Introduction to Machine Learning (CSCI-UA.473)
Capstone Project Report

Dekun Ma

December 14, 2022

In this report, I will summarize my work on the final project of Introduction to Machine Learning (CSCI-UA.473) at NYU. The code of my final project is available at: https://github.com/dekunma/intro-ml-project.

1 Method

1.1 Input Data

I used all the RGB and depth images provided in the training dataset. I concatenated these images on the first dimension (dimension 0), so the input data to my models have 12 channels. The shape of the input tensor is $[batch\_size, 12, 224, 224]$.

For RGB image normalization, I used $transforms.ToTensor()$ provided by *torchvision* to normalize the RGB images into range $[0, 1]$. Then I normalized them into mean of 1 and standard deviation of 0 based on each channel's global mean and standard deviation values. For depth image normalization, I used min-max normalization $depth\_img = (depth\_img - min\_depth)/(max\_depth - min\_depth)$ to normalize them into range $[0, 1]$, where min_depth is the global minimum value of the depth and max_depth is the global maximum value.

For data augmentation, I tried $PixDropout$ suggested by Rezaei et al.[5], where some fraction of pixels will be randomly dropped based on a selected probability during training. However, I did not notice an increased performance after applying this data augmentation technique. Visualizations are shown in figures 1 and 2

1.2 Models

Thanks to my modularized and scalable design of the codebase, I was able to incorporate six different models into the project: ResNeSt 269, ConvNeXt, HRNet, PoseResnet 152, Resnet 50, and Resnet 152.

ResNeSt[10] is a CNN architecture based on Resnet, which applies the channel-wise attention on different network branches to leverage their success in capturing cross-feature interactions and learning diverse representations. ConvNeXt[3] is a CNN architecture inspired by Vision

Figure 1: Original depth image

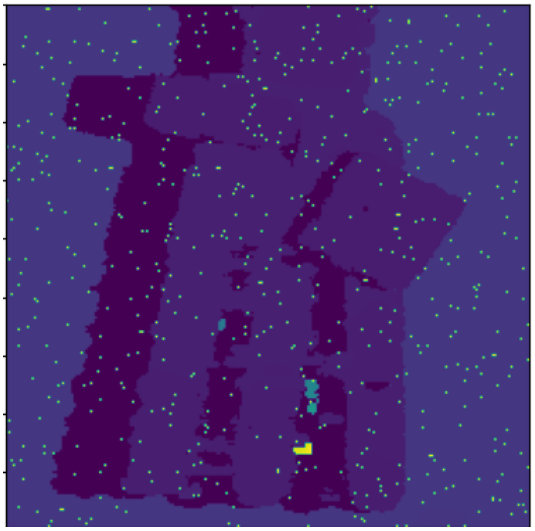

Figure 2: After applying *PixDropout*

Transformers. It utilized depthwise convolution to improve the performance. HRNet [6] is a CNN architecture that can maintain the input image in a high resolution across the network. PoseResnet [9] is a modification on Resnet which added deconvolution layers at the end. Finally, Resnet [2] is the classical CNN architecture that used residual blocks to solve the vanishing gradient problem.

1.3 Training

The training was done on NYU High Performance Computing (HPC) - Greene cluster. All models were trained on Nvidia RTX8000 graphics cards with 48GB dedicated memory. I tried using *MSE Loss* and *SmoothL*1 *Loss* as the loss function, and it turns out that their influence on performance did not differ by much. The best configuration for each model are as follows:

Model	**Batch Size**	**LR**	**Optimizer**	$\beta 1$	$\beta 2$	**Decay**	**Epochs**	**LR Scheduler**
ResNeSt269	32	0.001	Adam	-	-	-	200	lr*=0.5 at epochs [30, 60, 90]
Resnet152	32	0.001	Adam	-	-	-	180	lr*=0.5 every 30 epochs
ConvNeXt	32	0.004	AdamW	0.9	0.999	0.05	180	Cosine Annealing
HRNet	24	0.001	Adam	-	-	-	210	lr*=0.1 at epochs [170, 200]
PoseResnet152	32	0.001	Adam	-	-	-	140	lr*=0.5 every 30 epochs
Resnet50	32	0.001	Adam	-	-	-	30	-

2 Results

At the time this paper was submitted, I ranked number **1** on the public leaderboard, with an RMSE of **0.00217**. The results of all the models are as follows (ranked in ascending order by

RMSE):

Model	RMSE
ResNeSt269	**0.00217**
Resnet152	0.00232
ConvNeXt	0.00253
HRNet	0.00262
PoseResnet152	0.00327
Renet50	0.0397

3 Discussion

3.1 Fancier model ! = Better performance

I found that models that perform well on other tasks might not perform as well on our task. I implemented ConvNeXt, which even outperformed Vision Transformers on classification tasks on ImageNet. I also implemented HRNet, which was introduced in their corresponding paper [9] to be suitable for tasks that require high precision, such as human pose estimation, which I believe was quite similar to our task. However, none of these two models outperformed Resnet152, which I initially intended to use as a baseline model. Perhaps they are not the best for this particular task, but it could also be that I did not find the optimum configurations.

3.2 PixDropout

As proposed by Rezaei et al.[5], *PixDropout* is a data augmentation technique that randomly drops some fraction of pixels in the input depth images. It is similar to RGB image augmentation techniques, where rectangular regions were randomly masked out to simulate occlusion.

I was unable to reproduce the performance improvement suggested in their paper. The reason could be that the depth images in the test dataset, unlike the real-world ones, do not have occlusion. So, the performance of our model will not be improved by introducing noise during the training process. In fact, I noticed a slight performance decrease after applying *PixDropout*.

3.3 Depth Image Normalization

Some research papers on the human hand pose estimation task, such as [4, 1, 5], choose to normalize the depth image into the range $[-1, 1]$. However, I also noticed that they all implemented another function to normalize the depth image into $[0, 1]$ in their codebase, suggesting that this is also a feasible approach for normalization.

I tried these two types of normalizations on my model, and I found that they have similar performance, with the performance of $[0, 1]$ being slightly higher. The reason could be that the datasets used in those papers, such as NYU Hand Pose Dataset [8] and ICVL Hand Pose Dataset [7] work better with $[-1, 1]$ while ours works better with $[0, 1]$ normalization.

4 Future work

4.1 Ray Tune

Ray Tune is a python library for scalable hyperparameter tuning. I wrote scripts for Ray Tune, but I did not have enough time to run large-scale tuning experiments to find the best hyperparameters for my models. Instead, I applied suggested parameters for the models in their corresponding research papers and tuned them manually. The performance of my models could potentially be improved after having Ray Tune experiments, especially $ConvNeXt$, whose unsatisfactory performance could be caused by unsuitable hyperparameters.

4.2 More Data Augmentation

I did not find many data augmentation techniques suitable for our task. However, there could be some techniques in the literature which I haven't found yet that could improve the performance of my models.

References

[1] Hengkai Guo, Guijin Wang, Xinghao Chen, and Cairong Zhang. Towards good practices for deep 3d hand pose estimation. *arXiv preprint arXiv:1707.07248*, 2017.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[3] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[4] Markus Oberweger and Vincent Lepetit. Deepprior++: Improving fast and accurate 3d hand pose estimation. *International Conference on Computer Vision (ICCV)*, 2017.

[5] Mohammad Rezaei, Razieh Rastgoo, and Vassilis Athitsos. Trihorn-net: A model for accurate depth-based 3d hand pose estimation. *arXiv preprint arXiv:2206.07117*, 2022.

[6] Ke Sun, Bin Xiao, Dong Liu, and Jingdong Wang. Deep high-resolution representation learning for human pose estimation. In *CVPR*, 2019.

[7] Danhang Tang, Hyung Jin Chang, Alykhan Tejani, and Tae-Kyun Kim. Latent regression forest: Structured estimation of 3d articulated hand posture. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3786–3793, 2014.

[8] Jonathan Tompson, Murphy Stein, Yann Lecun, and Ken Perlin.

[9] Bin Xiao, Haiping Wu, and Yichen Wei. Simple baselines for human pose estimation and tracking. In *European Conference on Computer Vision (ECCV)*, 2018.

[10] Hang Zhang, Chongruo Wu, Zhongyue Zhang, Yi Zhu, Zhi Zhang, Haibin Lin, Yue Sun, Tong He, Jonas Muller, R. Manmatha, Mu Li, and Alexander Smola. Resnest: Split-attention networks. *arXiv preprint arXiv:2004.08955*, 2020.