Rotated Human Detection on Fisheye Cameras

Wyatt Mayor
University of Illinois Urbana-Champaign
Champaign, IL
wpmayor2@illinois.edu

Abstract

This project tackles the challenges of rotated bounding box prediction and overhead fisheye object detection. I train RetinaNet to produce rotated bounding box predictions on a fisheye image. This is achieved by training on the COPODF dataset [1]. This dataset is a collection of fisheye overview frames with rotated bounding box ground truth annotations. After only 10k iterations this model can produce promising results.

1. Introduction

Security cameras are littered throughout grocery stores, airports, buses, and almost any other place that most people visit. These security cameras offer a wide range of benefits that are not taken advantage of by most systems today. These security cameras can capture information about the flow of customers, the runway traffic of an airport, the interest of people in an art museum, and much more data that can be used to increase productivity, improve traffic flow, or increase sales. Creating technology that can collect this data would be useful in a multitude of industries. There is technology that exists but falls short. These technologies suffer from two problems which are axis-aligned bounding boxes and poor performance on fisheye cameras. In this project, I addressed these problems as they will lead to a profound effect on the larger goal of traffic flow data collection.

1.1. Rotated Bounding Boxes

Axis-aligned bounding boxes in the object detection task have been at the forefront of object detection. Most projects utilize axis aligned bounding boxes in object detection as it is easy to train and provides good results. However, For collecting traffic flow data, the orientation of the human or object could provide valuable insight into what is going on. This is especially true for fisheye cameras. For example, if a person was standing in a main aisle in the grocery store and the person was skewed to the left on the fisheye camera, It would be difficult to extract whether the person is in an aisle or just walking by it. This is because axis-aligned bounding boxes are often way larger than the specific object if the person is titled, which is a common occurrence from a fisheye perspective. This issue could lead to corrupted and unusable data. rotated bounding boxes allow for more precise boxes to derive data from and prevent as much data corruption as possible.

1.2. Fisheye Cameras

Many security cameras today are designed to capture a 360-degree perspective. These make up a large portion of security cameras which is why this software must work well on these cameras. There are a few challenges that come with these images. Objects are often skewed, disproportional, or not displayed fully as most of these cameras are from an overhead view. Another challenge is a model that is not trained on these types of images, which include a black border around a circular image, might miss classifying the black border. Training on a specific dataset that represents these cameras improves the data collection ability and accuracy.

Figure 1. Examples of fisheye camera frames.

2. Approach

Originally I wanted to train a YoLo model on the fisheye dataset. However, This had already been done by the creators of the dataset. I decided to alter the RetinaNet model to train on the fisheye dataset. This proved to be a greater challenge than I expected. There are many things that I had

to consider when I waas converting this model to a rotated bounding box prediction scheme. I had to alter the architecture, rework the anchor generation, implement new loss functions, implement a new dataset class and dataset utilities, and rework any previous helper functions to handle the new data structure.

2.1. Dataset

In this project, I trained on the CEPODF dataset. This dataset is a people detection dataset that includes frames from an overhead fisheye camera. This dataset was released in 2020 by a team at Boston University. This dataset consists of multiple categories which all contain different lighting filters and max number of people. I trained on the Lunch1 subset of data. This data consists of 1200, 2048 x 2048, frames with a max of 11 people in the frame and decent lighting. This subset of data seemed like a good place to start. The other subsets of data contained more people or lighting filters. This data follows the same annotation scheme as the popular coco dataset except the bounding boxes are represented as [centerx,centery,width,height,degree(clockwise)]. This dataset also has a fifth degree dimension. This dimension is added because this dataset was designed for rotated bounding box prediction. This dataset allows me to address the two problems that I stated in the introduction.

2.2. RetinaNet Architecture

The backbone of my model follows the same backbone as the RetinaNet in mp3 except for a few changes. I altered the classification head to output a single output for each anchor box. The classification head essentially predicts whether the anchor bounding box contains the background or a person. Then, I altered the number of anchors that are created per anchor point. Instead of nine anchors, The model generates twenty-four anchors per anchor point. Finally, The bounding box head outputs an extra dimension that includes the angle.

2.3. Anchors generation

Rotated anchors come with their own set of challenges. There is no rotated anchor generation function in PyTorch. I altered my previous implementation of anchor generation. This altered version includes a fifth dimension that holds the degree of the bounding box. Unlike axis-aligned bounding bounding boxes, the x1,y1,x2,y2 points must have the angle applied during the forward pass. During this process, I had to change this function to account for the correct rotation. Originally, I had rotated the bounding box around the origin, using the bottom left point. This type of rotation changes the center coordinates to the anchor box which would make it extremely difficult for the model to predict the angle. I was able to get substantially better results by rotating the anchor's center coordinates around the origin.

Another challenge that arises when using rotated anchors is the exponential growth of anchors. Axis-aligned anchors only deal with sizes and aspect ratios, this means the Number of Anchors = Sizes * Aspect Ratios whereas for rotated anchors the Number of Anchors = Sizes * Aspect Ratios * Angles. Due to this exponential growth, I chose four angles, three sizes, and two aspect ratios. This resulted in twenty-four anchor boxes per anchor point. I experimented with these numbers and this one allowed me to get promising results with only taking around three seconds per iteration. I tried other combos that covered more angle points, but I quickly realized that more than thirty bounding boxes significantly slowed down the already slow training process. This was an interesting trade-off that I had to navigate as some combinations would take more than ten Seconds an iteration. This was impractical for the computation power that I had access to.

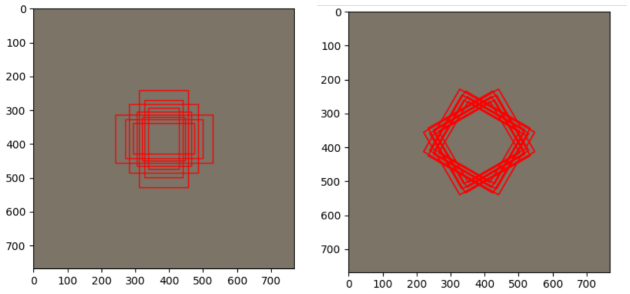


Figure 2. Axis Aligned Anchors (left) rotated Anchors (right)

2.4. Loss Function

The CEPODF dataset only contains one category which is a person. As stated previously the model is structured to output a single binary classification that represents background or person. I implemented the commonly used binary cross entropy loss function for the classification head. The bounding box loss function is more complicated. There are a few types of loss functions that could be used such as IoU loss, GIoU loss [2], DIoU [3] loss, and regression loss. I started by reusing the MAE loss function that was used in mp3. I quickly realized that MAE was not suitable for rotated bounding box prediction because MAE was very sensitive to outliers and did not produce any usable bounding boxes. I switched to an IoU loss, This was very slow and added a lot of time to training. This is because of the slow IoU calculation that will be discussed in a future section. At this point, I had read about Smooth L1 Loss, or Huber Loss, which is a regression loss that is less sensitive to outliers. Smooth L1 loss performed better than MAE and produced more promising bounding boxes.

$$L_\delta = \begin{cases} \frac{1}{2}(y-\hat{y})^2/\delta & if\ |y-\hat{y}| < \delta \\ |y-\hat{y}| - \frac{1}{2}\delta & otherwise \end{cases}$$

Figure 3. Smooth L1 Loss $\delta = 1$

2.5. Dataset utilities

At the start of this project, I had to implement the dataset class for CEPODF. There was not one provided in the CEPODF helper function GitHub. This was fairly simple, I worked from the coco dataset class and cur-rated it to the CEPODF dataset. I utilized the helper functions that were provided with the dataset such as the draw function that applied the rotated bounding boxes to images and their point conversion function. I also utilized their evaluation class, the class followed very closely to the coco evaluation class. Finally, I reused the normalization, unnormalization, and resizer functions. I decided to resize the images from 2048 x 2048 to 768 x 768, this causes a loss of information but is necessary for reducing training time. I experimented with this value to see what offers the best results at the most reasonable training time. 1024 x 1024 images would take around 10 seconds an iteration whereas 768 x 768 would only take around 2.5 seconds an iteration.

2.6. IoU Calculation

Calculating the IoU between two rotated bounding boxes can be a tricky task. It requires more computation and cannot be optimally vectorized like the IoU calculation of axis-aligned bounding boxes. This is due to the fact that the intersection of axis-aligned bounding boxes will always have the area of width * height, it is always square or rectangle. Rotated bounding boxes can have an intersection of an arbitrary shape which makes the intersection calculation a much more complex task. The CEPODF GitHub offers an IoU calculation function that uses masking and performs length encoding. The problem with their method is that it is performed on the CPU which requires coping tensors on and off the GPU and CPU. This copying can add a substantial amount of time for large tensors. I started by implementing another IoU function using the polygon library to try and reduce the calculation time. This library provides a function to calculate rotated bounding box IoUs using the four corners. This approach is simple as it is a built-in function. However, The major downfall of this IoU calculation is that it can't be vectorized. This makes the calculation very slow, making it almost impossible to train the model. I was not able to find any other IoU functions that could be vectorized and run on the GPU. I defaulted to using the given IoU calculation function from the CEPODF GitHub. I had to alter the function because the original implementation did not support tensors. This function is slow but it performs better than any method I was able to find.

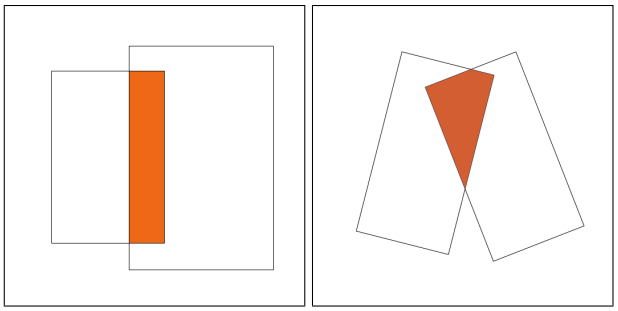


Figure 4. Axis Aligned Intersection Example (left) rotated Intersection Example (right)

2.7. Data Shapes

There are multiple ways that I could have worked with anchor generation. I decided to format the anchors in an x1,y1,x2,y2,a format. This decision required me to convert the anchor and ground truth shapes between the two formats [x1,y1,x2,y2,a] and [centerx,centery,width,height,degree(clockwise)]. I had to make multiple conversion functions that would allow me to work between these two functions. I regret not formatting the anchors in the same shape as the ground truth or reading the ground truth in the same format as the anchors. This would have saved me a lot of time throughout the process, I ran into a lot of errors and bad predictions because of incompatible shapes.

2.8. Converting Previous Functions

I worked off some of the functions from mp3. This was to reduce the time it would take to implement all of the functions from scratch as I am working on this by myself. In the end, I am not sure if it saved me much time because almost all of the helper functions had to be rewritten. I converted all of the functions to be able to handle angle calculations. I also reworked the function to allow it to handle the new shapes from the output of the classification head. Finally, I had to rework the functions so the input shapes formatting matched. Throughout the code, the anchors go from [x1,y1,x2,y2,a] to [cx,cy,w,h,a]. The IoU calculation requires both the ground truth and anchors to be in [cx,cy,w,h,a] format, I had to account for this and make sure they got converted back.

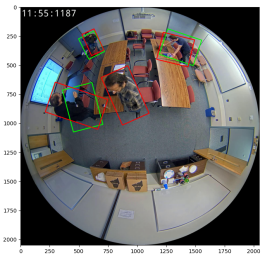
2.9. Training

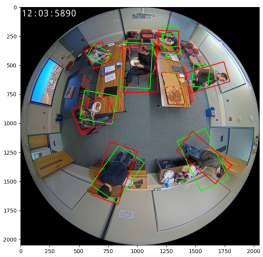
Training times had the biggest impact on my final results. I was able to train the model around 500 epochs every 30 minutes by the end of the project. As discussed previously, rotated bounding box prediction is a computationally expensive task. I was not expecting the training time to take so long. In mp3, I was able to train 120k epoch in 2 hours on my 3080. In this model, I was only able to train 15000 epochs in 15 hours. The slow training is mainly due to the

complexity of the IoU calculation. Unfortunately, there is not much I could do about this. I ran the PyTorch bottleneck profiler and the IoU calculation was the only significant time bottleneck. I researched IoU functions that could be run in parallel on the GPU but there weren't any available for rotated bounding boxes. The only methods that were out there were CPU based which added time by switching between GPU and CPU or IoU calculation that could not be vectorized such as the method using the polygon library. The result of these long training times prevented me from doing an in-depth hyper parameter study. For the final model, I used the hyper parameters that we used in mp3.

3. Results

The Results were promising for the number of epochs that I could train with my computation bottleneck. I trained the final model for 10k epochs which took around 10 hours to complete. In Figure 5, The green represents the 10k iteration results and the red is the ground truth. From a visual perspective, it seems to be converging fairly well early in the training process. This trend is also supported by the bounding box loss. The loss function is still decreasing. It would be interesting to see how good this model could get at 50k-100k iterations. In mp3, The model didn't hit the optimal model path until 75k and it was an easier prediction task. I would assume that if I were able to train this model I could see much better results. I also analyzed the classification loss. This loss fluctuates more in nature than the bounding box loss. However, on the loss plot, most of the points are to the bottom which means it is still on a downward trajectory. The goal of this project was to implement a model that could predict bounding boxes. I am not as concerned with classification loss as there are only two classes background and person.





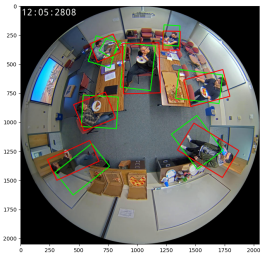
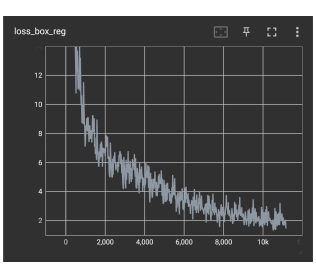


Figure 5. 10k iterations (Green) vs Ground Truth (Red)



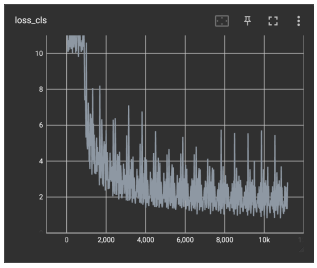
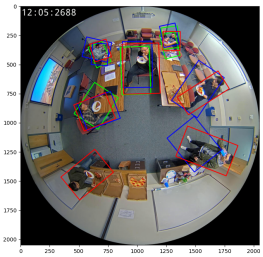


Figure 6. Loss Functions

3.1. Iteration Comparison

I wanted to compare earlier results to the 10k iteration results to make sure they were getting more accurate. It seems that there is definitely an improvement from 5k to 10k iterations. I noticed that the biggest improvement so far is that it is able to predict more bounding boxes. Some of the scenes can have up to 11 people and they are fairly small bounding boxes. As the iterations went farther in the training more bounding boxes were predicted as seen in Figure 7.



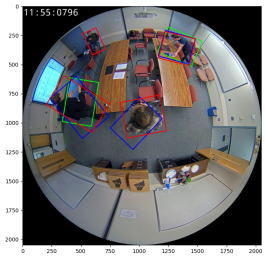


Figure 7. 10k iterations (Blue) vs 5k iterations (Green) vs Ground Truth (Red)

3.2. Average Precision

Finally, The model was able to achieve a .15 Average Precision score. This is low but the model is only trained to 10k iterations. The AP plot in Figure 8 shows that it is growing. I think with the insight from the loss functions and the improvement of results this would likely continue to grow.

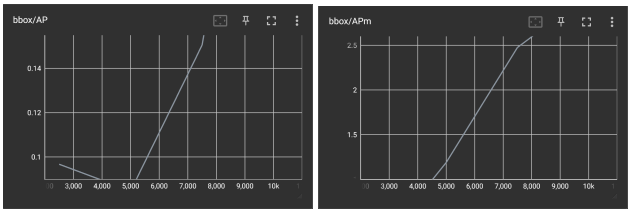


Figure 8. Bounding Box AP (left) Bounding Box APm (right)

4. Discussion and Conclusion

I learned some interesting things about object detection from this project. I have minimal experience in object detection especially in rotated bounding box detection. This is a less common approach to object detection. I found very little literature on rotated bounding box detection compared to axis alignment. I can understand why after training one of these models. This type of detection is tedious and hard to train. The computations are also much more complex than axis aligned. I am surprised there is not more work in this domain at the same time. I think that these types of models can produce much more accurate and form-fitting bounding boxes. There could be many uses for these models and improve some of the uses of axis-aligned projects.

4.1. Pitfalls and Techniques

There are many techniques and pitfalls that I will utilize and avoid in the future because of this project. I think one of the main lessons I learned from this project is that you should not overestimate the amount of computational resources that are required. Going into this project I was under the assumption that I would have no problem with the computational resources required. I have a 3080 GPU, which has allowed me to do object detection in the past but not with the rotated bounding boxes. This required much more computational resources than I had available. This project gave me a good understanding of what you should expect for computation costs when working with these types of models.

I also learned through this project that I should keep the shapes between the ground truth and anchors consistent in the object detection pipeline. I worked with multiple shapes in this project. This is because i did not read in the ground truth shapes to match the anchors. This caused a lot of errors and debugging time that could have been avoided. I also had to implement multiple conversion functions that could be used within the pipeline which was unnecessary. For future projects, I will know how to avoid this pitfall.

4.2. Conclusion

Overall, This project was a success. I was a little ambitious in my project proposal. However, my minimum goal was to train a model on the CEPODF dataset. I didn't really know what this entailed with the rotated bounding boxes. It was more complicated than I originally planned. However, I was able to train the model and output promising results. I would say I definitely met my minimum goal. I also learned a lot about object detection and good practices while working on this project which I can utilize in future object detection endeavors. This project was meant to be the start of a larger traffic flow data collection project. I believe that this project laid a good foundation for that future endeavor.

5. Statement of Contribution

I worked on this project by myself. All work and research that went into it was done by me.

6. Video Report

https://youtu.be/8TY4BRddonw

7. Code References

https://github.com/duanzhiihao/CEPDOF_tools

References

[1] Zhihao Duan, M. Ozan Tezcan, Hayato Nakamura, Prakash Ishwar, and Janusz Konrad. Rapid: Rotation-aware people detection in overhead fisheye images. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 2700–2709, 2020. 1

[2] Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, and Silvio Savarese. Generalized intersection over union: A metric and a loss for bounding box regression. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 658–666, 2019. 2

[3] Zhaohui Zheng, Ping Wang, Wei Liu, Jinze Li, Rongguang Ye, and Dongwei Ren. Distance-iou loss: Faster and better learning for bounding box regression. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(07):12993–13000, Apr. 2020. 2