Assignment 2: 2D Human Pose Estimation

CPSC 532R/533R Visual AI
by Helge Rhodin and Yuchi Zhang

This assignment focuses on computer vision aspects of deep learning. The goal is to infer the 2D human pose in a given image, the pixel location of the human body parts as shown in Figure 1. This assignment will compare three approaches. Straight regression with a black-box neural network, per-pixel classification in form of heatmap detection, and a combination of detection and regression via so called *integral pose regression* [1].

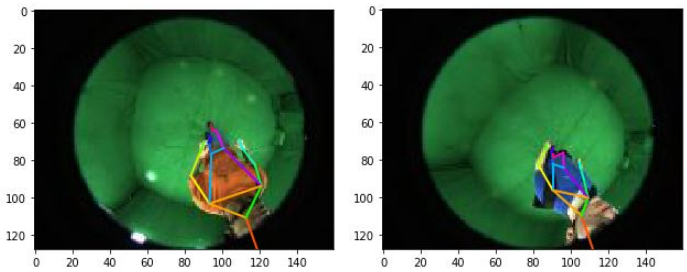

Figure 1: Egocentric images and corresponding pose annotation.

Preliminaries. While PyTorch supports most python expressions for defining a neural network (including control flow), using tensor operations instead of `for` loops is favorable, to facilitate parallel execution on GPU and CPU (SIMD operations). We prepared the `Assignment2_preliminaries.ipynb` notebook to exercise tensor operations. This part is not graded but highly recommended to prepare for the main task.

Main tasks. We provide a dataset of egocentric images and corresponding 2D pose in form of a pytorch dataset, a pre-defined neural network, and plotting utility functions. The data stems from the publication [2]. We also provide training code for regressing 2D pose directly from the image, similar to what you implemented in Assignment 1. Either start off from the `Assignment2.ipynb` notebook that we provide or integrate this new framework into your Assignment 1 solution (which might have additional functionality such as tensorboard). In both cases, make sure that that the sample trains and gives an output such as shown in Figure 2. Make sure that you are familiar with the code we provide before moving on to the subsequent tasks.

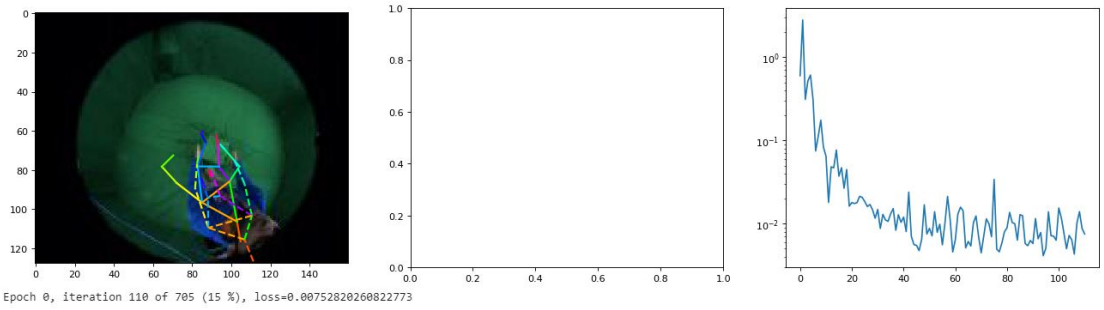

Figure 2: Output of the `Assignment1.ipynb` notebook on the regression task. The blank graph (center) is a placeholder for displaying heatmaps in the subsequent tasks.

- **Task I:** Implement human 2D pose detection through heatmap estimation [3]. The network must output a stack of $N = 18$ heatmaps, one for each joint of the human skeleton. It is your task to train the network by matching the output with reference heatmaps and to read out the predicted keypoint location as the arg max at inference time. The notebook provides skeleton code for the heatmap handling as a guidance. Proceed step-by-step, e.g., start with the processing of a single heatmap (as in the preliminaries notebook), later a stack of them, and then a mini batch of stacks. Entry points for your code are marked with `# TODO`.
- **Task II:** Implement human 2D pose regression through heatmap integration [2]. In this case, the heatmap prediction must be followed by a custom integration layer that computes the expected location over the heatmap. The loss on the resulting 2D pose can then simply be the MSE to the ground truth joint location, as for the direct regression methods we provided. The notebook provides a skeleton for the individual steps. Note, the integration layer must not use a for loop to gain full points. Make sure that your network trains from scratch, not from the result of **Task I**.
- **Task III:** Compare the three approaches (regression, classification, integral regression). Which of them attains the highest accuracy (lowest error) on the provided validation set?

Hints and comments.

- The lecture slides provide instruction of how to use pytorch with GPU support on UBC servers. Google colab with GPU acceleration enabled works too.
- It is easy to get lost when dealing with high-dimensional tensors. Break your implementation into individual steps and make sure that each of them works by printouts of tensor dimensions and tensor values as an image.
- By default, tensor operations are element wise, e.g., `A + B` generally assumes the same dimensions for `A` and `B`. So called *Broadcasting* is a powerful operation that kicks in if the dimensions differ. Make yourself familiar with that concept.
- If your are unfamiliar with tensor operations, implement your function first using loops and turn it into equivalent tensor operations once your loop version works.
- Aggregation functions, such as `torch.sum(mat)`, compute a single value for an entire tensor. One or multiple dimensions can be specified to apply the operation only along that axis, e.g., `torch.sum(mat, dim=-1)`. Positive and negative indices are permitted, with negative ones counting 'from the right'. While this operation returns a tensor with dimension `dim` removed, it is often useful to specify `keepdim=True` to maintain that dimension in the output (with size 1).
- `torch.linspace` is an other useful tool for creating sequences of numbers.
- We utilize a custom dictionary, `DeviceDict`, to store mini batches from the dataset and network output. It eases data transfer to and from the GPU.

Optional add-ons if you are curious (not graded):

- Implement functionality to save your neural network weights after every epoch and load the most recent version at start time. ***Highly recommended for google colab users where your session could be terminated at any time.***
- Try to tune the hyperparameters: learning rate (scheduler), alternative loss function, additional criteria (e.g., a prior that favors a localized heatmap). Can you augment the training data (e.g., rotation, color shift, gamma correction) to further improve? We will create a leader board, who can get the best performance?

References

[1] Xiao Sun, Bin Xiao, Fangyin Wei, Shuang Liang, and Yichen Wei *Integral Human Pose Regression*, ECCV, 2018.

[2] Helge Rhodin, Christian Richardt, Dan Casas, Eldar Insafutdinov, Mohammad Shafiei, Hans-Peter Seidel, Bernt Schiele, and Christian Theobalt, *EgoCap: Egocentric Marker-less Motion Capture with Two Fisheye Cameras*, SIGGRAPH Asia, 2016.

[3] J. Tompson, R. Goroshin, A. Jain, Y. LeCun, and C. Bregler. *Efficient object localization using convolutional networks.* CVPR, 2015