Human Pose Estimation

Kuan-Lin Chen
UC San Diego
kuc029@ucsd.edu

Hsiao-Chen Huang
UC San Diego
hsh030@eng.ucsd.edu

Eddie Tseng
UC San Diego
edtseng@eng.ucsd.edu

Abstract

We study and implement four different models for human pose estimation based on deep convolutional neural networks. The problem of pose estimation is formulated as learning with DNN-based regression. According to our experimental results, the deep convolutional neural network is capable of capturing the full context of each body joint and make accurate inferences. Finally, We present a detailed empirical analysis by visualizing the architectures, training process and prediction results.

1. Introduction

The problem of human pose estimation, defined as the **problem of localization of human joints or parts** [10]. In this project, we aim to design and implement a single-person pose estimator by formulating the pose estimation as a joint regression problem.

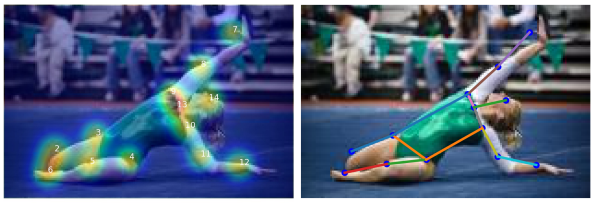
The convolutional architecture (AlexNet) proposed by Krizhevsky *et al.* [6] has shown a great potential on object detection and localization [9]. Recently, there has been much research ongoing related to pose estimation using **convolutional architectures**. Toshev and Szegedy [10] have proposed a method of directly regressing the Cartesian coordinates by a series of refining regressors.

In this paper, we are going to exploit the convolutional architectures and learn the underlying principles of designing a learning machine for pose estimation. Fig. 1 is a selected result of human pose estimation using our **Baseline-1** DNN-based regressor.

2. Method

2.1. Prediction

Given an input image $\mathbf{x} \subset \mathcal{X} \in \mathbb{R}^{W \times H \times 3}$, we will use a given decision function $g(\mathbf{x}; \mathbf{w}) : \mathcal{X} \rightarrow \mathcal{Y}$ to predict the



(a) Joints. (b) Skeleton.

Figure 1: An example of 2D pose estimation. We generate these results by using our Baseline-1 model which we trained it from scratch.

annotation $\hat{\mathbf{y}} \subset \mathcal{Y}$ where

$$\mathcal{Y} = \begin{bmatrix} \boldsymbol{\omega} & \mathbf{u} & \mathbf{v} \end{bmatrix} = \begin{bmatrix} \omega_1 & u_1 & v_1 \\ \omega_2 & u_2 & v_2 \\ \vdots & \vdots & \vdots \\ \omega_k & u_k & v_k \end{bmatrix} \tag{1}$$

with $\omega_1, \omega_2, ..., \omega_k \in \{-1, 1\}$, $u_1, u_2, ..., u_k \in \{1, 2, ..., W\}$ and $v_1, v_2, ..., v_k \in \{1, 2, ..., H\}$. Here, we consider the number of joints or parts we need to find in a given image $\mathbf{x}$ is k. For example, one can easily define $k = 14$ and construct the joints mapping in Table 1. For every $\omega_i, i = 1, 2, ..., k$, it indicates the presence of the i^{th} joint in the given image $\mathbf{x}$. That is,

$$\begin{cases} \omega_i = -1, & \text{if } i^{th} \text{ joint is not in } \mathbf{x} \\ \omega_i = 1, & \text{if } i^{th} \text{ joint is in } \mathbf{x} \end{cases} \tag{2}$$

For every $(u_i, v_i), i = 1, 2, ..., k$ pair, it represents the coordinate in the Cartesian coordinate system or location in the given image $\mathbf{x}$. For $\omega_i = -1$, the coordinate tuple (u_i, v_i) is ignored because it is meaningless to consider the location of an invisible joint [3]. Therefore, the goal is to design an estimator which can produce an estimate given an image $\mathbf{x}$.

k	Joints
1	Right ankle
2	Right knee
3	Right hip
4	Left hip
5	Left knee
6	Left ankle
7	Right wrist
8	Right elbow
9	Right shoulder
10	Left shoulder
11	Left elbow
12	Left wrist
13	Neck
14	Head top

Table 1: The joints mapping

To be more clear, we summarize the prediction in Eq. 3.

$$\begin{aligned}\hat{\mathbf{y}} &= g(\mathbf{x};\mathbf{w}) \\ &= \begin{bmatrix}\hat{\boldsymbol{\omega}} & \hat{\mathbf{u}} & \hat{\mathbf{v}}\end{bmatrix}\end{aligned} \tag{3}$$

Note that $g(\mathbf{x},\mathbf{w})$ is the output of the learning machine to input $\mathbf{x}$ when its parameters have value $\mathbf{w}$ (from lecture slides) and the scope here is limited in single-person pose estimation.

2.2. Learning with DNN-based Regressor

Given a set of input images $\{\mathbf{x}_1,\mathbf{x}_2,...,\mathbf{x}_n\} \subset \mathcal{X}$ and their corresponding annotations $\{\mathbf{y}_1,\mathbf{y}_2,...,\mathbf{y}_n\} \subset \mathcal{Y}$, we want to learn a decision function $g(\mathbf{x};\mathbf{w}) : \mathcal{X} \rightarrow \mathcal{Y}$ such that the structural risk is minimized. Now, consider a set of functions $S = \{g(\mathbf{x},\mathbf{w}), \mathbf{w} \in \mathcal{W}\}$ implemented by a deep neural network of fixed architecture with the weights $\mathbf{w}$ and a structure through $S_p = \{g(\mathbf{x},\mathbf{w}), \|\mathbf{w}\| \leq C_p\}$ with $C_1 < C_2 < ... < C_n$ [11]. The minimization of the empirical risk within the structure S_p is

$$R_{emp}(\mathbf{w},\beta_p) = \frac{1}{n}\sum_{i=1}^{n} L(\hat{\mathbf{y}}_i,\mathbf{y}_i) + \beta_p\|\mathbf{w}\|^2 \tag{4}$$

where β_p is the scale of introducing "weight decay" to a deep neural network.

Practically, we design a network with branches to do the multi-label classification and coordinate regression simultaneously in human pose estimation. By observing the (ω_i, u_i, v_i) tuple, it is obvious that we can decompose the problem to have classification head and regression head as we show in Fig. 2. To be more specific, the loss function of the classification head can be chosen by

$$L(\hat{\boldsymbol{\omega}}^i,\boldsymbol{\omega}^i) = -\frac{1}{k}\sum_{j=1}^{k}\left[\omega_j^i \log\hat{\omega}_j^i + (1-\omega_j^i)\log(1-\hat{\omega}_j^i)\right] \tag{5}$$

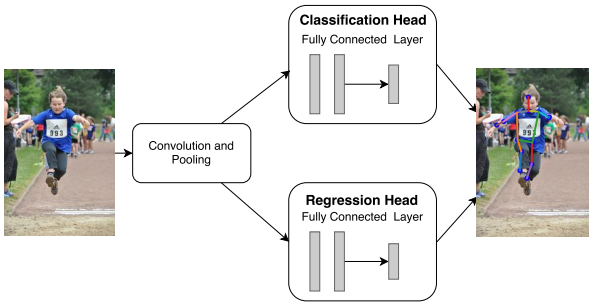


Figure 2: The illustration of Multi-label classification and coordinate regression. With the annotation estimate $\hat{\mathbf{y}} = (\hat{\boldsymbol{\omega}}, \hat{\mathbf{u}}, \hat{\mathbf{v}})$, one can connect every joint $(\hat{\omega}_i, \hat{u}_i, \hat{v}_i), i = 1, 2, ..., k$ to produce the skeleton.

and the loss function of the regression head can be chosen by

$$L(\hat{\mathbf{r}}^i,\mathbf{r}^i) = \left\|\hat{\mathbf{r}}^i - \mathbf{r}^i\right\|_2^2 \tag{6}$$

where

$$\mathbf{r} = \begin{bmatrix}\mathbf{u}\\ \mathbf{v}\end{bmatrix} \tag{7}$$

After specifying the loss functions, we can then apply *stochastic gradient descent* to train a deep neural network. Note that we still need to pick a specific architecture of the deep neural network to fit into our application of human pose estimation. In this paper, we pick **deep convolutional neural networks** including **AlexNet** [6] and **VGGNet** [8] as our regressors.

The advantages of using **DNN-based regressor** is that we do not need to define a domain specific pose model. The deep neural networks have the capability to implicitly learn from the data.

2.3. Learning with A Series of Refining Regressors

Toshev and Szegedy [10] have proposed a method (DeepPose) of directly regressing the Cartesian coordinates by a series of refining regressors. As we illustrate in Fig. 3.

At a particular stage, the input layer only takes a cropped image (smaller) around the joint location produced by the previous stage. They utilize a popular deep convolutional architecture (AlexNet) as building blocks and by cascading a series of DNN-based regressors, the overall network can see the finer and finer details in the input image. As as result, the overall accuracy increases.

In general, cropping image is one of the way to refine the inferences from the previous stages. Notice that DeepPose proposed by Toshev and Szegedy [10] is just one of the way to design a series of refining regressors. Learning with a series of refining regressors is not limited in only taking subimages. We want to emphasize that it is a general concept which should not be limited to one application.

2.4. Learning with Pose Machines

Suppose that the correct $\boldsymbol{\omega}$ is given. Now we concentrate on the localization problem, that is, finding $(\hat{\mathbf{u}}, \hat{\mathbf{v}})$. In this section, we consider a **cascade** of deep convolutional neural network that consists of a sequence of regressors [7]. At each particular

Figure 3: An illustration of the architecture of DeepPose, proposed by Toshev and Szegedy [10] (cropped from original paper).

stage $t \in \{1, 2, ..., T\}$, the regressor predicts a confidence map for assigning a location to each joint $(u_i, v_i) = (z_w, z_h), z_w = 1, 2, ..., W, z_h = 1, 2, ..., H$ based on a given input image $\mathbf{x}$ as well as the contextual information from the previous stages. With this kind of architecture, a cascade of deep convolutional neural network can actually provide increasingly refined estimate for the locations of joints. The confidence map for the assignment $(u_i, v_i) = (z_w, z_h)$ at each stage t is computed by

$$b_t\left[(u_i, v_i) = (z_w, z_h)\right] = g_t^i\left(\mathbf{x}; \bigoplus_{i=1}^{K} \psi[(z_w, z_h), \mathbf{b}_{t-1}^i]\right) \tag{8}$$

where

$$\mathbf{b}_{t-1}^i = \{b_{t-1}[(u_i, v_i) = (z_w, z_h)]\}_{z_w=1,2,...,W, z_h=1,2,...,H} \tag{9}$$

is a collection of confidence maps from the previous stages computed at every location $z_w = 1, 2, ..., W, z_h = 1, 2, ..., H$ for the i^{th} joint. Note that the function ψ computes the contextual features from the confidence maps generated by the previous stages and the operator $\bigoplus$ means vector embedding.

One of a notable state-of-the-art work proposed by Wei *et al.* [12] in 2016 is convolutional pose machines, which is shown in Fig. 4.

Note that they utilize the concepts of pose machines and modify each stage as a convolutional neural network to achieve better performance. In general, we can think of pose machines as a good data representation method of applying refining regressors. Therefore, by picking architecture at each stage and designing a helpful data representation are the keys towards success.

Compared with a series of DeepPose, we can think of convolutional pose machines capture features in a soft assignment way. Since the cropped image loses all the pixel information outside the brounding box, the potential discriminant features may be removed or partially distorted. In contrast, convolutional pose machines can still retrieve the information far away from the joint locations. Hence, the performance of convolutional pose machines outperform DeepPose.

2.5. Evaluation

To evaluate the pose estimator on each testing image and the corresponding annotation, we need to define an evaluation metric. One of the popular evaluation metric is **Percentage of Correct Part** or "PCP" metric [4]. It defines that a joint is said to be correctly localized if the estimated body segment endpoints are within 50% of the ground-truth segment length from their true locations. Also, there are many metrics such as "PCPm", "PCK" and "PCKh", etc. Among those evaluation metrics, we would like to choose "PCKh" which define the matching threshold as 50% of the head segment length [2] because it makes the metric articulation independent.

There are cases when the evaluation metrics are not applicable. For instance, the "PCKh" metric is incapable when head size cannot be computed from the annotation. Therefore, the dataset itself will restrict the possible evaluation metrics. Since all that matters is the matching threshold from location estimates, we can actually slightly modify the threshold in "PCKh" to a fraction of the image size.

3. Experiments

3.1. Dataset

In this project, we use **Leeds Sports Pose Extended Training Dataset** (LSP extended) provided by Johnson and Everingham [5] to train our deep convolutional neural networks. This dataset contains 10,000 images gathered from Flickr searches for the tags 'parkour', 'gymnastics', and 'athletics' and consists of poses deemed to be challenging to estimate. The images have been scaled such that the annotated person is roughly 150 pixels in length. Each image has been annotated with up to 14 visible joint locations [5]. Note that the dataset contains the joint annotations in a $3 \times 14 \times 10000$ matrix with x and y locations and a binary value indicating the visibility of each joint. The joint mapping has been illustrated in Table 1.

3.2. Platform

Since the availability of the powerful Graphics Processing Units (GPUs) will be a crucial key to train a deep convolutional neural network. We use the compute engine with one NVIDIA® Tesla® **K80** GPU on **Google Cloud Platform** (GCP). Note that the memory size of K80 GPU is 24G, so normally it has ability to accommodate most of implementation in deep convolutional architectures. The training speed is also a critical factor to the success in deep learning. Note that the processing power of K80 is $1864 - 2912$ GFLOPS in double precision, so it fit in our expectation of training deep networks.

To build networks, we choose to use **Tensorflow** [1] as our library due to the following tractabilities. First, by using **TensorBoard**, we are able to visualize and track our training progress in terms of accuracy and loss. Secondly, by coding in Tensorflow, we get a chance to understand the details in customizing estimators and loss functions. We did not choose Keras simply because it lacks degrees of freedom in designing networks and losses.

3.3. Models

We implement DNN-based regressor in four different types of deep convolutional neural networks. As we have mentioned in the previous section, the pose estimators need to do multi-label classification and coordinate regression in the same time, so the strategy here is first to train the classification head and then append two fully connected layers at the branch point to train regression head.

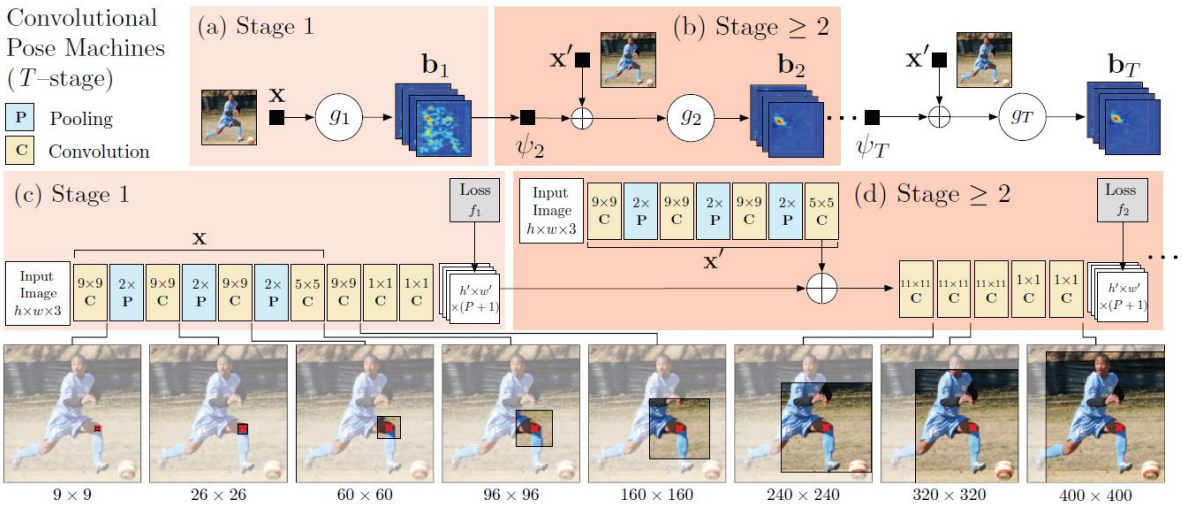


Figure 4: An illustration of the architecture of convolutional pose machines, proposed by Wei *et al.* [12] (cropped from original paper).

Notice that the classification head outputs 14 probabilities $\hat{\omega}_1, \hat{\omega}_2, ..., \hat{\omega}_{14}$ that indicate the presence of each joint and the regression head outputs 14 coordinates $(\hat{u}_1, \hat{v}_1), (\hat{u}_2, \hat{v}_2), ..., (\hat{u}_{14}, \hat{v}_{14})$ that give the location of each joint.

The descriptions of four different types of deep convolutional neural networks are presented as following.

3.3.1 Baseline-1

In classification head, Baseline-1 consists of two convolutional layers, which are all followed by max-pooling layers, and three fully-connected layers. In regression head, Baseline-1 cascades the branch point at the output of the first fully-connected layer in classification head, and are followed by three full-connected layers. The architecture is illustrated in Fig. 5.

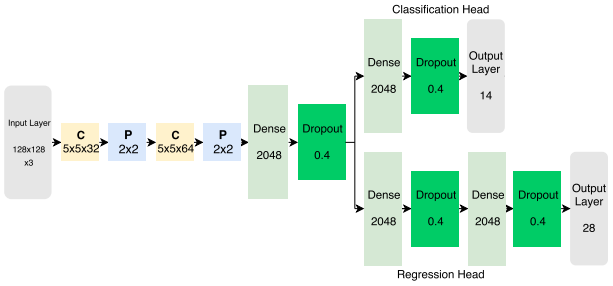


Figure 5: The architecture of Baseline-1. C denotes the convolutional layer, P denotes the max-pooling layer and Dense represents the fully connected layer.

3.3.2 Baseline-2

In classification head, Baseline-2 consists of three convolutional layers, which are all followed by max-pooling layers, and three fully-connected layers. In regression head, Baseline-1 cascades the branch point at the output of the first fully-connected layer in classification head, and are followed by three full-connected layers. The architecture is illustrated in Fig. 6.

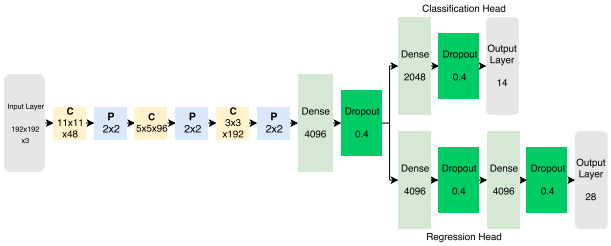


Figure 6: The architecture of Baseline-2.

3.3.3 Non-separated AlexNet

We use a non-separated AlexNet since K80 can almost accommodate the whole network (we add a pooling layer at the output of the fourth convolutional layer reduce the number of weights to fit the whole network in K80). The architecture is illustrated in Fig. 7.

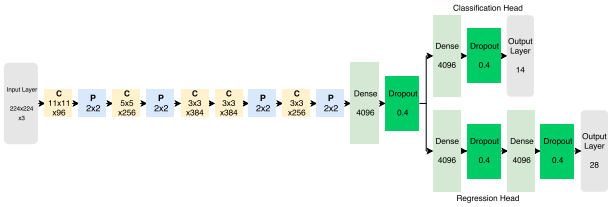


Figure 7: The architecture of non-separated AlexNet.

3.3.4 VGGNet

VGGNet is the runner-up in ILSVRC 2014 which was proposed by Karen Simonyan and Andrew Zisserman. Their main contribution is that showing the depth of the network is a critical component for good performance. In this project, we use VGG-19. The architecture is illustrated in Fig. 8.

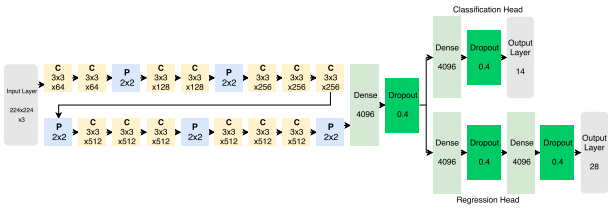


Figure 8: The architecture of VGGNet (VGG-19).

3.4. Results

3.4.1 Training

With the number of training iteration increases, we compare four different classification heads by the following perperties.

- Training accuracy for total joints
- Time to loss convergence
- Training speed

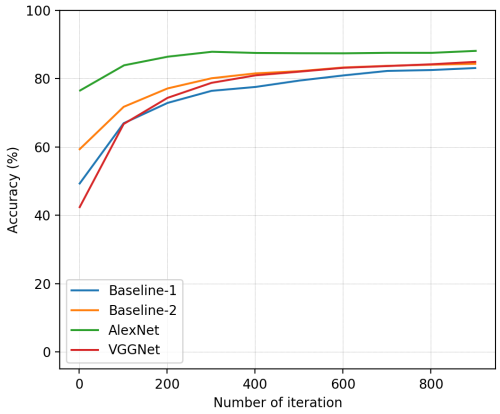


Figure 9: The training accuracy of four different models.

As we can observe in Fig. 9 and 10, there is a trade-off between accuracy and the difficulty of training a network. Since we can think of the training speed as an indicator of the difficulty to train a network.

3.5. Evaluation

Due to the limitation of LSP-extended dataset, we cannot compute the 50% head segment length as our matching threshold. To address this problem, we chooos the a fraction of image size as the matching threshold. That is, we say a joint is correctly localized if the L_2 distance is less than or equal to $\frac{1}{12}$ of image height (input image will be rescaled to a square which has equal height and width to fit into our input layer). For classification, the accuracy is simply copmuting the percentage of matching with the ground-truth. In Table 2, we summarize the accuracy of four different classification heads.

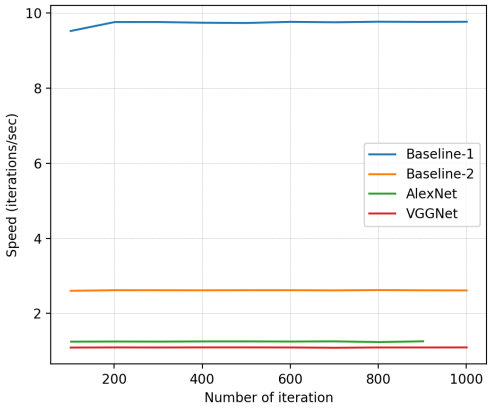


Figure 10: The training speed of four different models.

Model	Accuracy
Baseline-1	82.53%
Baseline-2	83.21%
Non-Separated AlexNet	88.82%
VGGNet	84.44%

Table 2: Accuracy of four different classification heads

3.5.1 Prediction

According to our evaluation result, the performance of Baseline-2 is not so bad. Therefore, we will discuss some results of Baseline-2 in this section. Recall that the output of the DNN-based regressor will produce the estimate $\hat{\mathbf{y}} = \begin{bmatrix}\hat{\boldsymbol{\omega}} & \hat{\mathbf{u}} & \hat{\mathbf{v}}\end{bmatrix}$. In detail, Fig. 11 shows the location of each joint by imposing a probability density on it. Fig. 12 and Fig. 13 show several good cases and bad cases, respectively.

To address the difficulties in occlusion and invisible joints, the configurations of different parts has been proved essential for detecting correct parts [6, 10, 7, 12]. Therefore, the way how we encode the large spatial contextual information by using branches in convolutional architectures will be crucial to the performance. As we can see in Fig. 13, some images contain more than one person, together with the invisible joints, the performance of Baseline-2 tends to be unstable.

4. Conclusion

We have studied some sequential deep convolutional neural networks (DeepPose and CPM) that encode contextual information to refine the inferences iteratively over the stages. Moreover, we have implemented four different DNN-based regressors from scratch by using Tensorflow.

5. Future Work

Due to the limitation of time and resource, we haven't implemented the state-of-the-art architecture which utilizes feature pyramids [13]. In the future, we will improve the performance of the prediction and investigate novel architectures which could be potentially better tailored towards localization problems in general. Also, we will extend our work to real-time multi-person pose estimation.

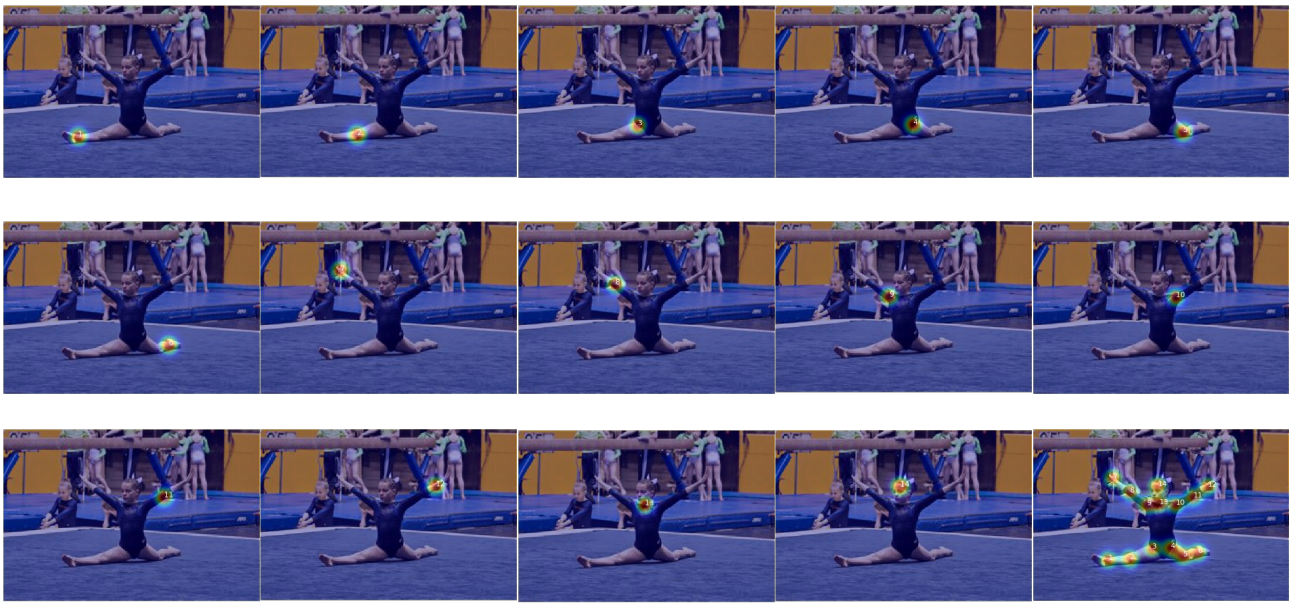


Figure 11: Localization of each joint. The last figure combines all inferences to predict human pose.

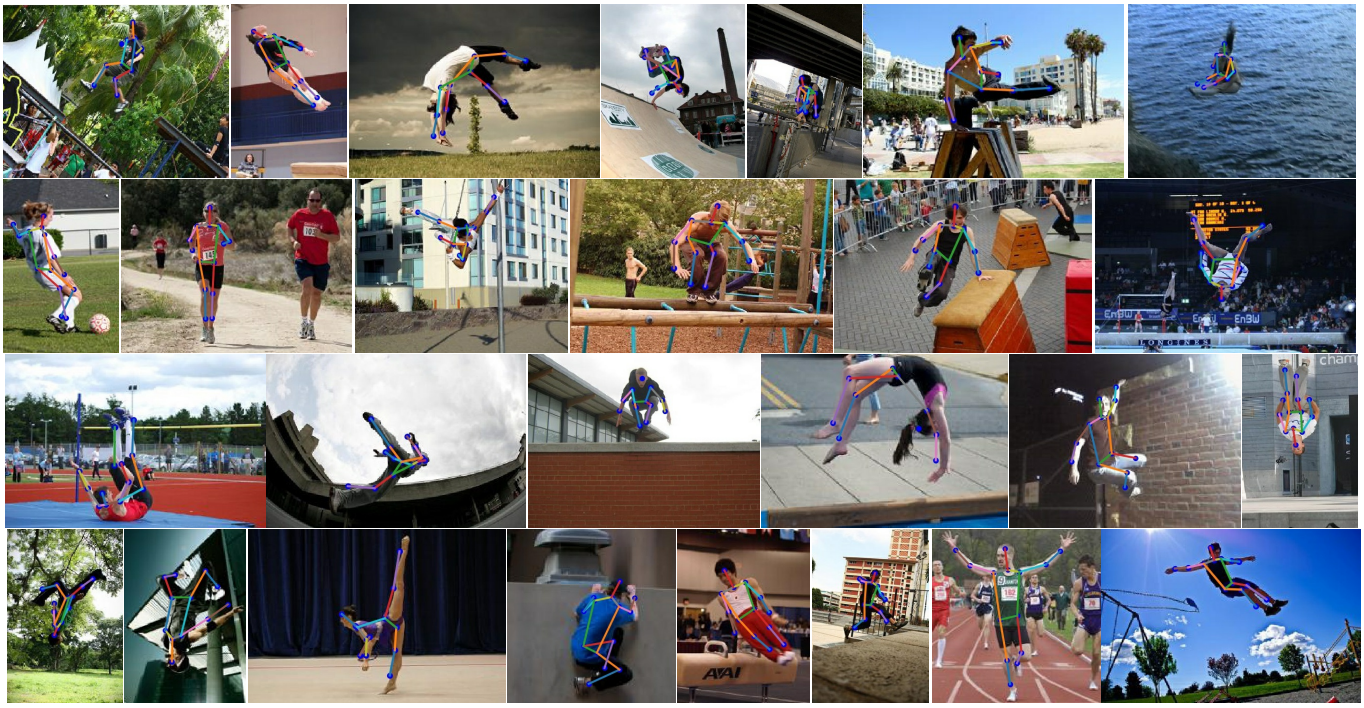


Figure 12: Several good cases of using Baseline-2 model.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray,

Figure 13: Several bad cases of using Baseline-2 model.

C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] M. Andriluka, L. Pishchulin, P. Gehler, and B. Schiele. 2d human pose estimation: New benchmark and state of the art analysis. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3686–3693, June 2014.

[3] M. B. Blaschko and C. H. Lampert. Learning to localize objects with structured output regression. In *European conference on computer vision*, pages 2–15. Springer, 2008.

[4] V. Ferrari, M. Marin-Jimenez, and A. Zisserman. Progressive search space reduction for human pose estimation. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[5] S. Johnson and M. Everingham. Learning effective human pose estimation from inaccurate annotation. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2011.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA, 2012. Curran Associates Inc.

[7] V. Ramakrishna, D. Munoz, M. Hebert, J. Andrew Bagnell, and Y. Sheikh. Pose machines: Articulated pose estimation via inference machines. In D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 33–47, Cham, 2014. Springer International Publishing.

[8] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[9] C. Szegedy, A. Toshev, and D. Erhan. Deep neural networks for object detection. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 2553–2561. Curran Associates, Inc., 2013.

[10] A. Toshev and C. Szegedy. Deeppose: Human pose estimation via deep neural networks. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1653–1660, June 2014.

[11] V. Vapnik. Principles of risk minimization for learning theory. In *Advances in neural information processing systems*, pages 831–838, 1992.

[12] S. E. Wei, V. Ramakrishna, T. Kanade, and Y. Sheikh. Convolutional pose machines. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4724–4732, June 2016.

[13] W. Yang, S. Li, W. Ouyang, H. Li, and X. Wang. Learning feature pyramids for human pose estimation. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 1290–1299, Oct 2017.