Design of practical computer vision system with real-time object detection capability

by

Guanyu Chen

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2024

Author's Declaration

Abstract

Computer vision nowadays relies heavily on machine learning techniques to interpret useful information from images or videos. Object detection is one such computer vision technique for identifying and locating objects in images. This type of application is of great interest for its potential use in various fields including product inspection, analysis, security, etc.

As another important technique in computer vision, object recognition for identifying objects in images has been accomplished earlier. Classic models including LeNet and VGG16 have already adopt CNN-like architectures. In comparison, an object detection model would not only identify objects, but also label each detected object with a bounding box. Provided ground truth labels about both object class and bounding box coordinates, object detection models can be trained regularly for making both predictions. Certain families of object detection models are listed as follows: In R-CNN, the Region Proposal Network (RPN) produces region proposals, corresponding to rectangular regions in the image in which targeting object is possibly present. YOLO divides the input image into grids and predicts the bounding box and class confidence simultaneously for each grid. SSD is a similar model to YOLO but has better accuracy by using features at different scales. As a result of improved hardware performance and innovative network architecture in recent years, real-time object detection has become possible with both satisfying speed and accuracy.

The goal of this thesis is to implement a real-time object detection system based on some of the already published models, with the Proposal Connection Network (PCN) discussed in more detail. PCN in simple terms is a two-stage, anchor-free object detection model with unique advantages. Following the demonstration of system design and setup are training and experimental processes, focusing primarily on performance analysis and comparison among models.

Acknowledgements

As this thesis concludes, I would like to first show gratitude towards my supervisor, Prof. Pinhan Ho. I'm very grateful for having him as my supervisor since he offered me a graduate student position in the first place, allowing me to explore further in academic fields and attain higher achievements.

Like many other graduate students, I had no research experience before entering graduate school. I also had to make a transition from taking classes and following instructors' orders to leading my research-based projects. Some had it easy, but for me, it could make me feel disoriented and so I would sometimes run into road bumps. Whenever I faced difficulties, Prof. Ho was always glad to give me useful advice and offer me multiple opportunities to help me get things done in the right way. It took me a longer time to finish all my work, but now I have learned how to conduct research properly. It wouldn't be possible without the guidance from him.

I'm thankful for my family members and friends who have been behind my back and helping me get through some of the difficult times. Due to pandemics and other personal issues, my graduate study plan has been considerably affected, in which I would need to postpone important academic milestones. During this situation, I was able to collect my mind and move on because my loved ones have given me immense patience, understanding, and support. Their kindness has been and will always be the source of my success. No words can demonstrate how much appreciation I have for them.

Finally, I would like to also thank the readers of my thesis, Prof. Sagar Naik and Prof. Otman Basir, for spending their precious time on the examination committee, as well as the staff at the ECE department for their assisstance when I had problems about the program.

Dedication

This thesis is dedicated to everyone I love and care about.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter will start with illustrating some basic concepts of computer vision and its historical development. Specific roles of machine learning in modern day computer vision will be discussed, as machine learning has become the core technique to implement complex computer vision features nowadays. Finally, showing contributions and organization of this thesis finishes this chapter.

1.1 Early Image Processing

The concept of manipulating digital images can be dated back to the early 20th century. Starting from transmitting pixelated newspaper images over telegram, to CCD sensor invented in 1980, digital imaging techniques have already been developing even before the age of modern computers. Various imaging related applications have also emerged during this period, including satellite imaging, medical imaging, digital photo, and so on [1]. However, due to the limit of the early day imaging technologies, imaging equipment could only produce low quality pictures. For this reason, the earliest image processing techniques were mainly aimed to improve image qualities, such as noise removal, repairing visual defects, and compression for more efficient data storage. Certain image processing techniques were also capable of detecting simple geometric shapes in the image. Examples of these image processing techniques are given below, and comparison between original and processed images using these techniques are shown in Figure 1.1-1.14 [2].

1. Anti-aliasing: Reconstructing low resolution image through interpolation from neighboring pixels.

2. Radon transform: An transformation technique that can reveal straight edges in the image.
3. Radial distortion: Can be used to reverse distortion caused by geometry of camera lens.
4. Histogram stretching: Increases certain intensity values in the histogram to enhance brightness.
5. Averaging: Can be used to reconstruct damaged image.
6. Median filtering: Removes noise from the image.
7. Laplace smoothing: Detect and sharpening edges in the image.

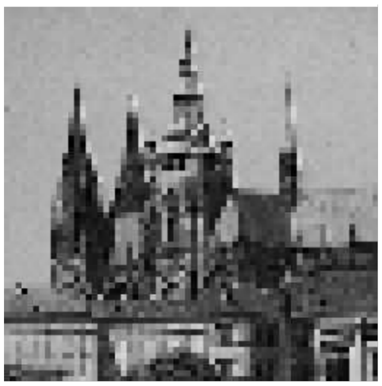
Figure 1.1: Before anti-aliasing

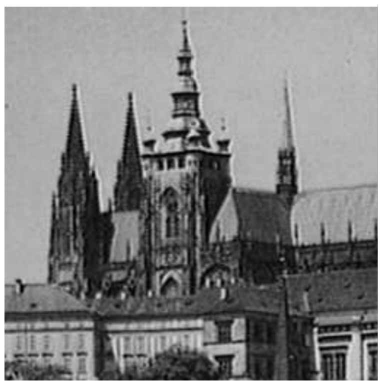
Figure 1.2: After anti-aliasing

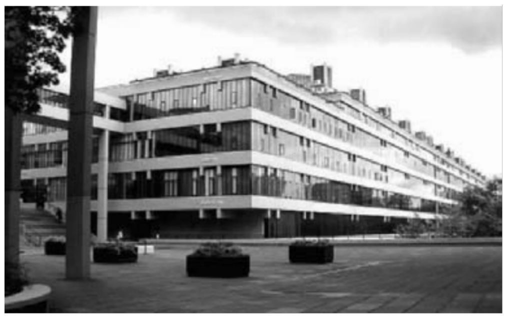
Figure 1.3: Before Radon transform

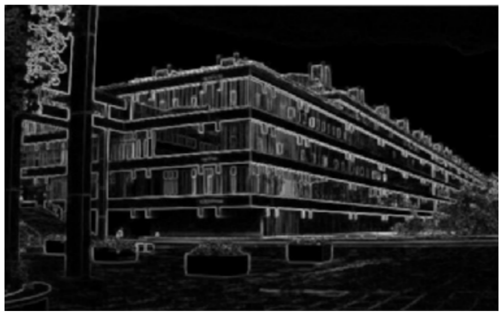
Figure 1.4: After Radon transform

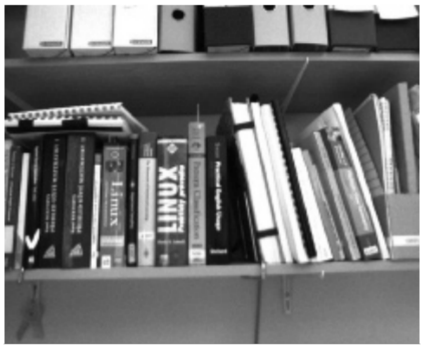

Figure 1.5: Before radial distortion

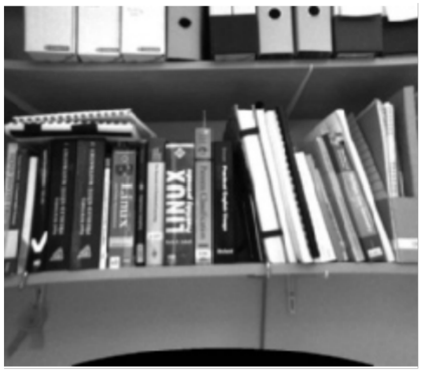

Figure 1.6: After radial distortion

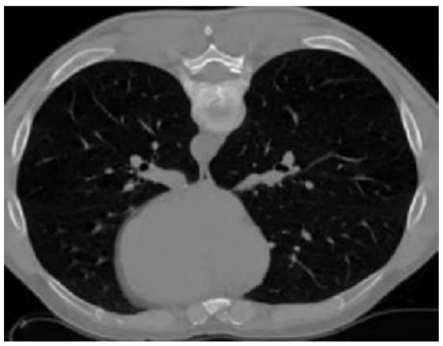

Figure 1.7: Before histogram stretching

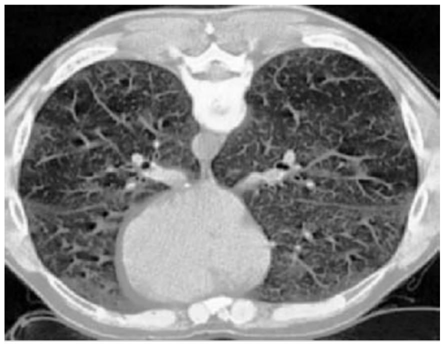

Figure 1.8: After histogram stretching

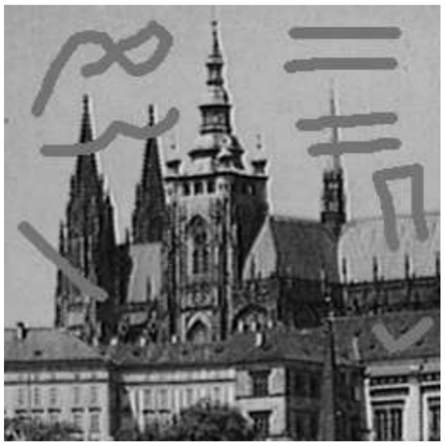

Figure 1.9: Before averaging

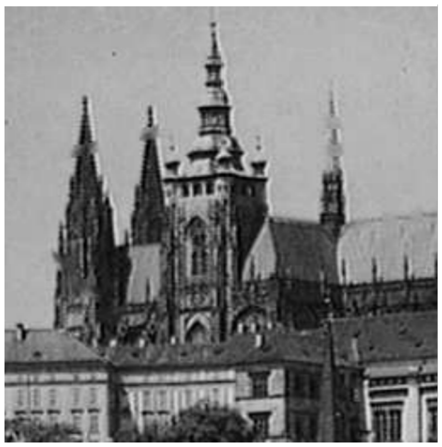

Figure 1.10: After averaging

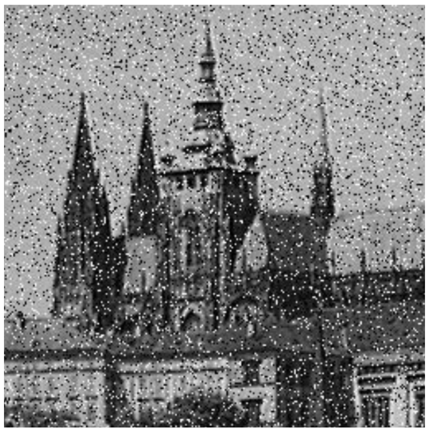
Figure 1.11: Before median filtering

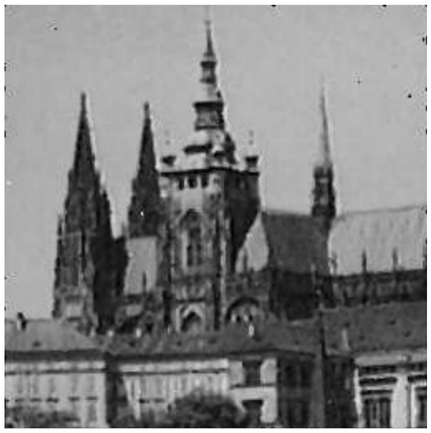
Figure 1.12: After median filtering

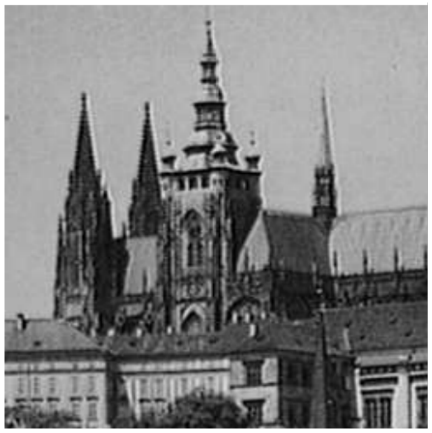
Figure 1.13: Before Laplace smoothing

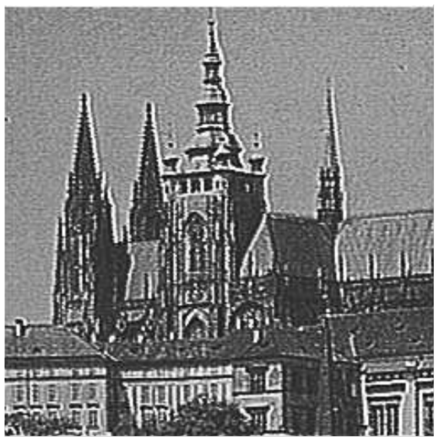
Figure 1.14: After Laplace smoothing

1.2 Computer Vision and Machine Learning

1.2.1 Early Applications

Unlike the basic image processing techniques introduced above, computer vision specifically requires the computer to make meaningful interpretation from the images. A computer vision program should be able to achieve this goal through extracting and recognizing key features from images, but it can be challenging because these key features are usually difficult to define explicitly in the code. With the use of machine learning, however, these features can be automatically acquired during the training stage. Applying machine learning has therefore become more important as the training process requires minimum human intervention, while a properly trained model can still produce accurate results with decent performance. In fact, machine learning has already become an indispensable component of computer vision systems [3].

One of the earliest computer vision applications leveraging the power of machine learning is Optical Character Recognition (OCR). Classifying hand written characters by computers is known to be especially challenging due to large variations in hand writings. Manually designing a classifier that would cover these different styles of hand writing would be very difficult. Instead, a dedicated computer vision model can utilize machine learning to discover patterns from varying strokes in hand writings and will correctly classify characters after training. Classifying characters by the OCR may comprise of the following steps: Each character would first be segmented from the image [4] and typically down sampled individually to reduce the input size of subsequent modules. Since characters can still be well recognizable under low resolution, simply down sampling would still retain character features at a good level while making the model more trainable. Down sampled characters are then fed into the classification module. For character classification, SVM or fully connected neurons is commonly used as the classification head. An example of OCR classifier is shown in Figure 1.15.

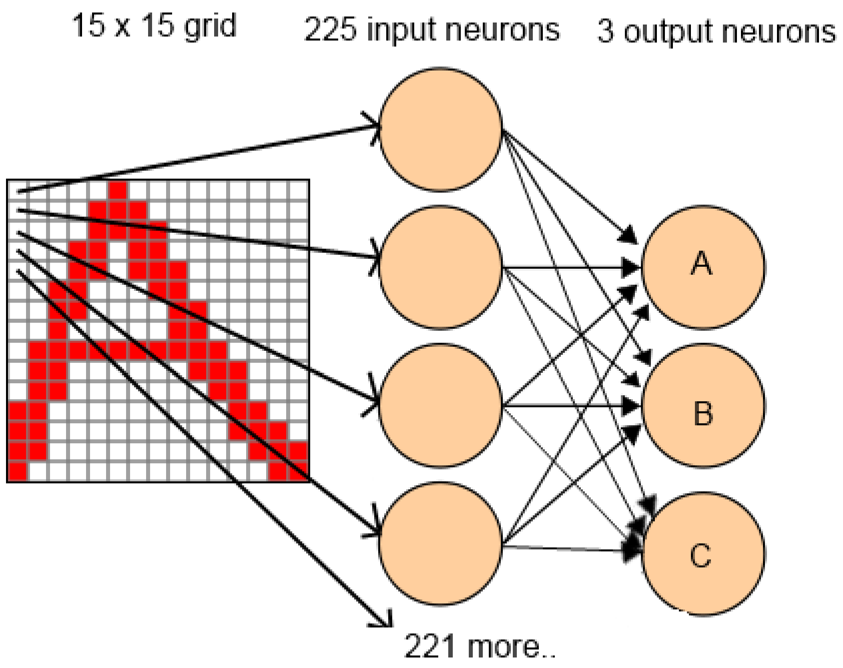

Figure 1.15: OCR example

1.2.2 Convolutional Neural Network

For higher resolution images containing detailed features, using Convolutional Neural Network (CNN) is a more popular approach for feature extraction. It imitates the process of how the brain perceives images: When a neuron in the visual cortex receives a stimuli, its neighboring neurons with overlapping reception fields will also activate. CNN carries out a similar mechanism by convolving filters (or kernels) with the input. Resulting features will become less sensitive to transformations including space shifting, scaling and rotation. Overall, each stage of CNN feature extraction can be concluded in the following way:

1. Convolution layer: Performs convolution through calculating the dot product between kernels and each small patch of the input. The filter will slide along the input until all areas of the input were covered. This layer produces a feature map that stores all calculated dot products.

2. Activation: Feature map from the convolution layer would need non-linear activation to start learning, similar to the case of fully connected neurons.

3. Pooling layer: Divides the feature map into grids and only records the most significant element in each grid. Its output is a feature map of reduced size.

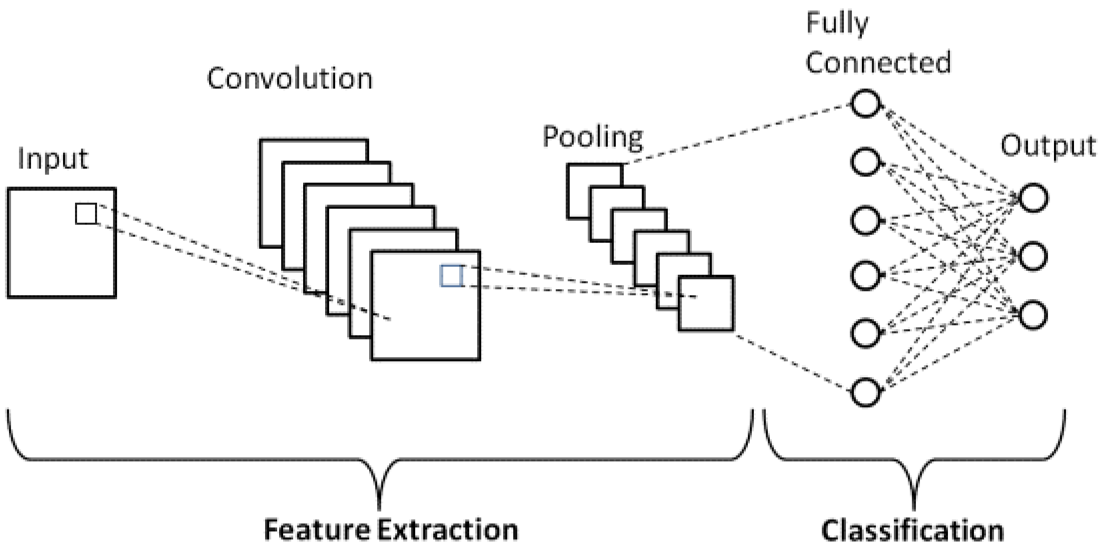

Figure 1.16: A demonstration of CNN layers

Layers illustrated above make up a single hidden layer in CNN. CNN feature extraction typically cascades multiple hidden layers with the same layout as the backbone, followed by a classification head to consist a complete classifier. This architecture is also used by LeNet [5], one of the classic and earliest CNN implementation as demonstrated in the Figure 1.16.

Throughout the development of modern computer vision models, CNN has become the major feature extraction backbone. For example, VGG16 as in Figure 1.17 is a classical implementation of object classification system to detect the presence of specific objects in the image. It has 16 CNN layers to convert each of the 224x224 input image into a 7x7 feature map before feeding into dense layers for classification [6].

Object detection is another popular field in the modern day computer vision. Other than identifying objects, object detection should also be able to locate the object and

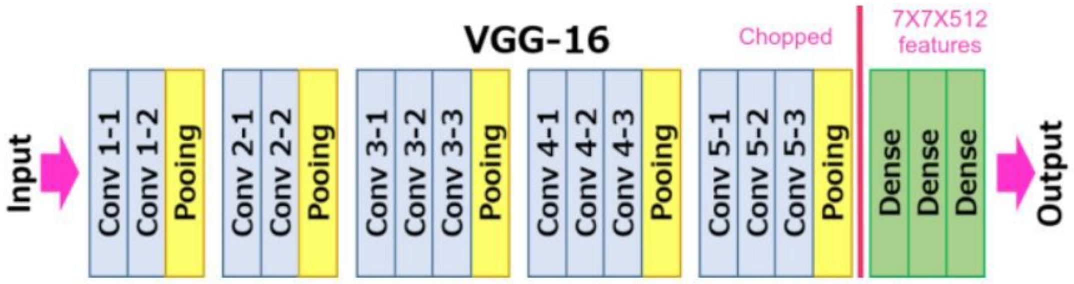

Figure 1.17: Layers of VGG16 network

calculate bounding box around each of them in the image. There are 2 major families of object detection models: Two stage detector like R-CNN has decent detection accuracy at different scales but slower. One stage detector like YOLO and Centernet can perform both class and bounding box predictions in the same module, and so the network architecture is simpler [7]. Generally, one stage detector can achieve higher speed, a desirable feature for real-time object detection.

1.3 Goals and objectives

Object detection has gained increasing popularity among many applications especially in industrial fields, and there have been well-developed models for these purposes. Nevertheless, there's no one model that fits all use cases and so choosing the right model becomes the important first step. Adjusting the model including its parameters is usually required to fit specific use case as well.

Following the rising trend of object detection related applications, the goal of this thesis to design an object detection system. This system can eventually be utilized by enterprises to specifically perform product inspection. The contribution of this thesis is it can help enterprises to have more efficient workflow by automating product checking processes. The object detection feature will be implemented based on already available architectures, so this work focuses more on demonstrating how to setup of a functioning system through hardware and software integration. It should be noted that these models does not equate out-of-the-box, or turn-key solutions. Instead, these models would still need to be tweaked as previously mentioned in order to produce satisfying results. Then, other than building the actual system, one of the main objectives of this thesis is to choose the best model and evaluate how it performs after training.

Lastly, works related to this thesis can be summarized as the following: Reviewing the implementation of PCN and other object detection models, showcasing an object detection system setup, and experimental analysis of these models.

1.4 Organization

Finishing up introducing some computer vision concepts, the remaining of this thesis will be presented in the following order:

- Chapter 2: Reviewing some popular object detection models as well as their architectures to understand how object detection works.
- Chapter 3: Introducing PCN architecture and its design considerations.
- Chapter 4: Illustrating the overall computer vision system architecture, including both hardware and software setup.
- Chapter 5: Running experiments to test the performance of multiple object detection models.
- Chapter 6: Finding out possible places to improve and plan for future works after summarizing this work and its respective results.

Chapter 2

Related Works

Following the introduction to computer vision related concepts and the goal of this thesis, this chapter will focus on illustrating several classic object detection models and their architectures. It would be useful to first review these models in order to thoroughly understand their design considerations.

2.1 YOLO

2.1.1 Overview

YOLO is a popular object detection model that has attracted attentions after its first appearance on the CVPR 2016 conference [8]. As a one-stage model, YOLO has several advantages over R-CNN and other two-stage predecessors. In a two stage model, the first stage would predict bounding boxes as ”region proposals”, and the second stage would predict the object class for each of these proposals. The purpose of such two-stage architecture is to allow the classifier to focus on regions of interest (ROIs) only. YOLO instead integrates these two functions into one stage so that a monolith module can output bounding box and class predictions simultaneously. It simplifies network architecture and can achieve higher speed, making it more desirable for real time object detection.

Similar to R-CNN, YOLO also starts with dividing the input image into grids. Both models would generate a given number of bounding boxes in each grid. In R-CNN, these generated bounding boxes are referred to as anchors. Anchors are a set of pre-defined rectangles of different shapes and sizes distributed throughout the image. To select the closest anchor to ground truth, each of these anchors is compared to any ground truth bounding box by calculating their IOUs. Higher IOU indicate an anchor better resembles the ground truth and so anchors with highest IOUs will become region proposals. The calculation of IOU is given as the following:

$$IOU = \frac{\text{Area over overlap}}{\text{Area over union}} \tag{2.1}$$

In comparison, YOLO's backbone would directly make bounding box predictions in each grid instead. The same backbone in YOLO would also predict the class score of objects in each grid and calculate confidence of bounding boxes by the following:

$$conf = P_{obj} * IOU \tag{2.2}$$

where P_{obj} is the probability of a grid containing an object, and IOU is calculated between prediction and ground truth. Those bounding boxes with low confidence will be discarded because they are very unlikely to contain objects.

At this point, each of the bounding boxes should contain an object, but the same object may be labelled by multiple bounding boxes. So YOLO would further perform Non-Maximum Suppression (NMS) to further remove bounding boxes with lower confidence. After NMS, only the bounding box with the highest confidence for each object would remain. This bounding box, along with confidence and class score, together becomes the result of YOLO object detection.

YOLO has great significance in computer vision because it proposes a new and concise method for object detection. Over the years, YOLO has kept evolving and there have already been several versions of YOLO developed. For example, YOLOv2 adapts a similar anchor mechanism to Faster R-CNN to improve accuracy [9]. YOLOv3 introduces feature pyramids for better detection capability of objects at different scales [10]. At the time of composing this thesis, YOLOv5 has become the mainstream implementation. It has proven to be a robust, efficient and accurate object detection model. For this reason, testing YOLOv5 will be a part of later experiments because its performance measurements can be seen as a good reference.

2.1.2 YOLOv5

The YOLOv5 model architecture is an evolved version from YOLOv3 and YOLOv4. Its variations are given as the following: YOLOv5s, YOLOv5m, YOLOv5l, YOLOv5x and

YOLOv5n. These variations come with different scales but sharing the same architecture. YOLOv5 models consist of input layer, backbone, neck and head networks as explained in the following [11]:

1. Input layer: YOLOv5 backbone takes a 640x640x3 input. Therefore, the most crucial role of the input layer is to resize the input image to this required dimension first. Resized image will retain its original aspect ratio to avoid distortion and then zero padded along shorter edges. Prior to resizing, the input layer will also perform data augmentation by adding random noise, stretching and rotating images, etc. This step will help the model generalize well.

2. Backbone: Backbone performs feature extraction from input images. In YOLOv5 backbone, the Focus structure will slice and re-concatenate an image along the channel dimension. It also employs CSP for residual connections, which allow some parts of the network to skip certain layers and thus feeding features forward. Both of these measures have helped accelerating YOLOv5 to better speed while retaining more features and decent accuracy.

3. Neck: It's a key network for connecting backbone to the head. In YOLOv5, FPN (Feature Pyramid Network) and PAN (Pyramid Attention Network) are used in the neck. It's responsible for merging features at different scales, which will improve the performance of detecting small objects.

4. Head: YOLOv5 has three heads with different output sizes. Like other computer vision models, each head in YOLOv5 will make both bounding box and class prediction. Multiple heads correspond to predictions at multiple scales.

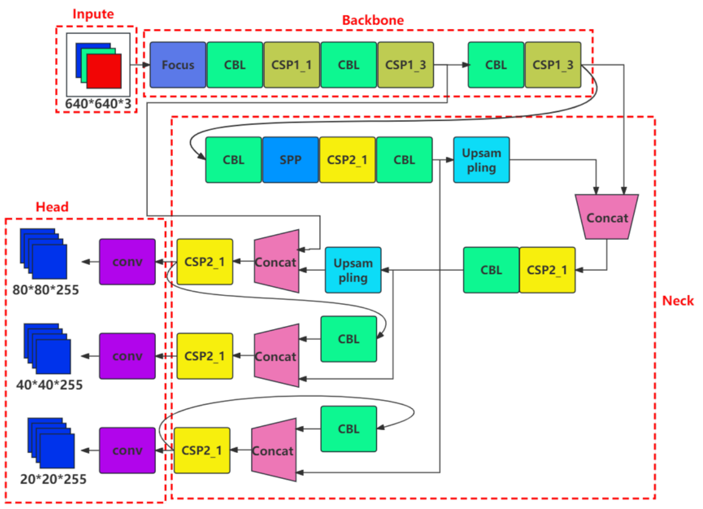

Figure 2.1: YOLOv5 architecture

2.2 Feature Pyramid Network

FPN is a network architecture first proposed in a CVPR 2017 paper for feature extraction [12]. In comparison to another feature extractor, the classical CNN introduced in the previous chapter, FPN has made several improvements to achieve better detection capability of objects at different scales.

Looking back at CNN, it typically has multiple hidden layers that each performs convolution, activation and pooling. After each pooling operation, one element in each grid is recorded while all other elements in the same grid are discarded. The final output of each hidden layer is a feature map with reduced size comparing to the input. By cascading these hidden layers, the CNN forms a pyramid structure where feature maps are obtainable at different sizes [13]. Layers closer to the input produce figuratively rich feature maps that still retain detailed patterns of the objects but have greater size. Moving towards the output, hidden layers will output smaller, semantically rich feature maps at the cost of losing important details. Some old computer vision models would only use feature map from the CNN output for subsequent prediction modules, leading to worse performance when detecting small objects.

Using feature map from the output layer is referred to as single feature map method, which is one of the possible ways to utilize feature maps from the pyramid structure.

Other ways to utilize these different feature maps from the pyramid include featurized image pyramid and pyramid feature hierarchy. Featurized image pyramid simply resizes the input image to different scales before extracting features from each of them. The resulting feature maps will retain the most information both figuratively and semantically but computationally expensive. Pyramid feature hierarchy is more similar to CNN in that features are extracted from the input image at different level, but feature maps from all layers will be used for prediction. It has faster speed than featurized image pyramid while utilizes more features than the CNN.

FPN is another method to produce both figuratively and semantically rich feature maps while having good efficiency. To achieve this goal, FPN borrows some concepts from the residual network to merge features from different levels. Starting with its backbone architecture, it still employs the classical CNN pyramidal structure with cascaded layers. This part is referred to as the bottom-up pathway. These two pathways Hidden layers within this CNN backbone usually has pooling size of 2 so that feature map output of each hidden layers would be half of the previous one. For example, a CNN consisting of stages C1. . . C5 will produce feature maps with size 1/2,. . . ,1/32 of the original input image.

In addition to the bottom-up pathway as in standard CNN implementation, FPN also incorporates top-down pathway that operates in the opposite direction: Semantically rich feature map from the CNN output layer has the lowest resolution. Using this feature map as the input, the top down module will generate a new set of feature maps from the top to the bottom layer. Each layer in the top down layer enlarges the feature map from the upper layer by up sampling the input by the same factor as the CNN pooling size. The resulting output would have the same shape as the CNN feature map at corresponding stage. The overall architecture of FPN is illustrated in Figure 2.2 below. The up sampling can be done through bilinear interpolation to enlarge the input features by inserting new pixels around existing ones. Key points in the bilinear interpolation process is shown in Figure 2.3.

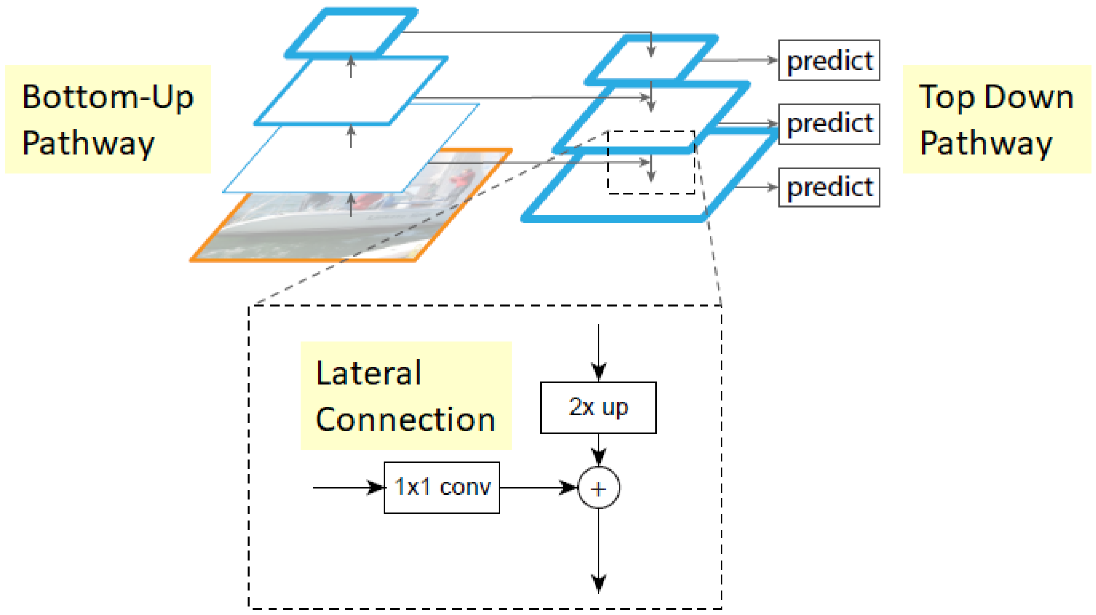

Figure 2.2: FPN architecture

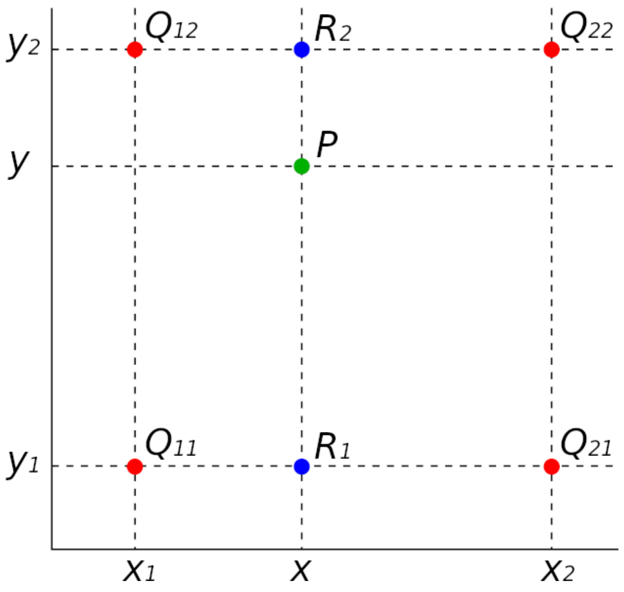

Figure 2.3: Bilinear interpolation

where:

$$\begin{aligned} R_1(x,y) &= Q_{11} * (x_2 - x)/(x_2 - x_1) + Q_{21} * (x - x_1) * (x_2 - x_1) \\ R_2(x,y) &= Q_{12} * (x_2 - x)/(x_2 - x_1) + Q_{22} * (x - x_1) * (x_2 - x_1) \\ P(x,y) &= R_1 * (y_2 - y)/(y_2 - y_1) + R_2 * (y - y_1)/(y_2 - y_1) \end{aligned} \tag{2.3}$$

The top-down approach brings semantically rich information from the top layer to the bottom. To combine the figuratively rich residual features from the bottom up module and interpolated semantically rich features from the top down module, at each layer the features from both modules are added element wise to generate new feature maps. By doing so, the bottom up and top down module complement each other and can generate high quality features of different sizes. Note that the element wise addition is possible because both modules use the same factor to scale up or down features to maintain the same feature size. Furthermore, residual features from bottom up layers typically goes through 1x1 convolution to flatten the channel before element wise addition.

Finally, 3x3 convolution is applied to these added features to eliminate aliasing effect. As a result of the bilinear interpolation in the up sampling process, the aliasing effect may produce uneven transitions in the interpolated regions and thus will leave zig zag patterns on feature maps. Adding an extra convolution helps removing these patterns and further improving the feature quality. These convolved features are served as the final outputs. They also come with various sizes and will be the input to subsequent modules. Each feature map output is assigned to a dedicated predicitor instance, trained or inference in parallel to detect objects at different scales.

2.3 Centernet

2.3.1 Key Point Approach

As its name suggests, the key point approach is a bounding box regression method that would directly locate key points of an object. This method does not require the model to divide the image and make predictions for each part individually. Instead, the key point approach works more intuitively: When looking at an object, the model would be able to focus on it through identify features at certain locations. These features can be specific patterns, contours, and so on. Those specific points to look may vary by model: For instance, Cornernet would identify the top left and bottom right corners of an object. Through finding out these two points, the rectangular bounding box can be uniquely defined [14]. It should be noted that features at the location of corner points may indicate either the object itself, or the background in the worst case. Using corner points solely for locating objects may have lower accuracy due to higher probability of having false results and so eliminating wrong predictions may become necessary. Another object detection model using key point approach would be Centernet. Details of Centernet architecture would be given in the following subsection.

2.3.2 Centernet architecture

The most notable characteristic of Centernet is that it would identify the object using the center point of the bounding box as well [15]. This allows the object feature to be extracted from the center location and so it can be used as the key feature for object identification. Centernet utilizing of center point other than two corner points as in Cornernet makes it has better accuracy with relatively small cost. For each bounding box, Centernet would also predict its center coordinate, along with its width and height. One of its major goals of training Centernet is to minimize the loss of these bounding box shape and location between the predicted and ground truth bounding box.

The structure of the standard Centernet implementation is shown in Figure 2.4 and can be summarized in the following way: After resizing the input image (usually to 512x512), the feature extraction backbone outputs heatmaps. Heatmap in Centernet is similar to CNN feature maps, but each element serves as a binary bit to tell whether an object center point exists: 1 indicates a center point exists in the corresponding area of the image, and 0 means otherwise.

Heatmap serves as the input to the following prediction heads: Each of these heads would predict center point, box height/width and class, respectively. Outputs from these modules are therefore the object detection result. To directly predicting key points from the heatmap, Centernet requires the heatmap to contain highly semantic features so that it captures more complete outlook of the object. It also requires the heatmap to have high resolution so that reading key point locations directly from heatmap becomes more accurate. These criterias can be satisfied by Hourglass network. It down samples the input to obtain semantically rich features with lower resolution and then up samples to increase its resolution again. In the standard Centernet, it cascades 2 hourglass networks as the backbone for heatmap generation.

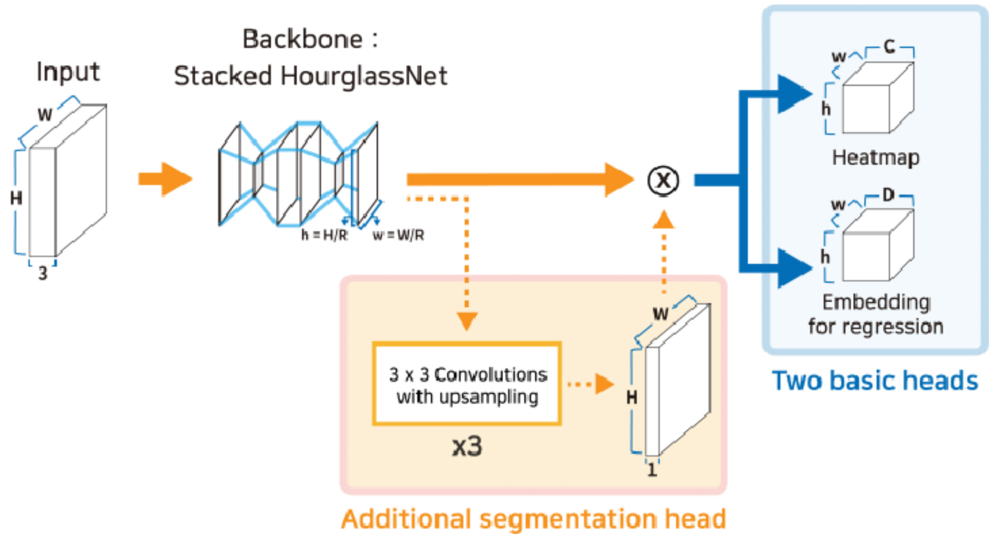

Figure 2.4: CenterNet architecture

When making predictions from heatmap, both Centernet and Cornernet would use corner coordinates to establish a bounding box. Suppose a bounding box is described by top left corner coordinate $[x_1, y_1]$ and bottom right corner coordinate $[x_2, y_2]$, the Cornernet would directly perform box regression using these two points. For Centernet, however, the bounding box regression would be performed upon its center point coordinate. The shape of the bounding box is defined by its height and width instead. Given aforementioned bounding box corner coordinates, the center point coordinate can be found by:

$$[x_c, y_c] = \lfloor \frac{x_2 - x_1}{2}, \frac{y_2 - y_1}{2} \rfloor \tag{2.4}$$

2.3.3 Focal Loss

Bounding box regression in Centernet consists of two independent tasks: For center point coordinate, it's preferrable to minimize the location difference between predicted and ground truth center. For bounding box width and height, both would be adjusted according to the ground truth as well. Also, only the center point would cause corresponding heatmap element to be 1, while all others would be 0. It would be more suitable for training if the heatmap can incorporate gradients. To do so, Centernet adds a 2D Gaussian around the ground truth center that helps predicted center point to move towards it through "gradient ascend". Heatmap elements in this case each represent the confidence from 0 to 1 at corresponding location instead [16]. The 2D Gaussian kernel is given by the following:

$$Y_{xyi} = \exp(-\frac{(\hat{x}-x)^2 + (\hat{y}-y)^2}{2\sigma_p^2}) \tag{2.5}$$

Y_{xyi} indicates the ith class confidence of predicted heatmap element at coordinate $(\hat{x}, \hat{y})$, and (x, y) is the ground truth center point coordinate. The addition of 2D Gaussian kernel is visualized as in Figure 2.5:

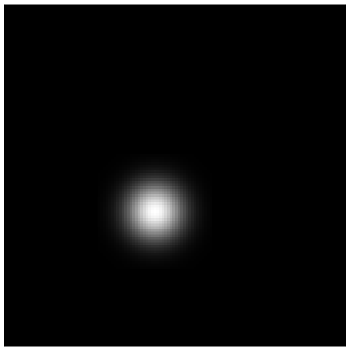

Figure 2.5: 2D Gaussian kernel demonstration

Determining whether a heatmap element should be 0 or 1 is analogous to solving the classification problem. Traditionally, the loss function for training a classifier is cross entropy given as the following:

$$CE = -\sum_i P_i * \log(P_i) \tag{2.6}$$

A training sample point further from the ground truth is more difficult to train and is referred to as the hard misclassified example. When other predicted points are already well-trained, these hard examples should be given more training weights. Therefore, a Focal term is introduced to the cross entropy loss function. so that as well-trained examples with higher training probability reduces the loss, and thus putting more weight on those hard ones. Resulting new loss function is Focal loss as illustrated in equation (2.7) below:

$$FL = -\sum_i (1-P_i)^\gamma * \log(P_i) \tag{2.7}$$

Assuming $\gamma > 1$: When the ith class confidence P_i is high, $(1 - P_i)^\gamma$ would reduce rapidly to suppress the loss.

The standard Focal loss above illustrates how the center point would be trained in the Centernet. However, other heatmap elements affected by the bell-shaped pattern introduced by the 2D Gaussian kernel would need to be trained as well. Training these peripheral elements is differentiated into two cases: A non-center point becomes misclassified when its predicted location is close to the ground truth center. One solution to this problem would be to increase its weight to retrain this point as much as possible. On the other hand, a center point predicted far away from the ground truth center is also considered as a misclassification. In both of these cases, adding another focal term would do us a favor. Therefore, the loss function for center point regression comes down to the following:

$$L_k = -\frac{1}{N}\sum_{xyi}\begin{cases}(1-\hat{Y}_{xyi})^\alpha \log(\hat{Y}_{xyi}), & Y_{xyi} = 1\\ (1-Y_{xyi})^\beta (\hat{Y}_{xyi})^\alpha \log(\hat{Y}_{xyi}), & \text{otherwise}\end{cases} \tag{2.8}$$

Since heatmap is "down sampled" from the input by a factor $R = 4$, there might be a slight difference between prediction from heatmap and ground truth on the input. Assuming the difference is defined by $p/R - \hat{p}$, defining another local offset term $\hat{O}_p$ would be helpful for minimizing the difference between these two terms. Furthermore, the difference in prediction and ground truth bounding box shape should be minimized as well. Combining these factors, its overall loss function would become:

$$L = L_k + L_{off} + L_{size} \tag{2.9}$$

where offset loss:

$$L_{off} = \frac{1}{N}\sum_p |\hat{O}_p - (\frac{p}{R} - \hat{p})| \tag{2.10}$$

and size loss:

$$L_{off} = \frac{1}{N}\sum_{p=1}^{N} |P_k - \hat{P}_k| \tag{2.11}$$

2.4 Summary

This chapter has reviewed state of the art network architectures of YOLO, FPN and Centernet. The next chapter will start explaining PCN in detail, especially layers and loss function. Knowledges from this chapter would be helpful for understanding PCN's design considerations.

Chapter 3

Proposal Connection Network

The previous chapter reviews the architecture and design of some well-known object detection models. Then, this chapter will take a look into another model: Proposal Connection Network (PCN) from a recent publication [17] by Kong et al. There are several interesting design considerations in PCN comparing to other models that are worth mentioning.

3.1 Network Architecture

Like Centernet and other anchor-less object detection models, the PCN backbone generates feature maps, which are used to further generate heatmaps for making region proposals. These region proposals would contain positive samples only, which are also known as the Region of Interests (ROIs). By focusing on positive samples, PCN avoids incorrectly classifying foreground/background and different classes of objects. Furthermore, weights are introduced to PCN classifier loss function to make a better use of training data and help combat the class imbalance problem.

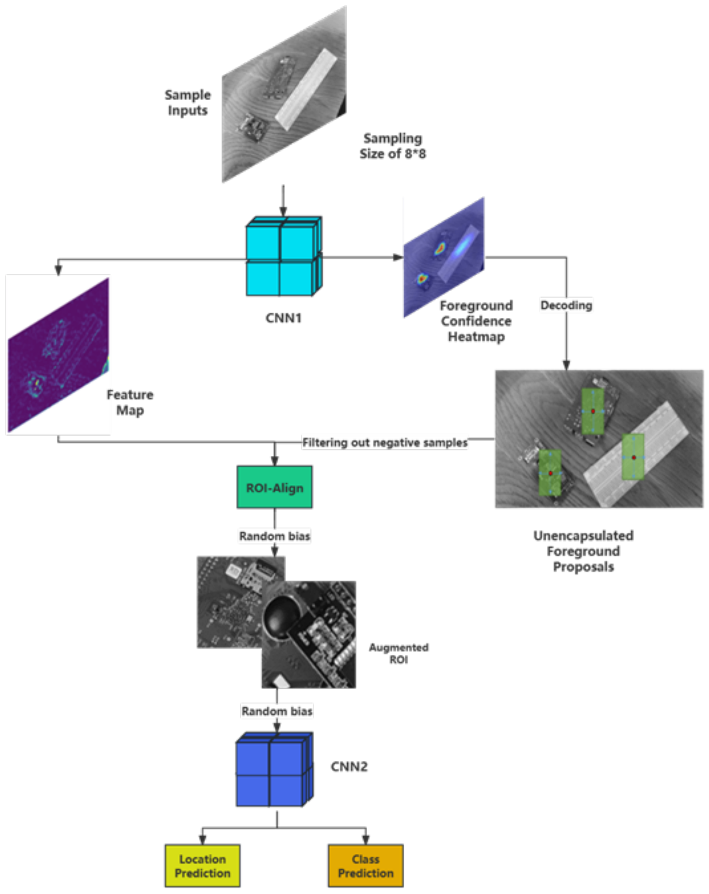

Figure 3.1: PCN architecture overview

The PCN architecture consists of two stages and is demonstrated in Figure 3.1. Stage one is the feature extraction backbone of PCN with feature maps and heatmap as the outputs. The heatmap is used for generating anchor-less foreground proposals. In stage two, the module would only look at proposed regions on the feature maps given by those proposals from stage one. Before pooling each proposal with ROI align [18], random displacements are introduced to generate augmented ROIs. Finally, these augmented ROIs are fed into prediction heads. The class head would predict the class probability of an object, and the location head would predict the center coordinate and size of the bounding box.

3.2 Generating Region Proposals

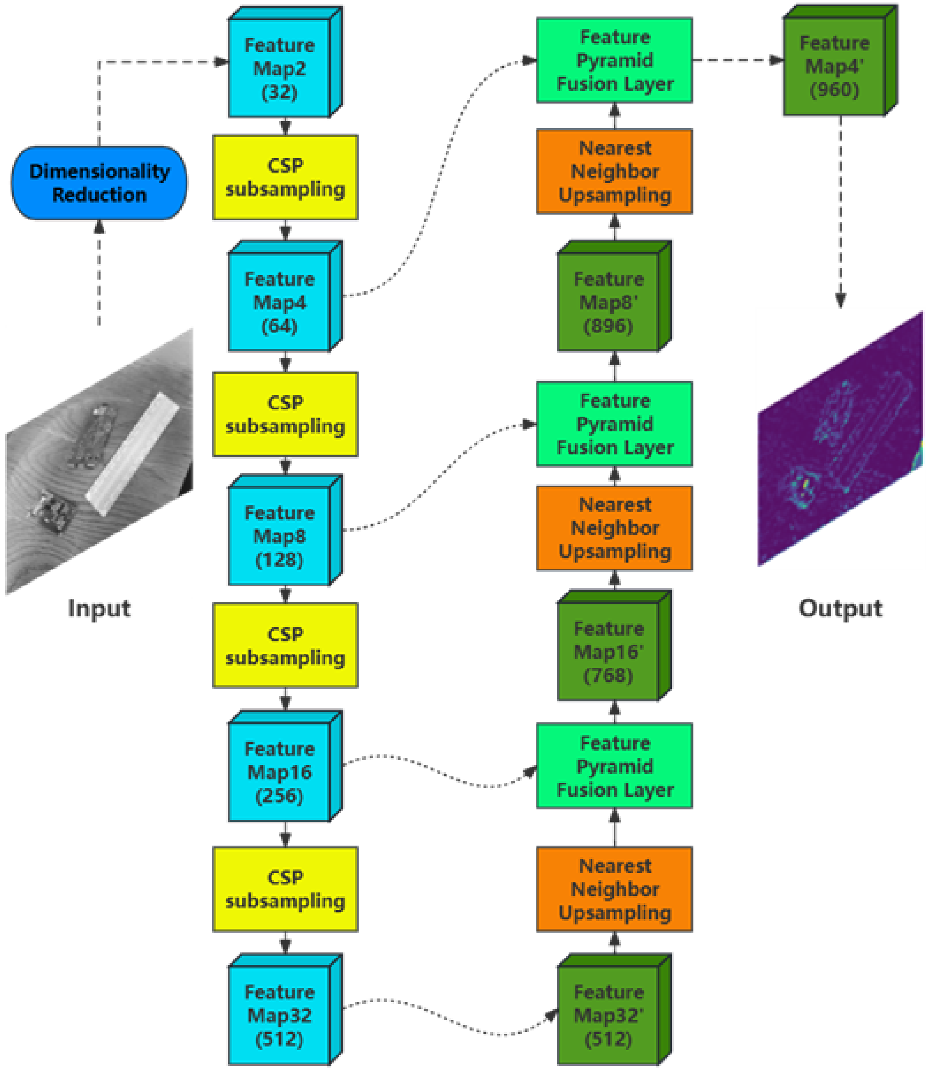

Figure 3.2: PCN stage one FPN

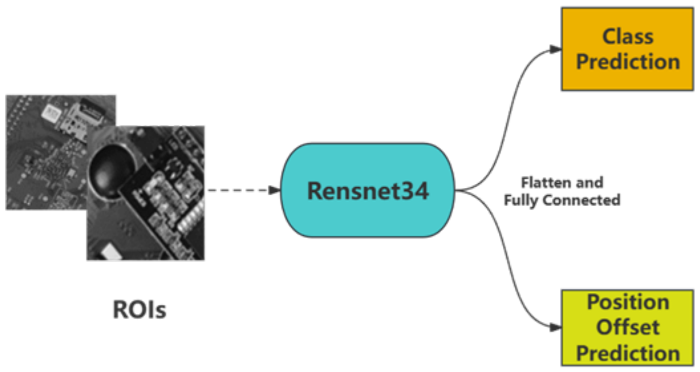

Figure 3.3: PCN stage two

As shown in Figure 3.2 above, feature extraction in PCN adapts the same architecture as the standard FPN configuration: Both bottom-up and top-down path contain 5 hidden layers, with strides of each hidden layer set to 2. Input is first scaled down to 1/32 of its size and then up sampled to its original size again. At each layer, the bottom-up feature after 1x1 convolution are merged with corresponding top-down feature. Merged features then each goes through a 3x3 convolution and finally serves as the output. Each convolution is followed by batch normalization to prevent gradient explosion in the network [19]. Also, convolved features have non-linear activation to ensure that kernels are trainable.

Figure 3.3 shows that after pooling and augmenting ROIs in the second stage, it would perform classification and predict offsets of the bounding box for each ROI. Resnet34 takes these ROIs as inputs and further extracts features before feeding into class and location head. Bounding box output from the location head would be a four-dimensional vector consisting of a 2D center point coordinate, width and height. In the class head, multi-class classification uses softmax as the activation function. The definition of softmax is given as the following:

$$\mathrm{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^{k} \exp(x_j)} \tag{3.1}$$

where k denotes the number of classes.

3.3 Proposal Connection

One reason the PCN would select only foreground proposals in its first stage is that it produces higher quality samples for subsequent modules, eventually leading to better performance. These selected foreground proposals provide all rectangular regions for the model to look on the feature map. Prior to entering prediction heads, proposals before ROI align are augmented to introduce more variations and thus making the model more robust. Proposal augmentation in PCN is done through randomly shifting the proposal location along either x or y axis. By default configuration, the maximum offset would be 10% of the proposal width or height, depending on the axis of shifting. Figure 3.4 and 3.5 illustrate how augmented proposals reduces the numbers of ROIs needed from region proposal module. By requiring less ROI, this practice also leads to significant speed improvement.

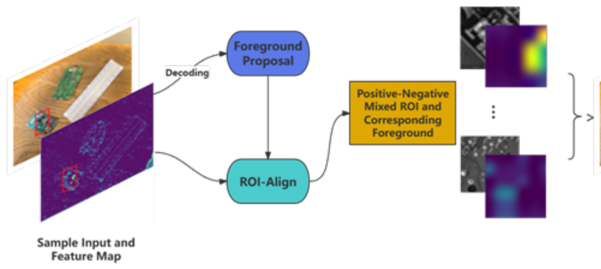
Decoding
Foreground Proposal
Positive-Negative Mixed ROI and Corresponding Foreground
ROI-Align
Sample Input and Feature Map
>

Figure 3.4: Traditional proposal connection

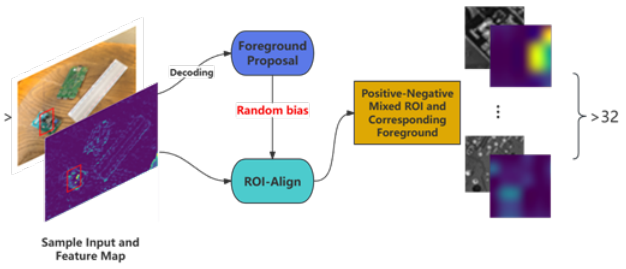
Decoding
Foreground Proposal
Random bias
Positive-Negative Mixed ROI and Corresponding Foreground
ROI-Align
Sample Input and Feature Map
>32

Figure 3.5: Augmented proposal connection

Generating proposals from the heatmap in PCN is analogous to the case in Centernet. The center point coordinate of the bounding box is encoded with 2D Gaussian kernel. Foreground proposals can then simply selected through max pooling over the heatmap. Gaussian kernel in the PCN is placed in the following way:

$$\Phi(x) = \exp(-\frac{x^2 + y^2}{2\sigma^2}) \tag{3.2}$$

where x, y are the offset from the center point, and σ is calculated according to the following rules:

$$
\begin{aligned}
r_1 &= \frac{(h+w) + \sqrt{(h+w)^2 - 4hw\frac{1-a}{1+a}}}{2} \\
r_2 &= \frac{2(h+2) + \sqrt{4(h+w)^2 - 4hw\frac{1-a}{1+a}}}{2} \\
r_3 &= \frac{-2a(h+w) + \sqrt{4a^2(h+w)^2 - 16hwa(a-1)}}{8a} \\
\sigma &= \frac{1}{3}\min(r_1, r_2, r_3)
\end{aligned}
\tag{3.3}
$$

where h, w are ground truth height and weight, and a is the IOU threshold for foreground proposal. r_1, r_2, r_3 each corresponds to the cases shown in Figure 3.6. Upon the max pooling operation, proposals further away from the ground truth are automatically excluded from the pooling region, and only the proposal with the closest proximity would be selected. These properties of pooling eliminates the need for NMS to filter proposals. Pooling also allows back propagation unlike NMS. Overall, it can be concluded that max pooling is more efficient and more suitable for end-to-end training.

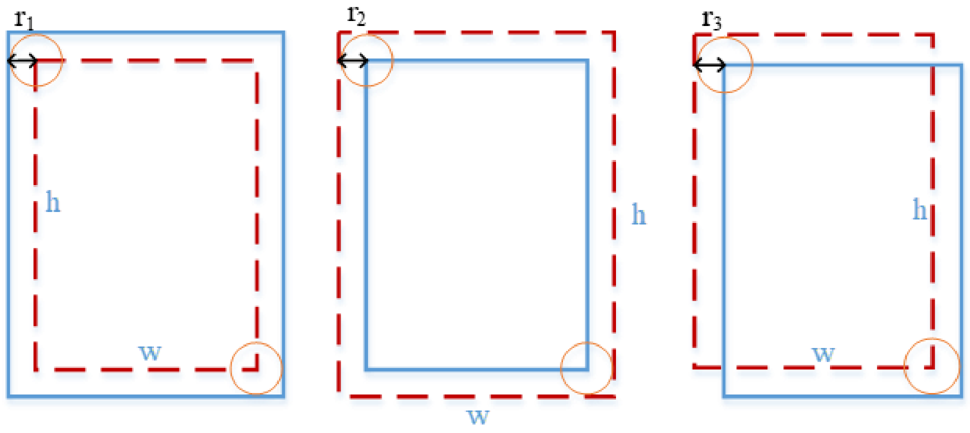

Figure 3.6: Anchorless bounding box overlays

3.4 Loss Function

Considering that outputs of PCN stage one are foreground proposals, it's also a classifier that determines whether each proposal is foreground or background. In stage one, Focal loss serves as the classification loss function instead of typical cross entropy loss, because Focal loss can put more weight on hard samples. The Focal loss function in PCN is given as the following:

$$L_{class1} = -\frac{1}{N}\sum_{i=1}^{N} a(1-P_i)^{\gamma}(y_i)\log(p_i) + (1-a)(p_i)^{\gamma}(1-y_i)\log(1-p_i) \tag{3.4}$$

localization loss uses L1 loss instead:

$$L_{loc1} = -\frac{1}{N}\sum_{i=1}^{N} |p_i - y_i| \tag{3.5}$$

where p_i and y_i are predicted and ground truth values, respectively. N is the number of total number of samples. Focal loss coefficients are set to $a = 0.5$ and $\gamma = 2$.

Class head in PCN stage 2 also performs classification, but it's only trained using foreground proposals and can potentially have class imbalance problems. Therefore, the stage 2 classification loss would be a modified cross entropy loss by adding a weight term. Localization loss function for location head would be smooth L1. Both of these loss functions are given as the following:

$$W_i = \frac{N^2}{N_i * \sum_{j=1}^{n}(N/N_j)} \tag{3.6}$$

$$L_{class2} = -\frac{1}{N}\sum_{i=1}^{N} W_i(y_i\log(p_i) + (1-y_i)\log(1-p_i)) \tag{3.7}$$

$$L_{loc2} = -\frac{1}{N}\sum_{i=1}^{N} SL_1(p_i - y_i) \tag{3.8}$$

where N_i denotes the total number of class i samples. SL_1 is the smooth L1 function. The sum of all these terms would be the overall loss for PCN:

$$L = L_{class1} + L_{loc1} + L_{class2} + L_{loc2} \tag{3.9}$$

3.5 Summary

PCN is a two-stage object detecion model with two main design considerations: Region proposal and class weight. PCN focuses on foreground proposals and augmenting ROIs for better efficiency, while applying class weights and focal loss helps with the class imbalance problem. The next chapter will then show the hardware and software setup of an object detection system. This system will be the platform for experimenting with PCN and other object detection models.

Chapter 4

System Design

This thesis has been extensively covering computer vision and object detection models up to this point. Instead, this chapter will demonstrate how to build a functioning object detection system through reviewing its hardware and software setup. Object detection models from the previous chapters will be directly integrated into this system.

4.1 Overview

One of the classic computer vision application in manufacturing and logistic related industries is product inspection. An industrial product inspection system typically has cameras connected to edge devices at the frontend, and cloud servers at the backend to run computer vision or image processing related functions. Among all computer vision techniques, object detection is the suitable method for identifying different products or interpreting visual pattern on those products. For example, object detection can be used to find out defects on a product, which may include contamination, misalignment, broken parts, incomplete components or any other irregularities detectable from photos. At a production line, such product inspection system is capable of automatically check products and then execute corresponding tasks based on detection results. This type of work has long been done manually before computer vision system becomes commercially viable. Adapting computer vision to replace human labor can speed up the process while eliminating human error. Nowadays, these product inspection systems have seen more and more use to automate a variety of production lines.

The object detection system mentioned throughout this thesis is specialized for product inspection purposes. This system is designed to identify each product individually and in real time. In certain instances where multiple types of products may be mixed up, it would need to be able to differentiate these products from one another. From the list of potential use cases below, one can see that the application of this product inspection system is not constrained to manufacturing, but other fields as well:

1. Counting the number of manufactured product for each product type on the production line.
2. Keeping track of the quantity of each type of product in a storage facility.
3. Classify final products and directed them to corresponding packaging lines.

4.2 Camera Setup

Feature	Specification
Brand and model	HikRobot MV-CU020-19GC
Image Sensor	IMX290
Maximum resolution	1920x1080
Maximum FPS	56
Dynamic range	90dB
SNR	41dB
Gain	0-30dB
Exposure time	128μs-260ms
Output color channel	Mono8
Data link interface	GigE

Table 4.1: Camera specs

The core components of hardware platform include two industrial grade cameras whose specifications of these cameras are given in Table 4.1 above. The usage of these cameras in industrial applications requires the camera body to be attached onto a mounting point. Since distance from the camera to inspected objects may vary depending on the setup, these industrial cameras usually have zoom lenses installed with adjustable focuses, which these cameras also support. Upon camera testings, both cameras are capable of delivering clearly captured images when the focus is properly set. However, the camera does not perform well under normal lighting conditions. Only when the objects are exposed under strong lights can the cameras output images with normal brightness. Very good environmental brightness is typically required by these industrial cameras and so adding an external light source is also mandatory in this case.

As previously mentioned, cameras are first mounted on a mounting plate, which is then mounted on a microscope stand. Also, both of them requires higher brightness level to take videos properly, so a light source is also mounted along with these cameras to create an ideal brightness in front of cameras. The cameras are placed with lenses facing down, and objects will be placed right below the cameras for visual inspection. This setup places cameras and objects at close proximity. With the help of light source, pictures of the objects can be clearly taken from the top view after properly adjusting the lenses focus. These configurations have the following advantages:

1. Objects are placed at a fixed distance from the cameras, so the camera system would not need further calibrations once the system has optimal setup, including focus of lenses.

2. Fixed distance from the cameras to objects also makes visual measurement easier when needed.

3. Allow cameras to take videos clearly in any environmental brightness.

4.3 System Architecture

In addition to the mounting platform, cameras mentioned above would also be connected to an edge device through Gigabit Ethernet. The complete camera setup is presented in Figure 4.1 below. Video frames will be transmitted from cameras to the edge device under GigE protocol. Edge device is a PC runs pre-installed Windows 10 operating system, along with official camera SDK and driver from the manufacturer to make cameras properly function on the system. For the current setup, videos from both cameras will be streamed but only one of the camera is needed and used for object detection. The other camera is reserved for future extensions, including stereo vision or providing redundancy.

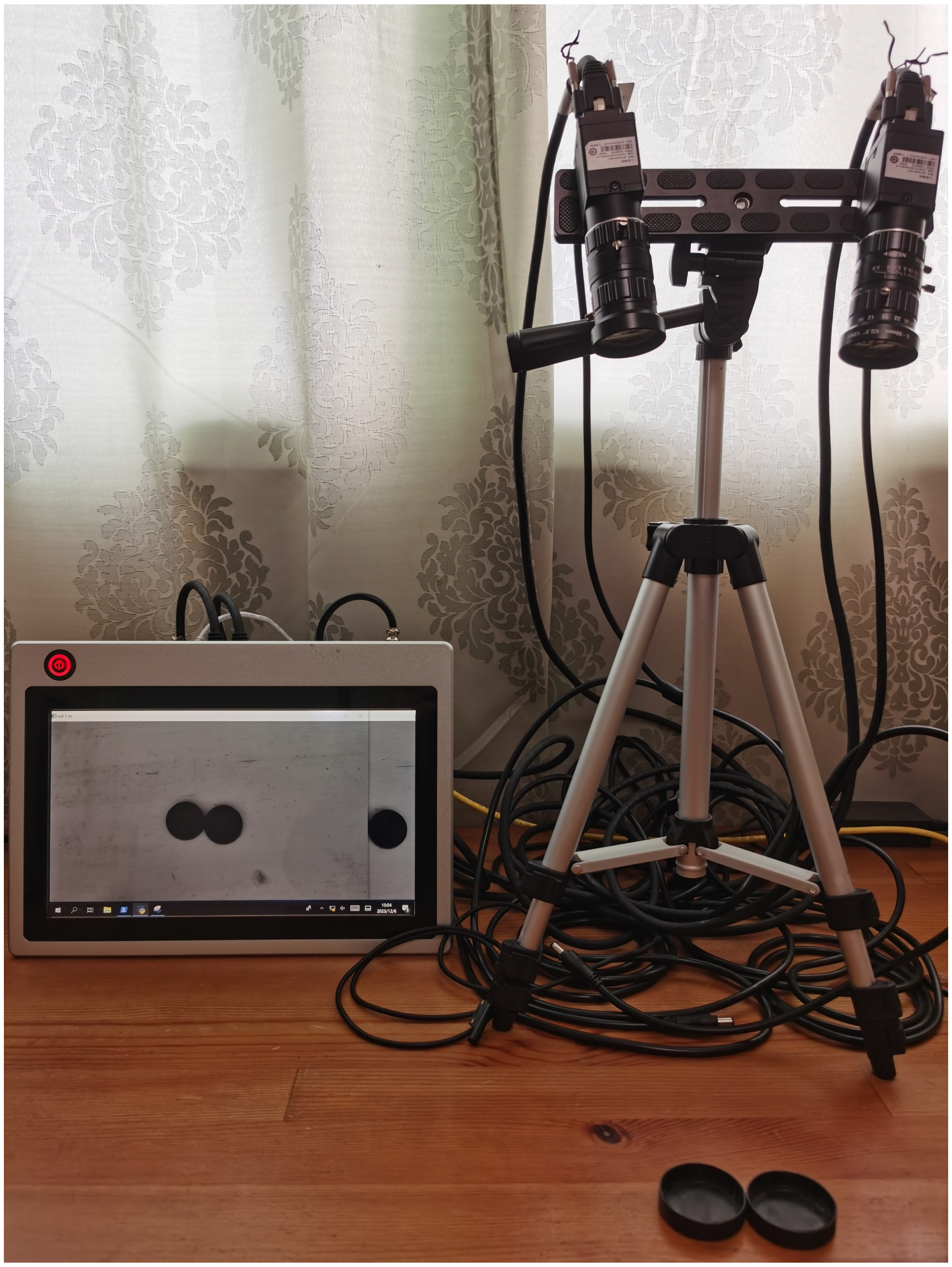

Figure 4.1: Camera setup

Feature	Specification
CPU	Intel Celeron J4125 @ 2.0GHz
RAM	8GB
Dedicated GPU	N/A
Input	Touch screen
Network interfaces	2x 1.0Gbps Ethernet, external USB2.0 to 100Mbps Ethernet dongle
OS	Windows 10 Pro
Installed softwares	Camera SDK, Python 3.11, OpenCV 4.8.0, FFMpeg

Table 4.2: Edge device specs

Specifications in Table 4.2 reveal that the edge device might not have sufficient processing power to run the object detection model locally. Models would need to be deployed on another computer with a dedicated GPU so that the object detection can operate smoothly in real time. These two machines would be connected directly over an Ethernet cable for more reliable connection. In this case, the system connection can be described by a simple client-server link: The edge device would act as a client for pushing video streams from cameras. The camera setup and edge device together consist the frontend of object detection system. The frontend in this case refers to all devices that would be installed on the field, in which objects are inspected. Meanwhile, the GPU machine becomes the server of the system. It's the backend for running computational heavy tasks, which in this case specifically indicates real time object detection.

Since both built-in Ethernet ports on the edge device have already been occupied by cameras, a USB 2.0 100Mbps Ethernet dongle is added to the edge device for the client-server connection. USB2.0 has the theoretical data rate of 480Mbps and so Ethernet port on the dongle becomes throughput bottleneck. After considering these factors, the maximum data rate of the client-server link would be 100Mbps/8 = 12.5MB per second. Each camera is capable of outputting 1080P video at 30 fps, whose bitrate is usually from 3 to 6Mbps after H.264 encoding. Therefore, lowering the resolution to reduce bitrate would become necessary to have stable streaming. After testing the cameras and edge device, streaming two cameras at 540P can ensure stability while still maintains decent image quality.

For the software implementation, all of the coding would be based on Python, OpenCV and machine learning frameworks: Tensorflow and PyTorch. The official camera SDK installed at the frontend provides camera interfaces to control and grab video frames from cameras. Basic image processing would be done by OpenCV. Tensorflow and PyTorch would be deployed at the backend server to run object detection models. When setting up the model environment, it's preferrable to leverage the GPU power to its full extent. All aforementioned frameworks and many other essential packages are available to Python, so the same programming language would be used for building the system. This arrangement makes implementation and maintenance relatively easy without causing significant performance loss, considering that all computationally intensive functions are already compiled to native libraries. Python codes only defines the overall routine, and the actual computations are carried out by calling these native libraries instead.

The overall workflow of the product inspection system is presented in Figure 4.2 and summarized as the following: The edge device at the frontend opens and grabs video frames through the camera SDK interface. The video refresh rate is defined by parameter FPS. Grabbed video frames then go through image processing routines in OpenCV. Possible steps may include color space conversion and resizing. Processed video frames would be streamed over RTSP, a common media streaming protocol that has been developed and used by many media services. Specifically, video frames are written into a pipe opened by a FFmpeg subprocess for RTSP transmissions. Video streams are received by OpenCV at the backend server. Received video frames will be fed into the object detection model. At the end, the model would output frames with bounding box and class label added around each detected object. These processes will be running for each video frame, and detection results returned from the model may be displayed continuously in the form of live video.

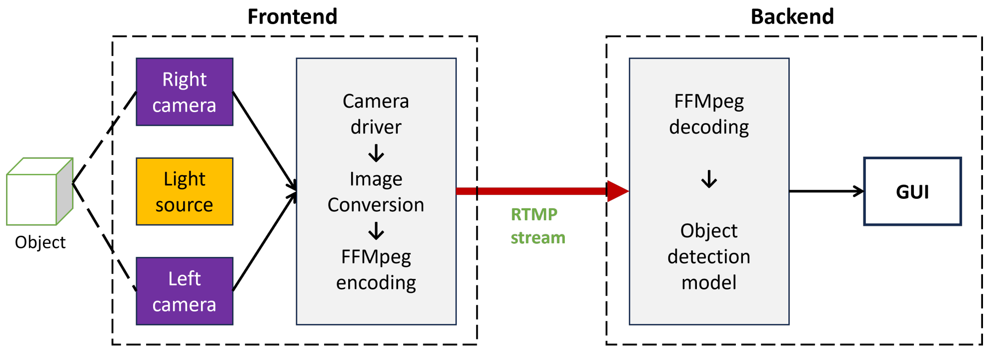

Figure 4.2: Product inspection system architecture

4.4 Summary

Following the introduction to object detection related topics in previous chapters, in this chapter showcases a practical product inspection system. This system consists of two industrial cameras, an edge device as the frontend, and a server as the backend. PCN and other models are deployed to this system, and in the next chapter their performance will be tested as a part of the experiment.

Chapter 5

Experiments and Results

After reviewing object detection models, this chapter will start with showing the training setup, dataset preparations and other important details. Models introduced in this thesis will be put under tests to see how they perform. Performance analysis will be based on evaluating key metrics, including accuracy and speed when the training is completed. Lastly, parameters and functions will be tweaked to see how they perform accordingly.

5.1 Training Dataset

In respect to product inspection related application, photos in the training dataset were prepared to include three types of product: Single Board Computers (SBCs), PCB and breadboard. The training dataset contains 50 SBC, 46 PCB and 39 breadboard photos, adding up to a total number of 135. The ratio of each of these products is 37%, 34.1% and 28.9% among all, respectively. These photos are first pre-processed by various augmentation techniques including rotation, stretching, displacement and contrast adjustment are also applied to the training datasets. All these photos will then be resized to 512x512 by bilinear interpolation and normalized before serving as the input to the entire network.

5.2 Key Performance Metrics

The first performance metric to consider would be MAP. MAP is one of the popular methods to measure precision and applicable to both classification and regression. Specifically, it calculates the average precision over all predictions made. The definition of MAP is given as the following:

$$\text{MAP} = \frac{1}{N} \sum_{N} \frac{\sum_{i=1}^{R} \text{precision}(i)}{R} \tag{5.1}$$

where R and precision(i) denote the number of predictions made and precision at the ith prediction.

It should be noted that upon evaluating the precision of bounding box outputs, each of these bounding boxes will be compared to their respective ground truth by calculating their IOUs. Only the bounding boxes with IOU over a threshold would be considered as a True Positive (TP).

For classification, it would also be of interest to look at the accuracy of PCN. Accuracy can be defined as the following:

$$\text{ACC} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{5.2}$$

where TP, TN, FP and FN denotes True Positive, True Negative, False Positive and False Negative, respectively.

The final metric of interest is FPS. It measures the number of images the model can process in each second. FPS may vary significantly by hardware specifications, system configurations, the extend of parallelism involved and many other factors. To eliminate these variations, 5 samples were selected for testing, and each sample would be tested for 100 times consecutively. The average FPS over these samples and trials would become the final measurement.

5.3 Experimental Results

Experiments were done by comparing the PCN and other 3 state-of-the-art object detection models as the baseline: YOLOv5s, YOLOv5l and Faster R-CNN. The same training datasets, parameters and performance measuring methods are applied to all 4 models for testing. These models are deployed to a dedicated backend server according to the system setup from the previous chapter. Specs of the server are listed in Table 5.1.

Feature	Specification
CPU	Intel Core i9-9900K @ 3.6GHz
RAM	64GB
Dedicated GPU	Nvidia GeForce RTX 2080Ti
Network interfaces	1.0Gbps Ethernet
OS	Ubuntu server 21.04
Installed softwares	Python 3.11, OpenCV 4.8.0, Keras 2.3.1, Tensorflow 2.13.0, CUDA 10.0, cuDNN 7.6, FFMpeg

Table 5.1: Backend server specs

Model	MAP	Accuracy	FPS	Parameters
YOLOv5s	0.878	0.783	42.0	7.5M
YOLOv5l	0.894	0.863	24.5	47.8M
Faster R-CNN	0.893	0.851	19.4	28.37M
PCN	**0.888**	**0.833**	**33.2**	**13.88M**

Table 5.2: Experimental results

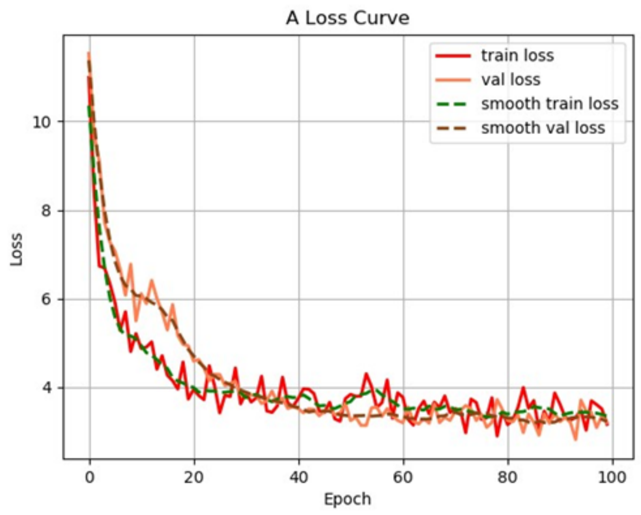

Figure 5.1: PCN loss curve

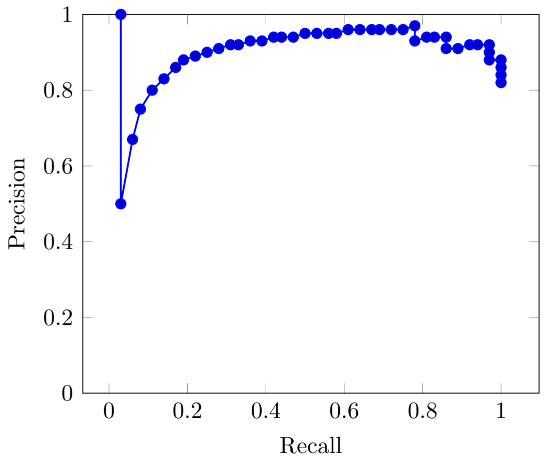

Figure 5.2: SBC recall vs precision

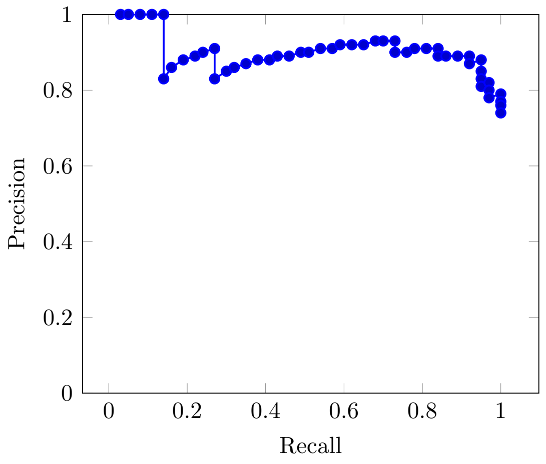

Figure 5.3: PCB recall vs precision

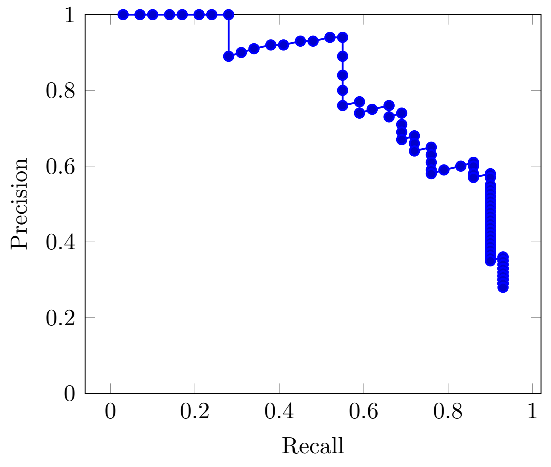

Figure 5.4: Breadboard recall vs precision

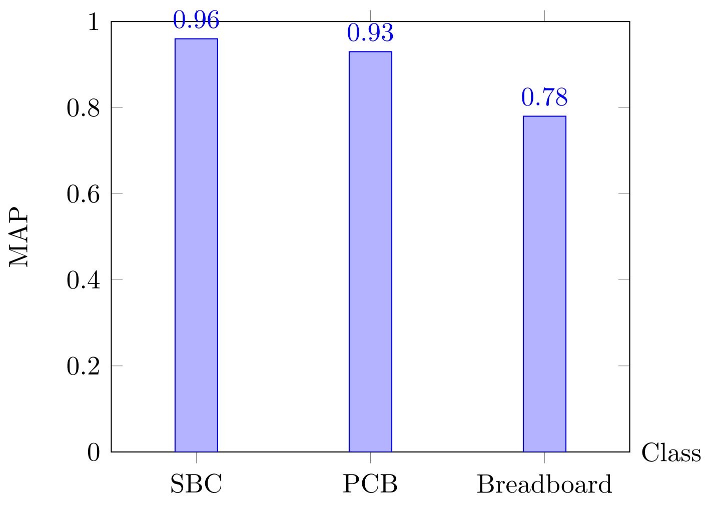

Figure 5.5: PCN average precision by classes

At the end of running the experiment, performance measurements were obtained as listed in Table 5.2 and Figure 5.1 to 5.5. Observations are be made from these measurements and listed below:

1. MAP of all four models are greater than 0.85, indicating low variations among predicted results.
2. All models other than YOLOv5s have good accuracy of over 0.8. YOLOv5l has the highest accuracy of 0.863 among all tested models, but PCN's 0.833 accuracy is only 2.4% lower in comparison. It reveals that PCN is also capable of detecting objects accurately.
3. PCN runs at 33.2 FPS, which is much faster than Faster R-CNN and YOLOv5l while having close MAP and accuracy. FPS measurement of YOLOv5s is still considerably faster, but its accuracy is not as ideal.
4. PCN has relatively less number of parameters among these models. Its 13.88 million parameters is lower than both YOLOv5l and Faster R-CNN. YOLOv5s has the least number of parameters but not as accurate in comparison to PCN.

Observations above reveals that PCN is the optimal model among all with balanced precision, accuracy, speed and size. However, PCN to some extent still suffers from class imbalance problem: Among average precision of all object classes displayed in Figure 5.5, the object class with higher number of training samples has better precision than others. It indicates that predictions made by PCN still demonstrate a direct relationship between class samples and precision.

Figure 5.6 to 5.15 visualize predictions made by different models on top of training sample photos. Predictions include bounding boxes drawn around the objects, with each having class labels and confidence immediately above. In most cases, these models can make right predictions on object classes and positions, but PCN makes less error than others. In addition, YOLOv5s can occasionally produce false positive bounding boxes on top of other object classes and the background. Those YOLOv5s predictions can be found in Figure 5.8 and 5.9. Other models are free from issues above and make higher quality predictions. However, in Figure 5.13 there is still one case where Faster R-CNN has failed to detect an object. PCN can successfully and correctly classify objects in all instances with satisfying accuracy and precision. It can therefore be concluded that the PCN is the best performing model throughout the experiment.

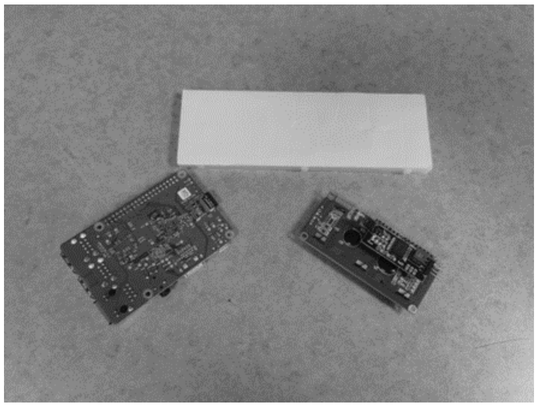
Figure 5.6: Training sample 1

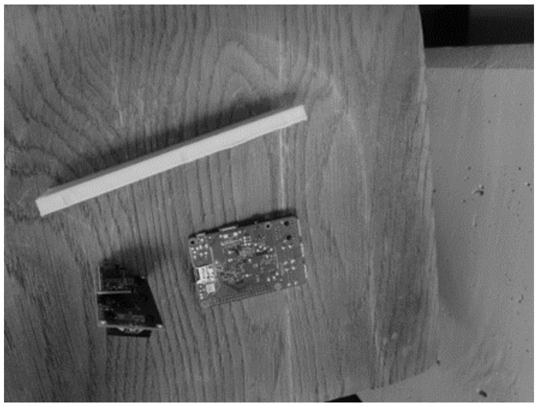
Figure 5.7: Training sample 2

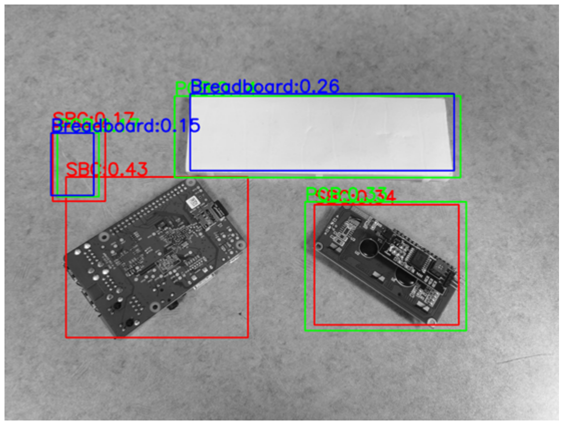

Figure 5.8: YOLOv5s prediction 1

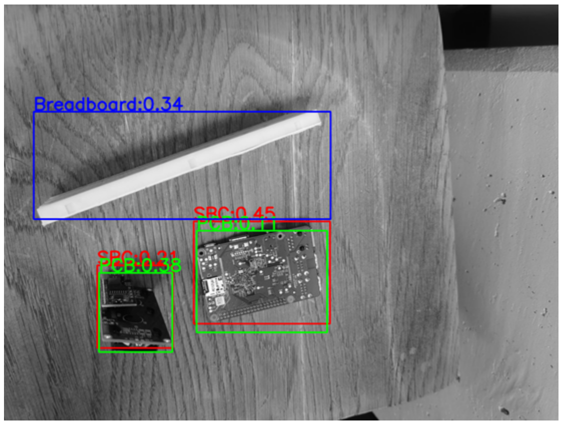

Figure 5.9: YOLOv5s prediction 2

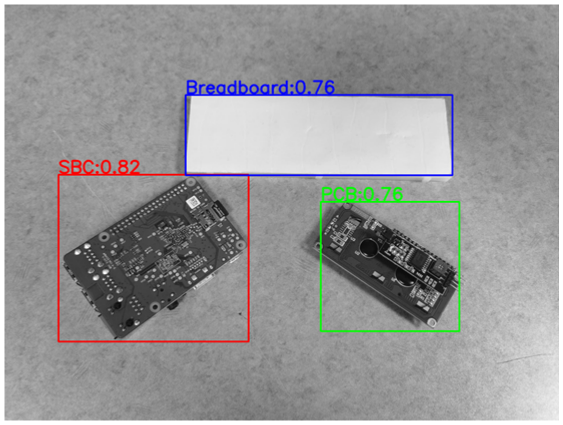

Figure 5.10: YOLOv5l prediction 1

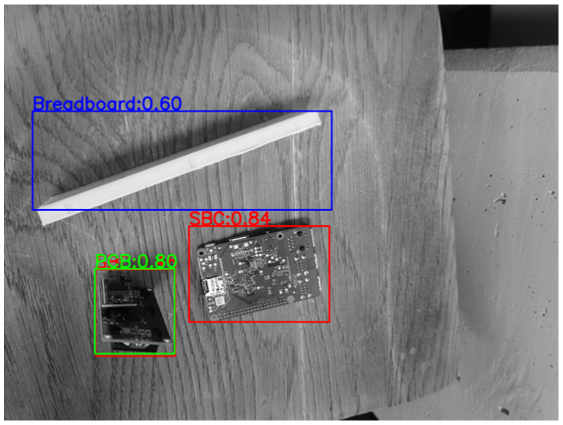

Figure 5.11: YOLOv5l prediction 2

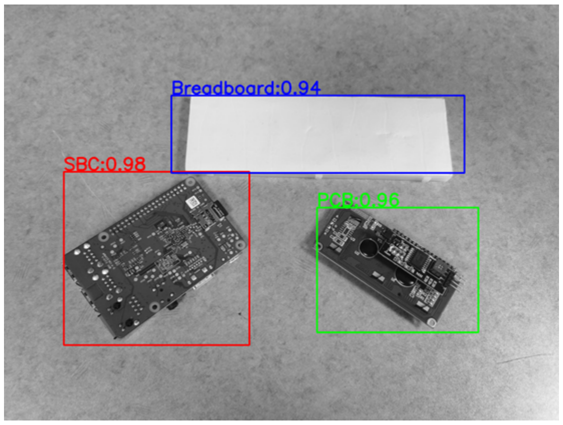

Figure 5.12: Faster R-CNN prediction 1

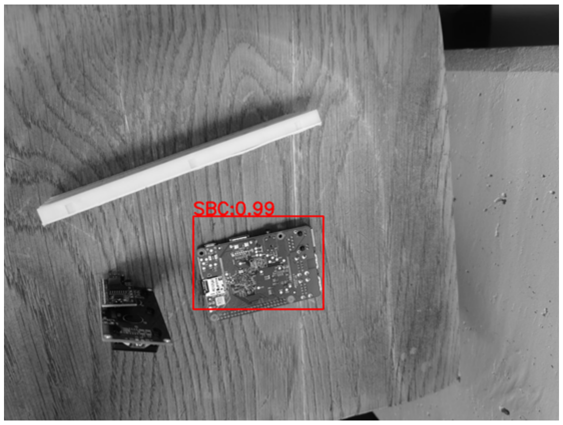

Figure 5.13: Faster R-CNN prediction 2

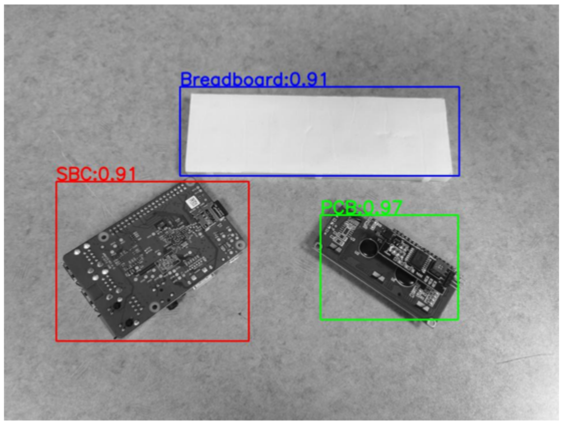

Figure 5.14: PCN prediction 1

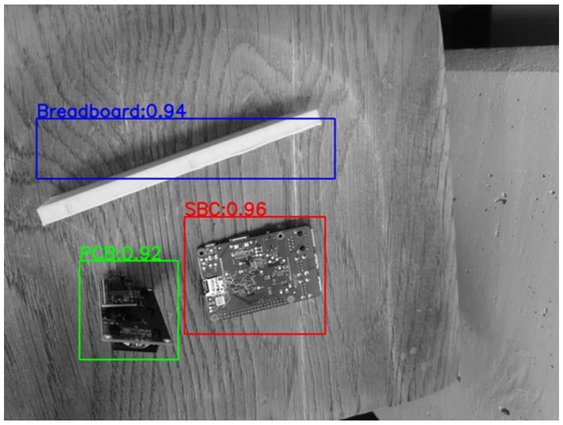

Figure 5.15: PCN prediction 2

5.4 Ablation Study

Class weight	MAP	ACC	FPS
Not included	0.842	0.812	31.7
Included	**0.888**	**0.833**	**33.2**

Table 5.3: PCN stage two class loss comparisons

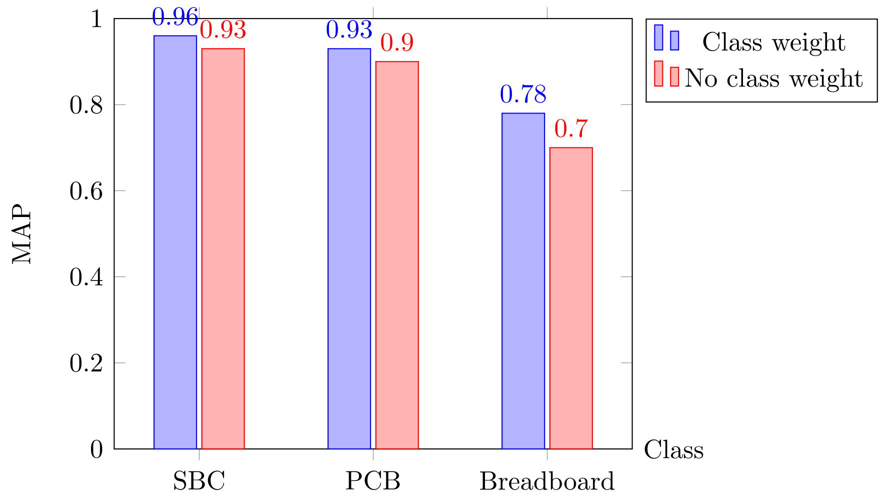

Figure 5.16: MAP comparison for each object class

Ablation study in machine learning is an analytical technique to gain a better understanding of the model through removing or replacing parts of the network and evaluate their impacts. Similarly, modifying loss function can help with understanding the effectiveness of certain design factors. For PCN, allowing $W_i = 1$ in (3.6) to remove the class weight. From the results listed in Table 5.3, both MAP and accuracy are higher when class weight is included. A possible explanation would be that the class weight alleviates the class imbalance problem. The same trend appears in Figure 5.16 when comparing MAP for each object class as well.

5.5 Summary

Through testing these models, PCN is found to be also capable of detecting objects at good accuracy, precision and speed like other well-known models. PCN reveals a balanced performance among all tested models. Lastly, this chapter also studies class imbalance and the effect of PCN loss function on this problem.

Chapter 6

Conclusion

6.1 Summary

This thesis has demonstrated a real-time product inspection system for practical uses. This system overall can be split into frontend and backend: Two industrial grade cameras at the frontend capture video of objects. Video frames from these cameras are then streamed to the backend for detection. Multiple object detection models were tested to study their performance with the focus on PCN, whose main features are highlighted as the following:

- FPN: The feature extraction backbone at stage one. It outputs heatmap for region proposals and feature map for further predictions at stage 2.
- Proposal connection: Augmenting region proposals at training stage. It uses less region proposals and is more efficient.
- Class loss function: PCN employs focal loss at stage one and class weights at stage two to alleviate class imbalance problems.

The experiment has compared and analyzed their key performance metrics related to accuracy, precision and speed. The main observations made from the experimental data are summarized as follows:

1. PCN has comparable accuracy and MAP to some state-of-the-art object detection models tested, namely YOLO and Faster R-CNN.

2. PCN has sufficient speed to perform object detection in real time. It has lower FPS than the fastest model but considerably higher accuracy and MAP.

3. PCN has less number of parameters, indicating less complexity and easier deployment.

4. Ablation study reveals that the class weight is the major contribution to improved accuracy and MAP.

6.2 Future Works

Testing these models has given us satisfying results. It proves the object detection and subsequently the entire product inspection system can function well and has potentially practical value. In addition, there exist more aspects of object detection to explore and extend related research in the future. Some future research ideas are presented as follows:

1. PCN currently adapts two-stage model to implement ROI augmentation. In the future, augmenting and making predicitions may be made from heatmap directly so that the model can become one-stage.

2. Class imbalance problem has been tackled by PCN but there are still places to improve. Since augmentation is a key feature of PCN, it may also be resolved by by directly adding the number of less training samples through this method.

3. During the experiment, objects are placed in front of the cameras under the same setup so that the distance to objects, environmental lighting and other factors remain unchanged. It would be interesting to see how PCN performs after introducing more variations, including objects at different scales and angles, dark environment, and so on.

References

[1] R.C. Gonzalez and R.E. Woods. *Digital Image Processing.* Prentice Hall, 2008.

[2] M. Sonka, V. Hlavac, and R. Boyle. *Image Processing, Analysis, and Machine Vision.* Cengage Learning, 2014.

[3] Athanasios Voulodimos, Nikolaos Doulamis, Anastasios Doulamis, Eftychios Protopapadakis, et al. Deep learning for computer vision: A brief review. *Computational intelligence and neuroscience*, 2018, 2018.

[4] Amarjot Singh, Ketan Bacchuwar, and Akshay Bhasin. A survey of ocr applications. *International Journal of Machine Learning and Computing*, 2(3):314, 2012.

[5] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[6] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[7] Zhong-Qiu Zhao, Peng Zheng, Shou-Tao Xu, and Xindong Wu. Object detection with deep learning: A review. *IEEE Transactions on Neural Networks and Learning Systems*, 30(11):3212–3232, 2019.

[8] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2016.

[9] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger, 2016.

[10] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement, 2018.

[11] Glenn Jocher et al. ultralytics/yolov5: v3.1 - Bug Fixes and Performance Improvements, oct 2020.

[12] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection, 2017.

[13] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 international conference on engineering and technology (ICET)*, pages 1–6. Ieee, 2017.

[14] Hei Law and Jia Deng. Cornernet: Detecting objects as paired keypoints. In *Proceedings of the European conference on computer vision (ECCV)*, pages 734–750, 2018.

[15] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. Objects as points, 2019.

[16] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection, 2018.

[17] Zhengmin Kong, Hui Ouyang, Yiyuan Cao, Tao Huang, Euijoon Ahn, Maoqi Zhang, and Huan Liu. Automated periodontitis bone loss diagnosis in panoramic radiographs using a bespoke two-stage detector. *Computers in Biology and Medicine*, 152:106374, 2023.

[18] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[19] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

[20] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2016.