High Performance Hand Pose Estimation Project Report

Xiaoqi Li, Yipeng Liu, Zhuocheng Guo

1. Overview

1.1 Background

As deep learning develops and the RGB-D camera becomes cheap, hand pose estimation is getting more attention in the computer vision field. And it has applications in multiple fields, such as Augmented Reality (AR), Virtual Reality (VR), Mixed Reality (MR), and Human Computer Interaction (HCI).[1]

Hand pose estimation is a process to detect the joints of hands and model the hands' skeletons in 2D images or 3D space. It is similar to body pose estimation but more difficult to predict precisely for hand pose. As the distance between the fingers is short and the movement of the fingers is small, self-occlusion is easy to occur when the hand is moving, and it affects the precision of hand pose estimation. Besides, the complexity of hand gestures is higher than body gestures. Since even the small movement of fingers and joints can make the hand gestures quite different, making a precise prediction of hand pose is difficult. As a result, hand pose estimation is always a challenging task.

In this project, our task is to find a hand pose estimation model, deploy it to the Atlas 200 DK board, and achieve a performance better than 20 fps for the whole pipeline, which includes pre-processing, model execution, and post-processing. We introduce our workflow in part 3 and analyze our results in part 5.

1.2 Methodologies

The methods for hand pose estimation could be divided into two classes, the traditional machine learning methods and deep learning methods, in which Convolution Neural Network (CNN) is the mainstream.

In traditional machine learning methods, there are two steps. The first step is feature extraction and the second step is classification or regression. For feature extraction, there are various methods that could be used, such as Histogram of Oriented Gradient (HOG), Principal Component Analysis (PCA), and Scale Invariant Feature Transform (SIFT). After obtaining hand features, traditional machine learning algorithms, such as decision trees, random forests[2], and support vector machines[3], are used for classification or regression.

In deep learning methods, ResNet, VGG[4], and Hourglass network[5] are widely used. Based on the outputs of the models, the deep learning methods could be also divided into two classes, which are detection-based methods and regression-based methods[1]. In

detection-based methods, the model generates probability density maps for each joint, and then the joints are extracted from the related heat maps. While, in regression-based methods, the model predicts coordinates for each joint directly. The difference between the two kinds of methods is that the regression-based methods require more data and training iterations.

In our project, we focus on the deep learning methods, and we try both the detection-based methods and the regression-based methods. The models that we use and their types are listed as follows:

Detection-based method:

- 2D Hand Pose Estimation RGB
- Real-time 2D and 3D Hand Pose Estimation

Regression-based method:

- HandPose X

1.3 Hand Model

There are various models to define the skeleton of the hand, including the 14-joint model, the 16-joint model, and the 21-joint model. In our project, all of the models that we select are 21-joint models.

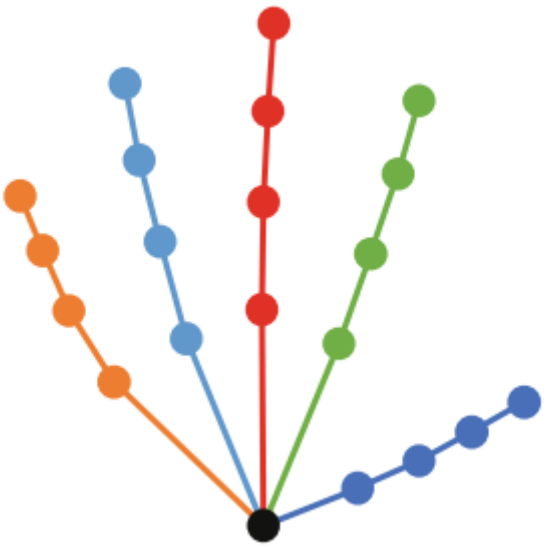


Fig 1[1] . 21 joints model[6]

1.4 Dataset

The image dataset for training hand pose estimation models could be divided into three classes, which are RGB-D images, RGB images, and RGB images with depth information. In our project, all of the models use RGB images for training.

The 2D Hand Pose Estimation RGB model and the Real-time 2D and 3D Hand Pose Estimation model use FreiHAND Dataset for training and testing. While, the HandPose X model's author uses a dataset that includes some low-duplicated data from the Large-scale Multiview 3D Hand Pose Dataset and other images, and the amount of data in that dataset is 49062.

2. Introduction of the Models

2.1 2D Hand Pose Estimation RGB model

This model uses a shallow UNet, which has fewer downsampling and upsampling layers. It modifies the calculating method of IOU to calculate the overlap of predicted heat maps and ground truth heat maps for the model's loss. The ground truth heat maps are generated from annotations and camera parameters matrices. Besides, this model uses the FreiHAND dataset, which is composed of RGB images, for training and testing.

For the inference workflow, firstly, it resizes and normalizes the images. Secondly, the model predicts heat maps. Finally, it extracts joints from related heat maps and shows the skeleton in the 2D image.

2.2 Real-time 2D and 3D Hand Pose Estimation model

This model is an improvement of the work by Ge et al.[5]. The work of Ge et al. proposes a Graph-CNN based method to reconstruct a full 3D mesh of the hand surface. The authors also propose a weakly-supervised training pipeline for the real-world dataset, to solve the problem of lacking annotated real-world data.

This model introduces a biologically inspired loss function to enhance the machine learning model generalization of the original one. It also uses the FreiHAND dataset to resolve the image occlusion problem.

This model uses the Hourglass network to predict heat maps and the multilayer perceptron to predict the relative depth. Then it combines those two outputs to compute the coordinates of joints in the camera coordinate system.

However, we do not deploy this model to the Atlas 200 DK device. The main reason is that this model includes the torch.inverse operation, which is not supported by ONNX. Though we can try to change that operation to numpy.linalg.inv as an alternative, we still choose not to deploy it because it requires tuning manually to make sure that the joints are in the correct positions of the real-world 2D image. Besides, the camera calibration process in pre-processing could also raise precision problems that need to be tuned. As a result, considering the limit of time for our project, we choose to try another model, which is the HandPose X model.

2.3 HandPose X model

This model uses a ResNet50 backbone and the Wing loss for training. It predicts the coordinates of joints in the image coordinate system directly. The dataset for training includes 49062 samples and some samples of the dataset are selected from the Large-scale Multiview 3D Hand Pose Dataset according to whether the level of duplication is low. Besides, this dataset is also composed of RGB images and the annotations are coordinates of joints in the image coordinate system.

3. Workflow Overview

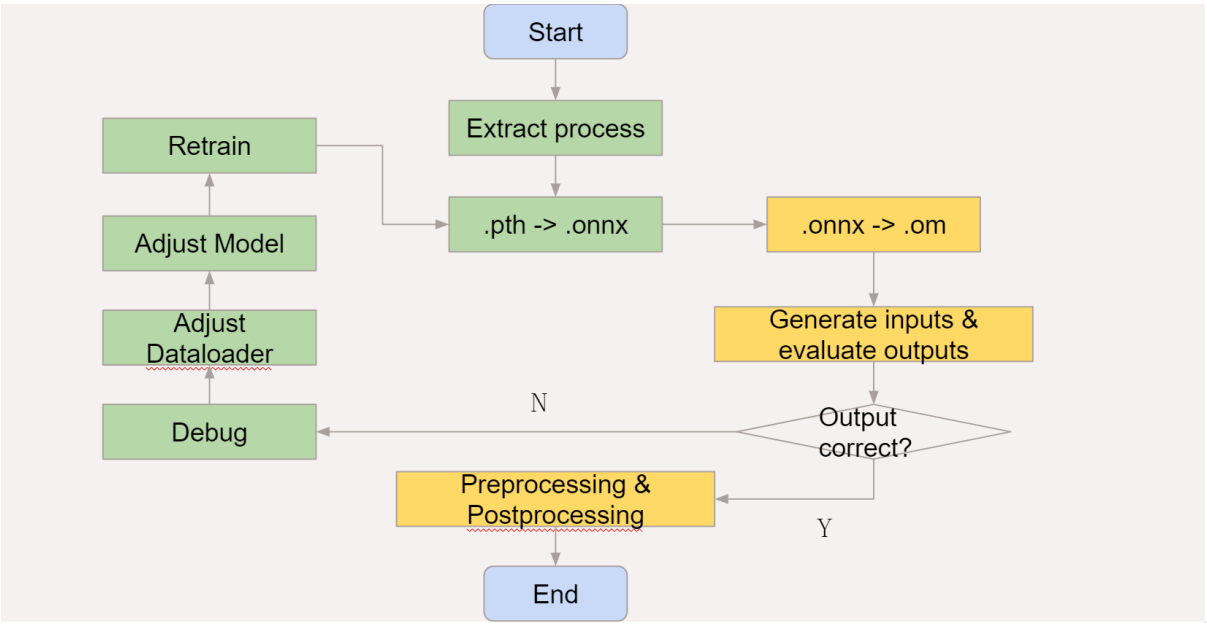


Fig 2. The workflow of deploying the model to the Atlas 200 DK device

Our workflow of deploying the model to the Atlas 200 DK device is as follows:

(1) Extract the main framework from the original model:
Before we write the script to do the model conversion, we simplify the original pipelines in the model and extract the main framework. This process aims to help us adjust the pre-processing process of the model and prepare for changing the model's architecture or retraining the model in the future. During this process, we also extract the input parameters which are needed when converting the PyTorch model to the ONNX model.

(2) Convert the model from the PyTorch model to the om model:
We use the extracted parameters to write the conversion script to convert the Pytorch model to the ONNX model, then we check the model's architecture through Netron. If the architecture is correct, then we use the following command to convert the ONNX model to the offline model (om model):

```
atc --input_format=NCHW --model=[model name].onnx --framework=5
--output=[model name] --soc_version=Ascend310
```

(3) Evaluate the results:
First of all, we initialize the ACL runtime. Secondly, we load our om model by the AclLiteModel method. Thirdly, we pre-process the test image and feed it to our model to generate predictions. Fourthly, for the detection-based model, we extract joints from heatmaps. Fifthly, we evaluate the output to check if it is the same as the output of the original Pytorch model. If the output is not as we expected, we will go back to the Pytorch model to debug. Sometimes, we change the original pre-processing process of the model's inputs or adjust the layers in the models. After fixing the bug, we will retrain the model and reconvert the PyTorch model to the om model.

(4) Write the pre-processing pipeline for reading the video and the post-processing pipeline for generating the gif image:
If the output of the offline model is as we expected, we will use our test video to evaluate the performance of the model for real-world inputs. We write the pre-processing code to capture our input video into frames for model inference, and we write post-processing code to combine the output images into a GIF image.

4. Problems and Solutions

During the process of the project, we encounter many problems. The problems and our solutions are as follows:

4.1 Data type mismatch

When deploying the 2D Hand Pose Estimation RGB model to the Atlas 200 DK device, we meet a data type mismatch problem. Because the data type we use for the inputs of the om model is different from the one of the PyTorch model. To solve this problem, we try different data types for the om model, however, none of them works. Eventually, we decide to change the pre-processing step for inputs of the PyTorch model to make sure the data type could keep the same as the one on the Atlas device. This solution raises the problem in part 4.2, and this problem is fixed after solving the problem in part 4.2.

4.2 nn.BatchNorm2D problems

This problem is raised by the solution in part 4.1. After we change the data type of the inputs for the PyTorch model, we meet an error that is caused by the mismatch between the data type of the model inputs and the model parameters. After debugging, we need to convert the eps parameter of nn.BatchNorm2D from long to float32. Besides, we also find that if we do not give a fake input for the num_batches_tracked parameter of nn.BatchNorm2D, it may cause some errors when converting to the ONNX model. After solving this problem and re-training the model, the problem in part 4.1 is also solved.

4.3 torch.inverse problems

When converting the Real-time 2D and 3D Hand Pose Estimation model from the PyTorch model to the ONNX model, we meet a runtime error, as ONNX does not support torch.inverse right now. Though we can try to use numpy.linalg.inv to replace this operation and re-train the model, we decide to change to another model. This is because the manually tuning requirements of the joints' 2D coordinates and the camera calibration process could raise some precision problems, which might be time-consuming for fixing. Due to the time limit, we change it to another model.

5. Results and Analysis

5.1 Final results

5.1.1 Result of the 2D Hand Pose Estimation RGB model

Fig 3. One frame of the output of the 2D Hand Pose Estimation RGB model

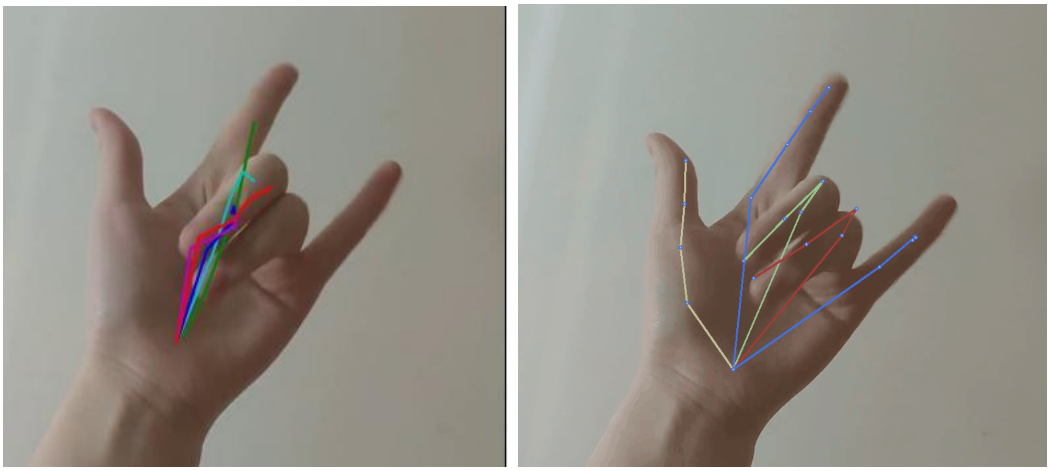
The prediction of the 2D Hand Pose Estimation RGB model does not have good accuracy. This model could work if we spread our fingers, however, it does not work well if we flex our fingers.

The FPS for this result is **39** frames per second. The output information is as follows:

```
image in list:  126
total time:  3.2245962619781494
fps: 39.07465920173972
```

Fig 4. The output information of the output in Fig 3.

5.1.2 Result of the HandPose X model



(a) (b)

Fig 5. The same captured from the output of two models for comparison
(a) A result of the 2D Hand Pose Estimation RGB model
(b) A result of the HandPose X model

Through the above comparison, we find that the prediction of the HandPose X model has much higher accuracy than that of the 2D Hand Pose Estimation RGB model. The HandPose X model also works well when we flex our fingers.

The FPS for this result is **27** frames per second. The output information is as follows:

```
image in list:  97
total time:  3.539405345916748
fps: 27.405733596437184
acl resource release all resource
```

Fig 6. The output information of the output in Fig 5 (b).

5.2 Analysis

Considering the different methods, losses, and network architectures of the two models, we infer that the **HandPose X model** has better predictions than the 2D Hand Pose Estimation RGB model due to the following reasons:

1. The HandPose X model predicts the coordinates of joints directly, which could avoid the possible precision problem when extracting joints from heatmaps.
2. The HandPose X uses the Wing loss, which is proposed for keypoints detection of human faces, and it could have a larger gradient when the error is small. It could work better than the IOU loss of the 2D Hand Pose Estimation RGB model.
3. The network architecture of the 2D Hand Pose Estimation RGB model is too shallow, which leads to its low accuracy.

6. Conclusion

Through this project, we learn the workflow of hand pose estimation and improve our debugging skills in the process of model conversion. We also learn new knowledge about how to convert and use the ONNX model and how to use the Atlas 200 DK device to accelerate the computation of CNN. We try different models and achieve a high accuracy model with FPS larger than 20 frames per second in the end. We appreciate that our professor and Huawei Technologies provide such a good opportunity for us to learn and experience the workflow of developing pipelines to use the models. We also appreciate that Hongyi helps us deal with problems and bugs and Derek provides pretty useful instructions for our project.

Reference

[1] Bardia D. Hand pose estimation: a survey. arXiv:1903.01013. 2019.
[2] Cem K. et al. Hand pose estimation and hand shape classification using multi-layered randomized decision forests. In European Conference on Computer Vision, pages 852–863. Springer. 2012.
[3] Kawulok M. et al. Hand pose estimation using support vector machines with evolutionary training. In IWSSIP 2014 Proceedings (pp. 87-90). IEEE. 2014.
[4] Tomas S. et al. Hand keypoint detection in single images using multiview bootstrapping. In CVPR, volume 1, page 2. 2017.
[5] Liuhao G. et al. 3D Hand Shape and Pose Estimation from a Single RGB Image. In CVPR 2019. arXiv:1903.00812. 2019.
[6] Chen Q. et al. Realtime and robust hand tracking from depth. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 1106–1113, 2014.