

# Custom caffe layers

Qingfu Wan

strawberryfgalois@gmail.com

## 1. DeepHumanModel

### 1.1. DeepHumanModelArgmaxHMLayer

**Source file** deep\_human\_model\_argmax\_2d\_hm\_layer.cpp

**Input**  $N \times J \times H \times W$  2d heatmap. J is number of joints

**Output**  $N \times (J \times 2)$ . predicted 2d joints.

**Functionality** Argmax on 2d heatmap.

### 1.2. DeepHumanModelConvert2D

**Source file** deep\_human\_model\_convert\_2d\_layer.cpp

**Input**  $N \times (16 \times 2)$  or  $N \times (32 \times 2)$  2D joints

**Output**  $N \times (32 \times 2)$  or  $N \times (16 \times 2)$  2D joints

**Functionality** Converts full 32 joints  $\Leftrightarrow$  usable 16 joints. (2D)

### 1.3. DeepHumanModelConvert3D

**Source file** deep\_human\_model\_convert\_3d\_layer.cpp

**Input**  $N \times (16 \times 3)$  or  $N \times (32 \times 3)$  3D joints

**Output**  $N \times (32 \times 3)$  or  $N \times (16 \times 3)$  3D joints

**Functionality** Converts full 32 joints  $\Leftrightarrow$  usable 16 joints. (3D)

### 1.4. DeepHumanModelConvertDepth

**Source file** deep\_human\_model\_convert\_depth\_layer.cpp

**Input**

1.  $N \times (J \times 3)$  3D joints or  $N \times J$  normalized depth of joints

2.  $N \times (J \times 3)$  3D gt.

**Output**  $N \times J$  Normalized depth of joints or original depth of joints (denormalize).

**Functionality** Conversion of normalized depth  $\Leftrightarrow$  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. DeepHumanModel-Gen3DHeatmapInMoreDetailV3

**Source file** deep\_human\_model\_gen\_3d\_heatmap\_in\_more\_detail\_v3\_layer.cpp

**Input**

1.  $N \times (J \times 3)$  camera frame 3d gt

2.  $N \times (J \times 2)$  gt 2d in cropped bbox

**Output**  $N \times (J \times D) \times H \times W$  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**  $N \times (J \times D) \times H \times W$  3D heatmap

**Output**  $N \times (J \times 3)$  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.  $N \times (J \times 2)$  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**  $N \times (J \times 3)$  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.  $N \times (J \times 2)$  predicted 2d in cropped bounding box [0, 1]
2.  $N \times J$  predicted depth(in camera frame)
3. bbx\_x1
4. bbx\_y1
5. bbx\_x2
6. bbx\_y2
7. image\_index

**Output**  $N \times (J \times 3)$  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**  $N \times C$  (C depends on the axis integral is performed along, for Z,  $C=\text{depth\_dims}$ ; for X or Y,  $C=H(W)$ )

**Output**  $N \times 1$

**Functionality** returns integral position for X (or Y or Z)  
 $\sum_{i=0}^{C-1} \text{prob}(i) \times \text{position}(i)$

**Param**

1. dim\_lb
2. dim\_ub

### 1.10. DeepHumanModelIntegralX

**Source file** deep\_human\_model\_integral\_x\_layer.cpp

**Input**  $N \times C \times H \times W$  (C is depth\_dims) 3D heatmap

**Output**  $N \times W$  (a row vector)

**Functionality** Integral to get X

### 1.11. DeepHumanModelIntegralY

**Source file** deep\_human\_model\_integral\_y\_layer.cpp

**Input**  $N \times C \times H \times W$  (C is depth\_dims) 3D heatmap

**Output**  $N \times H$  (a column vector)

**Functionality** Integral to get Y

### 1.12. DeepHumanModelIntegralZ

**Source file** deep\_human\_model\_integral\_z\_layer.cpp

**Input**  $N \times C \times H \times W$  (C is depth\_dims) 3D heatmap

**Output**  $N \times C$  (a vector along depth axis)

**Functionality** Integral to get Z

### 1.13. DeepHumanModelNorm3DHM

**Source file** deep\_human\_model\_norm\_3d\_hm\_layer.cpp

**Input**  $N \times (J \times D) \times H \times W$  (D=depth\_dims) 3D heatmap

**Output**  $N \times (J \times D) \times H \times W$ . Normalized 3D heatmap

**Functionality** Normalize 3d heatmap to make sure probability sums up to 1.0

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

**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**  $N \times J \times H \times W$  (J: number of joints) 2D heatmap

**Output**  $N \times J \times H \times W$  normalized 2D heatmap

**Functionality** Normalize 2d heatmap to make response values sum up to 1.0

Formula:  $\frac{\text{prob}(i)}{\sum_{i \in \text{ImageSpace}} \text{prob}(i)}$

**Param**

1. hm\_threshold: see last section

### 1.15. DeepHumanModelNumericalCoordinateRegression

**Source file** deep\_human\_model\_numerical\_coordinate\_regression\_layer.cpp

**Input**  $N \times J \times H \times W$  normalized 2d heatmap

**Output**  $N \times (J \times 2)$  predicted 2d joints

**Functionality** integral on 2d heatmap  $\rightarrow$  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.  $N \times J \times H \times W$  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** repeated field **scale\_factors** in *TransformationParameter*)

### 1.17. DeepHumanModelOutputJointOnSkeleton-MapH36M

**Source file** deep\_human\_model\_output\_joint\_on\_skeleton\_map\_h36m\_layer.cpp

**Input**

1.  $N \times 3 \times H \times W$  image to be overlaid on
2. image index
3.  $N \times (J \times 2)$  predicted 2d joints
4.  $N \times (J \times 2)$  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** radius 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**  $N \times (J \times D) \times H \times W$  ( $D = \text{depth\_dims}$ ) 3D heatmap

**Output**  $N \times (J \times D) \times H \times W$ . 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**  $N \times J \times H \times W$  unnormalized 2d heatmap

**Output**  $N \times J \times H \times W$  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  $\alpha$  such that after multiply 2d heatmap by  $\alpha$ , 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  $N \times D$
2. Loss 0 ground truth blob  $N \times D$
- ...
- 2 \* M - 1: Loss M - 1 prediction blob  $N \times D$
- 2 \* M : Loss M -1 ground truth blob  $N \times D$

**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**  $N \times D$  flattened vector

**Output**  $N \times D$  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.  $N \times D$  vector **A**

2.  $N \times D$  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.  $N \times 1$ : image index of source 1

2.  $N \times 1$ : image index of source 2

...

M:  $N \times 1$ : image index of source M

**Output**  $N \times 1$ : 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 bbox [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**  $N \times 1$ : 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**  $N \times 1$  sequential index for training/testing

**Functionality** Read current index from file, and then add one  $\rightarrow$  mod sample\_num to get new index  $\rightarrow$  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 bbox)

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.  $N \times (J \times 2)$  gt\_joint\_2d\_raw: ground truth 2d joint on raw image

**Output**

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

2.  $N \times (J \times 2)$ : 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:  $\frac{\text{image\_size}}{\text{heatmap\_size}}$
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.  $N \times C \times H \times W$  predicted map
2.  $N \times C \times H \times W$  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**:  $N \times 3 \times H \times W$  original RGB blob

**output**:  $N \times 3 \times H \times W$  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 **CPM-DataLayer** generates a blob by mean subtraction & division by 256.0. Sometimes it is required to  $\times 256$  or  $/ 256$  before visualization.

**Note** This layer can be totally replaced by **ScaleVector-Layer**, **ReshapeLayer** and **FlattenLayer**.

## 2.12. OutputBlob

**Source file** output\_blob\_layer.cpp

**Input**:  $N \times D$ ,  $D$  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 $\times$ Cols) *e.g.* Say you want to output a 3D heatmap that is of size  $(J \times D) \times H \times W$
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.  $N \times 1 \times H \times W$  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**  $N \times D$ : 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**  $N \times D$ : 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**  $N \times 3 \times H \times W$

**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**  $N \times 3 \times H \times W$

**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**  $N \times 3 \times H \times W$

**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**  $N \times D$  flattened vector

**Output**  $N \times D$  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, bbs *etc.* per sample, and store them to a **single** folder on SSD.
- A layer that fetches all annotation for each sample instead of reading separating 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.