Design and Implementation of Document Detection Based on Deep Learning

Abstract

With the development of computer vision, more and more computer vision applications have emerged, such as face recognition and license plate recognition. This article presents a document detection algorithm that can automatically identify possible target areas in the picture. This algorithm consists of two parts: edge detection and target area recognition.

Edge detection algorithms are classical problems in computer vision. With the rise of convolutional neural networks, this algorithm has made new developments. This article uses the RCF edge detection model, which is currently the best edge detection model. It can make full use of the information in each layer of the network, greatly improving the detection results without losing efficiency.

The target region recognition algorithm includes a series of image mathematical algorithms, such as image binarization, Hough transform, solution of plane intersection, rectangle detection, and perspective transformation. In the Hough transform, this paper improves the original Hough transformation for the RCF model, designs a straight line classification algorithm, and improves the straight line detection effect. In addition, in the rectangle detection, this paper proposes to calculate the "black rate" of the rectangle on the image to eliminate the interference rectangle, which greatly improves the accuracy of the model.

Experiments have proved that the whole model has good operation efficiency. The entire operation can be completed within 0.5 seconds, and the test results can accurately display the target area.

This paper proposes a new design pattern for the document recognition algorithm. After the improvement of the edge recognition model and the detection of the target area, this algorithm can achieve better results in practical applications.

Key words: RCF, Edge Detection, Deep Learning, Shape Recognition, Image Processing

Contents

Chapter 1 Introduction

Now, many applications provide document scanning function. Edge detection is the most important part of document scanning. In these years edge detection has a great improvement with the development of machine learning. In order to have a well result in object detection, an effective edge detection model is essential. This thesis will propose a smart notes system, which is based on latest edge detection algorithm and object locating algorithm.

In this section, current study of edge detection and background will be introduced. Then, contribution of this research will be discussed. The structure of this paper will be showed at the end of this chapter.

1.1 Research Background

Our life becomes more and more intelligent and convenient with the development of technology. Computer vison is also closer to everyone's daily life, like face recognition, business card scanner, object detection, car license recognition, etc. There are lots of interesting areas in computer vision.

People cannot live without information. Everyone is always capturing information, so the efficiency of getting information will have a deep impact on everyone's life. This thesis will focus on smart notes system which can automatically capture the document or notes from background. This system can make the work and study much easier. The cores of this system is edge detection and target area locating algorithm.

Edge detection is a fundamental problem in image processing and computer vision. It captures significant changes in image properties by identify points in the digital image where the brightness changes significantly, such as depth discontinuities, surface direction discontinuities, changes in material properties, and scene lighting changes. In other words, edge detection plays a very important role in computer vision. It can significantly reduce the amount of data and eliminate weakly related information. What's more important, it can keep important structural attributes of the images. It can be said like this, edge detection is always the first step for most of computer vison applications because they need to separate different objects at first.

Main information can be got after the filtering by the edge detection, but it is not enough. A good target area locating algorithm still needs to be applied to identify the target area. This algorithm is essential because it will directly affect the results of the locating.

1.1.1 Document Scanning Applications

There are some smart notes and document scanners and they have brought convenience to many people, like Evernote scanner, Dropbox, YouDaoYun. All of these applications support the document scanning. Users only need to open camera on the phone or input an image, the application will do the scanning and capture the possible documents or notes like the Figure 1.1 shows.

Figure1.1 Document scanner of Evernote

According to the investigation, all of these applications almost rely on deep learning-based edge detection algorithm. Two or three years ago, the neural network algorithm was only used on servers with extremely high computing power and it is impossible for phones to run such kind of tasks. However, in the last two or three years, there have been some new trends. First, with the maturity of neural network algorithms, some scholars have put their research interest on the computational cost of compressed neural networks. The neural network model can be compressed. Second, the computing power of the chip is growing rapidly, especially the development of GPU and DSP computing capabilities. With these changes, the mobile phone can also get the computational needs of the neural network.

Take YouDaoYun for example, the edge detection algorithm of this application is HED (Holistically-Nested Edge Detection). HED is deep learning-based model and can have a better accuracy and result compared to traditional algorithm.

Now, there are more and more computer vision model adopt deep learning-based model and there is a good prospect for research in this area. It is a trend that using machine learning to solve the practical problems related to computer vision.

1.1.2 Unsolved Problems

Although some applications have successfully used deep learning-based algorithm to their products. There are still some problems. While these new advanced model are much better compared to the traditional ones, they also cost more resources. What's more, developers need to pay more cost if they want to have more ideal result. Sometimes hardware cannot provide abundant resources for models to use like some phones and embedded system.

In this case, optimization is necessary. Optimization includes the operation time optimization and processing result optimization. This thesis will try to adopt a new model to improve the result without cost more resources.

1.2 Current State of Research

Edge detection algorithms can be divided into two types: traditional algorithm and deep learning-based algorithm. In order to better evaluate the edge detection algorithm, the Berkeley research group has established an internationally recognized evaluation set called the Berkeley Segmentation Benchmark.

Traditional algorithms refer to the method which uses operator to do the operations, like Sobel Operator, Canny Operator. These algorithms usually find edges by considering the local sharp changes, especially sharp changes in color, brightness, etc. However, Only using brightness and color is not sufficient to finish edge detection because these features will be useless in more complex scenes. From the result of Berkeley Segmentation Benchmark, it is known that even if algorithm can learn low-level features such as color, brightness, gradient, etc., but in special scenes, it is difficult to achieve robust detection based on such features alone. In some cases, some high-level information, such as object-level information, need to be able to remove the middle details of the texture, making it more in line with the cognitive process of people. For example, When the painters are drawing, they always only focus on the outline of objects first and ignore the details of objects.

In the few past years, the best result of traditional algorithms is 0.7 on the Berkeley Segmentation Benchmark. This is highly due to this kind of algorithms does not include high level object-level information and there are lots of false detections. However, the bloom of convolutional neural network (CNN) brings new solutions to edge detection. Many

scholars tried to use CNN to explore whether this problem can be solved by embedding a lot of high-level, multi-scale information.

Some algorithm based on CNN show up in these years like N4-Fields, Deep Edge, Deep Counter, HFL, HED. These models have better performance on the Berkeley Segmentation Benchmark compared to traditional methods. However, these algorithms require more resources and needs to do more optimization. In 2017, Professor MingMing Chen came up with a new edge detection model called RCF. This model has better performance than the previous models.

1.3 Research Contents

This thesis focuses on using texture knowledge to correct the wrong classified labels, in order to improve the performance, mAP. The research is divided into three parts.

In the first parts, the reasons decreasing mAP is carefully researched, which will be mentioned in chapter 4, implementation and experiments. At first, it is unclear whether the low mAP is caused by wrong bounding boxes or wrong classification. We fixed the bounding boxes which overlaps ground truth, and calculated the ratio of wrong classification of these boxes, the result is inspiring, the ratio is high enough, which means correcting the wrong classified labels will improving performance of the model.

In the second parts, textual knowledge will be used in training process by adding a new loss item in the model. This loss will teach the model to learn a feature – adjacent objects should be mapped to similar feature vectors, so that they will be close to each other in mapping space.

In the last parts, textual knowledge will be used in testing process by boosting the correct label scores. This could be separated to the two parts – firstly, detecting which region is recognized uncertainly, secondly, using the surrounding knowledge from text to boost the recognition score of this unsure region, which would turn the unsure region to certain region with correct label.

After experiments, both of two models above has achieved desirable result, even though, there still leaves some weaknesses, such as the balance between speed and performance.

1.4 Thesis Structure

This thesis is composed of five chapters and the structure of every chapters will show as following:

Chapter 1 Introduction. The basic knowledge of document scanning will be introduced.

The background and current of the research related to edge detection will be discussed. In addition, our research and contribution are also listed.

Chapter 2 Related Works. It will discuss the algorithms have been used for edge detection. Then comparison is made between different edge detection algorithms and introducing model as an improvement. In addition, basic theory of CNN is introduced in order to have a better understating of following chapters.

Chapter 3 Model Design and Implementation of Smart Notes System. This chapter proposes a new smart notes system with RCF model based on the analysis of previous research. The math model for locating rectangle area will be given more explanation.

Chapter 4 Implementation and Experiments of Smart Notes System. This chapter give the detail implementation of smart notes system and will do some testing. The result of testing will be discussed from the perspective of accuracy and efficiency.

Chapter 5 Conclusion. The word in this paper will be summarized. Especially the advantages and shortcomings will be discussed. In addition, this chapter also proposes how to improve the algorithm and develop in the future.

Chapter 2 Related Works and Basic Theories

In this chapter, several edge detection models will be discussed at first. Then, the basic theories of edge detection will be discussed after that.

2.1 Related Works

Tradition edge detection algorithms cannot deal with the complex scenes and these algorithms are applied to limited applications. With the development of CNN and optimization of deep learning model, many researchers make efforts to develop edge detection model based on deep learning. In this section, five deep learning edge detection models will be discussed. The analysis and comparison will also be made.

N^4-Fields was proposed in 2014, by Yaroslav Ganin and Victor Lempitsky, and was published in 2014 ACCV [1]. This is the first and most advanced results of the natural edge detection obtained by deep learning. This method is a mix of convolutional neural networks and nearest neighbor search. As the Figure 2.1 shows N^4-Fields is very straightforward: there are many patches in the image. The feature of each patch are calculated using a convolutional neural network (CNN). Then the search is performed in the dictionary to find similar edges. The integration of these similar edge information becomes the final result. The result have been better because of more powerful edge.

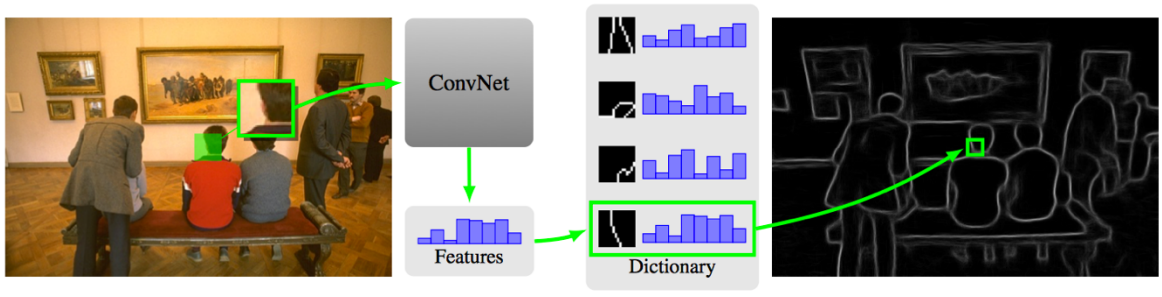


Figure 2.1 Processes of N^4-Fields [1]

Figure 2.1 shows that although N^4-Fields have better result than traditional ways, there are still some points need to be improved in this model. (1) This approach is slower and that means there will need more resources and time to run it. (2) The difference between input patches' neural codes an target labels is not small enough.

Based on N^4-Fields, Gedas Bertasius, Jianbo Shi and Lorenzo Torresani came up a new model, DeepEdge which is published in CVPR 2015 [2]. DeepEdge extended the above work, First, DeepEdge gets candidate contour points by the Canny Edge detection, and then create

patches of different scales on these points. These patches are input to two CNNs, all the way to classification, and all the way to regression. Finally, the probability of each candidate contour point is obtained.

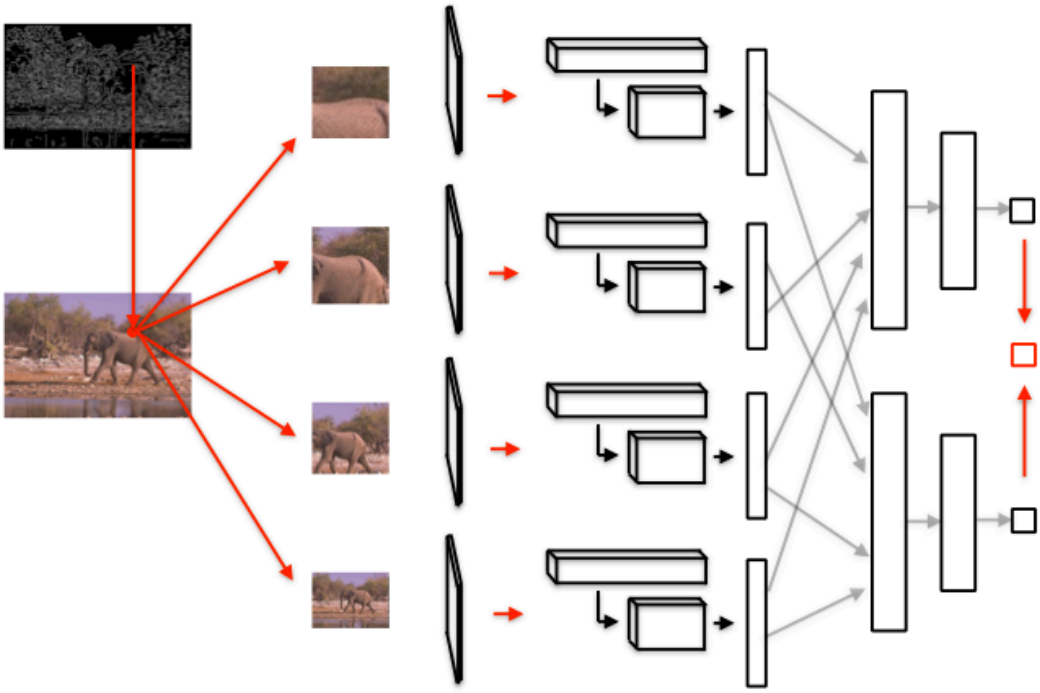


Figure 2.2 Architecture of Deep Edge [2]

As the Figure 2.2 shows, this model makes a creative change. This algorithm is from top to bottom compared to other bottom-to-up edge detection algorithm. The results show that the consideration of object's higher level information will increase the accuracy of the detection.

DeepCounter is antoher work in CVPR 2015, it is finished by the Xinggang Wang [3]. This model is similar to the former models. The work is still based on patch. First it seeks patch in the image, and then do the classification of different patches types, to judge what kind of edge, the edge belongs to finally put into different categories of edge to get the final result. The whole process as the Figure 2.3 shows.

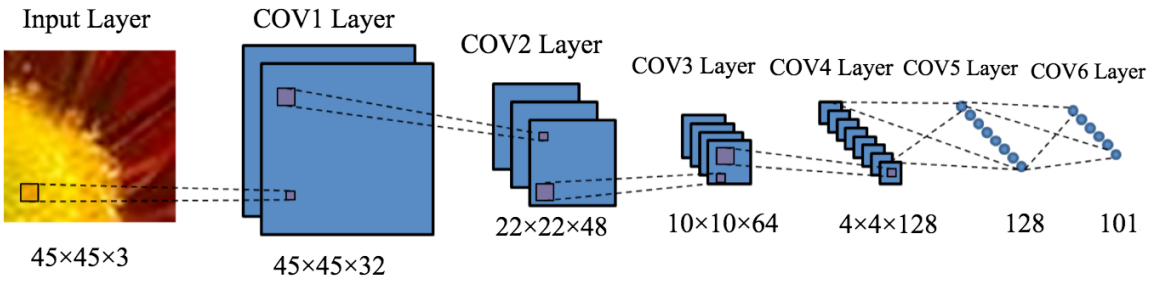


Figure 2.3 Demonstration of the architecture of CNN model in Deep Counter [3]

The author wants CNNs to learn distinguishing features. For this reason, the author considers that contours have different features and structures. Therefore, different contour features are classified and they are represented by different model parameters. Because the classification error between outlines is generally negligible or tolerable, but the classification error between the outline and the background is not tolerable, so in the loss function emphasizes the cost caused by the outline and background errors, is an additional increase Misclassification of outlines into backgrounds and backgrounds The cost of misclassifications to outlines. Through these two measures, CNN will learn more distinguishing features.

One more edge detection model in ICCV 2015 is High-for-Low (HFL) model proposed by Gedas Bertasius [4]. This model first get a group of candidate outline points. Then, it samples the images and pre-trained them with 16 convolutional layers for the target classification. Next, it finds the every candidate points' corresponding in the feature maps and does the feature interpolation. Each candidate point will generate feature vectors. Model feedbacks each of these vectors to two fully connected layers and stores the predictions to generate the final boundary mapping like the Figure 2.4 shows.

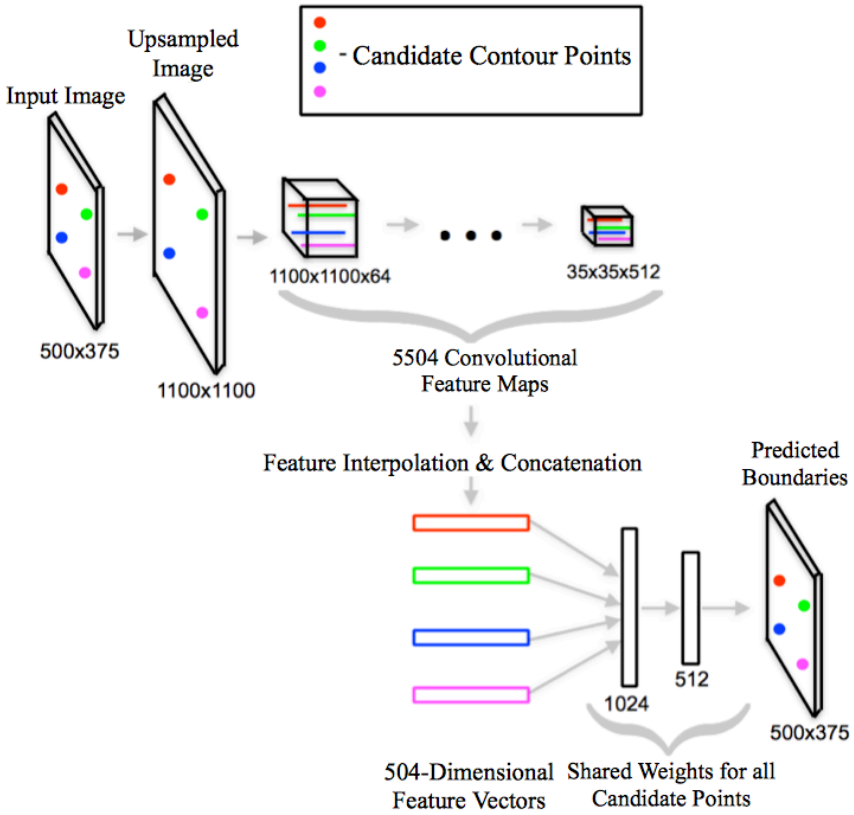


Figure 2.4 Architecture of HCL model [4]

HFL also uses CNN to judge possible candidate contour points. Due to the usage of VGG Net trained through high-level semantic information, high-level semantic information is used

to a certain extent, and therefore good results have been obtained.

All the edge detection models above based on local strategy, they are not efficient enough for the usage of high level information. Professor Sining Xie and Zhuowen Tu proposed Holistically-Nested Edge Detection in the ICCV 2015 [5]. The biggest bright spot of this work is that instead of changing the local edge detection method based on the local strategy, it uses a global image-to-image processing method. That is, no longer to operate on one patch, but on the entire image operation to facilitate the acquisition of high-level information.

In the specific algorithm structure, the author proposed holistically-nested method, mainly through several different multi-scale deep learning under the structure of comparative description. In terms of multi-scale, there are multi-scales formed by more internal networks and multi-scales formed by external networks. The former is to learn the different scale features of different layers in the neural network due to the difference in down sampling, combined to form multiple scales, and the latter is to obtain different scale information through multi-scale processing of the scale of the input image[5]. The author classifies deep learning at specific multiple scales into four categories, as the Figure 2.5 shows.

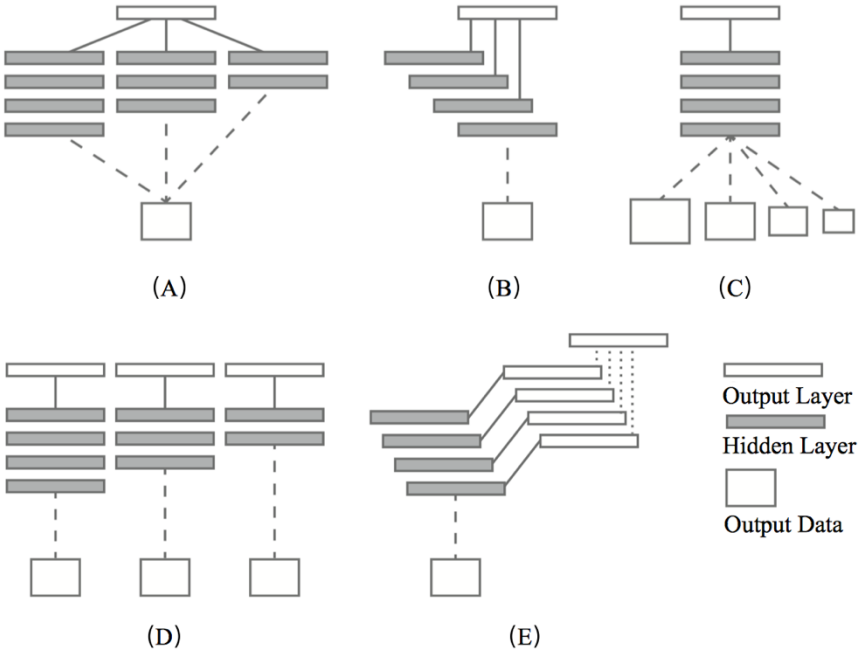


Figure 2.5 Different multi-scale deep learning architecture [5]

At the same time, this model inserts an output side-output layer on the side behind the convolutional layer and performs deep supervision on the side-output layer, making the result proceed toward the edge detection direction. At the same time, as the size of the side-output layer becomes smaller, the receptive field becomes larger. Finally, a multi-scale output is

obtained through a weighted-fusion layer [5].

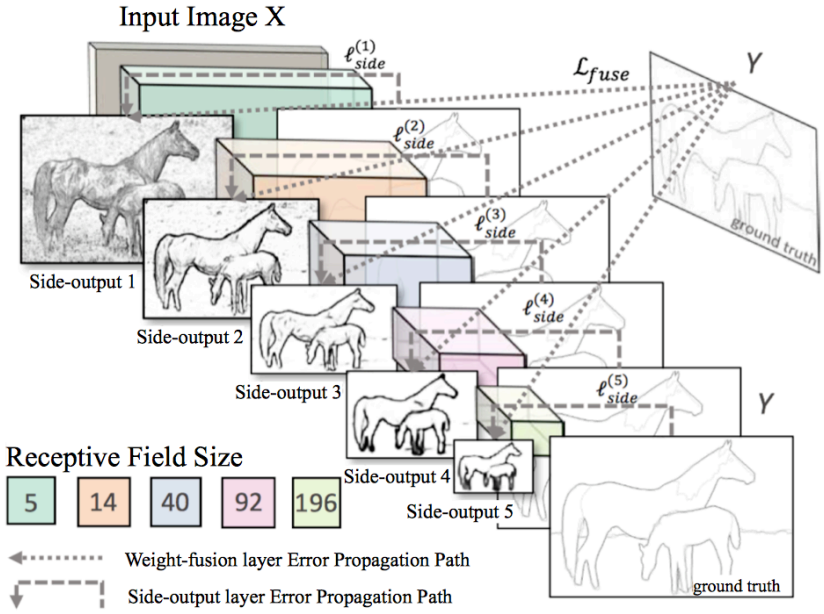


Figure 2.6 Network architecture of Holistically-Nested Edge Detection [5]

Figure 2.6 describes the development of an edge detection model that applies deep learning. Obviously, these models have achieved better results than traditional edge detection models, but there are still many problems, such as efficiency and accuracy. To get better edge detection results and faster operating speed still needs further exploration.

2.2 Basic Theories

In edge detection problems, the main algorithm is full convolutional network (FCN). Before introducing FCN, first briefly introduce the foundation of FCN, convolutional neural network (CNN).

2.2.1 Convolutional Neural Networks (CNN)

Convolutional Neural Network (CNN) was proposed as early as 1962, and the most widely used structure at present is the model proposed by LeCun in 1998 [7]. CNN, like common neural networks, consists of input and output layers and several hidden layers. Each layer of the CNN is not one-dimensional, but has three dimensions (length, width, number of channels). For example, if the input layer is a RGB image, the three dimensions of the input layer are (image height, image width, 3).

Compared with common neural networks, CNN has the following features [9] [10]:

(1) A node at layer n is not related to all nodes at layer n-1, but is only related to the n-1 layer's nodes which are near to it.

(2) All the nodes in the same layer share the weights.

(3) There will be one pool layer whose function is to scale down the length and width of this layer (usually halved) between every few layers. Commonly used pool methods are local maximum (Max) and local mean (Mean).

By adding a number of pool layers, the length and width of hidden layers in CNN continue to shrink. When the length and width are reduced to a certain degree (usually a single digit), the CNN is connected at the top with a traditional fully connected neural network, and the entire network structure is completed.

The reason why CNN can work is that it uses some of the constraints in the image. Feature 1 corresponds to the local correlation of the image (One point on the upper right corner of the image has little relation with one point on the lower left corner). Feature 2 corresponds to the translation invariance of the image (The shape of the upper right corner in the image will keep same when it moves to the lower left corner). Feature 3 corresponds to the invariability of the image's scaling (There is little information loss after the image is scaled). The addition of these constraints is like the discovery of "momentum conservation theorem" in physics. Conservation theory can make the motion of objects predictable, and the addition of constraints can make the identification process more controllable, the demand for training data is reduced, and overfitting is less likely to occur.

2.2.2 Fully Convolutional Networks (FCN)

Full convolutional network (FCN) is an algorithm developed on the basis of CNN. Unlike CNN, FCN has to solve this problem: the image is not identified by image-level tags, but pixel-level tags [8] [11] [12]. For example:

(1) Image Segmentation: The image needs to be divided into several categories based on semantics. Each pixel corresponds to a classification result.

(2) Edge Detection: The edge and non-edge parts of image need to be separated. Each pixel corresponds to “edge” or “non-edge”. This is the problem we need to solve in this thesis.

(3) Video Segmentation: Apply image segmentation into the continuous video images.

In CNN, the pool layer reduces the length and width of the hidden layer, while the FCN faces a full length and width label. There are two methods to solve this problems.

One way is to not use the pool layer, so that the length and width of each hidden layer is equal to the full length and width. The disadvantage of this approach is that, once the computation is quite large, especially when the computation is performed to the higher layers of CNN, the number of channels reach hundreds. Secondly, The result of this recognition does

not make use of global information because the convolution is always performed in the local area without using the pool layer.

Another method is transposition transpose, which can be understood as the reverse operation of the pool layer, or the up-sampling layer, and the hidden layer is scaled to its original length and width with interpolating . This is exactly what FCN uses. Of course, since the length and width of the last hidden layer of CNN is very small, there is basically only global information. If only the hidden layer is up-sampled, the local details are lost. For this reason, the FCN will perform the same up-sampling on several hidden layers in the middle of the CNN. Because the intermediate layer is scaled to a lesser extent and retains more local details, the result of the up-sampling will also include more local areas information. Finally, several up-sampling results are combined as output, so that global and local information can be better balanced.

FCN removes the fully connected layer at the top of the CNN connection. There is a classifier before each transposition of the convolutional layer. The output of the classifier is up-sampled (transpose convolution) and then added.

The hidden layers at the lower levels retain many picture details, while the hidden layers at the higher levels have better understanding of the global distribution. By combining the two, the result that contains both global information and local information.

2.2.3 Improved Cross-Entropy Loss Function

In the edge recognition problem, each pixel corresponds to a certain type of “edge” or “non-edge”. So, Every pixel can be seen as a training sample. This leads to a problem: Edges in the picture are usually much smaller than those in the non-edge, so the difference between two types’ sample number is very huge. In the problem of pattern recognition, unbalanced categories can cause many uncontrollable results and should be avoided as much as possible.

In order to solve this problem, oversampling(repeated sampling) of small sample or generating artificial data based on the spatial distribution of the original sample will be used[13] [14]. However, neither of these two commonly used methods cannot apply into this problem because one same picture contains too much samples.

HED (the model mentioned above) gives a new method that trying to change the definition of loss function. First, we outline the common functions used in CNN. In the two-class problem, cross entropy is defined as equation 2.1[5]:

$$l = -\sum_{k=0}^{n} (Q_k log p_k + (1 - Q_K) \log(1 - p_K)) \quad (2.1)$$

Here l is the loss value, n is the number of samples, k is the number of samples, Q is the value of the ta and the value is 0 or 1, p is the probability that the classifier calculates that the sample belongs to category "1". Although this function seems complicated, if we take the exponent (L=exp(-1)), we will find that this is the probability that all the samples are predicted correctly. For example, if the sample set's tag values are (1,1,0,1,1,0,…) then:

$$L = p_0 * p_1 * (1 - p_2) * p_3 * p_4 * (1 - p_5) * \ldots \quad (2.2)$$

In the equation 2.2, L is the likelihood function, which means that all samples predict the correct probability. HED uses the weighted cross entropy function [5]. For example, when the label 0 corresponds to a very few samples, the weighted cross entropy function is defined as:

$$l = -\sum_{k=0}^{n} (Q_k log p_k + W(1 - Q_K) \log(1 - p_K)) \quad (2.3)$$

In equation 2.3, W is a weight and needs to be greater than 1. Let us consider W = 2 and the likelihood function like the equation 2.4.shows:

$$L = p_0 * p_1 * (1 - p_2) * (1 - p_2) * p_3 * p_4 * (1 - p_5) * (1 - p_5) * \ldots \quad (2.4)$$

Samples belong to category 0 appear repeatedly in the likelihood function. In this way, although we cannot actually expand the sample size of the small sample category, we can achieve the effect of basic equivalence by modifying the loss function.

2.3 Summary

In this chapter, we first introduced several edge detection algorithms based on deep learning, such as the first to apply deep learning to edge detection algorithm: N^4-Fields. Then we discussed the DeepEdge, DeepCounter, HCL. These algorithm makes some improvements and the can make the result better. However, all of these models are based on the patches in the image and they cannot make a full usage of high level information in the picture. Then we place an important focus on HED model. Rather than the former algorithms tackle with the local patches, this model starts from the whole and operates on the entire picture. In addition, this model also adopts the deep supervision. HED also shows a good performance on Berkeley

Segmentation Benchmark. Currently, HED is widely used in the document scanning applications like YouDaoYun and Tencent.

After introducing the edge detection models, the basic theories applied in the deep learning-based algorithms were discussed. We first discussed the fundamental knowledge of CNN. We learned that the feasibility of CNN to do the image processing. After that, FCN was introduced. FCN is the development of CNN and it can have a better performance on image processing compared to CNN. However, it faces new problem. The solution to this problems and its result were discussed. In the end, the essential loss function in the algorithm was introduced. This function is used to deal with the problems that there is big gap between two classification groups.

In this chapter, we can know the related work of deep learning-based edge detection models and the basic theories in these algorithms.

Chapter 3 Algorithm Analysis and Design

In this chapter, all the algorithms included in the smart note system will be introduced in detail. First, the structure and flow of the entire algorithm will be shown, and the links between the various parts will be explained. Then, this chapter will discuss a new edge detection algorithm - RCF. Finally, the mathematical model for finding a rectangle will be introduced.

3.1 Algorithm Background

There are now many document scanning applications that appear in our lives. According to the survey, all document scanning applications include two parts, edge detection and target area search. The edge detection is to better separate all the objects contained in the picture and to pave the way for the next target area search. The effect of edge detection will largely determine the difficulty and accuracy of the target area inspection. Relative to the edge detection, the target area search algorithm flow is relatively single, but it still needs to be adjusted according to different edge detection models and application scenarios.

According to the above, the edge detection model adopted by most document scanning systems is HED. Compared with the traditional edge detection model and the early-depth learning-based edge detection model, the results of the HED model have made great progress. However, there are still defects in this model. Prof. Ming-Ming-Cheng's work on CVPR 2017[6] has improved the HED model. In short, since the information between different convolutional layers can be complementary, the problem with the traditional method is that the information is not used sufficiently, which is equivalent to only using the last convolutional layer before the Pooling. If all convolutional information is used instead of the last layer before pooling, such a very simple change will greatly improve the detection results. So in this system, we intend to use the Prof. Ming-Ming-Cheng's RCF model for edge detection tasks.

Although the edge detection model is more important, target detection is also an integral part of the entire system. On the basis of target detection, target detection involves a series of operations such as image preprocessing, binarization, Hough transformation, finding intersection points, finding rectangles, and evaluating rectangles. Finally, we can draw the user's desired result.

3.2 Algorithm Description

3.2.1 Basic Architecture

Figure 3.1 is the basic structure of the entire system, first enter the picture, after a layer of RCF model operation to get an edge detection picture. Subsequently, the resulting edge detection picture is grayed and binarized.

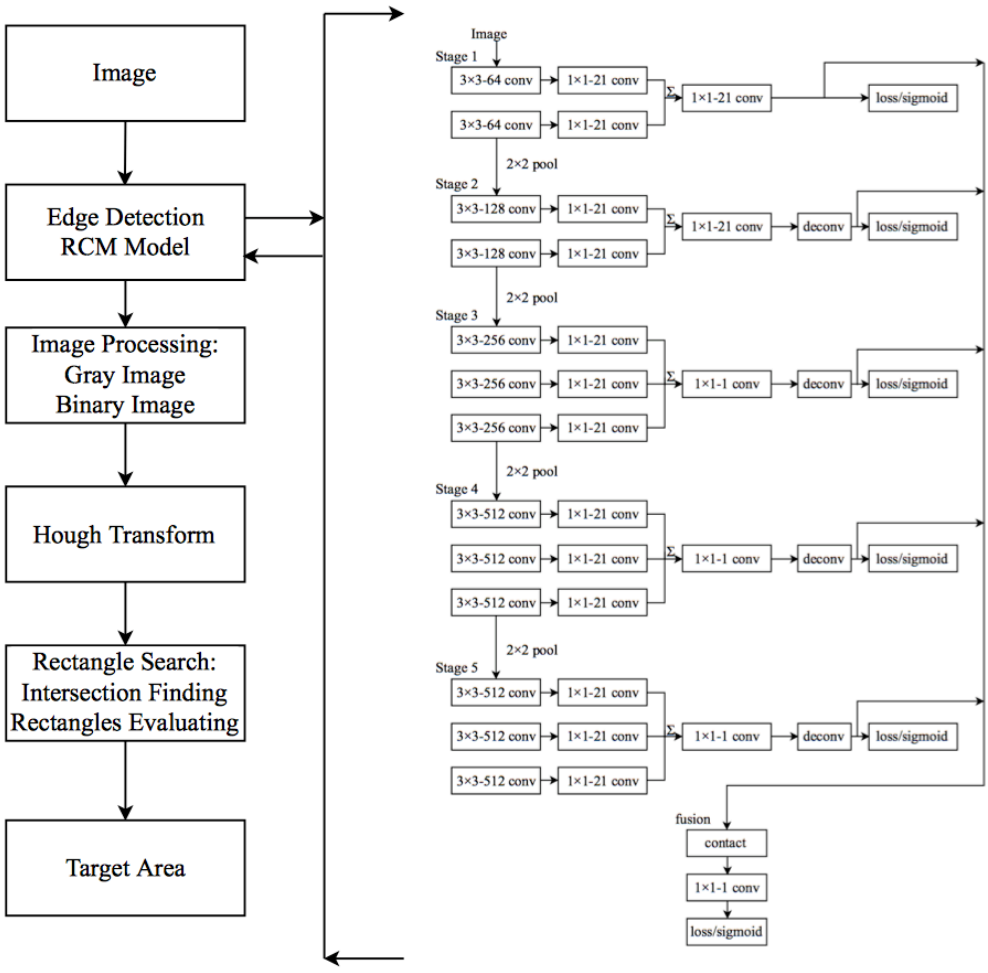


Figure 3.1 Architecture of document detection system

At this time, a clearer binary edge image can be got. The edges of each object in the image can be more clearly displayed. Use the Hough transform to find all the lines in the graph and calculate their intersections. By using a series of algorithms to find possible rectangles and calculate the possibility of the target area. Finally select the most likely target area.

3.2.2 Problem Definition

Smart notes system is similar to document scanning. The definition of the problems in smart notes will be shown.

Definition 3.1: Smart notes system is aim to find the target area from the input picture. Target area means the area that the users are interested in. For example, if a user is taking class, his interesting area should be blackboard. What's more, algorithm may detect many possible target areas, then, there is another algorithm to evaluate these areas and find the most possible one. In summary, the final target area should meet the equation 3.1:

$$final\_target = \begin{cases} [(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4)] \\ \\ Max(targes_{evaluting_{value}}) \end{cases} \tag{3.1}$$

Final target area contains the coordinates of four corners and its evaluating value is the biggest of all possible values.

Definition 3.2: As the equation 3.2 shows, after the edge detection and binarization of the input-image, all the pixel should only belong to two classes: "edge" or "no-edge".

$$pixel - value[x, y] = \begin{cases} 0, & pixel[x, y]\ is\ edge \\ \\ 255, & pixel[\mathrm{x,y}]\ is\ no - edge \end{cases} \tag{3.2}$$

3.2.3 Problem Explanation

The problem document detection model solved is to locate the interesting area and extract it from the picture. Document detection is usually applied in our work life. For example, sometimes it is necessary to record some important information and only want to save the important part of the picture and delete unnecessary information. This not only saves storage space, but also eliminates noise, laying the foundation for future expansion operations. With the development of technology, if smart glasses emerge later, this detection system can be used to quickly extract important information that is seen.

First, document scanning system will first do the edge detection on the input image, like the Figure 3.2 shows (Example from Dropbox Document Scanning [9]):

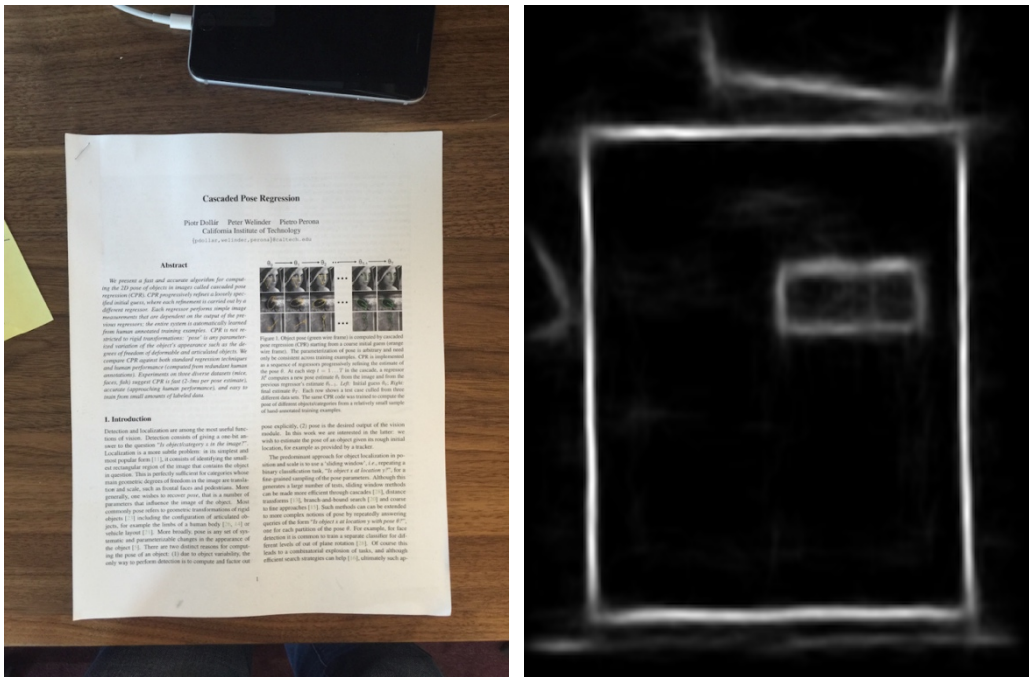


Figure 3.2 Input image and edge detection image

After finishing edge detection, we can use the result to do Hough Transform, we can find all the lines in the images, like the Figure 3.3:

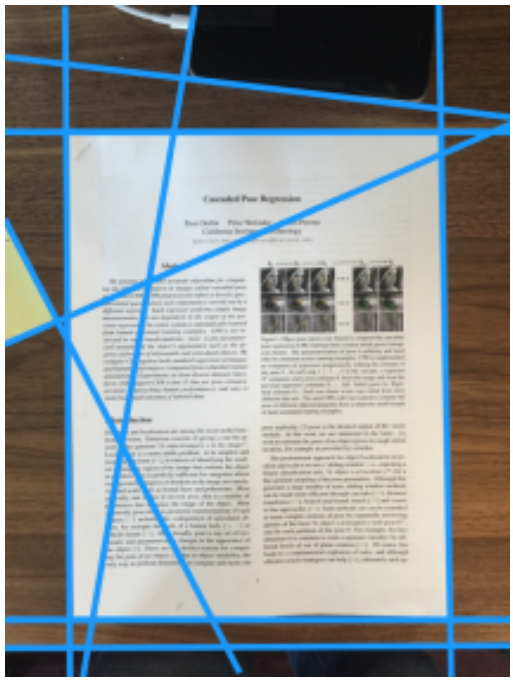


Figure 3.3 Image with Hough Transform

Then intersections of each lines will be calculated and showed on the image in the next step. After finding intersections, all the possible rectangles in the images can be got. We can finally find the final most likely rectangle in the end by algorithm.

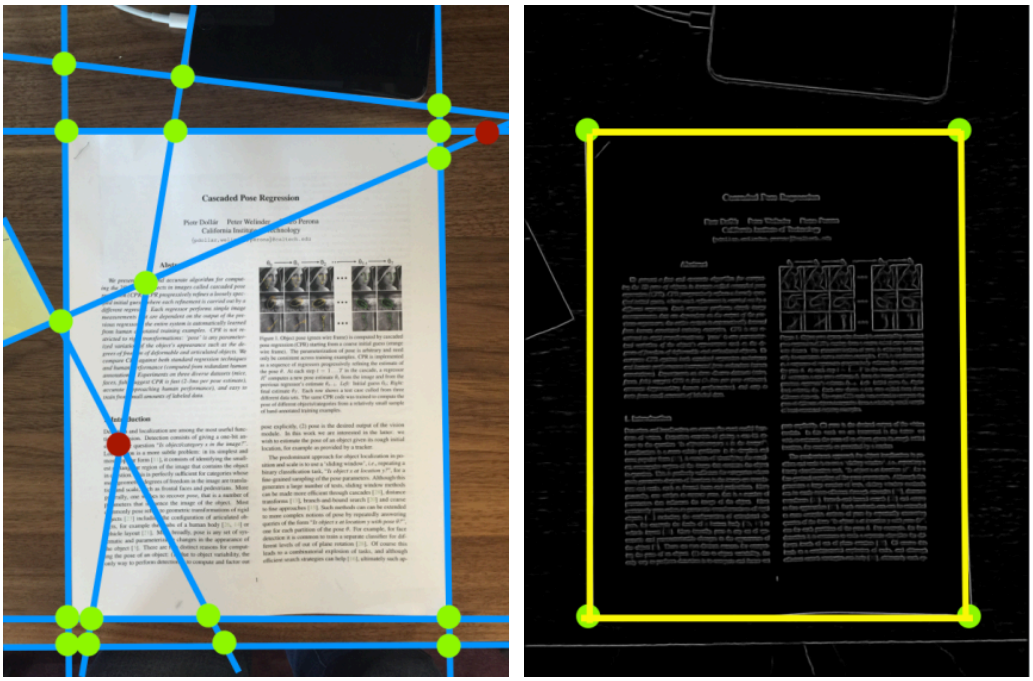


Figure 3.4 Intersections and final result

As the Figure 3.4 shows, we can see a perfect document area can be got after applying this kind of system.

3.3 Outline Design of the Algorithm

3.3.1 Edge Detection (RCF)

In this thesis, not like the common applications' algorithm, we will use a new edge detection algorithm – RCF. RCF is an enhanced version of HED. RCF points out that for boundary detection, previous neural networks only use the last layer of features as output, and many feature details are lost during the convolution process [6]. Figure 3.5 shows the whole process of the flow of multiscale algorithm.

Obviously, using only the convolutional features of the last layer, at least in the boundary segmentation domain, is not recommended. HED is a cascade structure, using VGG as a model for fine-tune, outputting the output of 5 stages in VGG for fusion, but HED only uses the convolutional output of the last layer in each stage. In addition, RCF also adopts deeply-supervised nets. RCF achieves a very great result on Berkeley Segmentation Benchmark.

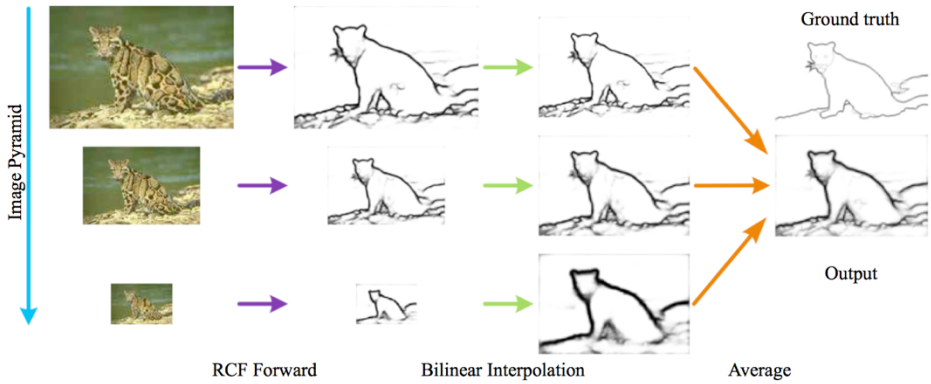


Figure 3.5 Flow of multiscale algorithm in RCF model [6]

3.3.2 Target Area Locating

Although RCF model is very important, smart notes system cannot finish its works without the later works – establish a math model to process edge image and find the target area.

(1) Generate Binary Image

Compared with the traditional edge detection algorithm, one of the advantages of the RCF model is that after processing the RCF model, there will be thicker edges and weaker details.

Based on this feature, this paper has made an innovation. After getting the image, we choose to binarize the image again by selecting an appropriate threshold. Because of the coarse edges, edge information can be well preserved. Because details are weak, details can be further moved. This will reduce interference and improve efficiency and accuracy for future work.

(2) Hough Transform

When a binary graph is obtained, the Hough transform is used to find the straight line in the graph. The Hough algorithm is a very classic image processing algorithm, and its basic principle is: First, a polar coordinate system is used to represent a straight line. Since there is a parameter representing the angles in the polar coordinate system. We can create a lot of slots to represent different lines. After creating the slots, iterate through the pixels in the image and vote for these slots. Finally, set a certain threshold, and select the slots that exceed this threshold to obtain the straight line in the Figure. A detailed description of the Hough algorithm and how to improve the Hough algorithm can be applied to this system.

(3) Calculating Intersections

The ultimate goal of the system is to look for rectangular areas that may contain critical information. When straight lines are obtained, the next step is to calculate the intersection of

these straight lines. Simultaneous linear equations can get the intersection points, and also calculate the angle of intersection. Because the final area is a rectangular area, vertices whose angle is far from 90 degrees can be excluded first. This can reduce the number of possible rectangular areas and improve the efficiency of the program.

(4) Filtering Possible Rectangles

After all the intersections are obtained, all possible rectangles can be enumerated. However, many of the rectangles are disturbing rectangles and an algorithm needs to be designed to screen them. The method used in this paper is the maximum black rate. The detailed algorithm will be introduced below. After the calculation of this algorithm, the most likely rectangle will be found.

(5) Correction of Target Area

After above operations, a "rectangular" target area will be obtained. The rectangle here is quoted because the target area may not be a standard rectangle. Sometimes due to the orientation and angle of the photography, the target area still needs further correction. In this case, it is necessary to use the corresponding algorithm to correct the area obtained in the previous step and finally output the target area.

3.4 Detail Design of Algorithm

3.4.1 Edge Detection (RCF)

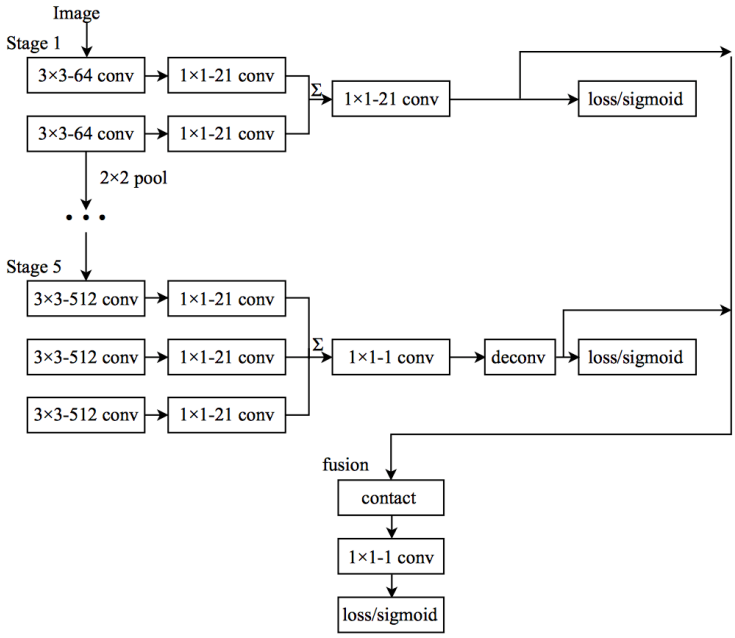


Figure 3.6 Architecture of RCF model

Figure 3.6 shows the basic flow of his model. This part I directly adopt the model of Prof.

Ming-Ming Chen's model. In the entire algorithm, the role of the RCF model is edge detection, and the result is a binary image. Ming-Ming Chen's model makes full use of the information at each level, and the results obtained are more ideal and close to human judgment.

3.4.2 Target Area Locating

(1) Generate Binary Image

Image Binarization means that there are only two values on the image, 0 and 255. 0 means black and 255 means white. First, the picture is turned into a grayscale image. Grayscale means that the value of each pixel is between 0 and 255. In fact, the picture we obtained from the previous step is a grayscale image. In order to make the edges more visible, the details are weakened. After getting the binary picture, it is more convenient for the next step.

(2) Hough Transform

In the field of image processing and computer vision, how to extract the required feature information from the current image is the key to image recognition. In many applications, lines or circles need to be quickly and accurately detected. One of the most effective ways to solve this problem is the Hough Transform, which is one of the basic methods for identifying image geometry in image processing. It is widely used and there are many improved algorithms. The most basic Hough transform is to detect lines (line segments) from binary images.

The Hough transform was first proposed by Paul Hough in 1962 [15]. The initial method needs to understand the analytical equation of the boundary line of the object, but does not need to know the position of the area in advance. The original Hough transform was used to detect lines and curves. One of the prominent advantages of this method is the robustness of the segmentation result, it is insensitive to data incompleteness or noise. The Hough transform uses a transformation between two coordinate spaces to map a curve or line in one space to a peak in another coordinate space, thereby transforming the problem of detecting an arbitrary shape into a statistical peak problem.

The following is a brief introduction of the Hough algorithm principle:

In cartesian coordinates, any straight line can be expressed in the following form in 3.3:

$$y = kx + c \tag{3.3}$$

However, there is a problem in polar coordinates. That is there is no range for the slope of the line. Especially when the slope of the straight line is large, even if the two straight lines are

very similar, the slope value is much different. Since the Hough transform is a voting algorithm, this problem will make it difficult to implement the voting algorithm. Therefore, it is necessary to use a polar coordinate system in the Hough transform. In the polar coordinate system, like the equation 3.4 shows:

$$\rho = cos\,\theta x + sin\,\theta\, y \tag{3.4}$$

As the Figure 3.7 shows. Here ρ means the perpendicular distance between coordinate origin and line. θ means the angle between the straight line and x line. If the distance above the x, ρ is positive. If the distance below x, ρ is negative.

Any straight line can be represented as (ρ,θ). Because θ and ρ have ranges, we can first create a two-dimensional array to enumerate all possible lines in the picture. The size of a two-dimensional array depends on the accuracy of the line. The initial value for each position is 0. Then for each possible straight line, the black pixels in the graph are traversed. If a pixel is on a certain line, the value of the line is incremented by one. After all the calculations are completed, count the value of each straight line. If the value of a line is large, it means that there are many black pixels on this line, indicating that the possibility of the existence of this line is greater. Finally we set a threshold and select the straight line that exceeds the threshold as the detected line.

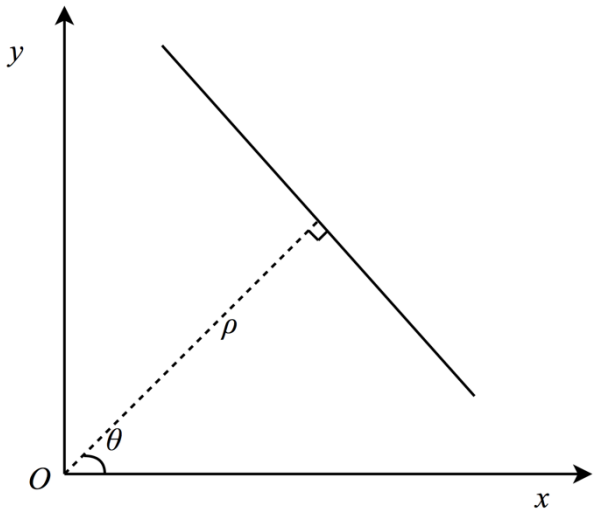


Figure 3.7 A line in polar coordinate

The pseudocode of Hough Transform is as the algorithm 3.1 shows:

Algorithm 3.1: Hough Transform

Input: (Binary) image
Output: Lists of (ρ, θ).
Begin
Let Array(R,T) be the counter array, R and T depends on the resolution, initialized as zeros.
For each pixel E(i,j) = 0, and for h = 1,…T
Let $\rho = isin\ \theta_d(h) + jsin\theta_d(h)$
Find index k so that $\rho_d(k)$ is closest to ρ
Increase Array(k,h) by one
End
Find all local max (k_p, h_p) Where $(k_p, h_p) > Threshould$
End

Although the RCF model brings more obvious edge detection images, this makes edge detection easier. But at the same time RCF will also bring "defects." Because this article uses OpenCV for image processing, many noise lines are detected using OpenCV's Hough transform, such as Figure 3.8. After analysis, this result is due to the fact that the edges of the RCF model are too much coarser than the traditional algorithm, and even if the Hough transform uses the 'Local Max' strategy, many noise lines will still be generated.

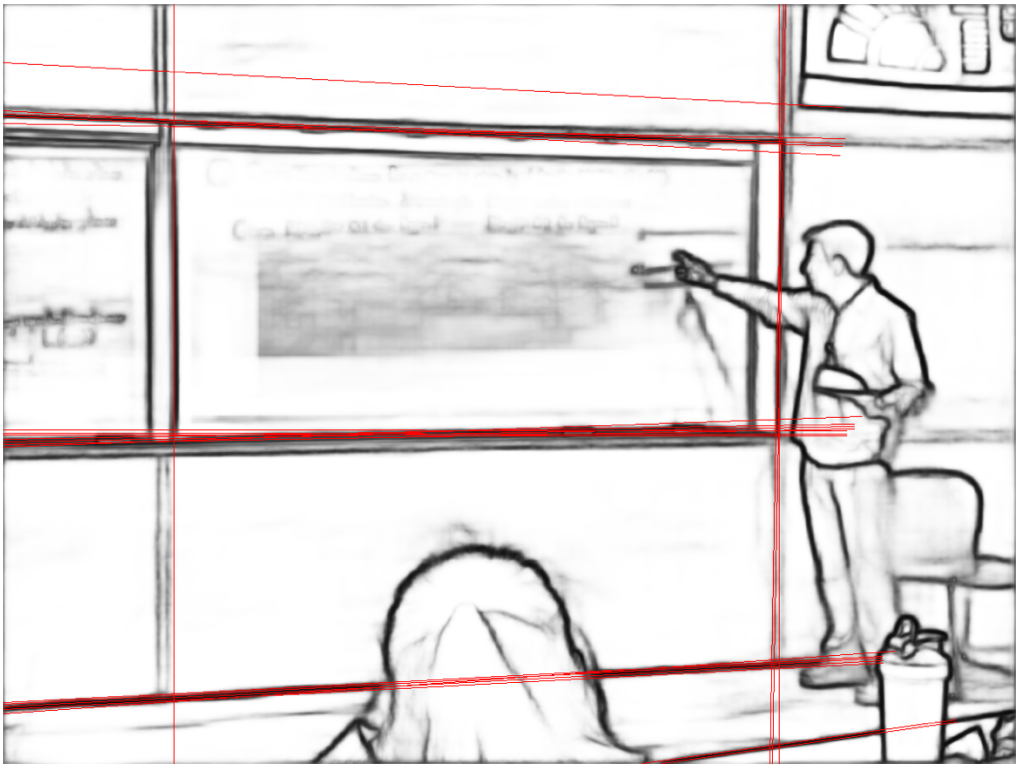


Figure 3.8 Noise Lines of Hough Transform in OpenCV

Although the test results contain the correct straight lines, these noise lines can greatly increase the amount of calculations in subsequent steps. The final result may be affected as a result. So we need to use an optimization algorithm to remove noise.

Algorithm 3.2 shows the upgraded Hough Transform and there is an extra classification step to convert the dense lines into one line.

Algorithm 3.2: Hough Transform Optimization Algorithm

Input: Lists of (ρ, θ)
Output: Lists of (ρ, θ) without noise.
Begin
Crate a line class Lines list to store each line class
For each line *l* in lists of (ρ, θ)
IF no such line class contains line *l*
Create a new line class in Lines[] and put *l* into the new class
ELSE
Add *l* to the existing class
End
For each line class *lc* in Lines list
Calculate the average value of (ρ, θ) to represent a new line
End
End

(3) Calculate Intersections

When we get a relatively clean lines, we need to calculate the intersection of each line in the graph. The intersection of the lines is the potential vertices of the rectangle. It is the essential step to get the final target area. To reduce the noise and search for the rectangles in the picture, a small filter is added here. To reduce the noise and search workload for the rectangles in the picture, a small filter is added here. Because the angle at the apex of the rectangle is 90 degrees, if the angle between the two straight lines is far from 90 degrees, then this vertex can be filtered.

Frist, calculating the intersection of two lines in the polar coordinate system, simultaneous equation solving like the equation set 3.5:

$$\begin{cases} \rho_1 = cos\,\theta_1 x + sin\,\theta_1\, y \\ \rho_2 = cos\,\theta_2 x + sin\,\theta_2\, y \end{cases} \tag{3.5}$$

Then, As the equation 3.6 shows, calculate the angle of two lines:

$$agnle = \cos^{-1}\left(\frac{cos\,\theta_1 * cos\,\theta_2 + sin\,\theta_1 * sin\,\theta_2}{\sqrt{sin\,\theta_1{}^2 + cos\,\theta_1{}^2} * \sqrt{sin\,\theta_2{}^2 + cos\,\theta_2{}^2}}\right) \tag{3.6}$$

(4) Filter Possible Rectangles

After calculating the intersections, there are two more tasks to follow. Find out all the rectangles that these points may constitute and select from those rectangles the one that is most likely to be the target area.

Algorithm 3.3 shows how to find all the possible rectangle by the intersections.

Algorithm 3.3: Finding Possible Rectangle

```
Input: A list of intersections (each intersection contains coordinate and indexes of two lines)
Output: A list of rectangles (each rectangle contains four indexes of intersections)
Begin
   Create a list intersection_combination.
   Create a list possible_rectangle
   Put all the combination (4 indexes combination)of all indexes of intersections
   For each combination in intersection_combination:
      If the intersections in the combination only contains 4 lines and each line appears exactly 2 times:
         Add the combination to the possible_combination
   End
End
```

Now, we need to select the most possible rectangle area from possible rectangle areas. There are two main problems needing to be solve. Like the Figure 3.9 and 3.10 shows:

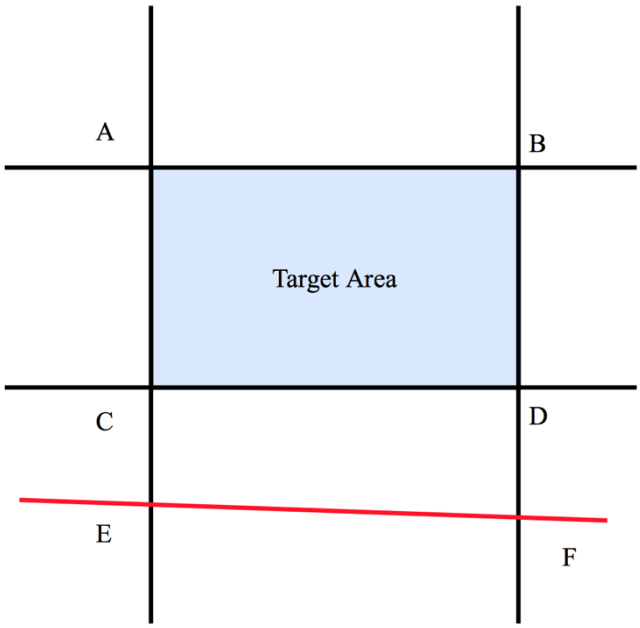


Figure 3.9 Noise Rectangle Example I

In order to maintain the accuracy of the detection, the results of the Hough transform in this paper are straight lines rather than line segments, because sometimes a certain edge may be discontinuous due to light or other reasons. If algorithm uses line segments, the result will be very poor. A straight line can be a good way to avoid this, because the line only needs to be guaranteed to have a large number of points on it. However, nothing is perfect. Adopting this strategy will also bring disadvantages and introduce some noise rectangles.

Usually, in the environment in which the target area is located, there is not only the target, but also other interfering objects. Because the interfering objects also have edges, the edge detection algorithm also detects these edges. This will create noise rectangles. As shown in

Figure 3.9, EF is the side of the interference object. AC, CD, DB, BA are the edges of the target area. Rectangular algorithm detection will not only get rectangular ABCD, but also CDEF and AEBF.

Before introducing the solution, we first give the definition of "Black Ratio" of a line segment. Black rate of a line segment means the ratio of the number of black pixels on the line to the length of the line segment. The black ratio of a rectangle like the 3.7 shows:

$$r-blackratio=\frac{blackratio_{line1}+blackratio_{line2}+blackratio_{line3}+blackratio_{line4}}{4} \quad (3.7)$$

By observation, compared to the target area ABCD, the noise rectangles ABEF and CEDF both include edges CE and DF in Figure 3.10. In fact, neither CE nor DF are real edges. They are due to the extension of the straight line. The two segments are white on the binary graph. This feature will cause the black ratio of the rectangular ABEF and CEDF to decrease. In this way, the target area ABCD can be distinguished from these rectangles.

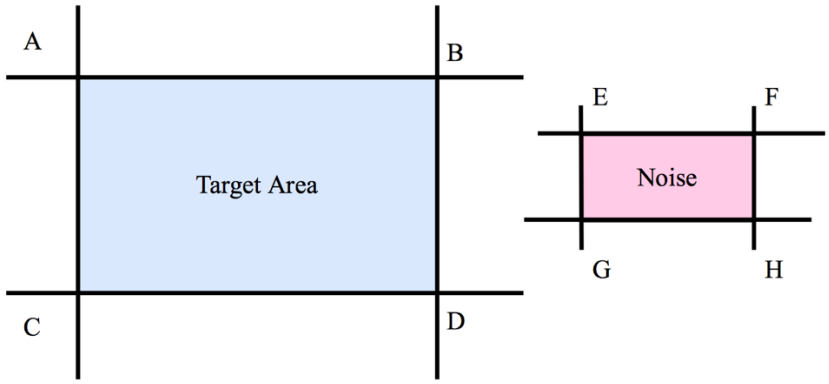


Figure 3.10 Noise Rectangle Example II

Although this kind of noise rectangle is solved, there is still another noise rectangle that cannot be solved by the above method, for example, Figure 3.12. There is a real area in the image. The rectangle ABCD is target area and the rectangle EFGH is the noise rectangle. Not like the above example, EFGH is the real area in the picture. So it has the similar black ratio with ABCD.

In order to solve this problem, this model starts from the user's use habits. The target area that people usually identify is the most prominent and larger area in the map. So when this happens, there are two rectangles with similar black ratios. The larger area is considered to be the target area.

(5) Correction of Target Area

Until now, most of the work has been completed. However, in order to give the user a better

experience. Still need to consider a situation where the user's photograph is not a standard rectangle because of the shooting position and angle as the Figure 3.11 shows. So after getting the target "rectangular", it still needs to be corrected to make the result more readable.

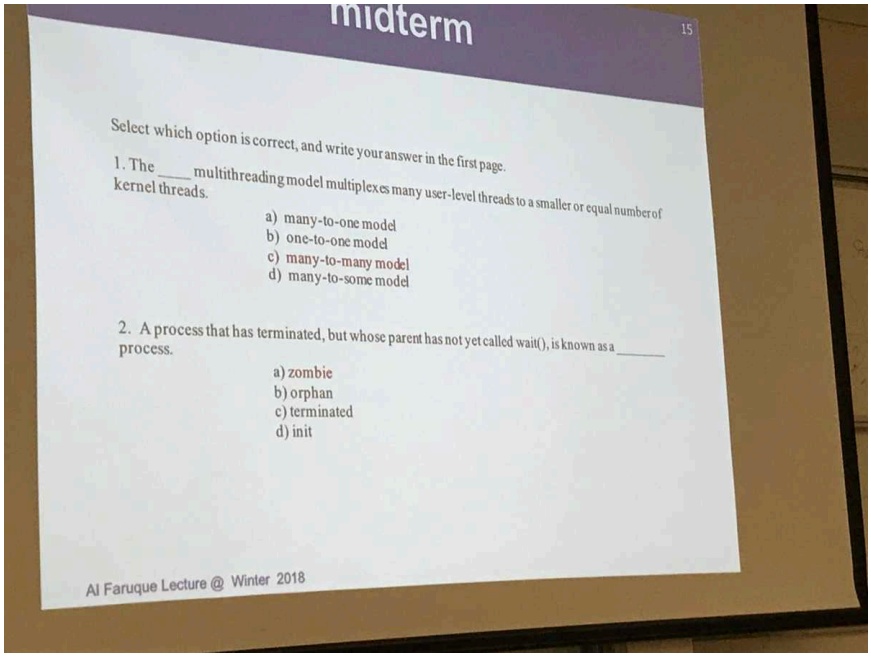


Figure 3.11 An input picture contains nonstandard-rectangle target area

The solution to solve this problem is applying perspective transform [16] [17] [18] to target area. Perspective Transformation projects a picture into a new viewing plane. The general transformation is formula 3.8:

$$[x', y', w'] = [u, v, w] \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \tag{3.8}$$

u, v is the coordinates before the transformation, and x', y' is the coordinates after the transformation. $\begin{bmatrix} a_{11} & a_{12} \\ a_{12} & a_{22} \end{bmatrix}$ means the linear transformation, like scaling, shearing and rotation. $[a_{31}, a_{32}]$ is used for panning. $[a_{13}, a_{23}]^T$ is user to generate perspective transformation.

In this problem, we only need to apply 4-points perspective transform because we already know the four corners of target rectangle area. The following is the pseudocode of this Algorithm.

Algorithm 3.4 shows how to use 4 points to do the perspective transform.

Algorithm 3.4: Four Points Perspective Transformation

Input: Coordinates of Four Corners of Target Area
Output: Correction Image of Target Area
Begin
Sort the four corners. # find the top-left, top-right, bottom-left, bottom-right points.
Calculate the new width of the target area.
Calculate the new height of the target area.
Compute the perspective transform matrix by OpenCV and apply it to the target area.
Generate the correction target area.
End

3.5 Summary

In this chapter, the background of the model-containing algorithm is introduced, and why the new RCF model and the detection of the target area are emphasized. Afterwards, a brief introduction to these algorithms is provided, so that the reader can have a general understanding of the entire model. Next, each algorithm is described in detail, especially for each step involved in the detection of the target area.

In the recognition of the target area, the methods and effects of image binarization are first introduced. Then discussed the principle of Hough transform and its defects, and proposed an improved method. Then the calculations of the coordinates and angles of the intersections of the various straight lines are introduced and the method of noise reduction is proposed. Then discussed how to find possible rectangles and use certain algorithms to eliminate noise rectangles, and finally get the target area. After obtaining the target area, we discussed how to use perspective transformation to correct the image to better display the results.

Chapter 4 Implementation and Evaluation

In this chapter, implementation and experiments of RCF edge detection and Target area location will be introduced. The whole project is based on Python 2.7, Caffe and OpenCV are also used in this system.

4.1 Algorithm Implementation

4.1.1 Edge Detection (RCF)

(1) Implementation and Deployment

RCF model is based on Caffe and Python 2.7. First need to build Caffe [19] and then put the pretrained models in to RCF. After finishing this, put the images that need to do the edge detection in the related folder and run the following code:

```
# Make sure that caffe is on the python path:
import sys
sys.path.insert(0, caffe_root + 'python')
import caffe
data_root = ' rcf_master/data/'
print(data_root)
with open(data_root+'test.lst') as f:
    test_lst = f.readlines()

test_lst = [x.strip() for x in test_lst]
im_lst = []
for i in range(0, len(test_lst)):
    print(data_root+test_lst[i])
    im = Image.open(data_root+test_lst[i])
    in_ = np.array(im, dtype=np.float32)
    in_ = in_[:,:,::-1]
    in_ -= np.array((104.00698793,116.66876762,122.67891434))
    im_lst.append(in_)
#remove the following two lines if testing with cpu
#caffe.set_mode_gpu()
#caffe.set_device(0)
# load net
net = caffe.Net('rcf_master/examples/rcf/rcf_pretrained_bsds.caffemodel', caffe.TEST)
save_root = os.path.join(data_root, 'test-fcn')
if not os.path.exists(save_root):
    os.mkdir(save_root)
start_time = time.time()
for idx in range(0, len(test_lst)):
    print(idx)
    in_ = im_lst[idx]
    in_ = in_.transpose((2, 0, 1))

    # shape for input (data blob is N x C x H x W), set data
    net.blobs['data'].reshape(1, *in_.shape)
    net.blobs['data'].data[...] = in_
```

```
# run net and take argmax for prediction
net.forward()
fuse = net.blobs['sigmoid-fuse'].data[0][0, :, :]
fuse = 255 * (1-fuse)
cv2.imwrite(save_root + '/' + test_lst[idx][5:-4] + '_fuse.png', fuse)
```

(2) Result Display

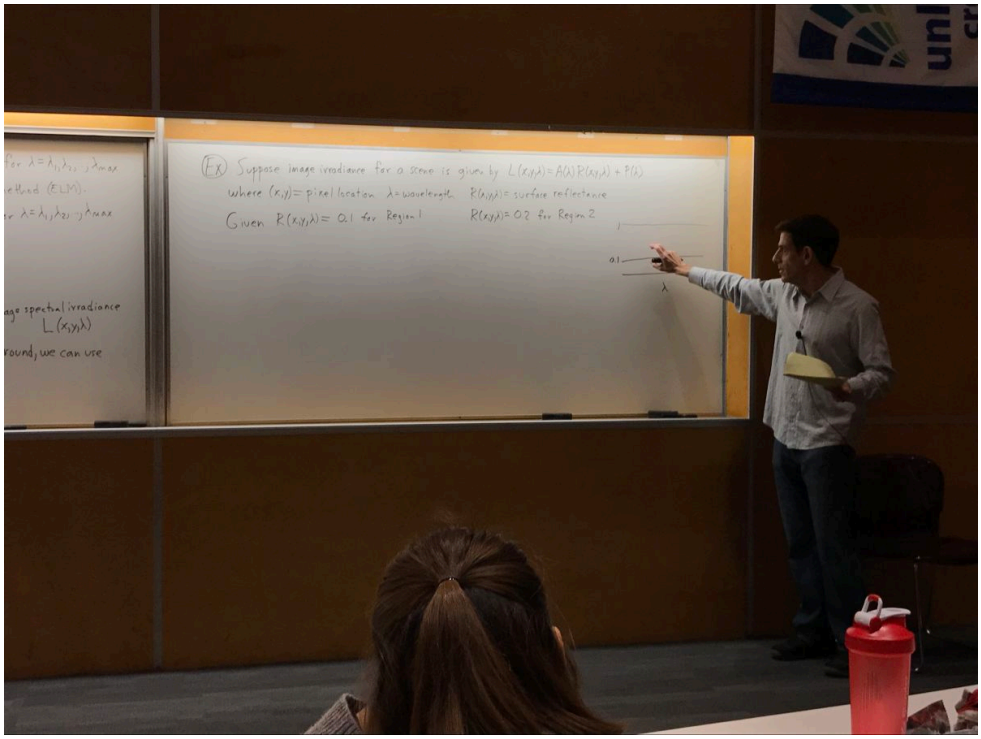


Figure 4.1 Original picture

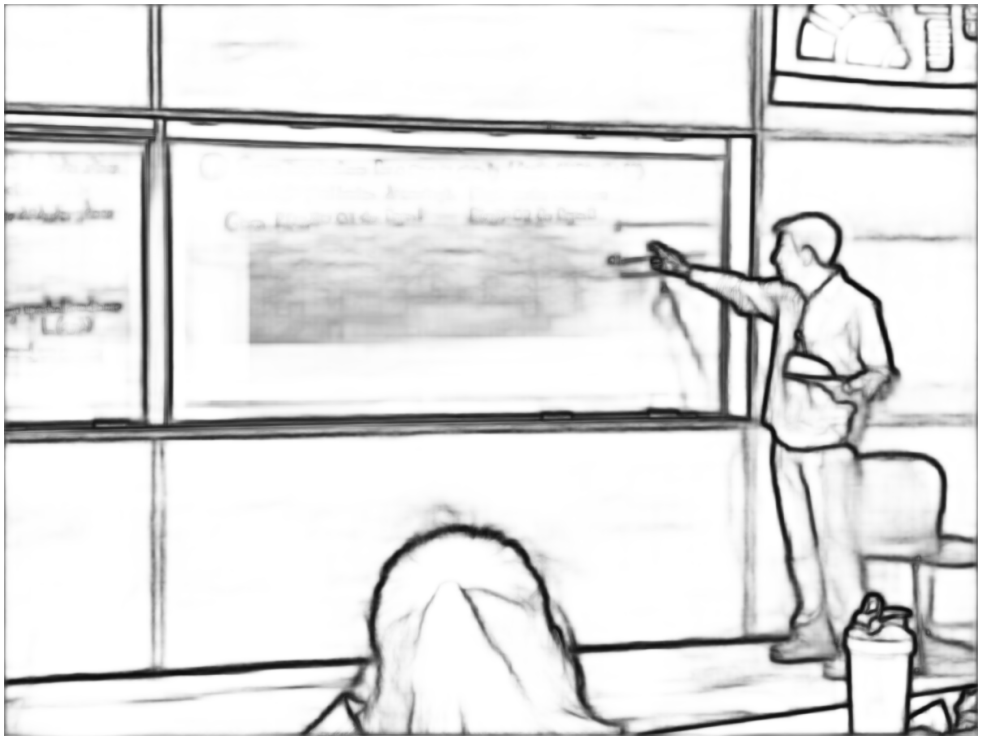


Figure 4.2 Edge detection picture of RCF

As the Figure 4.1 and 4.2 show, first we should input an original image and the RCF model will do the calculations on the image based on VGG16 network, RCF model will use all the information in the each layers of network and final get the edge detection image.

4.1.2 Generate Binary Image

(1) Implementation and Deployment

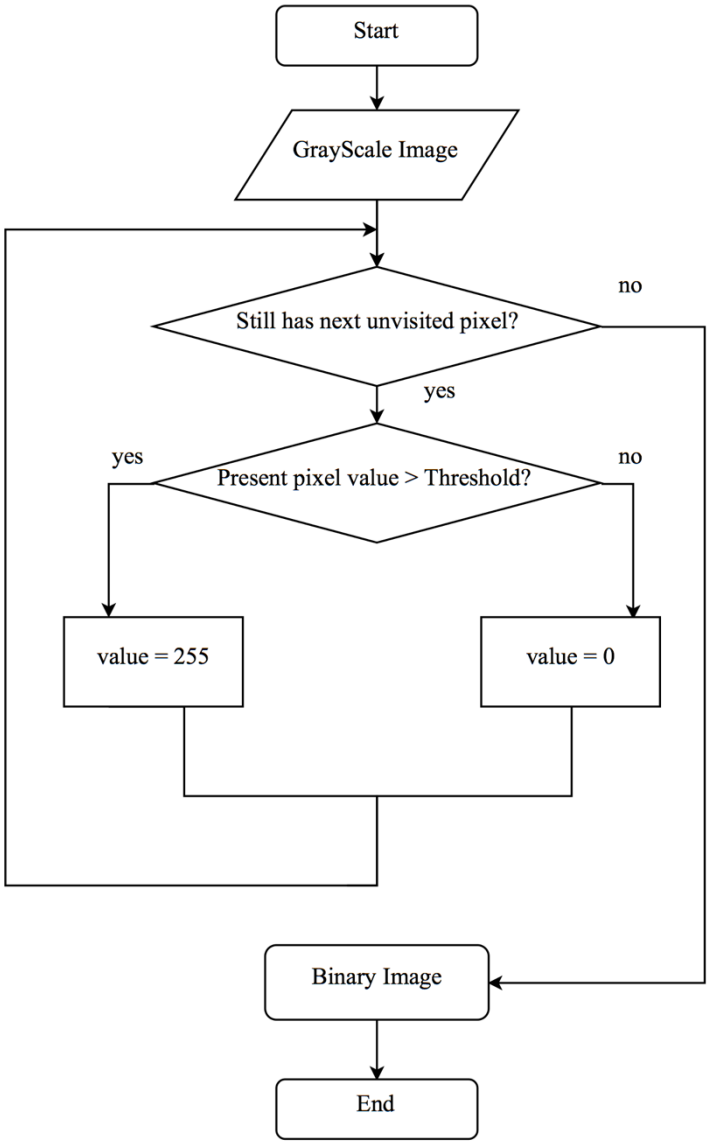


Figure 4.3 Flow chart of generating binary image

Figure 4.3 shows the flow of getting binary image. This part mainly use two functions of OpenCV Cv2.cvtColor and Cv2.threshold:

```
# generate gray level image
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
cv2.imwrite('2G.png', gray)
# generate binary image
ret, binary = cv2.threshold(gray, 50, 255, cv2.THRESH_BINARY)
```

```
cv2.imwrite('2B.png', binary)
```

(2) Result Display

As the Figure 4.4 shows: after getting the edge detection images, we will use the functions provided by OpenCV to get the binary image again, we can see that this step will remove more irrelevant details and get more accurate edges.

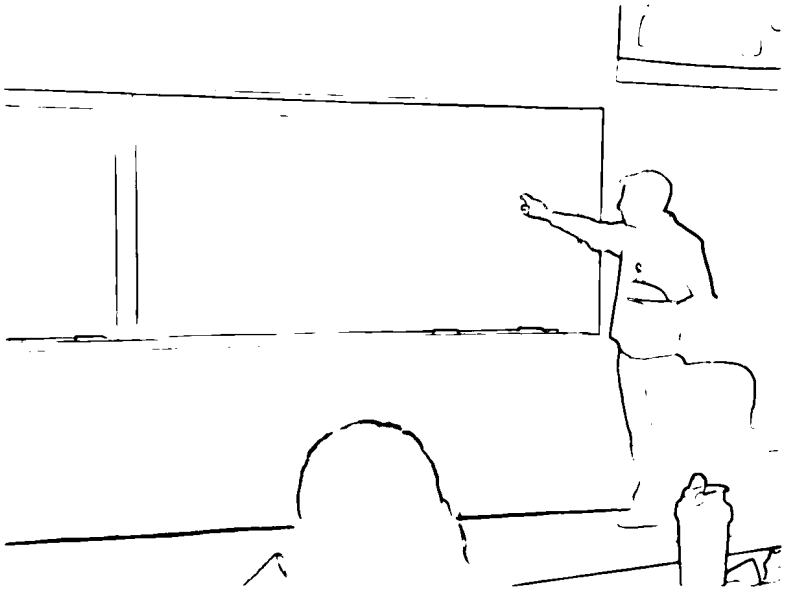


Figure 4.4 Image binarization

4.1.3 Hough Transform

(1) Implementation and Deployment

There are two parts in this step. The first is to use the Hough Transform of OpenCV. The second part is to use the de-noising algorithm designed by ourselves to optimize the straight line. The first part:

```
lines = cv2.HoughLines(edges, 1, np.pi/180, 150)
extra_img = cv2.imread('2_fuse.png')
for i in range(len(lines)):
        rho = lines[i][0][0]
        theta = lines[i][0][1]
        a = np.cos(theta)
        b = np.sin(theta)
        x0 = a*rho
        y0 = b*rho
        x1 = int(x0 + 1000*(-b))
        y1 = int(y0 + 1000*(a))
        x2 = int(x0 - 1000*(-b))
        y2 = int(y0 - 1000*(a))
        cv2.line(extra_img, (x1, y1), (x2, y2), (0, 0, 255), 1)
 cv2.imwrite('2H1.png', extra_img)
```

The second part:

```
    # store the different lines class
    line_class = []
    # show if lines can match the already classes
    match_label = -1    # 0 matched -1 no matched
    rho_difference_threshold = 300    # this is the max different between lines in the same class
    theta_difference_threshold = np.square(20.0/180 * np.pi)

    # rho_difference_threshold = 0    # this is the max different between lines in the same class
    # theta_difference_threshold = 0
    for ii in range(len(lines)):
        for rho, theta in lines[ii]:
            # scan line class
            # difference values mean the difference between lines
            for j in range(len(line_class)):
                for k in range(len(line_class[j])):
                    rho_difference = np.square(line_class[j][k][0]-rho)
                    theta_difference = np.square(line_class[j][k][1]-theta)
                    if rho_difference < rho_difference_threshold and theta_difference <
theta_difference_threshold:
                        line_class[j].append((rho, theta))
                        match_label = 0
                        break
                if match_label == 0:
                    break
            # if matched, continue the next lines
            if match_label == 0:
                match_label = -1
            else:    # if no, create a new class
                # print ("create new class" +str(rho) +" "+ str(theta))
                line_class.append([(rho, theta)])
    result_lines = []

    # calculate the average rho and theta in every class and put them into the result list
    for i in range(len(line_class)):
        mean_rho = 0
        mean_theta = 0
        for j in range(len(line_class[i])):
            mean_rho += line_class[i][j][0]
            mean_theta += line_class[i][j][1]
        mean_rho = (mean_rho * 1.0) / len(line_class[i])
        mean_theta = (mean_theta * 1.0) / len(line_class[i])
        result_lines.append((mean_rho,mean_theta))
return result_lines
```

(2) Result Display

As the Figure 4.5 and 4.6 show, there are lots of detected lines in the Figure 4.5. Some lines are very dense because the thick edge which will cause more calculation and will cause the loss of efficiency. In order to solve this problem, an optimization algorithm will be used to cancel the noise, this algorithm will classify straight lines. This algorithm will calculate a single final line for each classes. This method can greatly reduce the noise lines without losing important information. The Figure 4.6 shows the optimization result.

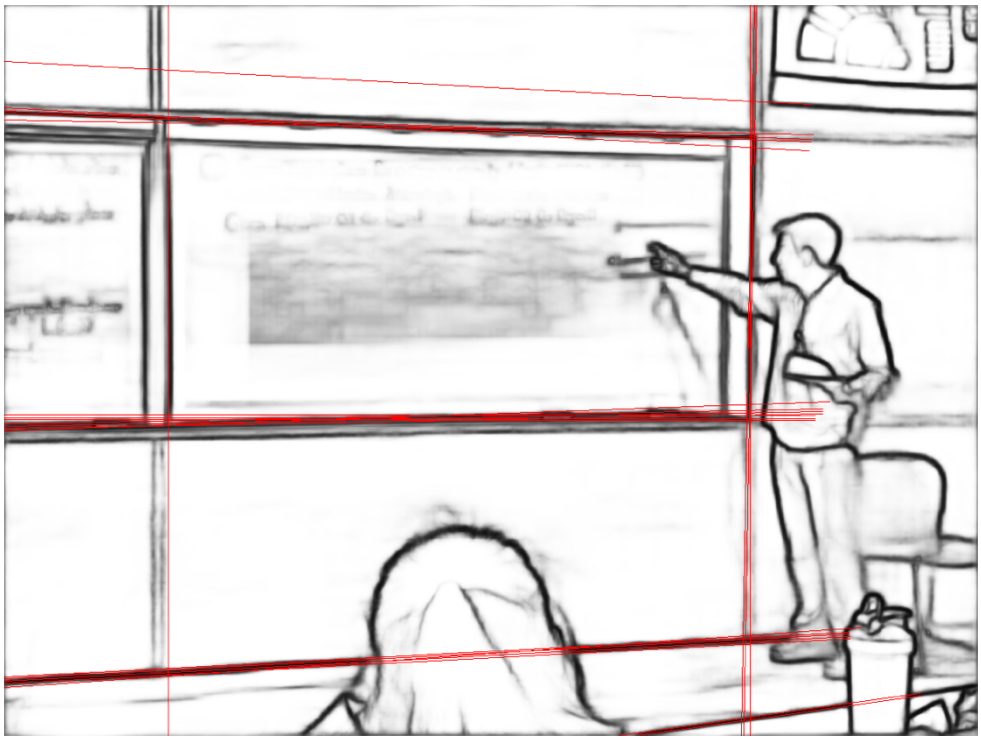


Figure 4.5 Hough Transform of OpenCV

Figure 4.5 shows the result of Hough Transform provided by OpenCV, there are some dense red lines and the Figure 4.6 shows the result of upgraded Hough Transform and there are obvious less red dense lines and they are converted into one line.

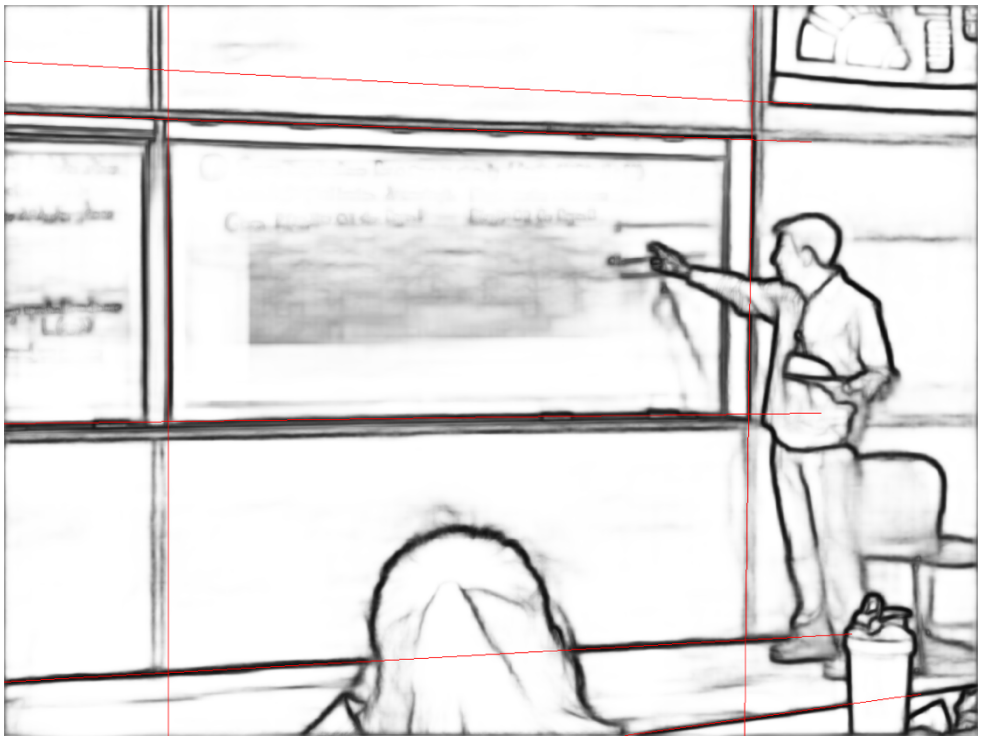


Figure 4.6 Optimized Hough Transform image

4.1.4 Calculate Intersections

(1) Implementation and Deployment

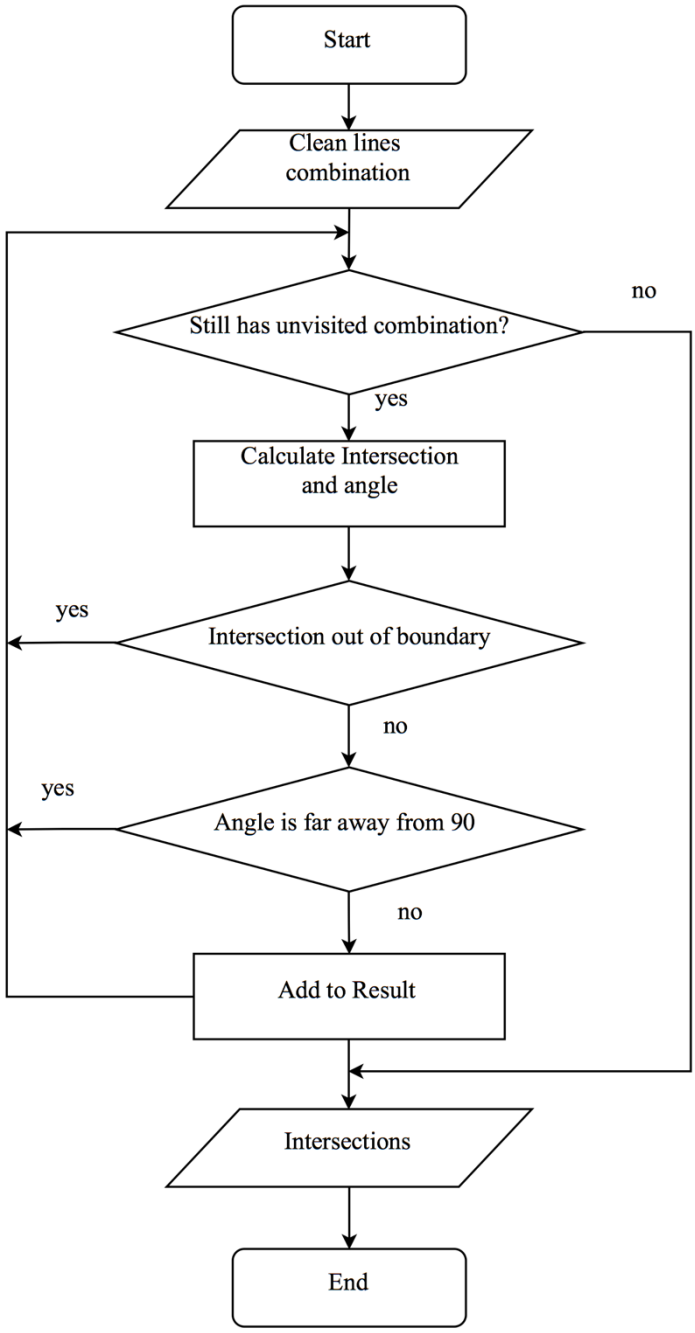


Figure 4.7 Flow chart of finding potential corners of rectangles in image

Like the Figure 4.7 shows, most of the knowledge involved in this part is geometry. Function get_intersection_anle(rho1,theta1,rho2,theta2) is used to get two intersections and angle of them. Input the two rho and theta and return the ordinate and angle of intersection:

```
def get_intersection_angle(rho1, theta1, rho2, theta2):
    # calculate intersection
    A = np.matrix([[np.cos(theta1), np.sin(theta1)], [np.cos(theta2), np.sin(theta2)]])
    a = np.matrix([rho1,rho2]).T
```

```
            result = np.linalg.solve(A,a)
            x = float(result[0])
            y = float(result[1])
            # calculate angle
            v1 = [np.cos(theta1), np.sin(theta1)]
            v2 = [np.cos(theta2), np.sin(theta2)]
            angle = np.arccos((v1[0]*v2[0] + v1[1]*v2[1])/(np.sqrt(np.square(v1[0])+np.square(v1[1])) *
np.sqrt(np.square(v2[0])+np.square(v2[1]))))
        return x, y, angle
```

(2) Result Display

The algorithm will first calculate the every intersections and the angles in the image. Then, the algorithm will delete the impossible intersections and keep the possible ones. As the Figure 4.8 shows, although there are little inevitable intersections in the image, all of the target intersections have been found.

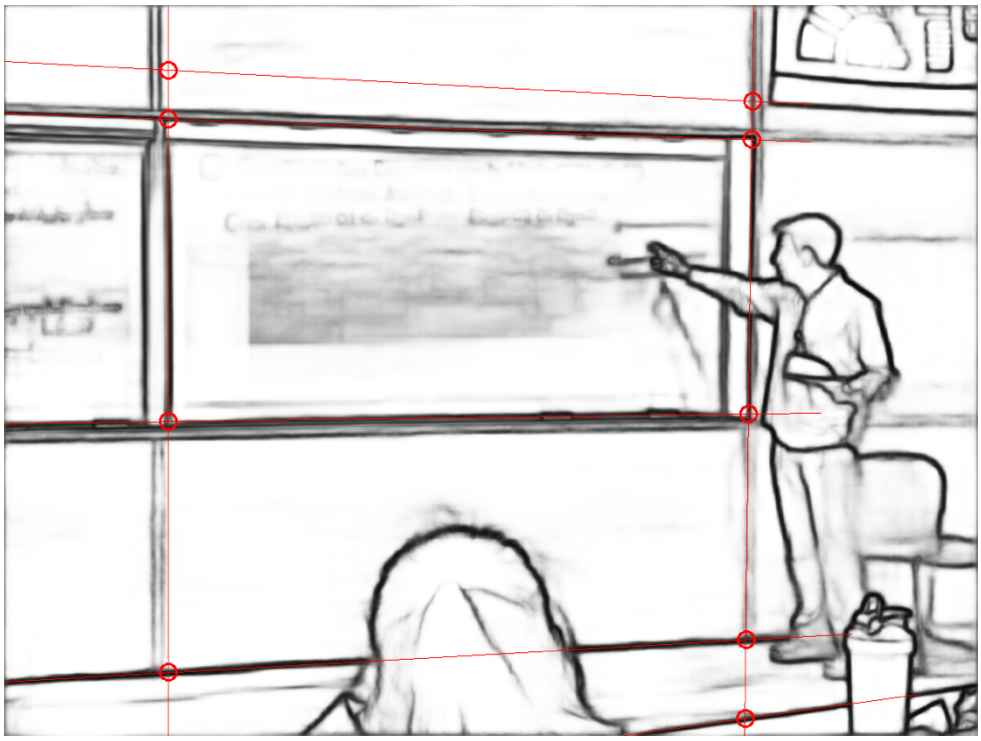


Figure 4.8 Eligible intersections image

4.1.5 Filter Possible Rectangles

(1) Implementation and Deployment

This step contains two parts: find possible rectangles and select most likely target area from these rectangles:

```
# find the combination contain only four lines and every line shows only two times
# these combinations are most likely rectangles
for combination in combination_array:
```

```
        isRectangle = True
        line_times = {}
        for point_index in combination:
            if points[point_index][0] not in line_times.keys():
                line_times[points[point_index][0]] = 1
            else:
                line_times[points[point_index][0]] += 1

            if points[point_index][1] not in line_times.keys():
                line_times[points[point_index][1]] = 1
            else:
                line_times[points[point_index][1]] += 1

        if len(line_times) == 4:
            for line in line_times:
                if line_times[line] != 2:
                    isRectangle = False
                    break
        else:
            isRectangle = False

        if isRectangle:
        rectangles.append(combination)
```

The following code is the function for calculating the black rate:

```
    def get_black_rate(x1, y1, x2, y2):
        min_x = int(min(x1, x2))
        max_x = int(max(x1, x2))
        min_y = int(min(y1, y2))
        max_y = int(max(y1, y2))
        black_number = 0
        length = np.sqrt(np.square(x1 - x2) + np.square(y1 - y2))
        if length < 50:
            return 0
        if max_x - min_x > max_y - min_y:
            for x in range(min_x, max_x):
                y = int((y2 - y1) / (x2 - x1) * (x - x1) + y1)
                if y < row_max and binary[y][x] == 0:
                    black_number += 1
        else:
            for y in range(min_y, max_y):
                x = int((x2 - x1) / (y2 - y1) * (y - y1) + x1)
                if y < row_max and binary[y][x] == 0:
                    black_number += 1
        black_rate = black_number / length

    return black_rate
```

Then evaluate the most likely target area from most possible rectangles:

```
    # black rate means the ratio of black points in one line
        # point1 connect point2 and point3
        # point2 and point3 also connect point4
        set1 = set(rectangle)
        set2 = set((firstPoint, secondPoint, thirdPoint))
        fourthPoint = (set1-set2).pop()
        fourthPointX = points[fourthPoint][2]
```

```
        fourthPointY = points[fourthPoint][3]
        blackrate1 = get_black_rate(firstPointX, firstPointY, secondPointX, secondPointY)
        blackrate2 = get_black_rate(firstPointX, firstPointY, thirdPointX, thirdPointY)
        blackrate3 = get_black_rate(fourthPointX, fourthPointY, secondPointX, secondPointY)
        blackrate4 = get_black_rate(fourthPointX, fourthPointY, thirdPointX, thirdPointY)
        # a rectangle black rate means the average of four lines' black rate
        blackrate = (blackrate1+blackrate2+blackrate3+blackrate4)/4.0

        if blackrate > max_blackrate:
            final_blackrate = [blackrate1, blackrate2, blackrate3,blackrate4]
            max_blackrate = blackrate
            final_rectangle = rectangle
        final_points = [(int(firstPointX), int(firstPointY)), (int(secondPointX), int(secondPointY)),
(int(thirdPointX), int(thirdPointY)), (int(fourthPointX), int(fourthPointY))]
```

(2) Result Display

After get the intersections, we can calculate all the possible rectangle is the image. The "Black Rate" will be used to select the most possible target area. As the Figure 4.9 shows, the target area can be shown after calculation and comparison.

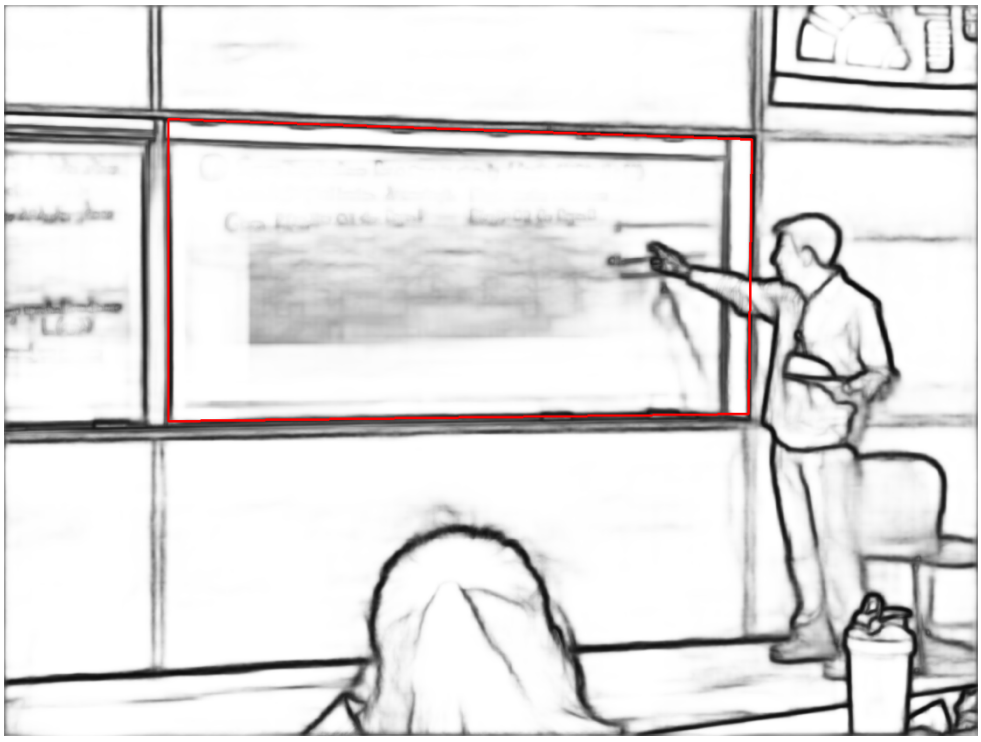


Figure 4.9 The final tarte area rectangle

4.1.6 Correction of Target Area

(1) Implementation and Deployment

The job of this step is to apply perspective transformation to the target area:

```
# get the perspective transform according to 4 points
def perspective_transform(img, points):
    # get ordered points
```

```
        rec = get_rectangle(points)
        (tl, tr, br, bl) = rec

        # calculate the new width of the image
        width_a = np.sqrt(((br[0] - bl[0]) ** 2) + ((br[1] - bl[1]) ** 2))
        width_b = np.sqrt(((tr[0] - tl[0]) ** 2) + ((tr[1] - tl[1]) ** 2))
        max_width = max(int(width_a), int(width_b))

        # calculate the new height of image
        height_a = np.sqrt(((tr[0] - br[0]) ** 2) + ((tr[1] - br[1]) ** 2))
        height_b = np.sqrt(((tl[0] - bl[0]) ** 2) + ((tl[1] - bl[1]) ** 2))
        max_height = max(int(height_a), int(height_b))
        dst = np.array([
            [0, 0],
            [max_width - 1, 0],
            [max_width - 1, max_height - 1],
            [0, max_height - 1]], dtype="float32")
        # compute the perspective transform matrix and then apply it
        matrix = cv2.getPerspectiveTransform(rec, dst)
        warped = cv2.warpPerspective(img, matrix, (max_width, max_height))
        # return the warped image
    return warped
```

(2) Result Display

The final step is used to rectify the final target area and show the better effect. As the Figure 4.10 shows, the final part will be transfer to a stander rectangle and show a better visual effect by perspective transform.

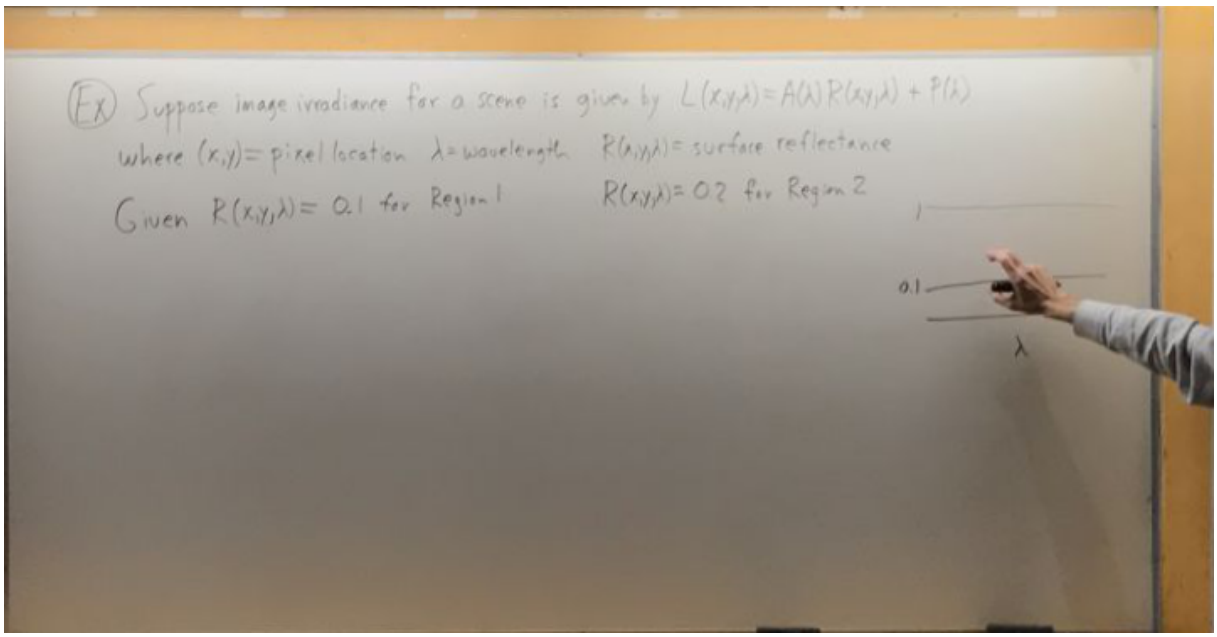


Figure 4.10 Final result by processing of all algorithms

4.2 Algorithm Evaluation

4.2.1 Environment

Our smart notes system is deployed on MAC OS with an 8 GB memory. The details about server configurations are as followings:

(1) Operation System: MAC OS

(2) Release version: Sierra 10.12.6

(3) GPU: None (Intel graphics doesn't support CUDA)

(4) Memory: 8 GB 1867 MHz

(5) CUDA version: None

(6) Python version: 2.7

4.2.2 Testing Result

The application direction of this system is to improve the user's learning and work efficiency. The system has tested these data and selected two representative pictures for display from these test results. Figure 4.11-4.16 show the process of finding the target area in the projection screen during the class. Figure 4.17-4.19 show the process of extracting data tests from the documents.

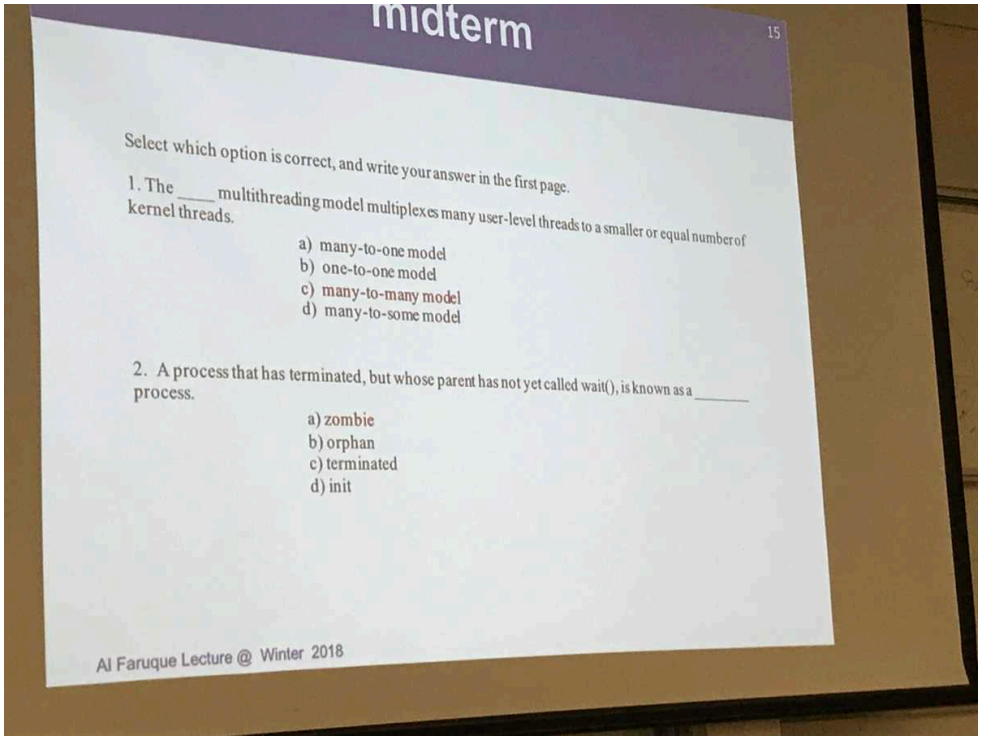


Figure 4.11 Projection screen image

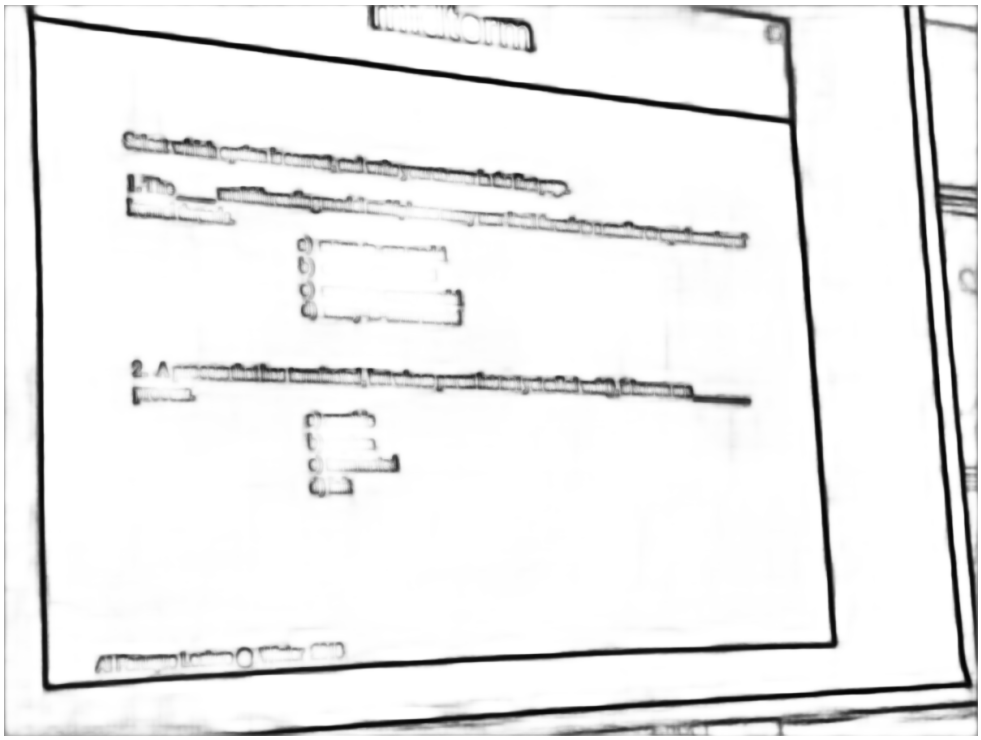


Figure 4.12 Projection screen image—RCF

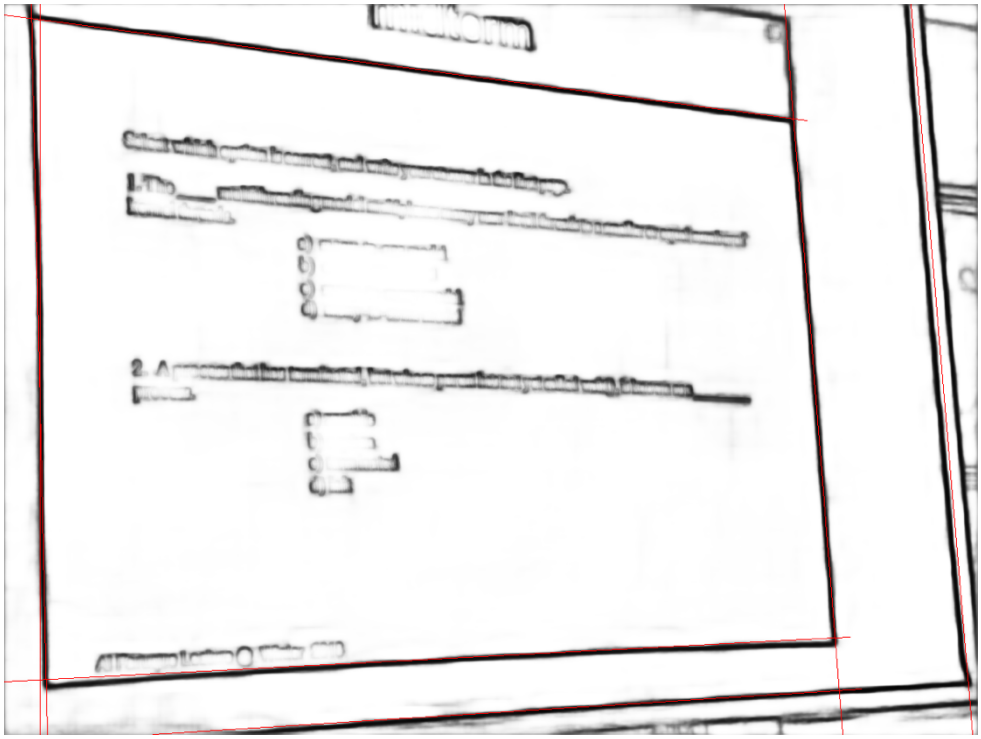


Figure 4.13 Projection screen image—optimized Hough Transform

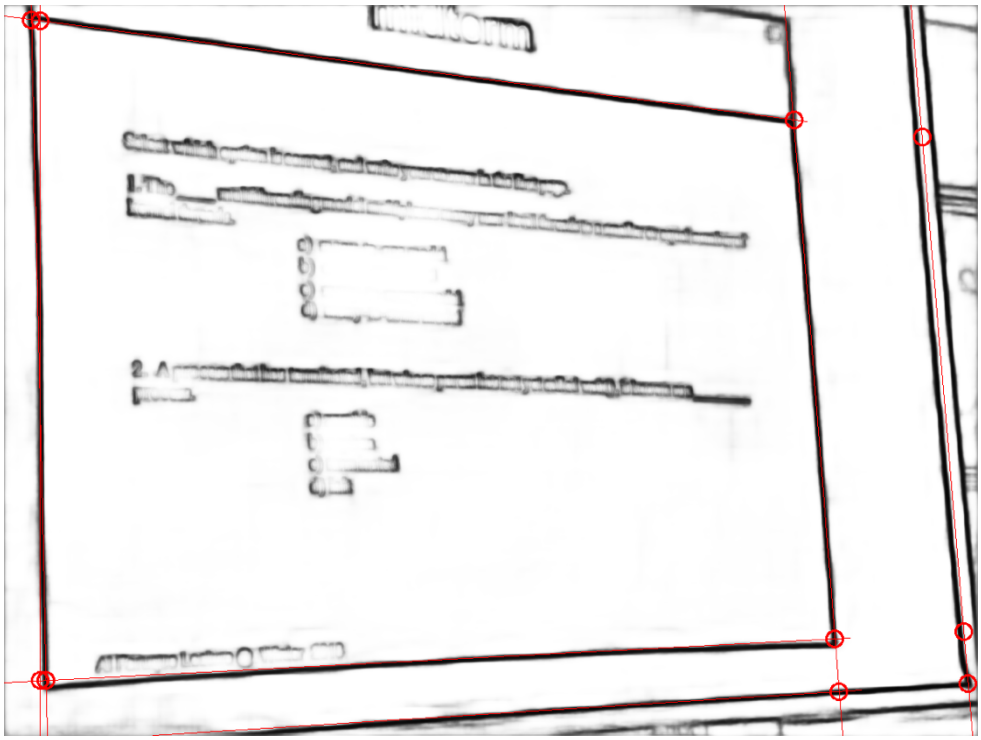


Figure 4.14 Projection screen image—calculating intersections

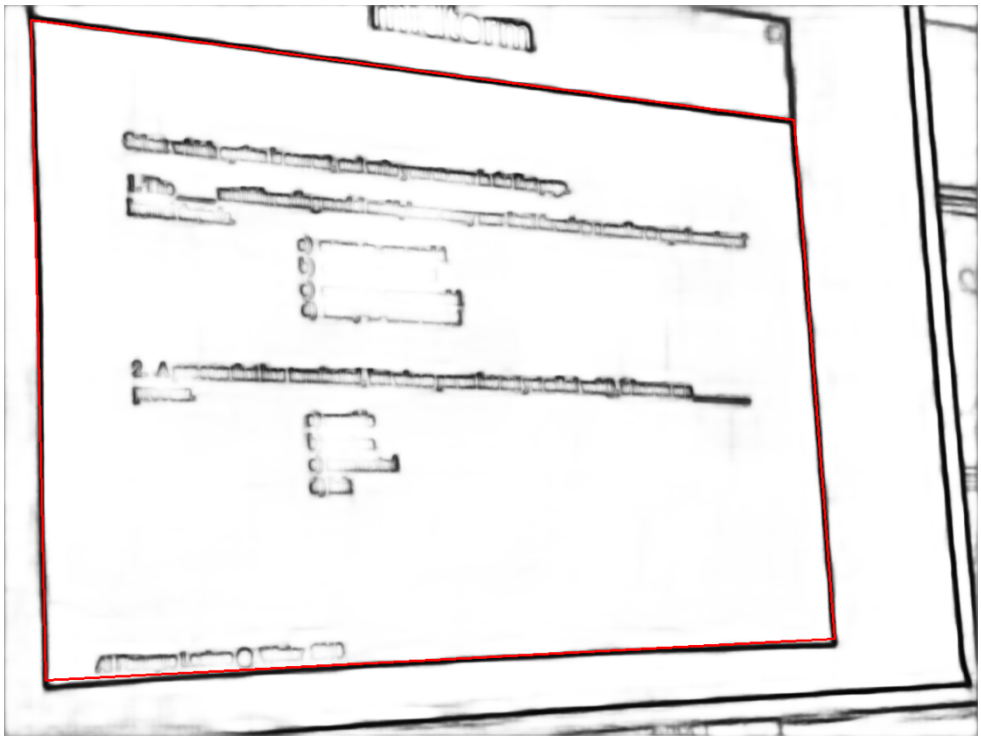


Figure 4.15 Projection screen image—target area

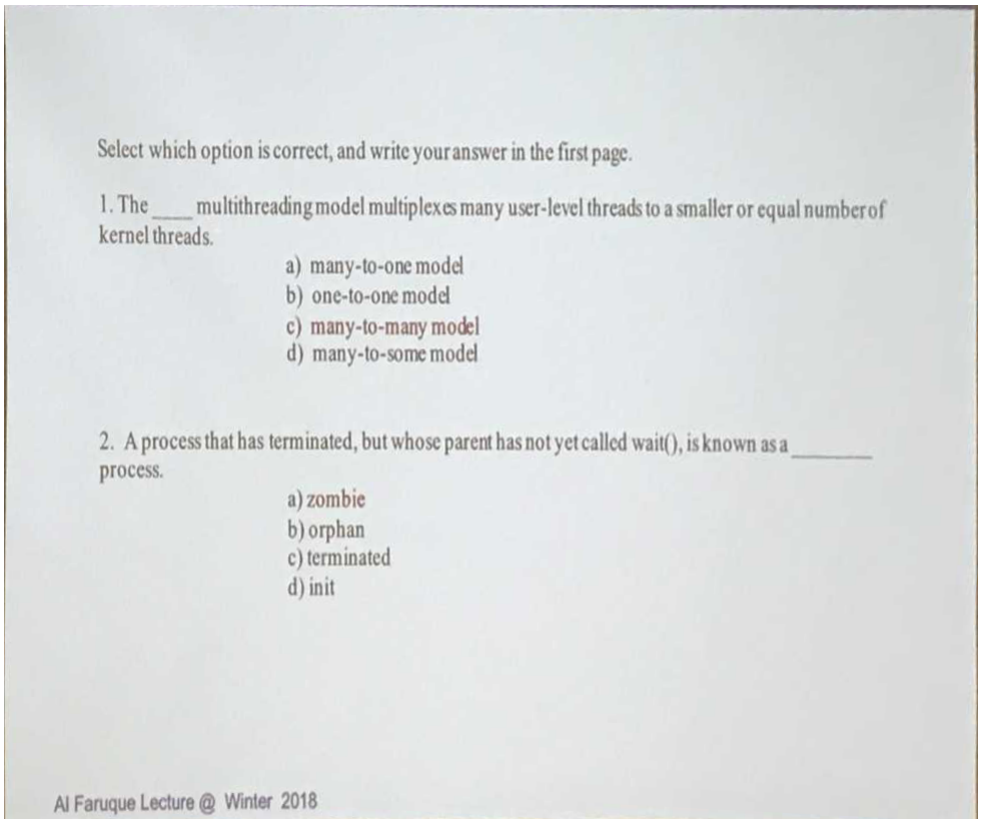


Figure 4.16 Projection screen image-perspective transformation—final result

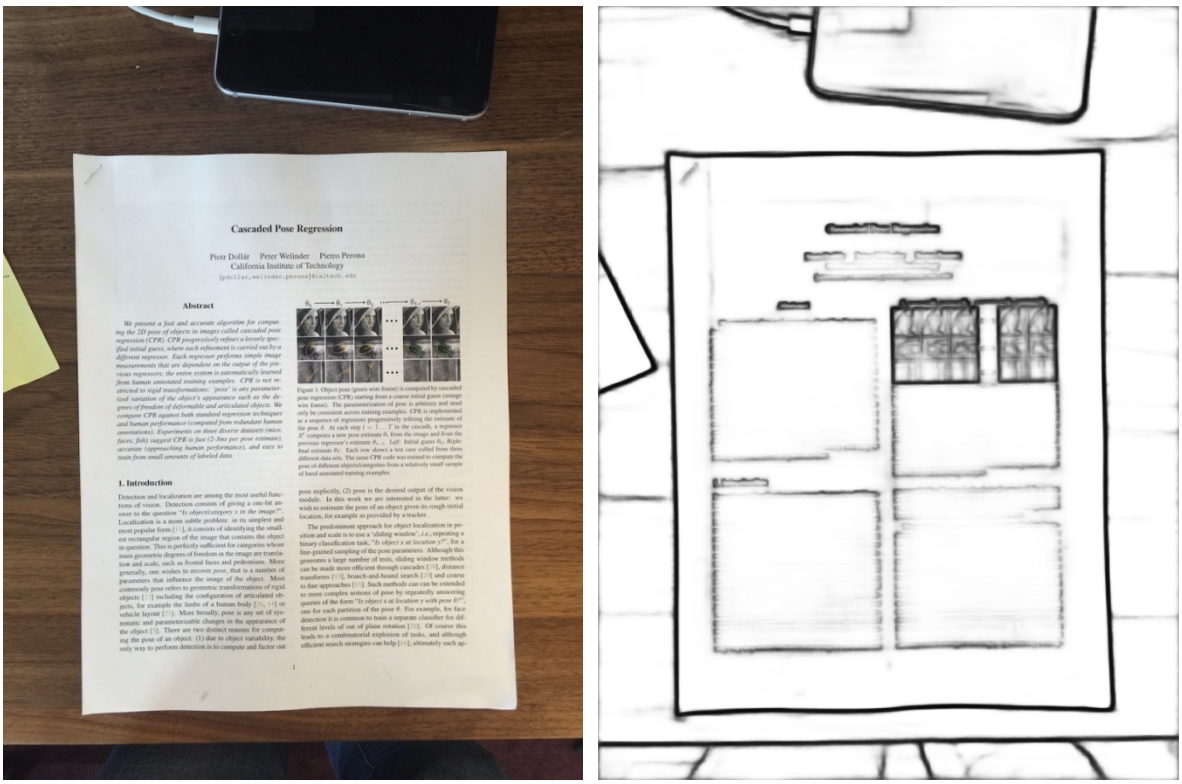


Figure 4.17 Document and image from RCF

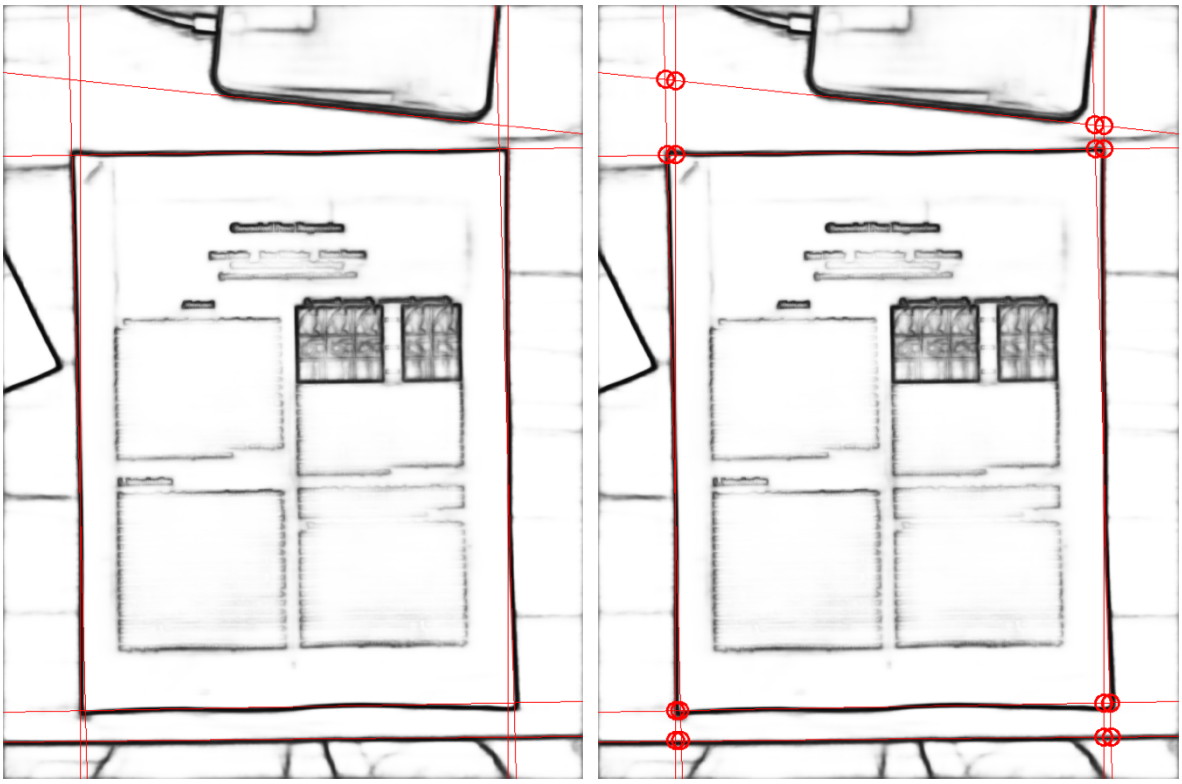
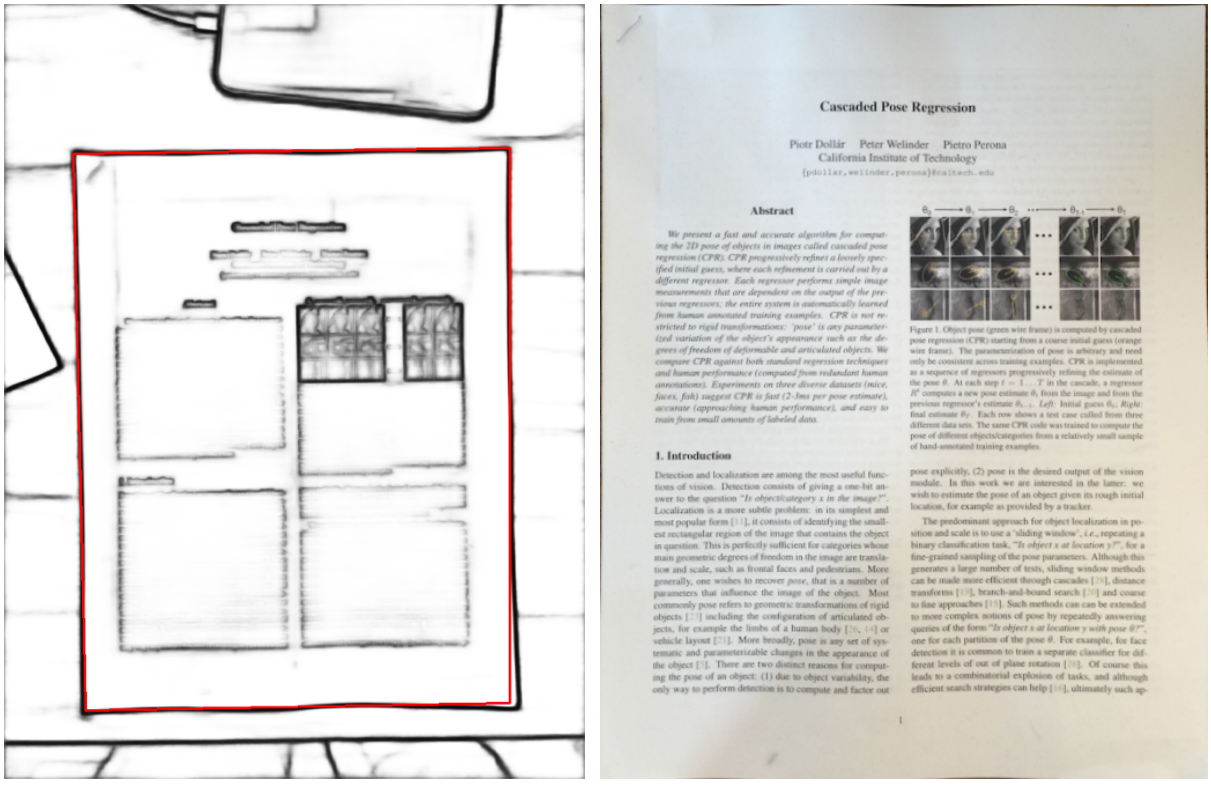


Figure 4.18 Hough transform and intersections images



Cascaded Pose Regression

Piotr Dollár Peter Welinder Pietro Perona
California Institute of Technology
{pdollar,welinder,perona}@caltech.edu

Abstract

We present a fast and accurate algorithm for computing the 2D pose of objects in images called cascaded pose regression (CPR). CPR progressively refines a loosely specified initial guess, where each refinement is carried out by a different regressor. Each regressor performs simple image measurements that are dependent on the output of the previous regressors; the entire system is automatically learned from human annotated training examples. CPR is not restricted to rigid transformations: 'pose' is any parameterized variation of the object's appearance such as the degrees of freedom of deformable and articulated objects. We compare CPR against both standard regression techniques and human performance (computed from redundant human annotations). Experiments on three diverse datasets (mice, faces, fish) suggest CPR is fast (2-3ms per pose estimate), accurate (approaching human performance), and easy to train from small amounts of labeled data.

1. Introduction

Detection and localization are among the most useful functions of vision. Detection consists of giving a one-bit answer to the question "*Is object/category x in the image?*". Localization is a more subtle problem: in its simplest and most popular form [11], it consists of identifying the smallest rectangular region of the image that contains the object in question. This is perfectly sufficient for categories whose main geometric degrees of freedom in the image are translation and scale, such as frontal faces and pedestrians. More generally, one wishes to recover *pose*, that is a number of parameters that influence the image of the object. Most commonly pose refers to geometric transformations of rigid objects [23] including the configuration of articulated objects, for example the limbs of a human body [26, 14] or vehicle layout [21]. More broadly, pose is any set of systematic and parameterizable changes in the appearance of the object [5]. There are two distinct reasons for computing the pose of an object: (1) due to object variability, the only way to perform detection is to compute and factor out pose explicitly, (2) pose is the desired output of the vision module. In this work we are interested in the latter: we wish to estimate the pose of an object given its rough initial location, for example as provided by a tracker.

The predominant approach for object localization in position and scale is to use a 'sliding window', *i.e.*, repeating a binary classification task, "*Is object x at location y?*", for a fine-grained sampling of the pose parameters. Although this generates a large number of tests, sliding window methods can be made more efficient through cascades [30], distance transforms [13], branch-and-bound search [20] and coarse to fine approaches [13]. Such methods can can be extended to more complex notions of pose by repeatedly answering queries of the form "*Is object x at location y with pose θ?*", one for each partition of the pose θ. For example, for face detection it is common to train a separate classifier for different levels of out of plane rotation [20]. Of course this leads to a combinatorial explosion of tasks, and although efficient search strategies can help [16], ultimately such ap-

Figure 1. Object pose (green wire frame) is computed by cascaded pose regression (CPR) starting from a course initial guess (orange wire frame). The parameterization of pose is arbitrary and need only be consistent across training examples. CPR is implemented as a sequence of regressors progressively refining the estimate of the pose θ. At each step $t = 1 \ldots T$ in the cascade, a regressor R^t computes a new pose estimate θ_t from the image and from the previous regressor's estimate θ_{t-1}. *Left:* Initial guess θ_0; *Right:* final estimate θ_T. Each row shows a test case culled from three different data sets. The same CPR code was trained to compute the pose of different objects/categories from a relatively small sample of hand-annotated training examples.

Figure 4.19 Target area and perspective transformation

4.2.3 Efficiency Analysis

Smart notes system is composed of Edge Detection Algorithm and Target Area Locating Algorithm. Compared to Edge Detection Algorithm, the running time of Target Area Locating is negligible. So we mainly talk about the efficiency of Edge Detection Algorithm.

Compared to HED, RCF has a higher performance. RCF is 1.1% higher than on the boundary task in ODS F-measure and 1.4%. As for edge detection, RCF is 0.9% higher than HED. What's more, HED's fluctuation is bigger than RCF's fluctuation which means RCF is more robust to tackle with different genres of images.

4.2.4 Data Set

There are mainly two data sets for the edge detection algorithms. BSDS500 and NYUD.

BSDS500 [20] means Berkeley Segmentation Data Set and Benchmarks 500. The database contains 200 training charts, 200 side views and 100 inspection charts. Ground truth is artificially identified. It has database image id as a unit and is saved as a mat format file. A file contains tag information of multiple taggers. Because the database includes contour and segmentation information, we can use this dataset to evaluate the effectiveness of the edge detection algorithm. Figure 4.11 shows some edge detection algorithms on BSDS500.

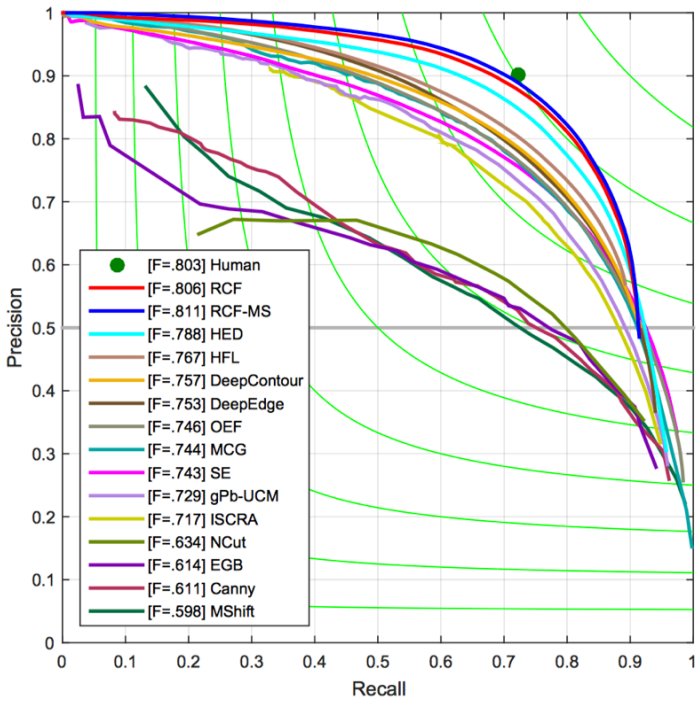


Figure 4.11 Evaluating result of RCF on BSDS500 [6]

The NYUD [21] dataset consists of approximately 1500 densely-labeled pictures. Many edge detection algorithms now use this dataset for evaluation. Professor Ming-Ming Chen divided the NYUD dataset into 381 training pictures, 414 verification pictures, and 654 test

pictures. Use this architecture to evaluate RCF models. Figure 4.12 shows edge detection algorithm performance on NYUD.

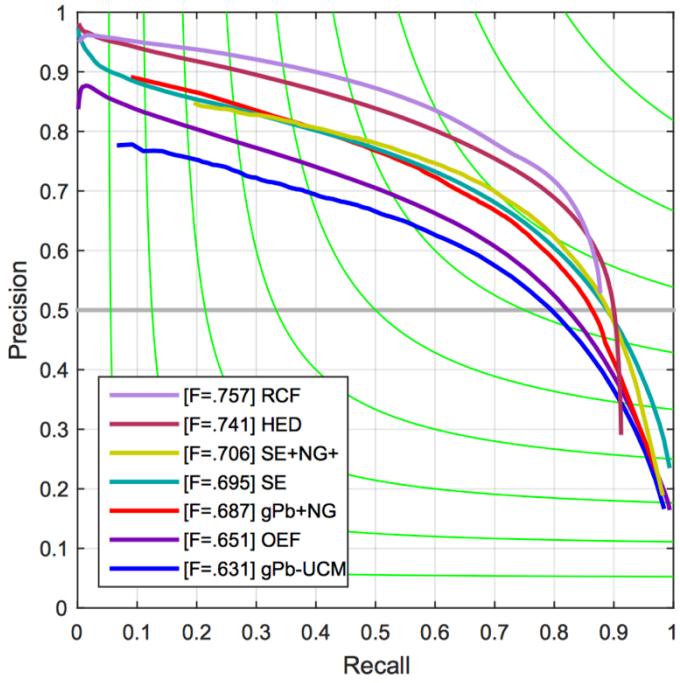


Figure 4.12 Evaluating result of RCF on NYUD [6]

From Figure 4.11 and 4.12 we can know that RCF shows good performance on both BSDS500 and NYUD [6] [22] .

4.3 Summary

In this chapter, the implementation of each algorithm is described in detail and the corresponding code is given. All the models and code are based on Caffe, OpenCV and Python2.7. In addition to the code, the corresponding results of the code are given. From the results it can be seen that the model works well.

In addition to the code and the resulting picture, this chapter discusses the efficiency of the RCF model. In evaluating the efficiency of the edge detection algorithm, two general data sets BSDS500 and NYUD were used. The test results show that the RCF model has good efficiency, and the accuracy rate is better than human performance on a certain data set. Prove the feasibility of the entire model and strong application.

Chapter 5 Conclusion and Prospect

5.1 Conclusion

In this thesis, We have developed a smart note recognition algorithm. The algorithm is applied to people's daily work, study and life, helping people to quickly and accurately identify and extract key information, such as documents and notes. This algorithm consists of two parts, the edge recognition algorithm and the target area localization algorithm.

As for edge recognition algorithm, we have adopted corresponding improvements and achieved good results. Most of the document detection applications on the market today use the HED model, which is a good model based on deep learning. However, it does not make full use of the information in each layer of the network. The RCF model has been improved on this point. Not only the operating time not increases significantly, but the results have been greatly improved. Therefore, we use the RCF model as the edge detection model of the system.

As for target area localization algorithm, a series of image mathematics models were used to find the target area, including Binarization Image, Hough Transform, Intersections Finding, Rectangle Searching and Perspective Transforms. Especially for Hough Transform and Rectangle Searching, We proposed our own improvement method. After obtaining the results of the ordinary Hough transform, we have also designed a straight line classification algorithm. This algorithm can greatly reduce noise lines and improve the efficiency of subsequent work. In the rectangular search process, we proposed our own definition of "black rate" and used this feature to find the most likely target area. The results show that using this feature of the algorithm can greatly improve the accuracy of the results.

5.2 Prospect

There are still some work needing to do in the future. First, we can make some changes to the network and try to get better results. Second, the training set of the RCF model includes various objects in life. In the future, we intend to use more pictures only contain documents or notes to train the model. This is more targeted and results can be even better. Then, the algorithm to find all the rectangles can be further improved and the running time can be further reduced.

References

1. Ganin Y, Lempitsky V. $$N^4$$-Fields: Neural Network Nearest Neighbor Fields for Image Transforms[J]. 2014, 9004:536-551.
2. Bertasius G, Shi J, Torresani L. DeepEdge: A multi-scale bifurcated deep network for top-down contour detection[J]. 2014, 52(3):4380-4389.
3. Shen W, Wang X, Wang Y, et al. DeepContour: A deep convolutional feature learned by positive-sharing loss for contour detection[C]// Computer Vision and Pattern Recognition. IEEE, 2015:3982-3991.
4. Bertasius G, Shi J, Torresani L. High-for-Low and Low-for-High: Efficient Boundary Detection from Deep Object Features and Its Applications to High-Level Vision[J]. 2015, 8(1):504-512.
5. Xie S, Tu Z. Holistically-Nested Edge Detection[J]. International Journal of Computer Vision, 2015, 125(1-3):3-18.
6. Liu Y, Cheng M M, Hu X, et al. Richer Convolutional Features for Edge Detection[J]. 2016:5872-5881.
7. O'Shea K, Nash R. An Introduction to Convolutional Neural Networks[J]. Computer Science, 2015.
8. Quan T M, Hilderbrand D G C, Jeong W K. FusionNet: A deep fully residual convolutional neural network for image segmentation in connectomics[J]. 2016.
9. Ying X. Fast and Accurate Document Detection for Scanning [EB/OL]. https://blogs.dropbox.com/tech/2016/08/fast-and-accurate-document-detection-for-scanning/#disqus_thread.
10. Girshick R. Fast r-cnn[C]//Proceedings of the IEEE International Conference on Computer Vision. 2015: 1440-1448.
11. Xu L, Ren J S J, Liu C, et al. Deep convolutional neural network for image deconvolution[C]// International Conference on Neural Information Processing Systems. MIT Press, 2014:1790-1798.
12. Shu H, Document Scanning: Practice of Deep Neural Networks on the Mobile Side [EB/OL]. http://techblog.youdao.com/?p=1237
13. Ilievski I, Feng J. A Simple Loss Function for Improving the Convergence and Accuracy of Visual Question Answering Models[J]. 2017.
14. Hu K, Zhang Z, Niu X, et al. Retinal vessel segmentation of color fundus images using multiscale convolutional neural network with an improved cross-entropy loss function[J].

Neurocomputing, 2018.

15. Pao D C W, Li H F, Jayakumar R. Shapes recognition using the straight line Hough transform: theory and generalization[J]. Pattern Analysis & Machine Intelligence IEEE Transactions on, 1992, 14(11):1076-1089.
16. Yan N. Discussion About Perspective Transform[J]. Journal of Computer Aided Design & Computer Graphics, 2001.
17. Niola V, Rossi C, Savino S. Perspective Transform and Vision System for Robotic Applications[J]. Wseas Transactions on Systems, 2006, 5(4).
18. Gobert J. Geometric method of transforming a two-dimensional image: EP, US 8666193 B2[P]. 2014.
19. Jia, Yangqing, Shelhamer, et al. Caffe: Convolutional Architecture for Fast Feature Embedding[J]. 2014:675-678.
20. Arbeláez P, Maire M, Fowlkes C, et al. Contour detection and hierarchical image segmentation.[J]. IEEE Transactions on Pattern Analysis & Machine Intelligence, 2011, 33(5):898-916.
21. Silberman N, Hoiem D, Kohli P, et al. Indoor Segmentation and Support Inference from RGBD Images[C]// European Conference on Computer Vision. Springer, Berlin, Heidelberg, 2012:746-760.
22. D C W, Li H F, Jayakumar R. Shapes recognition using the straight line Hough transform: theory and generalization[J]. Pattern Analysis & Machine Intelligence IEEE Transactions on, 1992, 14(11):1076-1089.

Acknowledgement

This graduation project is finished during my studying in University of California, Irvine. In this project, I encountered a lot of difficulties while learning a lot. First of all, I would like to thank Professor Gao Tianhan and Senior Engineer Gao Hongjie. Especially for Professor Gao Tianhan. Even if we are far away, he is very concerned about my studies and research progress. He helped me to select challenging topics and gave me instructions to let me know how to proceed with the entire project. And he actively pointed out my mistakes and put forward constructive opinions.

Thanks to Professor Glenn Healy. He is my mentor at University of California, Irvine. I followed him in the course of machine vision and guided me through some machine vision studies. These knowledge laid a solid foundation for my graduation project.

Thanks to my parents and girlfriend, They gave me spiritual support. In the heavy learning and research, they gave me the driving force to persevere and let me believe that I can overcome the difficulties encountered.

Last but not the least, thanks to my home university, Northeastern University. It provides me with an environment for learning and opportunities for development. I have learned a lot of erudite teachers and intimate classmates.