# Swimming pool detection from Aerial imagery

Anthony TAING
Stanford University
Department of Computer Science
anth248@stanford.edu

## Abstract

*State-of-the-art object detection methods are in two major groups: one stage like YOLOs and double stage with region proposal based methods like Faster R-CNN. The goal of this project is to detect the swimming pool from aerial imagery using Deep Learning. We study different techniques to detect the swimming pools in the aerial imagery. We implement FasterRCNN as well as other algorithms like CornerNet, which detects an object bounding box as a pair of keypoints, and DynamicHead which aims to unify object detection heads with attentions. These models have demonstrated very good performance on common benchmark like COCO [9]. We experiment on a custom dataset publicly available on the Kaggle platform. We show that CornerNet and DynamicHead models are still better than FasterRCNN model with our dataset focused on the detection of swimming pools.*

## 1. Introduction

Swimming pools are important for property tax assessment because they impact the value of the property. But Tax assessors at local government agencies often rely on expensive and infrequent surveys, leading to assessment inaccuracies. This task would be the detection of swimming pools using aerial imagery which is a better solution. It can also benefit swimming pool maintenance companies to help redirect their marketing efforts or for Public health and mosquito control agencies to detect pools and drive field activity and mitigation efforts. Many object detection rely on backbone models that were pretrained on famous dataset like Imagenet. Therefore, these models can recognize every objects learned during this training, however detecting differents objects that had never been learned before is very challenging. We will implement a swimming pool detection by using models that outperformed some common benchmark like COCO [1]. We will use some aerial images and try to detect swimming pools.More specifically, our goal is to improve the performance of Faster R-CNN by using other models and show the efficiency of these methods on this particular task. We use a dataset of patches of satellites images and detect swimming pools by using bounding boxes.

### 1.1. Aerial Imagery

Satellites images are widely used for many civil applications, such as geographic information system mapping, agriculture, traffic planning, and navigation.... There are a large number of object in a remote sensing image. However, the task has numerous difficulties such as the massive variations in the scale and orientation of objects, the extremely nonuniform object densities and large aspect ratios. Since this is very challenging, some benchmarks with their own datasets of aerial images have been created like DOTA [3] with 1,793,658 object instances of 18 categories or FAIR1M [14] with 1.02 million instances and 37 categories. They allow us to detect multiples categories of objects without the help of large-scale datasets of natural images, such as MS COCO and ImageNet. Therefore, some researchers have used aerial images to detect any objects like buildings,roads, ships, cars, planes... or experiment on a particular domain: to detect deforestation in Amazonia or the detection of flooded roads and buildings [11].

## 2. Related work

CNN based object detection algorithms can be divided into two major categories: we can find Single stage detector like YOLO,SSD and Two stage detector like RCNN and some variants.

### 2.1. Region Proposal Based Algorithms

Two-stage detectors generate a sparse set of regions of interest (RoIs) and classify each of them by a network, they was introduced by the R-CNN [6] or Region-based Convolutional Neural Network. This is composed of three parts: the first part generates region proposals using selective search. The second part is feature extraction through

[1] https://cocodataset.org/#home

CNN, in each proposed region. The third part is classification by SVMs, it also computes offset values to help in adjusting the bounding box of the region proposal.

Next, Fast R-CNN [5] improved the previous model by using several innovations to improve training and testing speed and performance.The authors introduced the RoI pooling layer which uses max pooling to convert the features inside any valid region of interest into a small feature map. Fast R-CNN uses a training process with one fine-tuning stage that jointly optimizes a softmax classifier and bounding box localisation with a linear regressor. R-FCN [1] replaces the fully connected layers with the position-sensitive score maps to incorporate translation variance into FCN(Fully Convolutional Network) for better detecting objects.The model ends with a position-sensitive RoI pooling layer, which aggregates the outputs of the last convolutional layer and generates scores for each RoI.

### 2.2. Regression Based Algorithms

On the other hand, one-stage detectors are usually more computationally efficient than two-stage detectors. These approaches remove the RoI extraction process and directly classify and regress the candidate anchor boxes. The YOLO [12] model uses a single convolutional network simultaneously to predict multiple bounding boxes and class probabilities for those boxes. Its design enables end-to-end training and realtime speeds while maintaining high average precision. Then many variants are available and improve the first one. For example, YOLOv2 uses more anchor boxes and a new bounding box regression method.

Furthermore, SSD [10] or Single Shot MultiBox Detector is also faster, it removes the use of the region proposal network and instead makes use of multi-scale features and default boxes. The network combines predictions from multiple feature maps with different resolutions to naturally handle objects of various sizes. Then CenterNet [4] is a keypoint-based approach, which directly detects an object using a triplets of keypoints. This model also implement two customized modules named cascade corner pooling and center pooling, which play the roles of enriching information collected by both top-left and bottom-right corners and providing more recognizable information at the central regions. This model outperforms all existing one-stage detectors by at least 4.9%.

## 3. Dataset

We use the dataset available on Kaggle [2], that contains 3,197 annotated pools with different shapes and hues. The original tile size is 25,000 x 25,000 pixels. Then it is cropped into patches of 512x512 pixels without overlaps

[2] https://www.kaggle.com/datasets/alexj21/swimming-pool-512x512

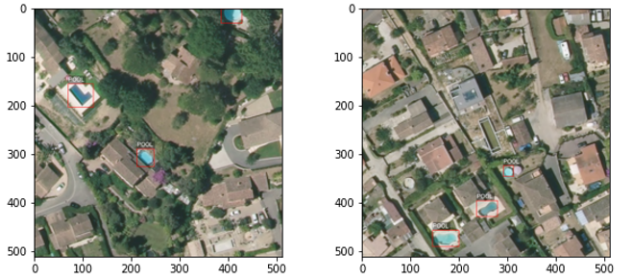

Figure 1. Example of images with detected swimming pools

and there are 1,224 patches. The patches have three bands (RGB) and the pixel values lie in the range [0, 255]. There are some existing preprocessing steps that extract information related to bounding boxes for each images from xml files then we stored in csv file, a structured format. But in order to experiment our new models, we had to implement other preprocessing steps that fit to each model, because the coordinates of bounding boxes are represented in different ways, they use either the COCO or PASCAL VOC[3] format. So, we get [xmin,ymin,xmax,ymax] for PASCAL VOC and [xmin,ymin,width,heigth] for COCO format. Then we split once the data using kfold validation and used the same split for all experiments. After we use Dataloaders to load data and experiment our models with some parameters like batch size. This pipeline is adapted to each model that we are trying to experiment. Therefore, it seems difficult to train theses selected models because they usually have been experimented with common benchmark datasets, so by using our custom datasets, we had several issues and more preprocessings steps to apply. During training, we apply an image transformations with HueSaturationValue,RandomBrightnessContrast, Horizon and vertical flips,random rotate, transpose, compression, resize and cutout. I also use cutmix augmentation in testing, we cut images and generate new ones with 4 to 6 samples of all images, using different orientations and size. We can see the input image in Figure 1 and an example of this augmentation in Figure 2. We can also visualize the number of annotated swimming pools per image in the Figure 3 .

## 4. Methods

### 4.1. Faster R-CNN

Faster R-CNN [13] improved the object detection architecture by replacing the selection search algorithm in Fast R-CNN [5] with a convolutional network called the Region Proposal Network (RPN).

It is a two stage object detector illustrated on the Figure 5. The entire system is a single, unified network for ob-

[3] host.robots.ox.ac.uk/pascal/VOC/

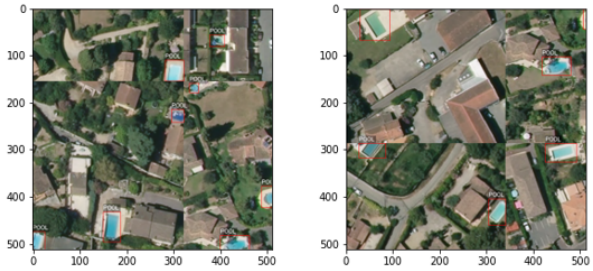

Figure 2. Example of image with data augmentation

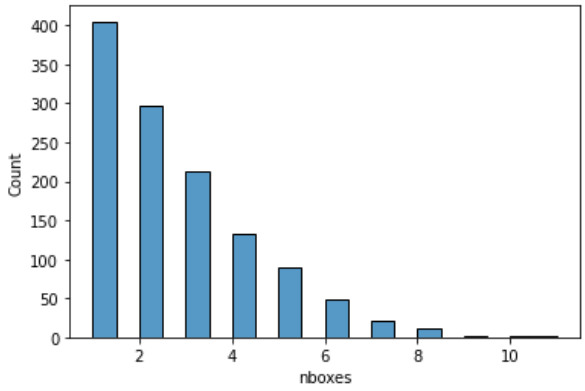


Figure 3. Number of boxes per image

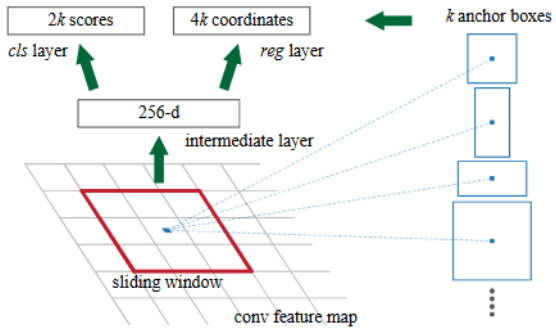


Figure 4. Example of a Region Proposal

ject detection. The first step consist to the Region Proposal Network illustrated in Figure 4, we construct an RPN by adding a few additional convolutional layers that simultaneously regress region bounds and objectness scores at each location on a regular grid. To generate region proposals, we slide a small network over the convolutional feature map output.

Next, each sliding window is mapped to a lower-dimensional feature and fed to n × n convolutional layer followed by two 1 × 1 convolutional layers for the regression and classification. This predict efficiently region proposals

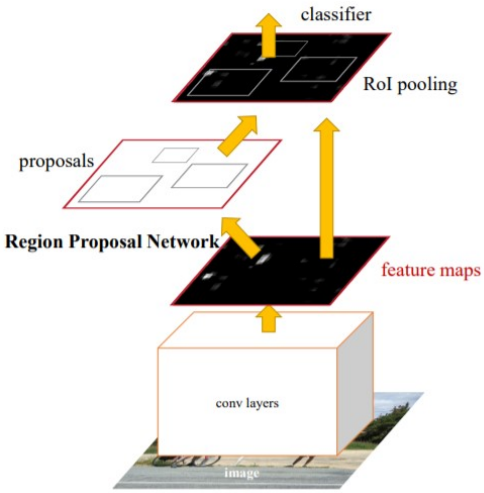


Figure 5. Illustration of Faster R-CNN

with a wide range of scales and aspect ratios.

Furthermore, the RPN and the detect network share the same backbone and the last shared layer of the backbone provides a feature map of the image that is used by the RPN to propose regions. Then at each sliding-window location, we simultaneously predict multiple region proposals called anchors. The outputs of the regressor layer and classifier are proportional to the numbers of proposals, and the output of the ”reg” layer encode the coordinates of k boxes, and the ”cls” layer outputs estimate probability of object or not object for each proposal.

Next, this model uses 3 scales and 3 aspect ratios, yielding 9 anchors at each sliding position and an anchor is centered at the sliding window and is associated with a scale and aspect ratio. Consequently, the anchor-based method is built on a pyramid of anchors, which is more cost-efficient.

Finally, Faster R-CNN combines two loss, one for the regression and another for the classifier, the regression loss is activated only for positive anchors and these two terms are normalized and weighted by a balancing parameter $\lambda$ or by the mini-batch size and the number of anchor locations as followed:

$$L(p_i, t_i) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*)$$

With $i$ the index of an anchor in a mini-batch and $p_i$ is the predicted probability of anchor i being an object, the ground-truth label $p$. $ti$ is a vector representing the 4 parameterized coordinates of the predicted bounding box, and $t_i^*$ is that of the ground-truth box associated with a positive anchor. Then the term $p_i * L_{reg}$ means the regression loss is activated only for positive anchors $i$.

The rest of the model architecture remains the same as

Fast R-CNN [5], the image is fed to a CNN to produce a feature map from which features for regions proposed by the RPN are selected and resized by a pooling layer and fed to an FC layer with two heads, a softmax classifier, and a bounding box regressor. [13]

### 4.2. CornerNet

The CornerNet model [7] [8] is a new one-stage approach, which detects an object bounding box as a pair of keypoints without using anchor boxes, the top-left corner and the bottom-right corner, using a single convolution neural network. This model also used a new type of pooling layer called corner pooling that helps the network better localize corners.

This approach uses a single convolutional network to predict a heatmap for the top-left corners of all instances of the same object category, a heatmap for all bottom-right corners, and an embedding vector for each detected corner. The embeddings serve to group a pair of corners that belong to the same object and the network is trained to predict similar embeddings for them.

Consequently, the network predicts an embedding vector for each detected corner such that the distance between the embeddings of two corners (the top-left and bottom-right) from the same object is small. Then we can group the corners based on the distances.

Next, to produce tighter bounding boxes, the network predicts offsets to slightly adjust the locations of the corners. Finally, given our predicted heatmaps, we apply a final post-processing step to obtain the final bounding boxes.

Besides, the corner pooling is also proposed because a corner of a bounding box is often outside the object, thus this requires to look horizontally towards the right for the topmost boundary of the object, and look vertically towards the bottom for the leftmost boundary. It is based on max pooling feature vectors to the right from the first feature map, and on feature vectors directly below from the second feature map. Then we add the two pooled results together. This can be visualized on Figure 6.

The Corner model uses the hourglass network as backbone (capturing global and local features in a single unified structure) followed by two prediction modules, one for the top-left corners, and the other for the bottom-right corners. Then we use corner pooling in each module to pool features from the hourglass network. The final result is a block of convolutions and pooling layers, and with 3 final branches to predict the heatmaps, embeddings and offsets as illustrated on Figure 7. Next, the key point with this model is during training, we try to reduce the penalty given the negative locations within a radius of the positive location instead of equally penalizing negative locations. The amount of penalty reduction is given by an unnormalized 2D Gaussian. When using the embeddings, we use two additional

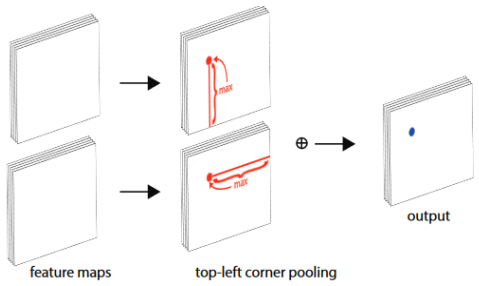


Figure 6. Illustration of Corner pooling, we take the maximum values (red dots) in two directions (red lines), each from a separate feature map, and add the two maximums together (blue dot)

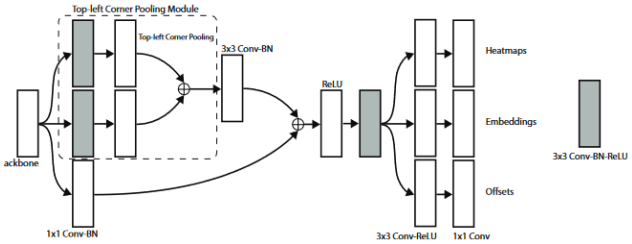


Figure 7. Illustration of CornerNet. The prediction module starts with a modified residual block, using our corner pooling module. The modified residual block is then followed by a convolution module and multiple branches for predictions.

loss: the "pull" loss to train the network to group the corners and the "push" loss to separate the corners with:

$$L(pull) = \frac{1}{N}\sum_{k=1}^{N}[(e_{tk} - e_k)^2 + (e_{bk} - e_k)^2$$

$$L(push) = \frac{1}{N(N-1)}\sum_{k=1}^{N}\sum_{j=1,j\neq k}^{N} max(0, \Delta - |e_k - e_j|)$$

where $e_k$ is the average of $e_{tk}$ and $e_{bk}$ and $\Delta$ is set to 1.

### 4.3. Dynamic Head, Unifying Object Detection Heads with Attentions

The goal of the dynamic head framework [2] is to unify object detection heads with attentions by combining multiple self-attention mechanisms between feature of 3 levels for scale-awareness, among spatial locations for spatial-awareness, and within output channels for task-awareness. In other words, this means that the model is scale-aware, it takes into account the fact that multiple objects with different scales often co-exist in an image, dimension of level. It learns features of different scales based on their semantic importance.

Then spatial-aware, because objects are with different shapes, rotations, and locations under different viewpoints,

it learns representations in spatial locations using the dimension of space (i.e., height × width). Thirdly, the head needs to be task-aware, since objects can have various representations (e.g., bounding box, center, and corner points) with specific objectives and constraints, it uses different feature channels.

Consequently, we can deploy attention mechanisms separately on each particular dimension of features, i.e., level-wise, spatial-wise, and channel-wise. Given the feature tensor $F \in R^{L*S*C}$ the general formulation of applying self-attention is

$$W(F) = \pi(F) * F$$

So we need to convert the attention function into three sequential attentions, each focusing on only one perspective

$$W(F) = \pi_C(\pi_S(\pi_L(F) * F) * F) * F$$

with three different attention functions applying on dimension L, S, and C respectively.This mechanisms is illustrated on the Figure 9. Scale-aware attentions is expressed by the following:

$$\pi_L(F) * F = \sigma(f(\frac{1}{SC} * \sum_{S,C} F)) * F$$

with a linear function approximated by a 1 × 1 convolutional layer, and a hard-max sigmoid function.

Furthermore, spatial-aware attention are based on regions co-existing among both spatial locations and feature levels. It is divided into two steps: first making the attention learning sparse by using deformable convolution (based on grid augmentation with offsets) and then aggregating features across levels at the same spatial locations.We have

$$\pi_S(F) * F = \frac{1}{L} \sum_{L=1}^{L} \sum_{k=1}^{K} w_{l,k} * F(l, p_k + \Delta_{p_k}; c) * \Delta_{m_k}$$

where K is the number of sparse sampling locations, $pk + \Delta_{pk}$ is a shifted location by the self-learned spatial offset, $\Delta_{pk}$ to focus on a discriminative region and $\Delta_{mk}$ is a self-learned importance scalar at location pk. Both are learned from the input feature from the median level of F.

Last, for the task-aware attention, it selects channels of features to favor different tasks separately,

$$\pi_C(F)*F = max(\alpha_1(F)*F_c+\beta_1(F), \alpha_2(F)*F_c+\beta_2(F))$$

where Fc is the feature slice at the c-th channel and $[\alpha_1, \alpha_2, \beta_1, \beta 2]^T = \theta(.)$ is a hyper function that learns to control the activation thresholds by using a global average pooling on L × S dimensions to reduce the dimensionality, two fully connected layers, a normalization layer, and finally applies a shifted sigmoid function to normalize the output to [1, 1].

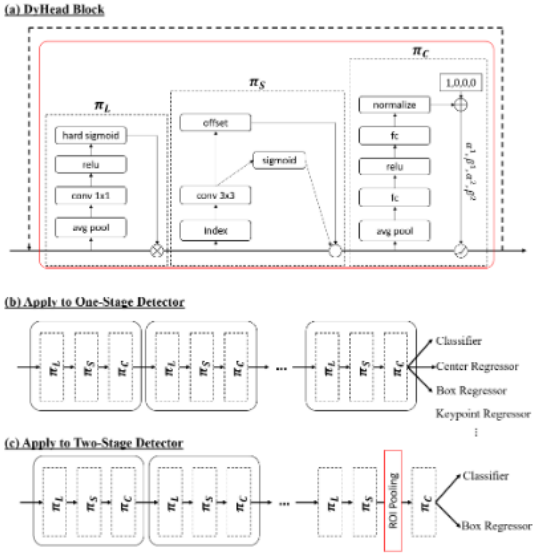


Figure 8. Design of the Dynamic Head. (a) the detailed implementation of each attention block. (b) design with one-stage object detector. (c) design with two-stage object detector

Moreover, these attentions can be stacked sequentially multiples times to any kinds of backbone network that are used to extract feature pyramid, because they resized to the same scale, forming a 3-dimensional tensor$F \in R^{L*S*C}$, and then used as the input to the dynamic head. Furthermore, the output can be used for different tasks and representations of object detection. This is represented in Figure 8.

For example in 2 stage detectors, we first apply our scale-aware attention and spatial-aware attention on feature pyramid before a ROI-pooling layer and then use our task-aware attention to replace the original fully connected layers.

We can also improve performance by individually adding each component to the baseline implementation.

Finally, this proposed dynamic head model can combine multiple attentions from different perspectives into a unified head, resulting in better efficiency. And this can be plug into an existing object detector framework to boost its performance.

## 5. Experiments

Experiments have been tested on Google Collab environment with GPU Tesla V100 then on AWS with a p2.xlarge ec2 instance.

The implementation of faster R-CNN are adapted using the pretrained version available with Pytorch library. We implement DynamicHead [2] and CornerNet [7] following their respective papers and links of their github repositories. We have adapted our implemented code to run these models with our dataset. For DynamichHead, we use a

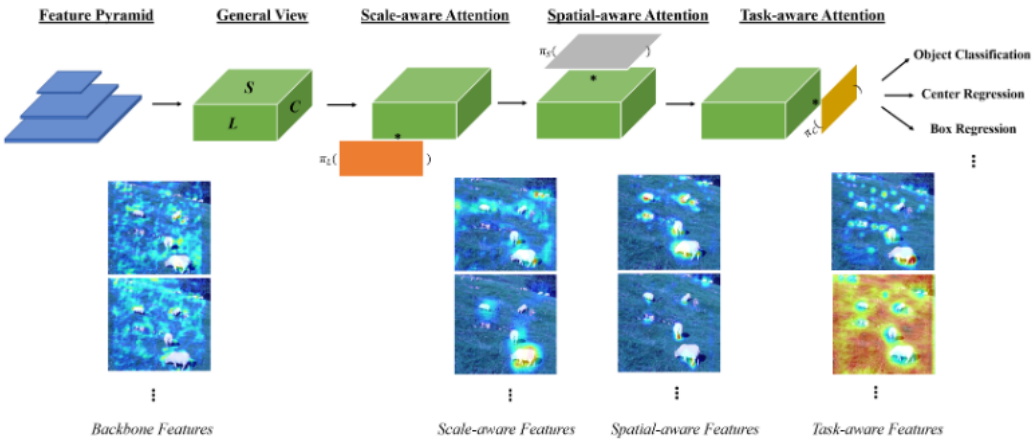


Figure 9. Example of a Dynamic Head approach. With three different attention mechanisms, each focusing on a different perspective: scale-aware attention, spatial-aware attention, and task-aware attention.

ResNet-50 as the model backbone and default parameters with an initial learning rate of 0.02 with weight decay of $1e^{-4}$ and momentum of 0.9. On the other hand, for CornerNet, we use default settings, with Adam optimizer, data augmentation techniques: random horizontal flipping, random scaling, random cropping and random color jittering (which includes brightness, saturation, contrast of an image) and PCA to the input image.

We apply popular metrics with the precision, recall and F1score as metrics.

$$Precision = \frac{\textbf{True Positive}}{\textbf{True Positive+False Positive}}$$

$$Recall = \frac{\textbf{True Positive}}{\textbf{True Positive+False Negative}}$$

$$F1score = 2 * \frac{\textbf{Precision*Recall}}{\textbf{Precision+Recall}}$$

Besides, we use a cross-validation with 5 folds, and use a scheduler on LRplateau, a batch size 4, number of epochs 5, the AdamW optimizer. For Faster RCNN, we keep the default parameters used in the Pytorch library. We use the pretrained weights available for the CornerNet and DynamicHead models.

We experiment the CornerNet-Saccade, which uses an attention mechanism to eliminate the need for exhaustively processing all pixels of the image, it is a subset of the variant CornerNet-Lite which combined two efficient variants: CornerNet-Saccade and CornerNet-Squeeze, which introduces a new compact backbone architecture.

### 5.1. Results

The final results are shown in Table 1. We found what we expected at the beginning of this project, CornerNet and DynamicHead models have better scores than FasterRCNN in our detection task. To better understand the implemented models, we visualize some object detected. We can see an example of images used in FasterRCNN in input Figure 10 and the RoI detected from the feature maps on Figure 11. After this first detection, the model will rescale the predicted bounding boxes to fit correctly the swimming pools. However, we found some errors of predictions, this can be illustrated with the Figure 11. We can see incorrect bounding boxes of swimming pools on the left. The model has detected some circle and rectangular objects that look like indoor swimming pools but they are not. Next we can find other particular images where the detection was difficult due to the environment and lead to a bad prediction, the model FasterRCNN and CornerNet predicted the blue zone as swimming pool in Figure 12 but in reality, this is some parking slots dedicated to people with disabilities, they have the same shape and color than a swimming pool. But they correctly identified Figure 13 with no swimming pools. This last example was easier because during training all models have learnt to recognize swimming pools surrounded by some houses and vegetation. The other implemented model solve this issue and achieve better performance with 0.95 precision and 0.94 F1score because DynamicHead is scale-aware, spatial-aware and task aware, so it is more robust to any variations in the image.

| Method | Precision | Recall | F1 score |
|---|---|---|---|
| Faster R-CNN | 0.88 | 0.95 | 0.90 |
| CornerNet | 0.92 | 0.92 | 0.92 |
| Dynamic Head | 0.95 | 0.93 | 0.94 |

Table 1. Performance of models.

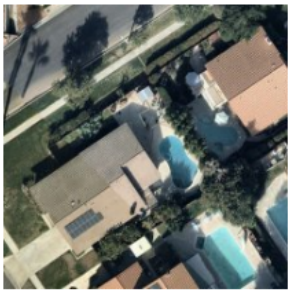

Figure 10. Input of FasterRCNN

Figure 11. RoI are detected from the feature maps(FasterRCNN)

Figure 12. Example of wrong prediction

The great performance of DynamicHead can be explained by the ability of the model to consider multiple objects with different scales in each image and by capturing different positions, orientations and contrast. This model is more robust to any changes applied during training with the data augmentation. Indeed, in our dataset, we can have image with no swimming pool and others images with a single or multiples swimming pools with different size and shape (circular, curve, rectangular).

## 6. Conclusion

In conclusion, we introduced object detection with multiples cutting-edge pretrained models FasterRCNN, CornerNet, and DynamicHead. We achieved best performance with DynamicHead and CornerNet compared to FasterRCNN on this specific object detection task, they were able

Figure 13. Example of bias correctly predicted

to consider differents variations and scales in images. Next step is to further improve our models by using their variants, which have been demonstrated to be more efficient on common benchmark. We can expect to experiment DynamicHead model with FasterRCNN or any other models because we can use it as a plugin block.

## 7. Acknowledgements

We thank the author of the dataset for providing publicly some labelled images of swimming pools from the aerial imagery. We are thankful for CS231 teaching staff for this wonderful class.

## References

[1] J. Dai, Y. Li, K. He, and J. Sun. R-FCN: object detection via region-based fully convolutional networks. *CoRR*, 2016. 2

[2] X. Dai, Y. Chen, B. Xiao, D. Chen, M. Liu, L. Yuan, and L. Zhang. Dynamic head: Unifying object detection heads with attentions, 2021. 4, 5

[3] J. Ding, N. Xue, G. Xia, X. Bai, W. Yang, M. Y. Yang, S. J. Belongie, J. Luo, M. Datcu, M. Pelillo, and L. Zhang. Object detection in aerial images: A large-scale benchmark and challenges. *CoRR*, 2021. 1

[4] K. Duan, S. Bai, L. Xie, H. Qi, Q. Huang, and Q. Tian. Centernet: Keypoint triplets for object detection. *CoRR*, abs/1904.08189, 2019. 2

[5] R. B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015. 2, 4

[6] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, 2013. 1

[7] H. Law and J. Deng. Cornernet: Detecting objects as paired keypoints. *CoRR*, abs/1808.01244, 2018. 4, 5

[8] H. Law, Y. Teng, O. Russakovsky, and J. Deng. Cornernet-lite: Efficient keypoint based object detection. *CoRR*, abs/1904.08900, 2020. 4

[9] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: common objects in context. *CoRR*, 2014. 1

[10] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg. SSD: single shot multibox detector. *CoRR*, 2015. 2

[11] M. Rahnemoonfar, T. Chowdhury, A. Sarkar, D. Varshney, M. Yari, and R. R. Murphy. Floodnet: A high resolution aerial imagery dataset for post flood scene understanding. *IEEE Access*, 9:89644–89654, 2021. 1

[12] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. *CoRR*, 2015. 2

[13] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2015. 2, 4

[14] X. Sun, P. Wang, Z. Yan, F. Xu, R. Wang, W. Diao, J. Chen, J. Li, Y. Feng, T. Xu, M. Weinmann, S. Hinz, C. Wang, and K. Fu. FAIR1M: A benchmark dataset for fine-grained object recognition in high-resolution remote sensing imagery. *CoRR*, 2021. 1