# AUTOMATIC KEYPOINT DETECTING OF WIREFRAME GATES

by

**Wei Sun**

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly Thursday July 11, 2019 at 10:00 AM

| | | |
|---|---|---|
| Student number: | 4711114 | |
| Project duration: | September , 2018 – July 2019 | |
| Thesis committee: | Prof. dr. M. J. T. Reinders, | TU Delft, chair |
| | Dr. J. C. van Gemert, | TU Delft, daily supervisor |
| | Dr. G. C. H. E. de Croon, | TU Delft, committee member |
| | Dr. Rob Wijnhoven, | Vinotion B.V., committee member |



TU Delft Delft University of Technology

# PREFACE

The report documents my thesis work. The report consists of two parts

- Scientific report includes introduction, methodology, implementation, experiments & results and discussions of the major thesis work. This part is written as conference-paper(CVPR) style. **This part is the most important part, if you have enough general background knowledge of deep learning for computer vision, then you probably prefer to read this part only.**
- The chapters after the scientific paper introduce some background knowledge needed to understand this work. Some meaningful failures are also presented in this part to give insights. **If you lack basic knowledge of deep learning for computer vision or you are interested in the failure cases, I would suggest you read related chapters of this part as well.**

This work would not have been possible without Dr. Jan van Gemert who was kind, considerate and supportive. He provided me with invaluable ideas and guidance while still affording me the freedom to test my own ideas. He also helped me to build the essential skills of how to do research and let me be well-prepared for my future study.

I would like to thank Yancong Lin who was my former thesis advisor. Although I terminated my cooperation with him, his early helps are still appreciated.

Many thanks to all the wonderful friends I met in Delft. I was fortunate to be accompanied by them on my journey towards my graduation.

Most importantly, none of these would have been possible without the support of my parents. I would like to thank them for always being by my side no matter what.

# CONTENTS

# 1

# Scientific Paper

# Automatic keypoint detecting of wireframe gates


## Abstract

*This work applies keypoint detection method to solve gate recognition problem. Unlike regular object detection task, gate recognition problem is made difficult by the fact that gate is empty wireframe which means that the object surrounded by gate-edge is not relevant and should not be taken into consideration when detecting. However, regular object detection algorithms will process on whole pixels of specific region and give the results as bounding box with object class. The architecture used in this project consists of two branches which are corner detector and edge detector respectively. Detected corners and edges are highlighted in heatmaps. We first verify the correctness of our model in the toy dataset and upgrade the model to work on more complex dataset. The experimental evidence shows the performance and functionality of our network intuitively.*


## 1. Introduction

**Project background.** Gate recognition is a popular task for drone race in the UAV (unmanned aerial vehicle) field. In this kind of race, the properties of drone system, such as control system, power system as well as perception system, would be evaluated in real world. Visual system is an important part of perception system, the images captured by the cameras on the drone would be processed by embedded visual system. The control unit of UAV makes decision based on the information extracted by the visual system.

Deep neural network has shown its abilities for solving complex visual tasks. More and more traditional computer vision algorithms in various fields are being replaced or improved by deep neural networks. Before deep learning, gate recognition could be seen as corner detection or line detection and relevant algorithms are Harris corner detector [7], Canny edge detector [1] and Hough line transform [3].

**Challenges.** Since the popularity of deep learning, some deep learning architectures for object detection have been published and achieved state-of-the-art performance over non-learning-based algorithms on public dataset(e.g. COCO dataset and ImageNet dataset) [29][18]. Generic bounding-box-based object detectors make decision based on whole pixels and localize objects by bounding box.

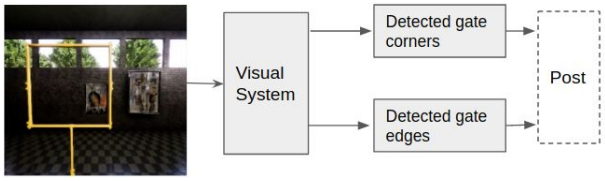



Figure 1: Overview of the thesis work. Visual System is the deep learning model we built. It processes the input image which simulates the real-world image captured by camera and detects corners and edges of gate simultaneously. The dotted-line polygon $Post$ stands for the grouping algorithm which can assemble gates based on detected corners and edges when the input image contains multiple gates. This part is not finished in this work but some discussions for future work are documented

However, for this specific task, generic bounding-box-based object detectors such as Faster R-CNN [26] and YOLO [22] are not suitable and could be error-prone [4]. The main reasons are: i) There is only one class – gate, so classification is not needed. ii) Unlike regular objects, gate is a kind of empty wireframe. So when we want to determine if the object is a gate or not, only edges and corners should be considered and the specific background objects surround by edges are not relevant. But for regular bounding-box-based object detection algorithms, every pixel in proposed region would be considered when the network makes decision.

**The better suited approach could be to locate the keypoints (i.e. corners) and edges of wireframe gate. Figure 1 illustrates the system. The keypoints method is inspired on human pose estimation [2] where keypoints of human body (i.e. joints) are detected and grouped per individual**.

**The major contributions of this thesis work are: 1) Link human pose estimation task to the wireframe gate detection task. 2) Build toy model based on the state-of-the-art methods [2][19] and verify on toy dataset. 3) Upgrade the toy model and test on more complicated dataset, modify the network when failure occurs. 4) Conduct some experiments to demonstrate the properties of our neural network. 5) Propose some recommendations for future work**.

## 2. Related work

### 2.1. YOLO-based neural networks for wireframe gate detection

P. Duernay [4] trained a simplified YOLO [24] to solve the wireframe gate problem. Training a deep learning model needs a lot of labeled images and the labeled images captured from real world is limited. So [4] created a labelled simulated dataset by using a video game engine(i.e. Unreal Engine[1]). The simulated images are labelled automatically by the game engine thus the intensive labour could be saved. We call the simulated dataset as Unreal dataset. Therefore, in this project, we use Unreal dataset instead of creating real dataset manually.

The results of an experiment in [4] show that YOLO-like model would get confused when the objects inside wireframe gate are not included in the train dataset. Figure 2 shows the corresponding results. We can see that the YOLO-like model gets confused and the accuracy will drop dramatically. Although those images(i.e. sign in gate and cat in gate) are unlikely to happen in real world, the experiments illustrate the drawback of using YOLO-like model to detect wireframe gate.

| Trained/Tested | Gate [$ap_{60}$] | Sign [$ap_{60}$] | Cats [$ap_{60}$] |
|---|---|---|---|
| Gate | $0.55 \pm 0.04$ | $0.01 \pm 0.00$ | $0.37 \pm 0.06$ |
| Sign | $0.02 \pm 0.01$ | $0.74 \pm 0.06$ | $0.05 \pm 0.04$ |
| Cats | $0.06 \pm 0.03$ | $0.03 \pm 0.00$ | $0.78 \pm 0.01$ |

Figure 2: Results from[4]


Figure 3: Example of gate, sign and cat

In our project, the keypoint-based method is expected to not get confused in the cases above.

### 2.2. Traditional computer vision algorithms

**Harris Corner Detector** . Harris Corner Detector [7] is a traditional computer vision algorithm to extract the corners of the input image. Figure 4 shows an example of Harris Corner Detector result. We can see that the Harris detector can not distinguish different corners. Specifically, what we want is corners of gate but the the Harris detector annotates all possible corners including corners of window and corners of light.

Figure 4: Harris Corner Detector result, red points are detected corners

**Canny edge detector**. Canny edge detector [1] is a multi-step algorithm that can detect edges with noise suppressed at the same time. Figure 5 shows the result of Canny edge detector. The result shows the same problem in Harris Corner Detector that edges of window and wall are also annotated.

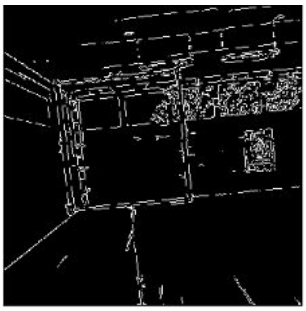


Figure 5: Canny Edge Detector result, white lines are detected edges

The traditional computer vision algorithms(i.e. Harris and Canny) can not handle background objects effectively. Non-deep-learning-based computer vision algorithms usually work in very low dimension so they are not able to classify different objects which have similar shape(i.e. corners or lines of different objects).

### 2.3. Deep learning algorithms

#### 2.3.1 Generic object detector

Deep-learning-based generic object detectors can be divided into two types – **Two-stage object detectors** and **One-stage object detectors**. R-CNN families – R-CNN [6] , Fast R-CNN [5] and Faster R-CNN [26] are good examples of two stage object detectors. Two-stage detectors first generate a set of regions of interest (RoIs) and then classify

[1] https://www.unrealengine.com/

each of them [15]. For instance, in faster R-CNN, the first step is finished by Region proposal network which generates region proposals from a set of candidate boxes. These proposed boxes are then forwarded to the second stage - classification sub-network. So, in two-stage algorithms, detection is explicitly divided into two stages – i) Localizing possible regions(RoIs). ii) Classifying objects of proposed regions. On the other hand, one-stage object detectors remove RoIs proposal part and detect objects directly on the complete image. YOLO [22] is a popular one-stage detector. YOLO predicts coordinates of bounding box directly from the input image. After a series of improvements, YOLO is now in the third version [24]. Generally, one-stage object detectors are more efficient and faster than two-stage detectors, while two-stage detectors can achieve higher accuracy [29]. So, for embedded applications, one-stage object detectors like YOLO are better suited than two-stage object detectors.

#### 2.3.2 Human skeleton pose keypoint detection

The method of this project is inspired by Human skeleton pose keypoint detection which is designed to detect the joints of human body and group the detected joints per individual correctly. Just like generic object detectors, Human skeleton pose keypoint detectors can be divided into two classes as well – bottom-up approach and top-down approach [20][18].

This thesis work is built on bottom-up approach which usually consists of two sub-networks. The first sub-network is designed to detect the joints of human body. This task is normally achieved by means of highlighting the positions of joints and producing the corresponding heatmaps[2]. The coordinates of local maxima are the coordinates of detected joints. The second sub-network is responsible for predicting clues with which the detected joints can be grouped correctly. **Associative Embedding** [19] uses embedding to achieve grouping assignments, they use vector embedding as identity tags in the context of joint detection and grouping [19]. Tag values of joints which belong to a same person would be close , otherwise the difference would be large. Associative Embedding has no ground truth, it uses two well-designed loss functions – pull loss and push loss, to cluster joint tags of same person together and separate joint tags from different person. **Part Affinity Fields(PAFs)** [2] is a 2D vector field for each limb between two joints, for each pixel in the area belonging to a particular limb, a 2D vector encodes the direction that points from one part of the limb to the other [2]. Unlike Associative Embedding, PAFs encode ground truth(2D vectors) as heatmap, and Mean squared error is used as loss function to measure the difference between ground truth PAFs and predicted PAFs.

[2] https://en.wikipedia.org/wiki/Heat_map

In this thesis assignment, both **Associative Embedding** and **Part Affinity Fields(PAFs)** have been investigated. **Associative Embedding** fails on toy dataset[3], so the final model of this work is based on the network architecture of **Part Affinity Fields(PAFs)**.

### 2.4. Backbone Network

In complex neural network architectures, backbone network(i.e. the feature extractor) plays a role in processing raw input images and generating high-dimensional feature maps. The generated feature maps are fed into following components like classification part, region proposal part in object detection networks or keypoint detection part. Choosing which backbone depends on the specific use case [10]. **ResNet** [8] innovates a new block – residual block, as illustrated in Figure 6. This new block is designed to solve vanishing gradient problem caused by deep convolutional layers. Residual block makes deep networks easier to train and achieve higher accuracy by increasing the depth of networks. ResNet and its variants are widely used [26] [10] as backbone network in deep neural networks which are designed to work on complex dataset. **SqueezeNet** [12] is a network architecture targeted for embedded applications where memory and computing resources are limited. Figure 7 shows the building block of SqueezeNet. The main idea behind SqueezeNet is that by replacing 3x3 filters with 1x1 filters which have 9 times fewer parameters and decreasing the number of 3x3 filters to achieve a reasonable accuracy with significantly smaller model size (i.e. less parameters). In [12], they state that SqueezeNet achieves AlexNet-level [14] accuracy on ImageNet with 50x fewer parameters. **In this thesis work, both of these two architectures are investigated as backbone network.**

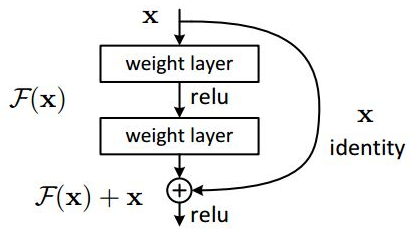



Figure 6: Residual block [8]. This innovative block allows us to build much deeper neural networks and will not suffer vanishing gradient problem

## 3. Simplified model on toy dataset

### 3.1. Toy dataset

Figure 8 shows concepts of toy dataset generated by myself. Each image (64*64) has two polygons to simu-

[3] failure case can be found in background part

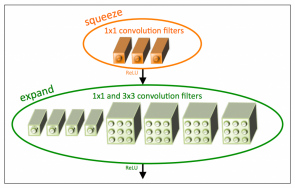



Figure 7: Fire Module, Micro-architecture of SqueezeNet [12]. This blocks help to reduce the number of parameters of standard convolutional blocks without losing too much accuracy

late gates, and the coordinates of corners for each gate are ground-true annotations.

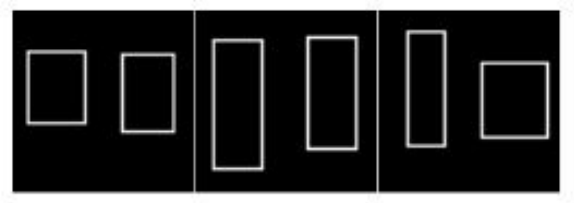


Figure 8: Toy dataset: the white polygons simulate gates

### 3.2. Model

Figure 9 shows the general overview of our model architecture. The model consists of three components:

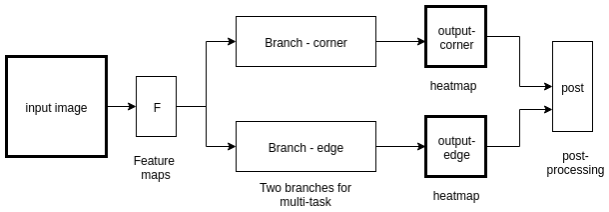



Figure 9: Overview of network architecture. The pipeline starts with the input image which is processed by backbone network $F$. The generated feature maps are fed into two branches simultaneously. Branch-corner is responsible for detecting gate corners and branch-edge is responsible for detecting gate edges

**i**) Backbone network(i.e. feature extractor) $F$. $F$ consists of several convolutional filters which are responsible for extracting high-dimensional features from the low-dimensional input image. Figure 10 shows the feature extractor in this toy network. The generated feature maps are then fed into following sub-networks for further tasks.

**ii**) After backbone network, the generated high-dimensional feature maps should be processed further before we can get the desired outputs (i.e. corners and edges of gate). Thus, two branches following backbone network are built for multi-task joint learning as shown in Figure 9. The upper branch is responsible for predicting corner heatmap where detected corners are highlighted. And the lower branch is for the edge heatmap where detected gate edges are with high confidence values. Figure 12 illustrates an output example.

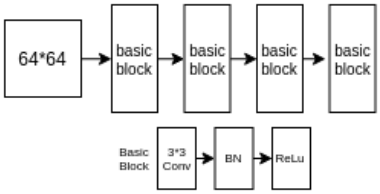



Figure 10: Feature extractor in toy network. Basic block consists of one 3x3 convolutional filter, one batch normalization layer[13] and one ReLu activation layer.

Each branch consists of several repeated stages [2], this iterative model can refine the intermediate output and give us more precise heatmaps for both corner and edge. Figure 11 illustrates the details of two branches where $Cn$ ($En$) is $n$th output (i.e. intermediate heatmap) of Branch-Corner (Branch-Edge). Figure 20 illustrates how intermediate heatmaps are refined. $LCn$ ($LEn$) is intermediate Corner loss (Edge loss) of $n$th stage.

$$LCn = \frac{1}{N}\sum_{i=1}^{N}(Cn_i - C_i)^2 \tag{1}$$

Where N means there are N pixels in the heatmap (i.e. 64*64 in this toy dataset case), $Cn_i$ is the $i$ pixel of $n$th predicted heatmap and $C_i$ is the $i$th pixel of ground truth heatmap[2]. $LEn$ shares the same formula with $LCn$.

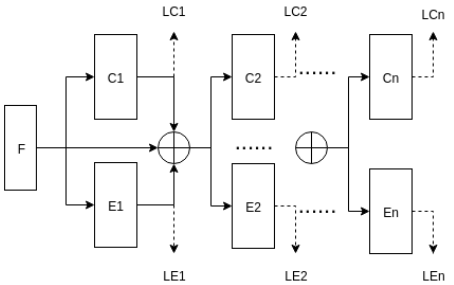



Figure 11: Details of two branches. F stands for backbone network (i.e. feature extractor). Cn (En) stands for $n$th block in corner-branch (edge-branch), their outputs (i.e. intermediate feature maps) are stacked with the output of F. LCn (LEn) stands for $n$th intermediate loss.

In the toy network, we use three stages (i.e.

n = 3) where each stage consists one 3x3 basic block (3x3Conv+BN+ReLu) and 1x1 basic block (1x1Conv+BN+ReLu).

The collective training loss is

$$L = \sum_{k=1}^{n} (a * LCk + b * LEk) \tag{2}$$

Where $a$ and $b$ are the weights of LC and LE respectively.

The corner detection branch is expected to produce four heatmaps to detect four types of corner (i.e. top-left(**tl**), top-right(**tr**), bottom-right(**br**), bottom-left(**bl**)) separately. And the edge detection branch is expected to output four heatmaps as well which are tl-tr, tr-br, br-bl and bl-tl. Figure 12 illustrates an example.

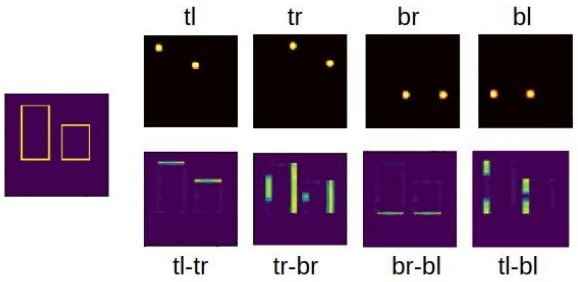



Figure 12: outputs of two branches. The upper four heatmaps are corner-heatmap, tl means the top-left corners of two polygons in the input image. The lower four heatmaps are edge-heatmap, tl-tr means the edge which links top-left corner and top-right corner.

**iii**) The post processing part is responsible for combining detected corners and edges. This part is also called **grouping method**. In toy model we follows the method used in[2].

### 3.3. Experiments

To verify the correctness of our model, we conduct some experiments on toy dataset. We randomly generate 500 images as the train dataset and 200 images as test dataset.

We evaluate corner detection by means of **recall** and **precision**. Recall is the fraction of true corners that are detected and precision is the fraction of corners that are indeed true positive[11]. Let **G** denote the set of ground truth corners and **D** denote the set of detected corners.

$$Precision = \frac{G \cap D}{D}, Recall = \frac{G \cap D}{G} \tag{3}$$

Given a detected corner **p**, if there is a ground truth corner **g** within $t$ pixel(s) (i.e. $|p(x) - g(x)| \leq t$ and $|p(y) - g(y)| \leq t$ ), then we set **p** as a true positive (i.e. it is in the set G ∩ D ), otherwise it is set as a false detection. To take both **precision** and **recall** into consideration when evaluating the overall performance of corner detection, F1 score[28] is used in this project.

$$F1 = (\frac{R^{-1} + P^{-1}}{2})^{-1} = 2 * (\frac{P * R}{P + R}) \tag{4}$$

Where $R^{-1}$ ($P^{-1}$) stands for the inverse of Recall (Precision).

For this simple toy dataset, we find that both recall and precision reach 1.0[4] when we set $t$ = 1 (i.e. the threshold is only 1 pixel). By checking the whole test dataset (200 images), we find that the detected corners are grouped correctly as well.

### 3.4. Discussion

This perfect experimental results on toy dataset indicate that our model is suitable for this task and the correctness has been validated successfully. **The model is able to classify the four types of corner and highlight the detected corners in four heatmaps correctly**. Thus the grouping method used in [2] can be used on the toy dataset directly. Although results on toy dataset can not give us more convincing insights because the toy dataset is too easy. Building and testing a model on toy dataset is necessary for training a model on the new dataset which the model has never worked on before.

## 4. Upgraded model on final dataset

After verifying the correctness of the model on toy dataset, we have to upgrade the model before applying it to Unreal dataset.

### 4.1. Unreal dataset

Unreal dataset is generated automatically by Unreal Engine by [4]. The biggest advantage of using Game Engine to generate dataset is that the labelled dataset is unlimited and free. However, automatic generating will cause a lot of noisy samples. For instance, some invisible objects are still labelled by the program.

The Unreal dataset can be categorized into three courses (Daylight, IROS, Basement) as shown in figure 13. Background , illumination and viewpoint vary per course . There are also a lot of noisy images in the dataset. Figure 14 illustrates some cases of noise data.

### 4.2. Upgraded model

As Unreal dataset is much more difficult than toy dataset, upgrading toy model is necessary for solving more complex image. We improve the model capacity by means of more powerful feature extractors (i.e. backbone) and more

[4]results can be re-produced by using my codes

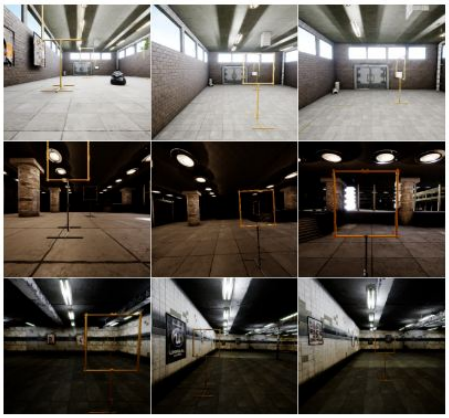


Figure 13: Some examples from three different courses

Figure 14: Some examples of noisy images

stacked stages in both corner detection branch and edge detection branch.

#### 4.2.1 More powerful feature extractors

There are various feature extractors [27][8][12][9] that are widely used in different computer vision tasks including image classification, object detection, semantic segmentation as well as keypoint detections. In this work we will investigate two popular feature extractors, ResNet and SqueezeNet, with necessary adjustment to fit this project.

Figure 16 and 17 show the architecture of two feature extractors in details. SqueezeNet [12] is optimized for embedded platform and this architecture has relatively less parameters and computing costs. ResNet[8] is a popular backbone network, many complex networks use ResNet or its variants as their feature extractors, the capacity of ResNet depends on how many convolutional layers used. In this project, we use a relatively small model for considering that the model should be able to ported into an embedded platform.

#### 4.2.2 More stacked stages

An important idea behind the model is that it uses multi-stage branches to refine the heatmaps. Adding more stages could give better output but it could also result more parameters. In this project, we investigate different number of stacked stages and how the number could influence the results.

#### 4.2.3 Failure case

We train our upgraded model on Daylight_train dataset and test on Daylight_test dataset. However, after extensive experiments with different training setting and hyper-parameters[5], **we conclude that the model is not able to classify the corners into four classes**[6] . Although we have already verified the model on toy dataset and it is able to give us desired outputs as shown in Figure 12. The first reason why the model fails could be that for Unreal dataset, it is too difficult for neural network to recognize the difference between the four type of courses. Although there is an obvious feature – rotation for different types of corner and it can be recognized easily by human eyes, the model still fails to learn this feature. Too much noise (6.3) in Unreal dataset could be another reason because mislabelled gates could misguide the network during training phase. Figure 18 illustrates an example.

**Then we modify the two branches of our model and use single heatmap to detect all corners and single heatmap to detect all edges**. So, unlike the toy mode, in the upgraded model, the final outputs are two heatmaps in total. The corner-heatmap highlights all detected corners in a single heatmap and edge-heatmap highlights all detected edges in a single heatmap as well.

#### 4.2.4 Exp 1: Compare two feature extractors

Bar charts in Figure 15 show the test accuracy of corner detection part on three different datasets. The only different component of the first two models (ResNet-5 and SqueezeNet-5) is the backbone. Both of them have five stages in two branches and are trained on Daylight-train dataset. Although ResNet is more 'powerful' than SqueezeNet and has more parameters, it does not give a better test accuracy on corner detection.

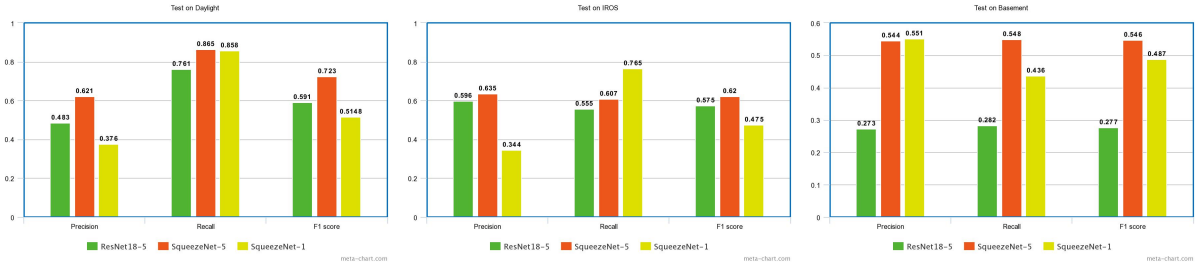
One possible reason is that SqueezeNet has better generalization ability on this network architecture when working as the feature extractor because both SqueezeNet[12] and ResNet[8] are originally designed for image classification and object detection instead of this keypoint detection problem.

#### 4.2.5 Exp 2: Iterative Model

In this experiment, we investigate the iterative model of two branches. The last two models (SqueezeNet-5 and SqueezeNet-1) in Figure 15 have same backbone (SqueezeNet) but different amount of stages. SqueezeNet-5 has five stages and SqueezeNet-1 has only one stage for

[5] we spent more than 6 weeks on tuning model, the log can be found in my Github repository

[6] failure examples can be found in background part




(a) Test results on Daylight-test dataset (b) Test results on IROS dataset (c) Test results on Basement dataset

Figure 15: Test results on different datasets

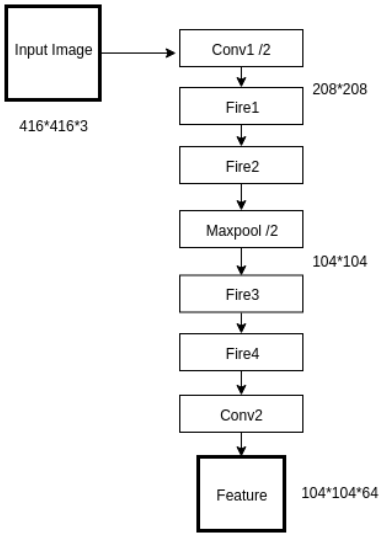



Figure 16: Model Feature extractor - SqueezeNet. Figure stands for the basic block of SqueezeNet as shown in Figure 7

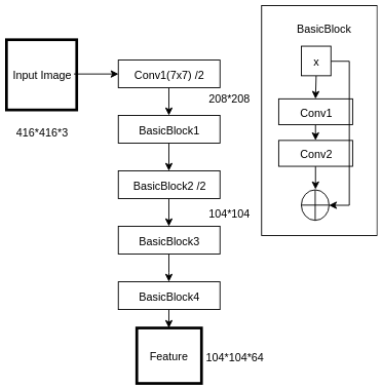



Figure 17: Model Feature extractor - ResNet

each branch. We can see that for three different test datasets, SqueezeNet-5 outperforms SqueezeNet-1 in F1 score.

Figure 19 and 20 visualize the outputs of each stage.

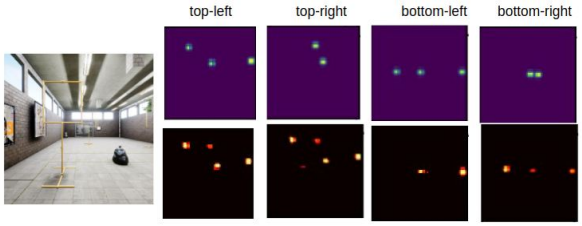



Figure 18: Example of failure. The upper four images are ground-true heatmaps and the second-row four images are corresponding predicted heatmaps. We can see that the network gets confused and is not able to make correct predictions

SqueezeNet-1 gives high confidence values on three circle lights on the ceiling in both corner detection part and edge detection part. SqueezeNet-5 is able to recognize the difference between lights and gates and give high confidence values on correct corners and edges.

Figure 20 also shows that the later stage has better confidence maps over previous one.

Figure 19: Model with single stage. The heatmaps are not precise enough

#### 4.2.6 Exp 3: Cat in Gate test

Figure 21 demonstrates an advantage of using keypoint detection to localize gates. This model is able to recognize the gate even we add some never-seen noise (i.e. cat), but

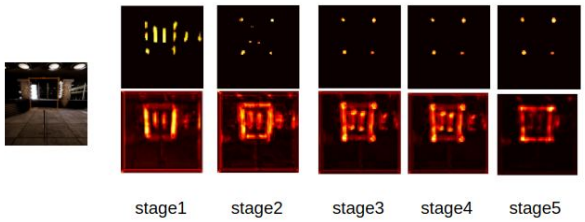



Figure 20: Model with five stages. We can see that the intermediate heatmaps are refined especially the edge heatmap.

YOLO [23] or other similar object detection algorithms will get confused in this case[4].

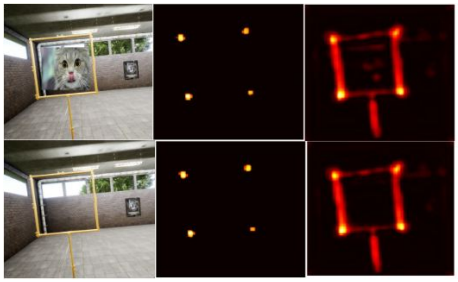


Figure 21: Cat in Gate. We can see that even we add a cat picture inside the wireframe gate. Our model is still able to detect the corners and edges correctly and does not get confused by the Cat.

#### 4.2.7 Exp 4: Compare performance to traditional computer vision algorithms

We compare our model to traditional algorithms – Harris corner detector and Canny edge detector. Figure 22 and 23 show the output of traditional algorithms and our model respectively. We can see that our model shows stronger performance on detection. The corners and edges of background object are filtered out by our model.

#### 4.2.8 Exp 5: Compare FLOPS to standard YOLOv3

Although currently our model could not group the detected corners and edges because of lacking effective grouping algorithm. Figure 24 shows FLOPS of the three models we have tested and YOLOv3-416. We can see that our models require significantly less floating-point calculations than original YOLOv3-416[24], although here we do not include post processing cost. If YOLOv3 is a baseline of real-time object detector for embedded platform, then there are still a lot of computing resources can be used for developing an effective grouping algorithm for our model.

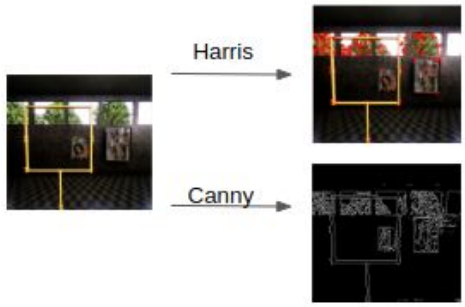



Figure 22: Example output of Harris and Canny. Corners and edges from background object are also detected which should have been filtered out.

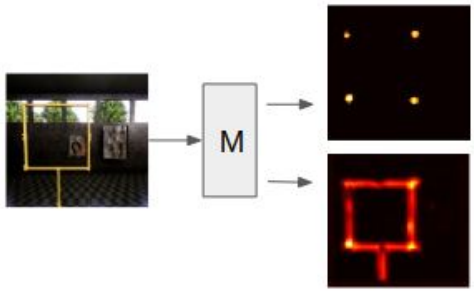



Figure 23: Example result of our model (M). Comparing with traditional computer vision algorithms, our model shows stronger and clear detection performance. Note that this input image is not included in the train dataset

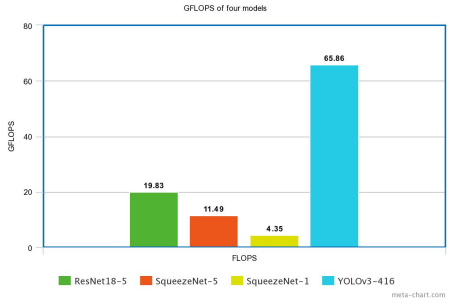



Figure 24: FLOPS of three investigated models and YOLOv3-416[21]

## 5. Discussions

We first verify the correctness of the method on toy dataset. The perfect result on the toy dataset motivate us to upgrade the model and conduct experiments on a much more difficult dataset – Unreal dataset which is generated by Unreal game engine. However, the model on Unreal dataset fails to give us desired results (i.e. grouping the detected corners per gate). It is not able to classify the corners into

four classes (i.e. top-left, top-right, bottom-right and bottom left).

The experiments on Unreal dataset also show some nice properties of our model. Experiment 4.2.6 demonstrates that our model can still work well even we add an object inside the gate which is not included in the training set. By comparison, YOLO-like model will get confused[4]. Experiment 4.2.7 compares our model with two traditional computer vision algorithms (i.e. Harris and Canny). Our model is much stronger because it filters out corners and edges of background or irrelevant objects. Furthermore, our model can do corner detection and edge detection simultaneously. Experiment 4.2.6 evaluates the computing workload of our model. By comparing with YOLOv3, our model needs significant less computing resources to reach real-time property.

# 6. Future work

This thesis work demonstrates the possibility of using keypoint detection to localize wireframe. There are still a lot of things should be investigated. And the model has to be improved further before implementing it on real drone. The major future work is to develop an effective grouping algorithm. There are recommendations for the possible future work.

## 6.1. Expanding the network

The main idea of this approach is that we can expand our current network by adding a sub-network following edge-heatmap and corner-heatmap. We can group those two heatmaps together and feed into the sub-network.

Figure 25 illustrates the enhanced architecture. The new network will be trained in a completely different way, the ground-truth will become the coordinates of each bounding box (gate) instead of heatmaps and the loss function will be based on the difference between predicted bounding-box and ground-truth bounding-box. If the two branches can generate corner heatmaps and edge heatmaps correctly under the completely new training setting, then the sub-network is expected to be able to extract the bounding box from the feature maps(i.e. edge heatmap and corner heatmap). Because there is no mathematical explanation to support this idea just like most of deep learning algorithms, so we are not sure if this idea will work or not before really implementing the idea.

## 6.2. Combining Junctions and Lines for Wireframe

There are some algorithms and approaches from Line segment detection research topic could be used in this task. [11] proposes a learning-based approach to detecting wireframe for images of cluttered man-made objects such as tables and doors. In their work, they trained two separate network for junction detection and line detection respectively

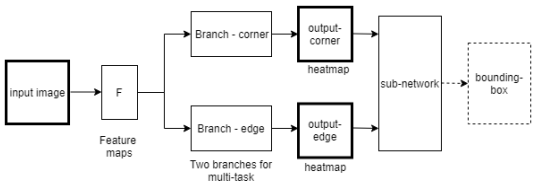



Figure 25: Add sub-network. The sub-network is added directly behind the two branches. It takes two heatmaps (i.e. corner heatmap and edge heatmap) as input and expected to produce the coordinate of bounding-box keypoints (i.e. top-left corner and bottom-right corner)

by producing two heatmaps which is similar to ours. To combine detected junctions and lines together, he proposed an algorithm which could give a decent results. However, we find that this algorithm is too complicated and the time-complexity could be very high. They also state in their paper that there should be more advanced ways to merge detected junction and line heatmaps.

## 6.3. Clean Unreal dataset

The Unreal dataset we used has a lot of noise so that the quality of this dataset is worse than standard COCO keypoint detection dataset. Because the images and labels are automatically generated by Unreal Engine, there are three types of noise.

- Invisible gates. Invisible but labeled gates occur in many images. Figure 26 illustrates this noise which could be caused by occlusion or low illumination intensity.

- No gate. There are also a lot of images which contain no gate at all but included in the dataset. Figure 27 illustrates this noise.

- Incomplete gates. Incomplete gates are also very often in the dataset as shown in Figure 28.

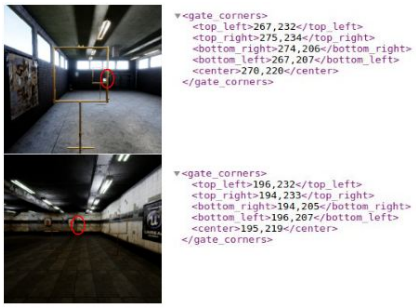



Figure 26: Invisible but labeled gates


Figure 27: No gate


Figure 28: Incomplete gates

We think the one possible reason why our upgraded model fails on classifying four types of corner is that there are so many noisy data and mislabelled data which could misguide the neural network during training phase[25]. However, cleaning the whole dataset manually is time consuming

### 6.4. Keypoint-based object detection

Keypoint-based methods are a relatively new direction in object detection [17] of one-stage framework. During my thesis work, there are some related works have been published [16][17][30]. The very first one is CornerNet[16] published in ECCV2018 which achieves state-of-the-art accuracy among one-stage detectors. The main idea behind CornerNet is to detect and group top-left and bottom-right corners of bounding box. However, CornerNet achieves the accuracy at high processing cost. It achieves an average precision of 42.2% on COCO at inference cost of 1.147s per image [17]. By comparison, YOLOv3[24] achieves 33.0 % with only 39ms on the same testing platform (Nvidia 1080Ti GPU and Intel Core i7-7700k CPU). To improve inference efficiency of CornerNet, CornerNet-Lite[17] is proposed and it achieves 34.4% at 34m which is faster and more accurate than YOLOv3.

Another keypoint-based detection algorithm is CenterNet[30] published on 25 April 2019. The name indicates this algorithm models object as single point – the center point of its bounding box. CenterNet detects objects by finding object centers and then regressing to their size[30]. The authors stage in the paper that CenterNet can be applied to various visual tasks including pose estimation and 3D detection. However, this novel architecture is not suitable for wireframe objects because the attentions of wireframe object are edges and corners (vertices) instead of the center point.

## References


[1] J Canny. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(6):679–698, June 1986.

[2] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. *CoRR*, abs/1611.08050, 2016.

[3] Richard O. Duda and Peter E. Hart. Use of the hough transformation to detect lines and curves in pictures. *Commun. ACM*, 15(1):11–15, Jan. 1972.

[4] P. Duernay. Detecting Empty Wireframe Objects on Micro-Air Vehicles. `http://resolver.tudelft.nl/uuid:82cb0f68-061e-4346-b536-a35a61621e51`, 2018.

[5] Ross B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015.

[6] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.

[7] Chris Harris and Mike Stephens. A combined corner and edge detector. In *In Proc. of Fourth Alvey Vision Conference*, pages 147–151, 1988.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[9] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

[10] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, and Kevin Murphy. Speed/accuracy trade-offs for modern convolutional object detectors. *CoRR*, abs/1611.10012, 2016.

[11] Kun Huang, Yifan Wang, Zihan Zhou, Tianjiao Ding, Shenghua Gao, and Yi Ma. Learning to parse wireframes in images of man-made environments. In *CVPR*, June 2018.

[12] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR*, abs/1602.07360, 2016.

[13] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 448–456. JMLR.org, 2015.

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[15] Hei Law and Jia Deng. Cornernet: Detecting objects as paired keypoints. *CoRR*, abs/1808.01244, 2018.

[16] Hei Law and Jia Deng. Cornernet: Detecting objects as paired keypoints. *CoRR*, abs/1808.01244, 2018.

[17] Hei Law, Yun Teng, Olga Russakovsky, and Jia Deng. Cornernet-lite: Efficient keypoint based object detection. *CoRR*, abs/1904.08900, 2019.

[18] Y. Liu, Y. Xu, and S. Li. 2-d human pose estimation from images based on deep learning: A review. In *2018 2nd IEEE Advanced Information Management,Communicates,Electronic and Automation Control Conference (IMCEC)*, pages 462–465, May 2018.

[19] Alejandro Newell and Jia Deng. Associative embedding: End-to-end learning for joint detection and grouping. *CoRR*, abs/1611.05424, 2016.

[20] Bharath Raj. An Overview of Human Pose Estimation with Deep Learning. `https://medium.com/beyondminds/an-overview-of-human-pose-estimation-with-deep-learn` 2019.

[21] Joseph Redmon. Darknet: Open source neural networks in c. `http://pjreddie.com/darknet/`, 2013–2016.

[22] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[23] Joseph Redmon and Ali Farhadi. YOLO9000: better, faster, stronger. *CoRR*, abs/1612.08242, 2016.

[24] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv*, 2018.

[25] Scott Reed, Honglak Lee, Dragomir Anguelov, Christian Szegedy, Dumitru Erhan, and Andrew Rabinovich. Training deep neural networks on noisy labels with bootstrapping. 12 2014.

[26] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[27] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[28] Wikipedia. F1 Score. `https://en.wikipedia.org/wiki/F1_score`, 2018.

[29] Zhong-Qiu Zhao, Peng Zheng, Shou-tao Xu, and Xindong Wu. Object detection with deep learning: A review. *CoRR*, abs/1807.05511, 2018.

[30] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. Objects as points. *CoRR*, abs/1904.07850, 2019.

# 2

# INTRODUCTION

## 2.1. PROJECT BACKGROUND

In fact, the project – **deep learning for gate recognition** has already been done by a former master student[1] who was a member in drone lab of Aerospace Engineering . He used Unreal Engine [1] to create some images as project dataset. Figure 2.1 and 2.2 show some examples of our project dataset. The advantage of using a game engine to make data is that objects(i.e. gate in our case) can be labeled automatically by the software and it will save a lot of manually labeling time. However, there are also a lot of noisy images generated simultaneously.

In his thesis work, he investigated YOLO[2] and made some modifications on it. He finally ported his model on drone and tested it in real world by modifying YOLO official C implementation [3] . If you want to know more about the real project background of this topic – Deep Learning for Gate Recognition , I would suggest you read corresponding chapters in his thesis[1].

This project will find a non-generic-object-detection algorithm for localizing gates. The initial motivation of this project is to create a brand-new neural network architecture by inserting Hough Transfrom(HT)[4] in regular neural network. The new model is expected to be more data-efficient because HT could work as a kind of pre-knowledge in the whole neural network and it could somehow guide the neural network to learn the feature of line by using less training samples[2].

However, after spending around 10 weeks on verifying the theory, the results indicated that the idea could be wrong. After several weeks literature survey, I linked human-pose-estimation[5] solutions to this task because :

- Gate consists of vertices and edges. Human-pose-estimation problem also consists of two parts: keypoints(joint) detection and grouping(link detected joints). The published algorithms for human-pose-estimation can be used in this problem by some modifications.
- Gate is a kind of empty wireframe, so the important parts are edges and corners of gate, the objects inside the wireframe are not important. Figure 2.4 illustrates the situation, YOLO-like algorithms are likely to fail in this cat in gate example If the object(e.g. cat and sign) inside gate is not included in training set[1]. By contrast, model built with keypoint-detection should work correctly on the two images in Figure 2.4 even the model has never seen the left one during training phase.

In [1], an interesting experiment are conducted and the results show that the YOLO detector gets confused when the object inside gate is not included in the training set. Figure 2.3 shows the results. We can see that the test accuracy decreases dramatically when testing on a different dataset where object in gate is different.

[1] https://www.unrealengine.com/

[2] Because Yancong Lin is still working on developing his HT theory and this work has not been published. So I can not explain more details about this theory for keeping confidentiality. If you are interested in his theory, I suggest you contact him(https://www.tudelft.nl/ewi/over-de-faculteit/afdelingen/intelligent-systems/pattern-recognition-bioinformatics/computer-vision-lab/people/yancong-lin/)

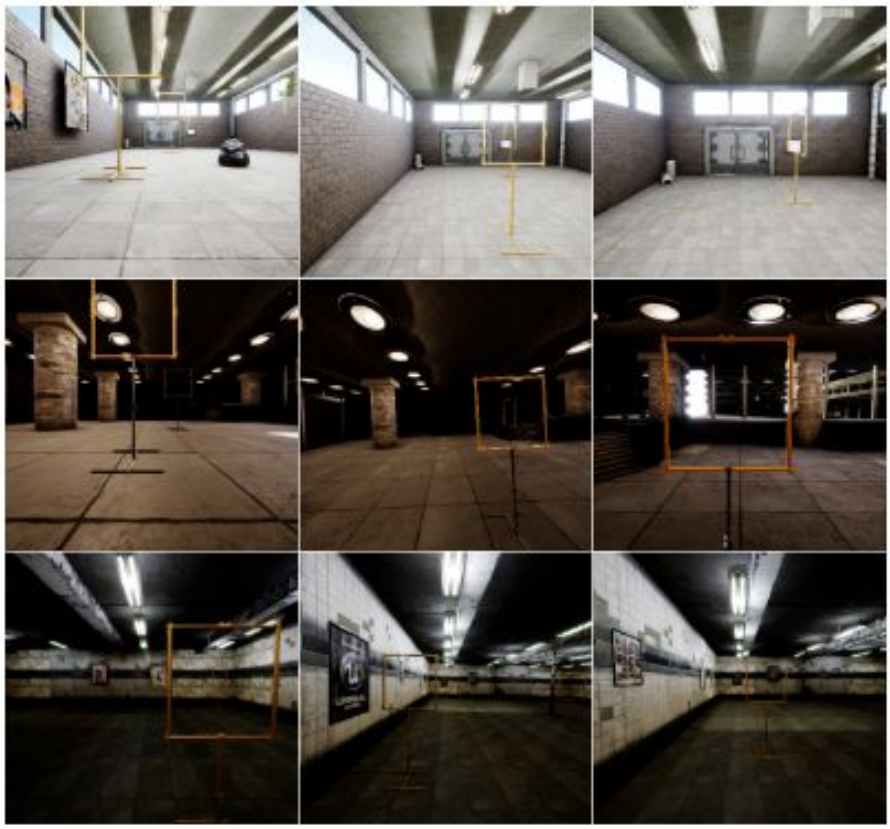


Figure 2.1: Images generated by Unreal Engine

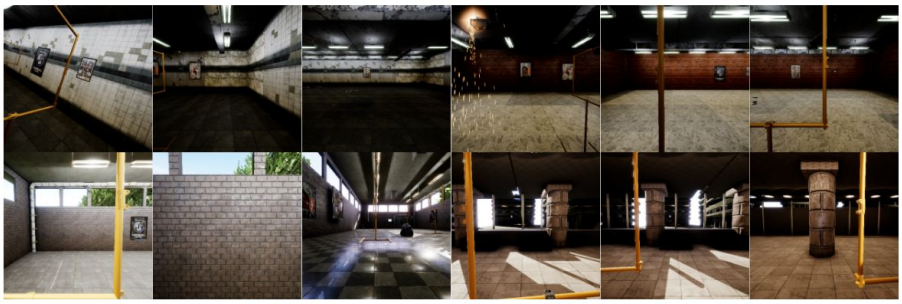


Figure 2.2: Examples of 'difficult' noisy images

## 2.2. RESEARCH OBJECTIVES

The research objectives of this thesis can be divided into two phases. The first phase is about HT-based CNN.

- Verify if HT-based CNN can be used to this gate localization problem.
- Compare the brand-new model with YOLO[2]

After realizing the HT-based model is not suitable for this project. The Research objectives are changed to:

- Investigate published human-pose-estimation algorithms and find a suitable one.

| Trained/Tested | Gate [$ap_{60}$] | Sign [$ap_{60}$] | Cats [$ap_{60}$] |
|---|---|---|---|
| Gate | $0.55 \pm 0.04$ | $0.01 \pm 0.00$ | $0.37 \pm 0.06$ |
| Sign | $0.02 \pm 0.01$ | $0.74 \pm 0.06$ | $0.05 \pm 0.04$ |
| Cats | $0.06 \pm 0.03$ | $0.03 \pm 0.00$ | $0.78 \pm 0.01$ |

Figure 2.3: Results from

Figure 2.4: Cat in Gate

- Adapt one suitable published algorithm to this gate localization problem .
- Validate my hypothesis that the model can work well under the situation in Figure 2.4.

## REFERENCES

[1] P. Duernay, *Detecting Empty Wireframe Objects on Micro-Air Vehicles*, http://resolver.tudelft.nl/uuid:82cb0f68-061e-4346-b536-a35a61621e51 (2018).

[2] J. Redmon and A. Farhadi, *YOLO9000: better, faster, stronger*, CoRR **abs/1612.08242** (2016), arXiv:1612.08242 .

[3] J. Redmon, *Darknet: Open source neural networks in c*, http://pjreddie.com/darknet/ (2013–2016).

[4] R. O. Duda and P. E. Hart, *Use of the hough transformation to detect lines and curves in pictures*, Commun. ACM **15**, 11 (1972).

[5] Y. Liu, Y. Xu, and S. Li, *2-d human pose estimation from images based on deep learning: A review*, in *2018 2nd IEEE Advanced Information Management,Communicates,Electronic and Automation Control Conference (IMCEC)* (2018) pp. 462–465.

# 3

# GENERAL BACKGROUND ON DEEP LEARNING

In this chapter some general background knowledge about deep learning will be given. If you have basic knowledge about deep learning and neural network, you can skip this chapter.

## 3.1. DEEP LEARNING

Deep learning is part of a broader family of machine learning methods based on artificial neural networks. Learning can be classified into supervised, semi-supervised or unsupervised method. In computer vision field, supervised learning is the most used one. Figure 3.1 [1] illustrates the relation between Artificial Intelligence(AI), Machine Learning(ML) and Deep Learning(DL).

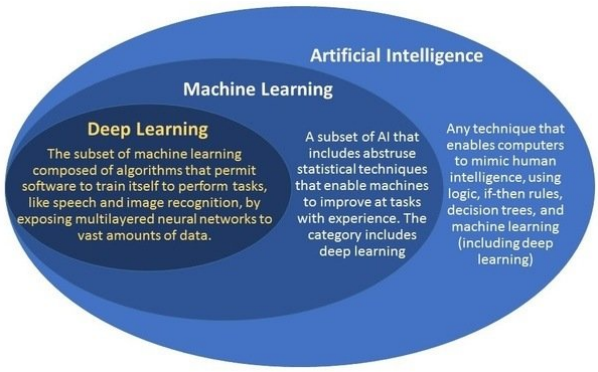



Figure 3.1: How deep learning is a subset of machine learning and how machine learning is a subset of artificial intelligence (AI).

## 3.2. ARTIFICIAL NEURAL NETWORKS (ANNS)

Neural networks are modeled as collections of neurons that are connected in an acyclic graph. Figure 3.2 illustrates the the mathematical model of neuron which is the unit of Artificial neural networks (ANNs). The weights $w_i$ are trainable and control the strength of influence and its direction of one neuron on an activation function. Figure 3.3 shows a regular three-layer Neural Network. Each layer consists several neurons in Figure 3.2.

Mathematically, each neuron applies an affine transformation of the input $X = [x_1, x_2, ..., x_n]$.

$$u = \sum_{i=1}^{n} w_i * x_i + b \tag{3.1}$$

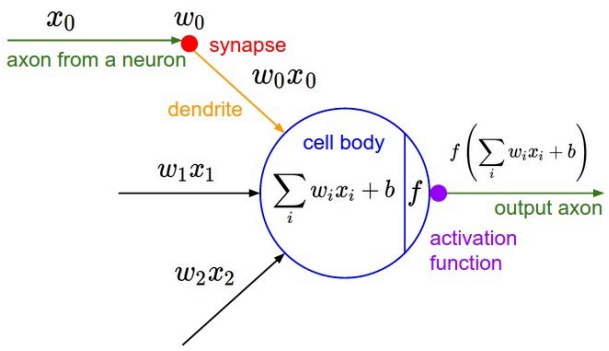



Figure 3.2: Mathematical model of neuron

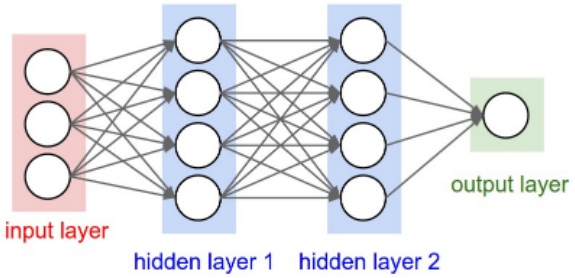



Figure 3.3: A regular 3-layer Neural Network

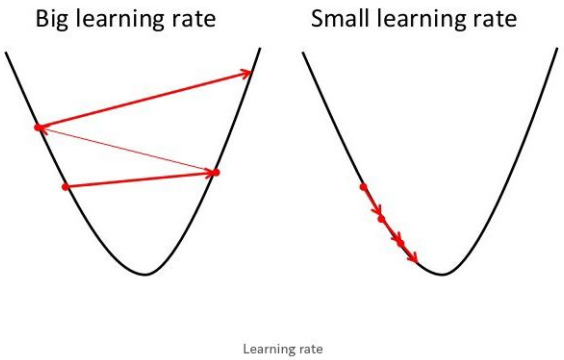



Figure 3.4: How learning rate influence the training process

Where $w_i$ is $i^{th}$ trainable weight for $x_i$ and b is the bias term. A non-linear activation function $f$ will be applied to u to give the non-linear transform ability of the neural networks. A collection of neurons can be used to approximate continuous functions which is a theoretical interpretation of deep neural networks.

## 3.3. TRAINING NEURAL NETWORK

Neural networks are usually initialized with small random weights, and these parameters are updated during training by minimizing a loss function $L$. Loss function is the evaluation function to tell neural networks the error between ground truth and its predictions. The learning process is achieved by gradient descent parameter update method with gradients calculated by backpropagation[2].

A bunch of techniques used to accomplish this training process has been developed such as Stochastic Gradient Descent(SGD)[3], Adagrad[4], Adam[5] and etc.

Tuning hyperparameters is one of the most important skills related to deep learning. Hyperparameters are the variables which determines the network structure(E.g. number of hidden units and number of channels each layer ) and the variables which determine how the network is trained(e.g.: learning rate, batch size, number of epoch). Hyperparameters are set before training the neural network. **Learning Rate** is one of the most important hyperparameters. Figure 3.4 illustrates how learning rate influences the training phase. Low learning rate slows down the learning process but converges smoothly. Larger learning rate speeds up the learning but may not converge. Other parameters related to the network size and structure are also very important. Tuning hyperparameters manually is time-consuming and boring and will usually result in sub-optimal results. Some techniques have been proposed to automate the process of tuning hyperparameters such as AutoML and NAS(Network architecture search).



## 3.4. MODEL SIZE AND FLOPS

In practical applications, model size and it FLOPS are two important evaluation metrics. Model size is used to evaluate how large the model is and how many memory resources it requires. Flops(floating point operations) is used to evaluate neural network computation workload. Because of weight sharing, number of weights on convolutional layers does not depend on input size. It depends on current layer depth, kernel size, and depth of previous layer. Increasing image input size would not cause a much larger model size, but it will bring more floating point operations.

## REFERENCES

[1] Wikipedia, *Deep Learning Wikipedia,* https://en.wikipedia.org/wiki/Deep_learning (2019).

[2] M. Nielsen, *How the backpropagation algorithm works,* http://neuralnetworksanddeeplearning.com/chap2.html (2018).

[3] L. Bottou, *Large-scale machine learning with stochastic gradient descent,* in *Proceedings of COMPSTAT'2010*, edited by Y. Lechevallier and G. Saporta (Physica-Verlag HD, Heidelberg, 2010) pp. 177–186.

[4] J. Duchi, E. Hazan, and Y. Singer, *Adaptive subgradient methods for online learning and stochastic optimization,* J. Mach. Learn. Res. **12**, 2121 (2011).

[5] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization,* (2014), cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

# 4

# DEEP LEARNING FOR OBJECT DETECTION

Object detection is the identification of objects in an image along with the classification and localization. A software/hardware system which is designed to solve object detection task is called object detector. The first object detector was introduced in 2011 [1] by using traditional computer vision algorithms like handcrafted features and shallow trainable architectures. Before deep learning, the progress is slow in this field and small gains are obtained by building ensemble systems with handcrafted feature extraction and employing minor variants of successful methods. Deep neural networks has significantly accelerated this field, and modern successful object detectors are based on deep neural network system.

Generic Object detection algorithms can be categorized into two types i) **Region proposal based framework** (R-CNN[2], Fast R-CNN[3] and Faster R-CNN [4]) . ii) **Regression/Classification based framework** (YOLO[5] , SSD[6]). Region proposal based framework and Regression/Classification based framework are also usually called **two-stage** and **one-stage** object detectors respectively. In this chapter we introduce two popular algorithms – Faster R-CNN[4] and YOLO[5].

## 4.1. FASTER R-CNN

Faster R-CNN is the third version of R-CNN family. It introduces a Region Proposal Network to solve the inefficiency of region proposal computation in its previous versions.

Figure 4.1 shows the model pipeline of Faster R-CNN. Region Proposal Network(RPN) is a sub-network which is responsible for creating most-likely region of interest(RoI). The proposed regions are fed into the next sub-network – classifier. RPN and classifier shares same feature maps generated by the backbone(conv layers). Sharing feature maps between classifier part and RoI projection part is a significant improvement over Fast R-CNN[3].

Figure 4.2 shows the architecture of Fast R-CNN. RoI projection and deep ConvNet(feature extractor) are conducted separately. Fast R-CNN uses selective search to generate RoIs, which is a slow and time-consuming process affecting the performance of the network.

## 4.2. YOLO

R-CNN based object detection algorithms use regions to localize the object within the image. For example, Faster R-CNN does not look at the complete image. Instead, it only check parts of the image with high possibilities containing an object.

You Only Look Once(YOLO) is an object detection algorithm much different from region proposal based algorithm. In YOLO, a single network predicts the bounding boxes and class possibilities. So it is categorized as regression/classification based algorithm or one-stage object detector.

One-stage frameworks like YOLO are based on global regression/classification which directly maps from image pixels to bounding box coordinates and class probabilities. It can reduce time expense on finding possible candidate regions(region proposal in two-stage algorithms).

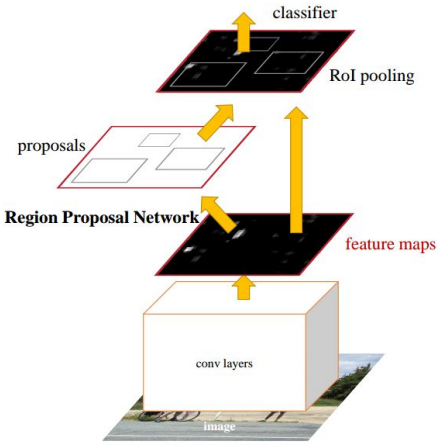



Figure 4.1: Faster R-CNN architecture[4]

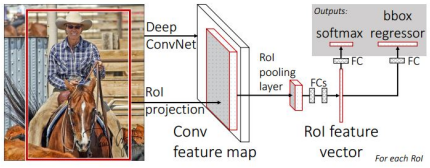



Figure 4.2: Fast R-CNN architecture[3]

Figure 4.3 illustrates main idea behind YOLO. The input image is split into S*S grid and each cell is responsible for predicting the object centered in that grid cell[7].YOLOv2[8] and YOLOv3[9] introduce further improvements for YOLO. YOLOv2 adopts several general strategies for improving neural network like Batch Normalization, anchor boxes and multi-scale training. YOLOv3 makes use of residual network to build more powerful feature extractor.

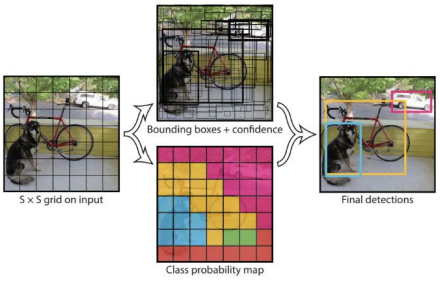



Figure 4.3: Main YOLO idea[5]

Usually, one-stage object detectors are faster and more efficient than two-stage object detectors. YOLO is a popular model in mobile application where memory and computing resources are strictly limited. On the other side, region proposal based algorithms are more likely to achieve higher accuracy over one-stage object detectors.

# REFERENCES

[1] P. Viola and M. Jones, *Rapid object detection using a boosted cascade of simple features,* in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, Vol. 1 (2001) pp. I–I.

[2] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik, *Rich feature hierarchies for accurate object detection and semantic segmentation,* CoRR **abs/1311.2524** (2013), arXiv:1311.2524 .

[3] R. Girshick, *Fast r-cnn,* in *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15 (IEEE Computer Society, Washington, DC, USA, 2015) pp. 1440–1448.

[4] S. Ren, K. He, R. B. Girshick, and J. Sun, *Faster R-CNN: towards real-time object detection with region proposal networks,* CoRR **abs/1506.01497** (2015), arXiv:1506.01497 .

[5] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, *You only look once: Unified, real-time object detection,* CoRR **abs/1506.02640** (2015), arXiv:1506.02640 .

[6] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg, *SSD: single shot multibox detector,* CoRR **abs/1512.02325** (2015), arXiv:1512.02325 .

[7] Z. Zhao, P. Zheng, S. Xu, and X. Wu, *Object detection with deep learning: A review,* CoRR **abs/1807.05511** (2018), arXiv:1807.05511 .

[8] J. Redmon and A. Farhadi, *YOLO9000: better, faster, stronger,* CoRR **abs/1612.08242** (2016), arXiv:1612.08242 .

[9] J. Redmon and A. Farhadi, *Yolov3: An incremental improvement,* arXiv (2018).

# 5

# DEEP LEARNING FOR MULTIPLE HUMAN POSE ESTIMATION

Human pose estimation is a problem of localizing body parts of individuals. The input images contain unknown number of people with unknown positions and scales. The algorithms are expected to infer the pose of people in the input images and give us the coordinates of each detected individual and associate detected joints correctly. Related algorithms can be categorized into two types i) top-down approaches and ii) bottom-up approaches. The following sections will give a brief introductions to both of two types.

## 5.1. TOP-DOWN APPROACHES

Top-down approaches employ a person detector and then perform single-person pose estimation. A person detector is responsible for finding individuals in the input image. A single-person pose estimation model then performs pose estimation on each of detected person. This pipeline is similar to two-stage or region-proposal-based object detector. However, this kind of simple top-down approach relies heavily on the accuracy and efficiency of the person detector. Figure 5.1 shows an example of how top-down approach works. Typically, top-down approach is easier to implement than bottom-up approach.

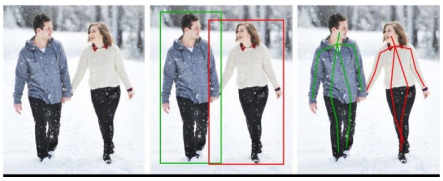


Figure 5.1: Typical Top-Down approach[1]

## 5.2. BOTTOM-UP APPROACHES

Unlike top-down approaches, bottom-up approaches globally detect keypoints of human body and then group the detected joints per individual. Figure 5.2 shows how bottom-up approaches work. In this thesis project, our model is built on a popular bottom-up approach called OpenPose[2][1]. Figure 5.3 shows the network architecture of OpenPose. It uses VGG19 as the feature extractor to process the raw input image and . The generated feature maps are fed into two parallel branches, the upper one is responsible for joint detection and the lower one is responsible for grouping detected joints per individual. The successive stages in the two-branch sub-network are used to refine the heatmaps and it can solve gradient vanishing problem during training phase[2].

[1] https://github.com/CMU-Perceptual-Computing-Lab/openpose

Figure 5.2: Typical Bottom-up approach[1]

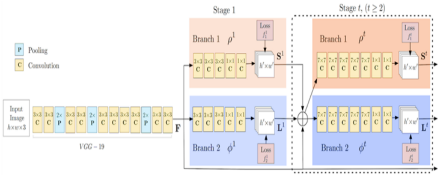



Figure 5.3: Flowchart of the OpenPose architecture[1]

5

## REFERENCES

[1] B. Raj, *An Overview of Human Pose Estimation with Deep Learning*, https://medium.com/beyondminds/an-overview-of-human-pose-estimation-with-deep-learning-d49eb656739b (2019).

[2] Z. Cao, T. Simon, S.-E. Wei, and Y. Sheikh, *Realtime multi-person 2d pose estimation using part affinity fields*, in *CVPR* (2017).

# 6

# FAILURE CASES

Failures are unfortunately one of the most important parts of my thesis work. In this chapter, two of them are presented to give insights.

## 6.1. FAILURE CASES: HOUGH-TRANSFORM-BASED CNNS

The initial motivation of this project is to applied Hough transform layer theory to this specific task - gate recognition. The main idea behind Hough-Transform-based CNNs is that by adding Hough Line Transform in traditional CNNs, the network is expected to perform better on line-detection related task such as this gate recognition task. We tried several possible architectures but all of them failed. In this section, we will introduce the failed architectures and provide some analysis.[1]

### 6.1.1. HYPOTHESIS

Hough Line Transform(HLT)[1][2] is a widely used and popular non-learning-based computer vision algorithm to detect lines. Generally, HLT is associated with Canny edge detector[3][4]. Figure 6.1 shows the pipeline of traditional line detector where $\theta$ is the angle of the line and r is the distance from the origin to the line[1]. If we insert HLT into traditional CNNs, it could guide the network to learn the feature of lines effectively and efficiently with less training samples and smaller network architecture. HLT is a kind of prior knowledge and by using this knowledge properly, the new network architecture is expected to give us a better results theoretically .

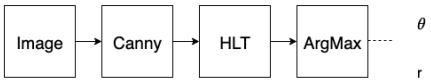



Figure 6.1: Pipeline Canny Edge detector + Hough Line Transform

**Hough Line Transform Layer** Yancong Lin developed a Hough Line Transform layer compatible with Pythorch[2] framework. The main idea is that by implementing the backpropagation of Hough Transform mathematically, the layer can be inserted in the normal CNNs can trained end-to-end by backpropagation. Because the theory is still underdevelopment by Yancong Lin as his P.h.D thesis work, so details are not presented in this report.

### 6.1.2. MODEL AND WHY IT FAILS

With Hough Transform Layer, we tried some models. Unfortunately, those models are not able to work at all, and they can not give us meaningful results to guide us to improve the model neither. Figure 6.2 shows one failed model which at least could give us some insights. .

[1] Our failure cases are based on the experiments at that moment.Currently, Yancong Lin has improved his theory and model but the new version has not been investigated in this work

[2] https://pytorch.org/

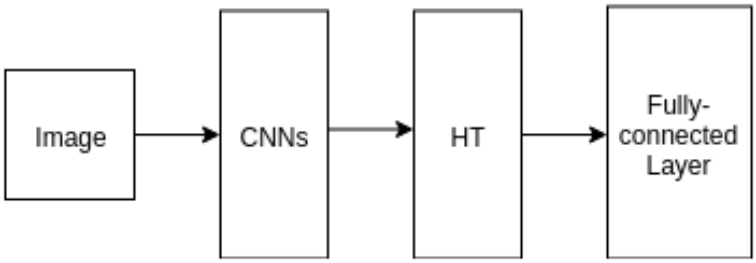



Figure 6.2: CNNs + HT

We expect that HT layer can guide the CNNs to learn the feature of line efficiently. The fully connected-layer can then extract the line information from the feature map generated by HT layer and give us $\theta$ and $r$ just like 6.1. In another word, we expect the CNNs can perform similar task as what Canny detector does in 6.1 and should be better and robust than Canny detector. For instance, canny detector could not classify edges of various object(e.g. edges of window, edges of door and edges of gate) but the CNNs should be able to extract the edges we want (i.e. edges of gate) and remove the edges of other objects. And the loss function of this model is

$$L = ||((\theta', r') - (\theta, r))||_2 \tag{6.1}$$

where ($\theta$', $r$') is prediction and ($\theta$, $r$) is ground truth.

However, after extensive experiments[3], the results show that this model could not even work on the toy dataset. It is very difficult to mathematically explain why it fails which is a common limitation of deep learning algorithms. Here we give some intuitive explanations.

The HT layer is just a Pytorch implementation of Hough Transform. In general, a line can be usually represented as y = m$x$ + b or parametric form as r = xsin$\theta$ + ycos$\theta$. HT transforms a line from (m,b) space to (r,$\theta$) space and generates a HT matrix as feature map. The lines in original space (i.e. (m,b)) can be located by performing a ArgMax operation on HT matrix and the coordinates(i.e.(r,$\theta$)) of the highest values are the (r,$\theta$) of lines. Figure 6.3 demonstrates how HT works. The input image contains two edges(i.e. lines), and after Hough Transform there are two brighter points. Their coordinates are r(i.e. Distance from center) and $\theta$(i.e. Angle) respectively. A comprehensive explanation can be found in [1].

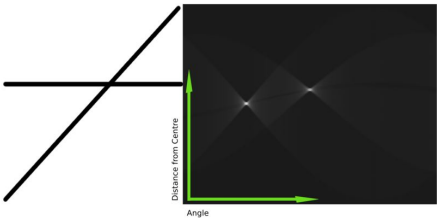



Figure 6.3: example of how HT works[5]

Therefore, whether the first model could work depends on

- whether the CNNs can generate feature maps which contain really lines. Because HT is designed for line detector, it will only make sense when the input images or feature maps contain really lines
- The fully connected layers can extract correct (r,$\theta$) from HT matrix. Because loss function is the L2 norm or Euclidean distance between prediction and ground truth.

We are not sure if the CNNs can generate desired feature maps in more complicated dataset, but during our experiments on toy dataset, the feature map indeed contains line as shown in Figure 6.4. Thus, we can conclude that for toy dataset, the CNNs is able to generate feature maps which contain really lines.

[3]Codes can be found in my github

Figure 6.4: Feature map example

However, the fully-connect layer is not able to work properly. The loss stops at very early stage during training phase and the test accuracy[4] is extremely low. Figure 6.5 shows the training loss of model (1).

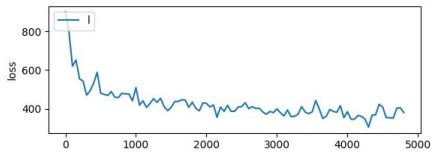



Figure 6.5: Training loss of model (1)



So, why it fails? I personally think the Hough Transform can not be used in such a brute-force way. It has to be modified to fit the general neural network architecture. The HT Layer is not trainable and it transforms the input feature map to a strange space where the values of each element is not important but the coordinates(r,$\theta$) of the values matter. And the backpropagation of HT layer should be checked further. The idea behind Hough Transform is indeed very brilliant(i.e. Hough voting), but there should be more smarter way to use it instead just inserting the Hough Transform in the neural network[5].

Overall, combing Hough Transform with CNNs is a brave attempt but it unfortunately fails.

- Hough Transform - Something to find line.
- CNNs - Convolutional Neural Network.
- Hought Transform(HT) – something we know why but it does not work very well.
- CNNs – something works very well but we do not know why(how to explain Deep Neural networks?).
- We expected : CNNs + HT = something works very well and we know why.
- But we got : CNNs + HT = something does not work and we do not know why.

[4] We simply set a threshold t, if $\|(\theta', r') - (\theta, r)\| \leq t$. The prediction is correct, otherwise it is wrong

[5] At very late phase of my thesis work, a brilliant way to use the key idea behind HT to improve deep learning was published[6]

## 6.2. FAILURE CASE: ASSOCIATIVE EMBEDDING

Associative Embedding[7] is proposed to solve the task of how to group detected joints per individual correctly. Figure 6.6 illustrates the overview of Associative Embedding used in multi-person pose estimation task. This approach is also used in CornerNet[8] to group detected top-left corners and bottom-right corners as paired keypoints of bounding box. The main idea behind associative embedding is to predict an 1D embedding(i.e. tag) for each detected candidates(i.e. joints). The predicted tags indicate how detected joints are grouped. The distance of every two tags of detected joint is expected to be small if they indeed belong to the same person. Otherwise the distance should be large. To achieve this, a collective loss function is developed which consists of **Pull Loss** and **Push Loss**.

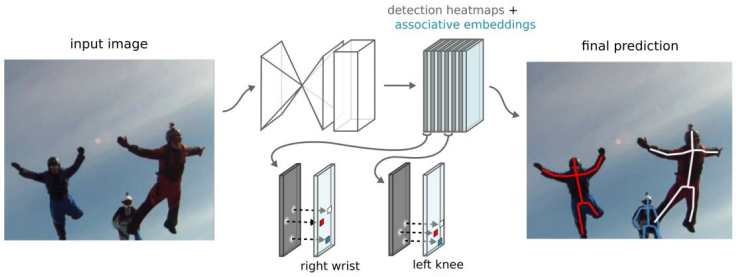



Figure 6.6: Overview of Associative Embedding used in multi-person pose estimation[7]

### 6.2.1. LOSS FUNCTION FOR ASSOCIATIVE EMBEDDING

The **reference embedding** for the nth gate in the image would be

$$\overline{h}_n = \frac{1}{4} * [h_{tr}(x_{n-tr}) + h_{tl}(x_{n-tl}) + h_{bl}(x_{n-bl}) + h_{br}(x_{n-br})] \tag{6.2}$$

where h(x) is a tag value at pixel location x, $x_{n-tr}$ is the ground truth pixel location of top-right corner of n-th gate.
The Associative Embedding loss is defined as $L_{AE}$

$$L_{AE}(h,T) = \alpha L_{pull} + \beta L_{push} \tag{6.3}$$

where

$$L_{pull} = \frac{1}{N}\sum_{K=1}^{n}[(\overline{h}_n - h_{tr}(x_{n-tr}))^2 + (\overline{h}_n - h_{tl}(x_{n-tl}))^2 + (\overline{h}_n - h_{bl}(x_{n-bl}))^2 + (\overline{h}_n - h_{br}(x_{n-br}))^2] \tag{6.4}$$

$$L_{push} = \frac{1}{N^2}\sum_{k=1}^{N}\sum_{j=1,j!=k}^{N} max(0, \Delta - |\overline{h}_k - \overline{h}_j|) \tag{6.5}$$

$\alpha$ and $\beta$ are hyper-parameters and should be tuned during experiments. Tuning the weights of these two losses (i.e. Push loss and Pull loss) is very important and time-consuming. Considering the limited time of a thesis work, I set a maximum threshold as four week.

### 6.2.2. TRAIN ON TOY DATASET

We first build our toy model following the architecture of Associative Embedding paper[7]. Figure 6.7 shows the architecture of our toy model, where **CNNs** stands for five 3*3 convolutional layers as feature extractor. The generated feature maps are fed to corner detection branch and Associative Embedding branch respectively. The corner detection branch will output four heatmaps (top-left,top-right,bot-right,bot-left) and Associative Embedding branch will output four corresponding tag-heatmaps. For corner detection branch we use MSE loss and for

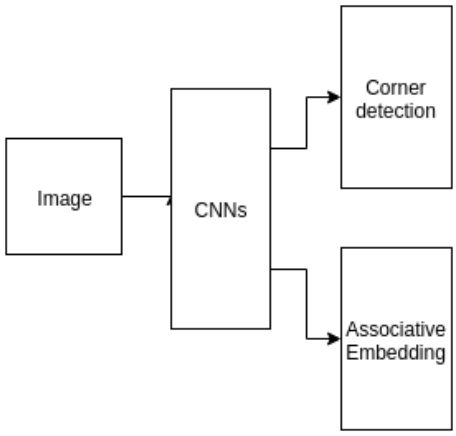



Figure 6.7: Overview of toy model architecture

Associative Embedding branch we use AE loss 6.2.1. We trained our model on toy dataset and spent around four weeks on tuning the parameters. However, the model still does not work. Based on our experiments, the corner detection part works and can detect the corners of polygons even they are not rectangle as shown in Figure 6.9. The loss for corner detection part converges smoothly as shown in Figure 6.8.



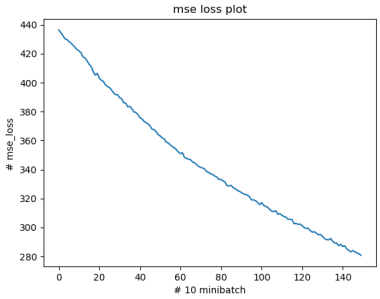



Figure 6.8: MSE loss

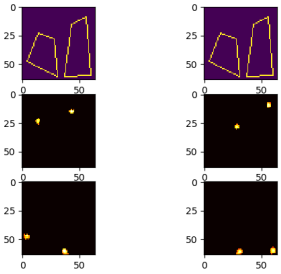


Figure 6.9: Detection

By contrast,associative embedding branch fails and it is not able to give us any meaningful results on the toy dataset. Figure 6.10 shows the training pull loss, we can see that it does not converge at all. For push loss, after a lot of efforts on tuning the parameters, it can decrease which is better than pull loss, but it stops converging at very early stage.

One possible reason why the AE does not work is that this approach does not have ground-truth. It makes training more difficult than supervised training. An other possible reason could be that our hyper parameters are still not optimal, and there could be some hyper parameters which could make the network perform better. However, without mathematical analysis and guidance, putting more efforts on hyper-parameter searching is not suitable for a master thesis project. So we decided to switch PAF[9] architecture which finally gives us something function.

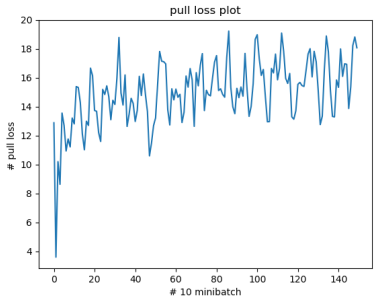



Figure 6.10: Pull loss

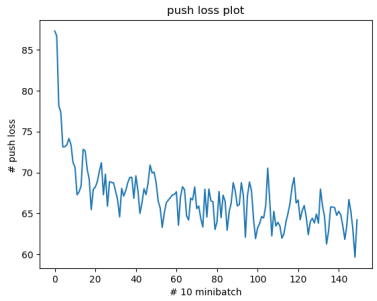



Figure 6.11: Push loss

## REFERENCES

[1] OpenCV, *OpenCV HLT*, https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_houghlines/py_houghlines.html (2014).

[2] R. O. Duda and P. E. Hart, *Use of the hough transformation to detect lines and curves in pictures*, Commun. ACM **15**, 11 (1972).

[3] J. Canny, *A computational approach to edge detection*, IEEE Trans. Pattern Anal. Mach. Intell. **8**, 679 (1986).

[4] OpenCV, *OpenCV Canny edge detector*, https://docs.opencv.org/3.1.0/da/d22/tutorial_py_canny.html (2015).

[5] Wikipedia, *Wikipedia Hough Transform*, https://en.wikipedia.org/wiki/Hough_transform (2014).

[6] C. R. Qi, O. Litany, K. He, and L. J. Guibas, *Deep hough voting for 3d object detection in point clouds*, CoRR **abs/1904.09664** (2019), arXiv:1904.09664 .

[7] A. Newell and J. Deng, *Associative embedding: End-to-end learning for joint detection and grouping*, CoRR **abs/1611.05424** (2016), arXiv:1611.05424 .

[8] H. Law and J. Deng, *Cornernet: Detecting objects as paired keypoints*, CoRR **abs/1808.01244** (2018), arXiv:1808.01244 .

[9] Z. Cao, T. Simon, S.-E. Wei, and Y. Sheikh, *Realtime multi-person 2d pose estimation using part affinity fields*, in *CVPR* (2017).