Hand Pose Detection

CMPT 733

by
Liam Sparling
Student Number: 301464281
Kritu Patel
Student Number: 301469200

Taught By
Dr. Ali Mahdavi Amiri
Date of Submission
April 16, 2022

PMP Visual Computing

Contents

List of Figures

List of Tables

Listings

Abstract

This project was proposed by Huawei and requires the execution of a Hand Pose Estimation model on an Atlas 200 Development Kit. The model must have a frame-rate which exceeds 20 FPS and must perform accurately.

The document below describes the research, evaluation, conversion, execution, and challenges involving setting up Hand Pose Estimation models onto the Atlas 200 Development Kit. The Mediapipe model [1], Victor Dibia's Hand Pose model [2], Minimal Hand model [3], and Open Pose model [4] were all evaluated on a desktop computer. The Open Pose had the worst frame-rate, around 2 FPS, but was considered for it's high accuracy and limitations in the code framework that could be optimized. From these models, the Open Pose model was successfully converted and run on the Atlas 200 DK with a frame-rate of 10 FPS. Optimizing the execution of this model led to a frame-rate of above 18 FPS. Modifying the model to use smaller images led to a reduction in accuracy but improved the frame-rate to 21 FPS. The other models evaluated previously were unable to be executed for reasons involving code versioning and support of the ATC (Ascend Tensor Compiler).

This document first explains the models that were tested and their comparison to one another. The conversion, execution and optimization of one of the models onto the Atlas 200 DK is then explained. Finally, the report concludes with an evaluation of the challenges that were faced, with some final remarks and additional notes.

1 Models

The 4 following models were the only ones the team managed to run so initially. The GIFS folder contains gifs for the first 3 models. You can also click the name of the model to open the related project url. Each test was run on a Ryzen 5800 CPU, without GPU acceleration.

Model	**Accuracy**	**FPS**
Mediapipe [1]	Best	34
HandPose-1.0 [2]	Worst	29
Minimal Hand [3]	Pretty Good	10
Open Pose [4]	Very Good	2

TABLE I
MODELS RUN ON AMD RYZEN 7 5800

As explained by team's Huawei contact Derek, the Mediapipe model [1] is implemented in TFLite, which is not supported on the Atlas 200 DK. However, this model is still listed as it is useful to have as a reference baseline.

1.1 HandPose-1.0 Model

The HandPose-1.0 model, created by Victor [2], doesn't have the typical key-point detection, and instead uses a CNN approach to determine different hand poses. This approach does not have a great hand pose classification, but does have a fairly good detection model. There is also a newer version which may have a better hand model.

1.2 Minimal Hand

The Minimal Hand model, created by Yuxiao Zhou [3] has good hand detection and tracking. It also adds the MANO model to output a graphical hand instead of key-points. However, this model is very slow, due to the time it takes for the detector model to run. This model by itself, would cause a frame-rate of around 14 fps. The tracking model or key-point model is much faster, resulting in an fps of 300 if it was run by itself.

1.3 Open Pose

The Open Pose model [4] has outputs results similar to the Mediapipe model as it tracks 22 points on a hand, but it has the worst frame rate out of the group. It was expected that this limitation in frame-rate is due to it's usage of Open CV to run the model, and the lack of GPU accelerated

processing in our tests. It was assumed for the model itself not to be limiting the frame-rate because as shown in Figure 1, the model is only 60 layers deep. This model takes an RGB image as input and outputs a probability map for a scaled down version of the image. The code then has to rescale that output map to the original image, then calculate the highest probabilities for each of the 22 points. If these points are above a set threshold (0.2 or 20%), then they are output to the image.

2 Atlas 200 DK Implementation

2.1 Open Pose Implementation

The Open Pose model was the only model that was successfully converted to the Atlas 200 DK. To understand the complications of the other two models, please refer to the **ATC conversion section 3.3**. To convert this model, the model file (*.prototxt) and weights file (*.caffemodel) were copied over to the Atlas 200 DK. After copying the file, command shown in listing 1 was run.

Listing 1: ATC Function Call

```
atc --model=pose_deploy.prototxt --weight=pose_iter_102000.caffemodel
    --framework=0 --output=openpose --soc_version=Ascend310 --
    input_shape="image:1,3,368,449"
```

This call creates an offline model that can be used on the board. In order to use this new model, it is needed to modify the original code structure. Open CV is supported on the Huawei board, so the team was able to run the model as is, using the original caffe models, with similar results to what was achieved on the desktop computer. However, these results are of too low of a frame-rate to consider, so no attempts were made to optimize or use the original Open CV code.

The model was converted using a combination of the original Open Pose code and the Huawei ACL lite code used in their Image Object Classification Atlas 200 DK examples. After converting the code using the structure of the Huawei implementation with the new offline model, a frame-rate of **10 FPS** was achieved. This rate was significantly better than what was initially achieved on the desktop computer, but did not meet the requirements so it was needed to optimize the model.

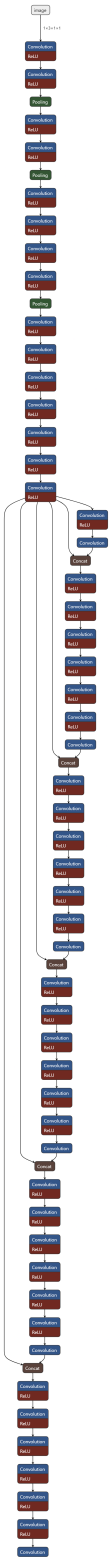

Figure 1. Open Pose Model

Here is a list describing of the folder and file structure of the repository.

- **inputs**
  Contains the input video files
- **outputs**
  Contains the output video files
- **model**
  Contains the OM model file
- **openposerun.py**
  The executable python file. Contains all the major code for opening a video and executing the model
- **model_processing.py** Contains code for pre-processing and executing the model using ACL resources.

2.2 Optimization Strategies

After the model had been converted and tested on the Atlas 200 DK, the team moved forward with checking where the model was taking the most amount of time. The following times that are shown in table II were checked and monitored . In order to meet the goal of 20 frames per second (FPS), the model was needed to take an average of 0.05 seconds per evaluated frame.

Model Section	Time (s)
Pre-processing	0.03
Model	0.04
Post-processing	0.03
Open CV Ouput Lines	0.00

TABLE II
MODEL TIMINGS

When originally converting the model, a step-by-step approach was used in the pre-processing step. This approach was significantly slower, taking the 0.03 seconds listed above, then the Open CV implementation function call, which does every step in a single call, used in the Open Pose code, which only took 0.015 seconds.

In the **Open CV and Frame Parsing** step, the output of the model is used to find the peaks on the probability map for the 22 points tracked on the hand.

Version	**#Threads**	**Task per Thread (Separated by &)**	**Average FPS**
1	2	Model Full Processing & Open CV Processing	14
2	2	Model Only Processing & Pre, Post, Open CV Processing	14
3	2	Model and Post Processing & Pre, Open CV Processing	17
4	3	Pre & Model, Post & Open CV Processing	>17
5	4	Pre & Model & Post & Open CV Processing	18-19

TABLE III
OPTIMIZATION STRATEGY COMPARISON

The following sections explain 3 of these optimization strategies in more detail and concludes with an additional optimization. In each of these sections, blocking queue pushes and non-blocking queue pulls are used. This approach allows the main thread to push an item to the queue, then skip a pull from the output queue before the other thread(s) are finished their tasks. This approach allows the main thread to push multiple items to the queue instead of waiting idle. The input queues are all max size of 1, as using a larger size created too much delay between the input video and output video frames (the Open CV lines will be written on a frame 5 frames later, for example, if the max queue size was 5).

2.2.1 Strategy 1

For this strategy, the main thread is responsible for reading in the frames from the video stream, and adding the points to the frame using Open CV after the model has been executed on it. The other thread is responsible for executing the entire model, the pre-processing, model execution, and post-processing steps, before passing the image back to the main thread. This approach speeds up the execution because the main thread can work on a previous output from thread 2 while the second thread is working on a different frame. Since these can be done simultaneously, we received a speedup of about 4 FPS, resulting in 14 FPS for this approach.

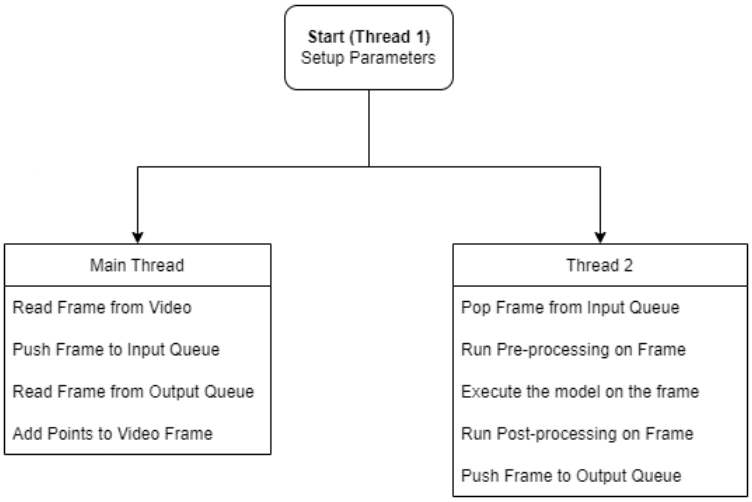

Figure 2. Strategy 1

2.2.2 Strategy 3

For this strategy, the main thread is responsible for reading in the frames from the video stream, running the post-processing steps, and using Open CV to add points to the image. The only difference from Strategy 1 is this approach allows a better distribution of work between the threads. This approach reduces the time the second thread takes per frame, and increases the time the main thread takes per frame. This allows each frame to take an average of 0.058 seconds per frame, resulting in a frame-rate of 17 FPS.

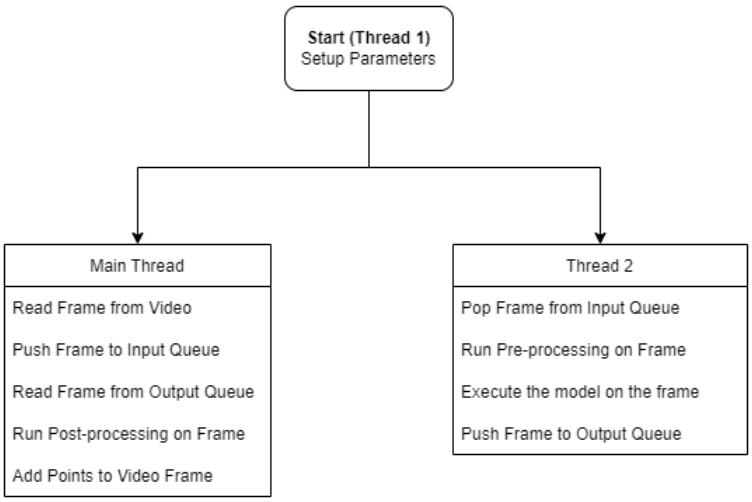

Figure 3. Strategy 3

2.2.3 Strategy 5

For this final strategy, the multi-threading approach was taken to it's limit. Here a total of five threads are used, one for pre-processing, one for model execution, one for post-processing, and one for reading in the video frame and writing the Open CV points. Although, this would theoretically result in a frame-rate of above 20 FPS, due to the unequal distribution of workload between the threads, a frame-rate of 18 FPS was achieved. There is also the setback of additional delay of the Open CV points on the output video. As there are 4 threads here, up to 3 frames of delay can occur on the output video, which can be compared to the 1 frame of delay on the 2 thread model. With the original implementation, this delay caused the video to be about 3 frames ahead of the open CV output points. A new file version, suffixed by `delayshift`, delays the outputs frames as well to make the video look better. However, if using live video, the output video would always be 3 frames behind.

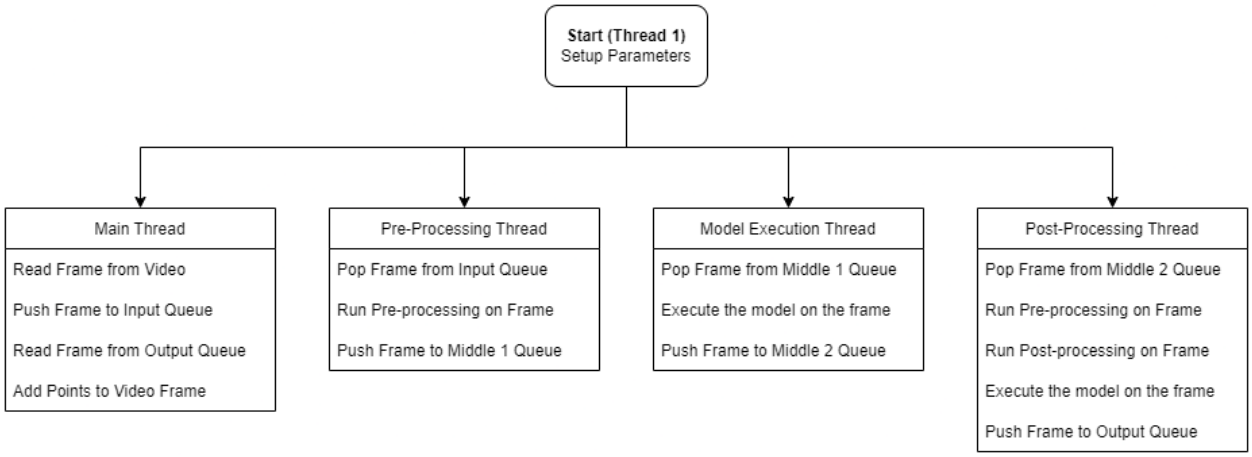

Figure 4. Strategy 5

2.2.4 Reduction of Image Size

In addition to these multi-processing strategies, it was discovered that the model's size could be reduced by specifying a smaller number for the input image size. A smaller image reduces the amount of operations at each step of the model, which reduces the amount of time it takes to process a single image. With reducing the input height from 364 to 300, the team was able to achieve results of above 21 FPS on the asl.mp4 video. The problem with this approach is that the results from the model also reduce with lower image sizes, and may improve with higher resolutions. To compare these two approaches, please look at the results in the output folder in the GitHub repository.

3 Challenges and Set-backs

3.1 Image Models

These challenges shaped our project direction.
We initially wanted to find multiple modern hand pose estimation models that were trained on modern datasets. These models had the problems listed as follows,

- Image Models
- Software Support
- Depth Models

As they were all image models, meaning they were trained and tested on specific datasets and trained on high performance computers, they were huge models with massive datasets that required a lot of setup. Additionally, the team had no expertise on the video performance of these models, as they may be extremely accurate models, but they sacrifice speed for that accuracy. Some of the models also had out-of-date software versions, such as Tensorflow version 1, which required rewriting some of the network to support Tensorflow version 2. Additionally, as almost all models were written in Ubuntu, they had packages that were either not supported, or required complicated setup for windows.
These models were often trained using depth images, or depth and RGB images. This approach means the models can not be directly converted to an RGB approach as required for our cameras. The following models fell under the above categories.

- MMPose [5]
- MeshGraphormer [6]
- Adaptive Weighting Regression (AWR) [7]
- HandAugment [8]
- V2V-PoseNet [9]

Note: Some models, like HandAugment, are based on RGB and Depth based input, so it may be possible to split the model on its two CNN pipelines and extract the part processing the RGB input data.
Also, considering that some of these models were written in Ubuntu, it may be possible to set them up on the Atlas 200 DK board with greater ease than on Windows. However, it would be unexpected for that to be the case so this approach should only be taken if necessary.

3.2 Converting the Model to ATC Supported Format

ATC (Ascend Tensor Compiler) supports a limited format of models for conversion. The formats supported by ATC are Tensorflow, ONNX, Minspore and Caffe. This limits our choice of model selection to run on our local systems.

- Two pytorch models, written by Yangli18 [10] and ZLLrunning [11] (github usernames), were found online. According to the guide posted by Huawei, these models needed to be first converted to an ONXX format followed by an ATC conversion. In order to convert the model, The model needed to be runnable on a local desktop computer. This conversion was not possible because as both of these models were using old code and was unsupported. Moreover it was built for a LINUX system and our Linux system did not have an external GPU, as was required for the setup.
- The YOLO Hand model [12] was written in a Darknet Model format. The team found three Darknet conversion tools to attempt to convert this model to a usable format. It was discovered that Darknet model can be converted to Tensorflow which is one of the supported format for ATC conversion to .om model. However the team faced similar issues trying to run them on the local systems and converting them, which were
  - The model didn't run at all on local system(deprecated libraries)
  - Or the model was built specific to LINUX system

The team decided to not proceed with these models due to the aforementioned reasons hence narrowing down our list of models for conversion to .om model.

3.3 ATC Conversion

To run the model on Atlast 200Dk board an .om model is needed. This model is an offline model format supported by the board. The ATC (Ascend Tensor Compiler) tool converts the network

models of open-source frameworks into offline models supported by Ascend AI Processors. Usage of this tool only requires single line command with command line parameters.

As mentioned previously, only models in Tensorflow, ONNX, Minspore and Caffe formats are supported.

Apart from trying to convert models from other formats to one of the supported types a couple of models were found which were in supported formats

- ATC supports Tensorflow but the Tensorflow model used in Victor Dibia's **victor** model contained multiple outputs (such as a detector box, number of detection, classes and scores) and it was assumed that this complexity didn't allow conversion to an offline model.The root cause of this problem couldn't be established but it was decided to not go any further with this model.
- Even though there is a limited model choices due to ATC support, the team was able to find a Caffe model, Hand-Keypoint-Detection [13], but it used functions which were deprecated and not supported for conversion.
- Apart from that, other models were found which were using either Pytorch ,CKPT files or Darknet format , which aren't directly supported by the ATC

The first conversion of the Open Pose model to an offline model did not result in a usable model. The original model supported multiple input image sizes, but to be usable as an offline model, a specific size needed to be specified. The original command line call is in listing 2.

Listing 2: Incomplete ATC Function Call

```
atc --model=pose_deploy.prototxt --weight=pose_iter_102000.caffemodel
  --framework=0 --output=openpose --soc_version=Ascend310
```

In this command, since no input image size was specified, the created model only accepted image sizes of shape (1, 1). This size is obviously useless for all images, and therefore the model was regenerated as explained in section 2.1.

4 Conclusion

Huawei originally tasked the execution of a Hand Pose model onto their Atlas 200 DK board with a frame-rate that exceeded 20 FPS and which performed accurately. These requirements were able to be met through a multitude of optimizations, and a model which performed on the requirements, upwards of 21 FPS on the Atlas 200 DK, was executed on the board.

With the experience gathered from the challenges listed in this document, in addition to the experiments performed through optimizing the model which was successfully converted to an offline format, the team would have clear direction on creating a model to perform well on the 200 DK. Should the team decide to return to this project and attempt to find a better model, with a much faster frame-rate, they would first analyze the existing body-pose models available on the board. As the execution time of the model is the limiting factor in the Open Pose implementation, and is expected to be for other models, given the small-size of the Open Pose model, it is hypothesized that the layout of models, which already perform efficiently on the Atlas 200 DK, may give insight on models that work efficiently on the board. Additionally, should one of these models be already considered the best for body-pose estimation, it may be possible to transfer learn and retrain the model using a hand-pose data-set.

References

[1] V. Bazarevsky, G. Ivan, G. B. Eduard, *et al.*, *3d hand pose with mediapipe and tensorflow.js*, 2020. [Online]. Available: https://blog.tensorflow.org/2021/11/3D-handpose.html.

[2] V. Dibia, *Handtracking: Building a real-time hand-detector using neural networks (ssd) on tensorflow*, 2021. [Online]. Available: https://github.com/victordibia/handtracking.

[3] Y. Zhou, M. Habermann, W. Xu, I. Habibie, C. Theobalt, and F. Xu, "Monocular real-time hand shape and motion capture using multi-modal data," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2020.

[4] V. Gupta, *Hand keypoint detection using deep learning and opencv*, Oct. 2018. [Online]. Available: https://learnopencv.com/hand-keypoint-detection-using-deep-learning-and-opencv/.

[5] *Mmpose open source project*, 2022. [Online]. Available: https://www.opensourceagenda.com/projects/mmpose.

[6] K. Lin, L. Wang, and Z. Liu, "Mesh graphormer," in *ICCV*, 2021.

[7] W. Huang, J. Wang, Q. Qi, H. Sun, and P. Ren, *Papers with code - awr: Adaptive weighting regression for 3d hand pose estimation*, Jul. 2020. [Online]. Available: https://paperswithcode.com/paper/awr-adaptive-weighting-regression-for-3d-hand.

[8] Z. Zhaohui, X. Shipeng, C. Mingxiu, and Z. Haichao, *Papers with code - handaugment: A simple data augmentation method for depth-based 3d hand pose estimation*, Jan. 2020. [Online]. Available: https://paperswithcode.com/paper/handaugment-a-simple-data-augmentation-for.

[9] dragonbook, *V2v posenet pytorch open source project*, 2020. [Online]. Available: https://www.opensourceagenda.com/projects/v2v-posenet-pytorch.

[10] yangli18, *Yangli18/hand$_d$etection: A light cnn based method for hand detection and orientation estimation*, 2019. [Online]. Available: https://github.com/yangli18/hand_detection.

[11] Zllrunning, *Zllrunning/hand-detection.pytorch: Faceboxes for hand detection in pytorch*, 2019. [Online]. Available: https://github.com/zllrunning/hand-detection.PyTorch.

[12] F. Bruggisser, *Cansik/yolo-hand-detection: A pre-trained yolo based hand detection network.* 2022. [Online]. Available: https://github.com/cansik/yolo-hand-detection.

[13] Hzzone, *Hzzone/hand-keypoint-detection: Hand and keypoint detection by caffe.* 2018. [Online]. Available: https://github.com/Hzzone/Hand-Keypoint-Detection.

[14] J. Romero, D. Tzionas, and M. J. Black, "Embodied hands: Modeling and capturing hands and bodies together," *ACM Transactions on Graphics, (Proc. SIGGRAPH Asia)*, Nov. 2017. [Online]. Available: http://doi.acm.org/10.1145/3130800.3130883.

[15] M. Loper, N. Mahmood, J. Romero, G. Pons-Moll, and M. J. Black, "SMPL: A skinned multi-person linear model," *ACM Trans. Graphics (Proc. SIGGRAPH Asia)*, vol. 34, no. 6, 248:1–248:16, Oct. 2015.

[16] *22 hand pose estimation open source projects.* [Online]. Available: `https://www.opensourceagenda.com/tags/hand-pose-estimation`.

[17] *Papers with code - freihand benchmark (3d hand pose estimation).* [Online]. Available: `https://paperswithcode.com/sota/3d-hand-pose-estimation-on-freihand`.

Appendix

The following list is files that we have come across, for reference and information.

File Names

Tensorflow Files

- .pbtxt = (Tensorflow Graph Text) Readable pb file
- .pb = protbuf = contains the graph definition as well as the weights of the model (i.e. the entire model)
- .ckpt-meta = the graph for the model
- .ckpt-data = the values for the variables (weight and bias, possibly more)
- .ckpt-index = a linking for the two previous files

Caffe Model Files

- .prototxt = The graph for the model
- .caffemodel = The weights for the model

Python Files

- .pkl = pickle file (python format)

Offline Model Files

- .om = Offline model file (graph and weights)

Additional Notes

The following models, MANO [14] and SMPL [15], are used in creating a mesh which accurately simulates and hand or body.

The team found the following hand pose models listed on the website [16], particularly useful for finding hand pose models. Using a comparison of results on a particular data-set can also be useful, such as the FreiHand dataset [17].