State of the Art: Automated object recognition frameworks in satellite imagery and geophysical surveys

Martin Olivier

May 6, 2020

Contents

List of Figures

List of Tables

1 Introduction

This documents presents a concise, but not comprehensive state-of-the-art on automated detection in satellite imagery, as of the 23rd April 2020. A few recent articles could not be consulted online as of this date, and so are not analyzed in this document, namely: Remote Sensing Object Localization with Deep Heterogeneous Superpixel Features by Yang et al.[1] and BMF-CNN: an object detection method based on multi-scale feature fusion in VHR remote sensing images by Dong et al. [2].

The general direction of the research in this area is toward Deep Convolutional Neural Networks that uses some kind of feature fusion at different layer height of the network. This feature fusion is supposed to improve detection rates in High Resolution Imagery by using information from the lower layers learned features, which has a higher resolution combined with the information from the higher layer features, which possess more semantic information. Using those techniques, researchers are able to obtain much higher precision scores on datasets such as VEDAI[3], NWPU[4] or DOTA[5] than traditional detection frameworks such as YOLOv3[6] o FasterRCNN[7]. In most of the cases, the network is also much faster and smaller, and is able to analyze larger swaths of terrain.

The paper in this document are always presented in the same manner. First a presentation of the general architecture is given. Then the proposed novel modules are introduced, and finally results on the different datasets are given.

1.1 Issues in Automated Detection in Remote Sensing Imagery

Remote sensing, particularly with satellite imagery possess a number of specific problems that often renders existing detection pipelines inefficient be it in terms of Frame Per Seconds, or precision scores.

First, the objects of interest are often very small and densely clustered. This differ from the large objects often seen in ImageNet. In satellite imagery, the absolute resolution can be extremely large, but since those image also cover a very large area. Depending on the source, a pixel can have a physical size of 30 cm for very high resolution image to 3-4 meters. Small objects, such as cars, **will only be 15 pixels** at most with the highest resolution.

Secondly, objects viewed from satellite can have any orientation. This means that **complete rotational invariance** is needed.

Lastly, **the input image size are often extremely large**. Downsampling, which is done by most algorithm to reduce the dimensionality to the feature maps to a reasonable degree[1] is not an option here.

Those issues are similar to the one that can be seen in the field of remote sensing automated detection, in particular with the input image resolution being very high. There is a need for specifically designed algorithms, since traditional methods fail to capture all the objects.

[1]For example, YOLOv3 downsamples an input image up to 32 times

2 You Only Look Twice[8]

You Only Look Twice is a model developped by Adam Van Etten, and is focussing on rapid multi scale detection for satellite imagery. This approach uses a modified YOLO[6] framework. A new backbone architecture is used, with finer grained features and a denser final grid, to better detect the very small objects found in satellite imagery. This approach also uses an ensemble method, where multiple networks are run simultaneously at different scales. Finally, the problem of large input image size is mitigated by partitioning the images using a sliding window.

This network was trained on small snippets or segments of large images from 3 differents sources to detect 5 classes of objects: cars, airplanes, boats,building footprints and airports.

2.1 Network Architecture

The YOLO network has been modified to better detect heavily packed objects, often found in satellite imagery. The YOLT network takes as input a 416×416 pixel image, which it downscales 16 times. The network outputs a 26×26 prediction grid, which is much finer than the 7×7 prediction grid offered by a "regular" YOLO network. **This finer prediction grid is what allows the network to detect densely packed objects.**

A passthrough layer is also used to pass coarse features from the earlier and high resolution layers to the final low resolution layers. This passthrough layer is similar to the identity layer used in ResNet[9] and was first used in YOLOv9000[10].

Each layer uses batch normalization[11] and uses the leaky-ReLU activation[12], except for the last layer, which uses a linear activation. This final layer provides the predictions for the bounding boxes and class. Its output size N_f is computed using the following formula:

$$N_f = N_{boxes} \times (N_{classes} + 5) \tag{1}$$

Where N_{boxes} is the number of boxes per grid square (with the default being 5) and $N_{classes}$ being the number of classes.

2.1.1 Scale Mitigation

The author uses two different detectors on the input images running simultaneously. One is trained to detect small scale objects, like vehicles and building, while the other is trained to detect airports and large structures. The size on the input images of these detector is different, as one takes in 200 meters segments, while the other uses 2000 meters segments.

As there is about 100 times less 2000 meters segment in the original images as there is 200 meters segments, the large scale network runs much less often than the small scale network. This limits the reduction of inference speed that running two detectors would do.

2.2 Training

2.2.1 Data and Preprocessing

The author uses training data from three sources: DigitalGlobe satellites, Planet satellites and aerial platforms. The author also uses some data augmentation techniques, with random rescaling and rotations to get more examples, as the dataset for some classes such as airports or airplanes is small.

2.2.2 Training Hyperparameters

The author trains the network using SGD with an initial learning rate of 0.001, a weight decay of 0.0005 and a momentum of 0.9. This training takes about 2-3 days on a NVIDIA Titan X GPU.

2.3 Results

It should be noted that the author initial tried to train only one detector and obtained very poor results, due to the large scale difference between some of the objects. The results presented here are the one using the two detector approach.

Table 1 shows the F1 Score for each class. It should be noted that while the absolute value of the F1 Score for the building class is lower in comparison the other classes, the best contestant in the SpaceNet Challenge 2, where the contestants where asked to detect building outlines using the same dataset as the one used here obtained a F1-Score of 0.69. This puts this detector in the Top-3.

Object Class	F1 Score
Car	0.90 ± 0.09
Airplane	0.87 ± 0.08
Boat	0.82 ± 0.07
Building	0.61 ± 0.15
Airport	0.91 ± 0.14

Table 1: YOLT Detection performance on all classes

The network is also very fast, being able to analyze $32km^2/min$ for the small scale network and $6000km^2/min$ for the large scale network.

3 Satellite Imagery Multiscale Rapid Detection with Windowed Networks[13]

This article present a detection pipeline that supposedly supercedes previous work by the same author; YOLT[8] presented in section 2. The author introduces a singular framework using not only YOLT but also various other detection models, such as SSD[14], Faster-RCNN[7] and R-FCN[15]. This approach allows the comparison of those different models in the context of object detection in satellite imagery.

3.1 Network Architecture and Training Method

Since the author uses a multitude of different models, we will give only the modifications and parameters chosen for each model.

3.1.1 YOLO

A standard YOLOv2[10] was used with a 13×13 output grid. Each layer uses batch normalization with a leaky ReLU activation. The training was done using an initial learning rate of 0.001, a weight decay of 0.0005 and a momentum of 0.9 using Stochastic Gradient Descent with a batch size of 16 for 60K iterations.

3.1.2 YOLT

The parameters used are similar than the one used in the original paper[8]: model coarseness was reduced by only downsampling by a factor of 16 instead of 32 used in the standard YOLO model. This yield a 26×26 prediction grid. This helps to detect small, densely packed object, often seen in satellite imagery.

A passthrough layer was also included to help detect small objects, that concatenates the final 52×52 layer onto the last convolutional layer.

Training was done using the same hyperparameters used in the YOLO implementation.

3.1.3 SSD

The SSD implementation was done similarly as the one described in a paper comparing the speed and accuracy of various object detectors by Huang et Al[16]. The author also experiments with two different backbone networks: Inception V2 [17] and MobileNet [18].

Training was done using an initial learning rate of 0.004 and a decay rate of 0.95 for 30K iterations with a batch size of 16. The "high resolution" settings were used, using 600×600 pixel images sizes.

3.1.4 Faster-RCNN

Again, the implementation of [16] was used, and uses the ResNet 101 [9]. The author also uses the "high resolution' settings, using 600×600 pixel image sizes.

Training is done with a batch size of 1 and an initial learning rate of 0.0001. The author does not specify the amount of iterations done.

3.1.5 R-FCN

Again, the same hyperparameters as the ones used in [16] are used. The backbone is also a ResNet 101 architecture, with the same parameters as the one used in Faster-RCNN

3.2 Training and Testing Procedure

3.2.1 Data and Preprocessing

The datasets used are the same as in the original YOLT papers, and a more complete description can be found in section 2.

Training was done on a similar timescale, or about 24-48 hours for each of the model tested, and followed the same principle as the original YOLT pipeline, where for each architecture two separate models were trained, one designed for vehicles (or small scale objects in general) and the other for airports (or large scale objects).

Testing was done using a similar procedure as the one used in the original YOLT paper, and a more complete description can also be found in section 2.

3.3 Results

The classifer was run a two different scale, 200m and 5000m. The first scale is designed for vehicles while the larger scale is optimzied for larger infrastructure.

The validation image is broken into appropriately sized segments and passed onto the appropriate classifier. Results from both detectors are combined into one final image, and overlapping detection are merged using Non Maximal Suppresssion.

Results for R-FCN and Faster-RCNN are poor, as it would seem that both models struggle in detecting objects with different sizes, and are very sensible to background conditions. Even with much more longer training runs, up to 300K iterations, different input image size, first stage stride, and batch size, no marked improvement is made over the original hyperparameters described in [16].

Airport Detection is poor for all models. The author argues that this is likely a result of the small training set size for airports, but that YOLO/YOLT do perform better on those objects.

Architecture	mAP	Inference Rate (km^2/s)
Faster RCNN ResNet101	0.23	0.09
RFCN ResNet101	0.13	0.17
SSD Inception	0.41	0.22
SSD MobileNet	0.34	0.32
YOLO	0.56	0.42
YOLT	**0.58**	**0.44**

Table 2: Precision comparison of the different models tested. Inference speed is also presented

Table 2 shows a performance comparison between YOLT and the other tested algorithms. **YOLT obtains the best performance, both in terms of mAP and inference speed.**

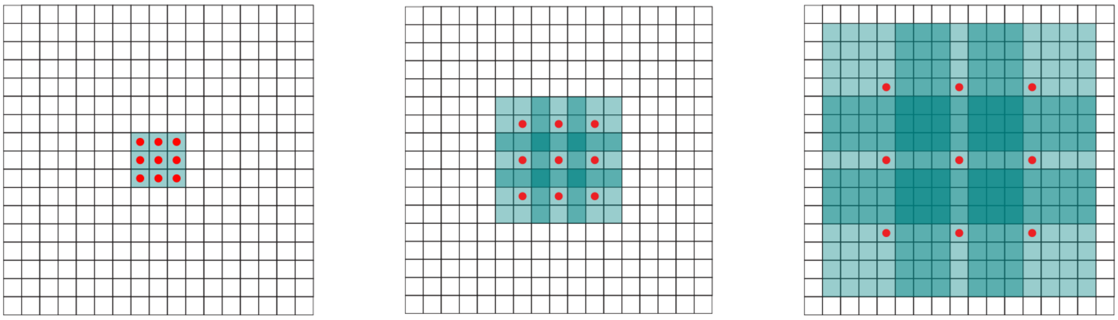
4 A Simple and Efficient Network for Small Target Detection[19]

In this article by Ju et al; the authors try to address the issue of low small target detection performance in classical detection networks. The authors put forth 3 modifications to improve detection performance: first, a "dilated module" that helps to expand the recepetive field of convolutional layers without loss of resolution. Secondly, feature fusion is applied on the feature maps of different layers of the network . Finally, a passthrough layer, similar to the one described in You Only Look Twice[8], described in section 2 and in ResNet[9] is applied to get the finer-grained information from the earlier layers and the more semantic information coming out of the deeper layers.

The performance of the network is evaluated on the VEDAI[3] dataset along with the DOTA[5] dataset, and obtains state of the art results, with FPS performance comparable to a tiny YOLOv3 network[6] but with average precision comparable to a "full size" YOLOv3 network.

4.1 Proposed Modules

4.1.1 Dilated Modules



(a) 1-Dilated Convolution (b) 2-Dilated Convolution (c) 4-Dilated Convolution

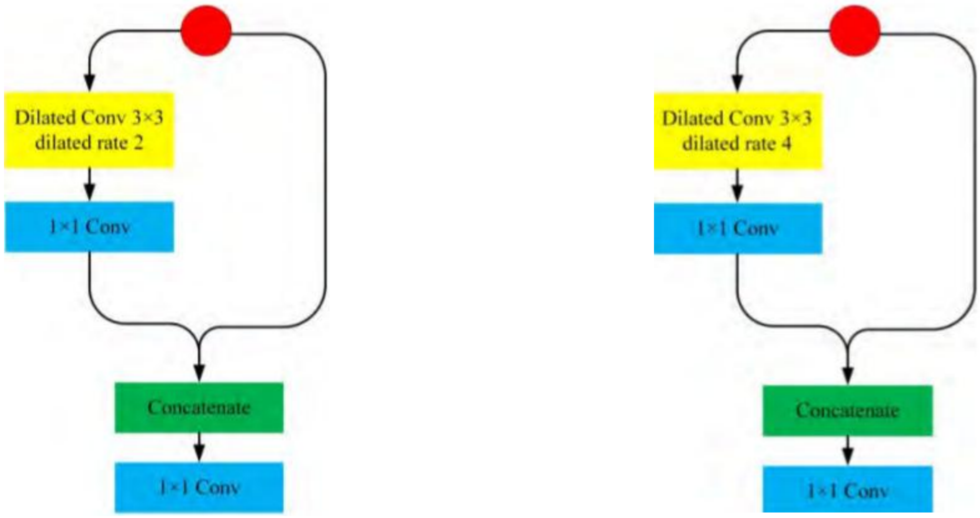
Figure 1: Dilated Convolution Example

Dilated convolution are used to expand the receptive field of the convolution operation without increasing the size of kernel or without reducing the size of the feature maps and losing information about small targets. Figure 1 shows the dilatation of a convolution kernel and the impact on the receptive field. Dilated Convolution has been introduced by Yu and Koltum in "Multi-scale Context Aggregation by Dilated Convolutions"[20].

Dilated modules are used to help to locate the small targets accurately and aggregate multi-scale contextual information. Dilated convolution is used as a basic element to build a dilated module. The module reuse features from earlier and deeper layer by concatenation. A 1×1 convolution is used to reduce the dimension of the module, as can be seen in figure 2.

4.1.2 Passthrough module

In a detection network, earlier layer contains more fine grained information which can be usefull to detect and accuretaly determine the location of small objects. Usually this information is "lost" in the deeper layers. A passthrough layer with a stride of 2 is used to utilize those earlier features. The passthrough layer transform the feature map from a $2N \times 2N \times C$ to $N \times N \times 4C$



(a) Module A with a 2-dilated convolution (b) Module B with a 4-dilated convolution

Figure 2: Dilated Module details

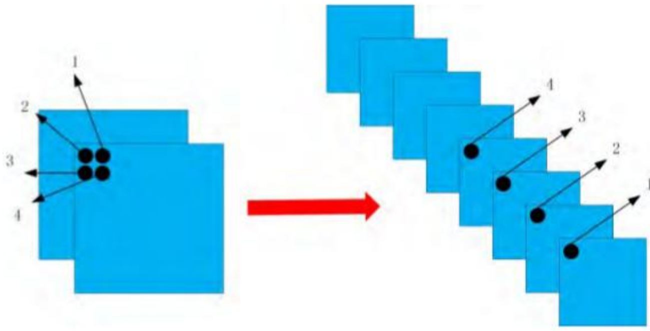


Figure 3: Passthrough layer

as shown in figure 3. This passthrough layer is used to construct the passthrough module. This module merges features from earlier layers with the ones in the deeper layers. Again, a 1×1 convolution is used to reduce the dimension of the module. Figure 4 shows the architecture of the module.

4.1.3 Feature Fusion

Here, concatenation is used to merge features from earlier layers with ones coming from deeper layers. There are two different kind of fusion used in this paper.

The first is concatenating the feature maps between different dilated modules, as can be seen in figure 5. Since the dilated modules don't change the dimension of the feature maps, the merging can be directly done by concatenation.

The second kind is the passthrough layer described in figure 3. Since the feature maps undergoes downsampling, their dimensions changes. The paper propose to unify their dimension by using another passtrough layer and upsampling.

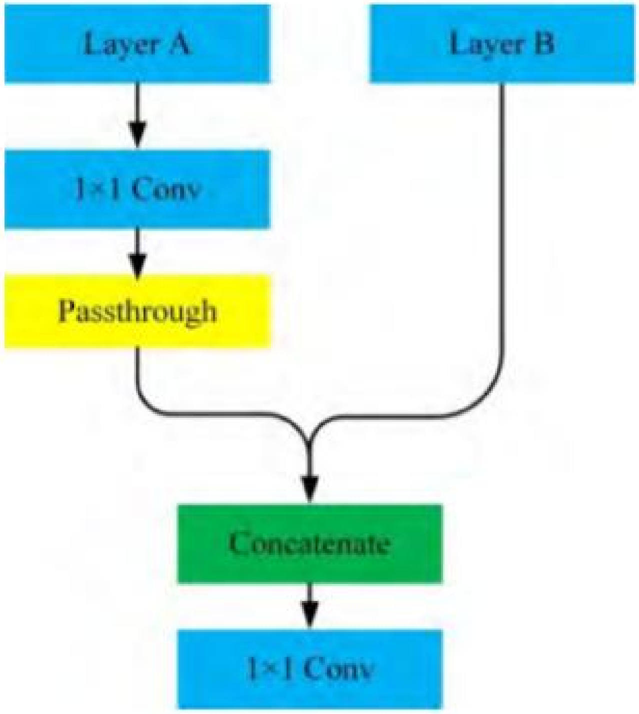


Figure 4: Passthrough module

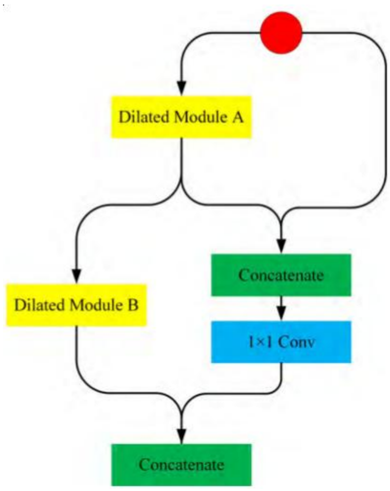


Figure 5: Feature fusion between the different dilated modules

4.2 General Architecture

The proposed architecture is inspired by the tiny YOLOv3, but uses deeper layers along with dilated modules and feature fusion. 1×1 convolution are used to reduce the dimensions, which helps increase the speed and efficiency of the network.

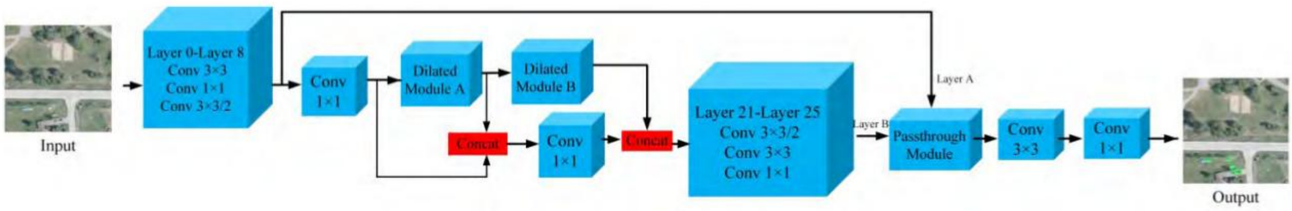


Figure 6: General Architecture

Since the goal of the network is to detect small targets, large downsampling as the one used in YOLOv3 are not adequate. However, the number of downsampling layers affects the size of the receptive field, which in turns determines the amount of contextual information of small targets. Two dilated modules are used to expand the receptive field. The feature maps are downsampled twice and used as fine grained infromation and combine with feature maps that are downsampled thrice using a passthrough modules.

The final layer provides the results of the prediction, which contains the location of the bounding box, and the class of the targets. The size of the last layer is $N = N_{boxes} \times (N_{classes} + 4 + 1)$ with 4 being the number of offsets for the bounding boxes, and 1 for the "objectness" prediction.

4.3 Results

The models were trained and tested on the VEDAI[3] and DOTA datasets[5]. The authors notes that the targets in VEDAI are smaller than the ones in the DOTA datasets, but the quantity of targets in DOTA are higher than of VEDAI.

Detection Algorithm	Input	TP	FP	FN	P	R	AP
YOLOv2	512×512	283	296	147	48.9%	65.8	57.33
Tiny YOLOv3	512×512	307	305	123	50.2	71.4	58.17
YOLOv3	512×512	**373**	**69**	**57**	**84.3**	**86.7**	**85.37**
Proposed Model	512×512	362	88	68	80.5	84.2	80.16

Table 3: Results of each detection algorithm on VEDAI

To obtain a good comparison between existing architectures and the proposed model, the author ran 4 experiments on both datasets, using YoloV2, Tiny YOLOv3, YOLOv3 and their own model. The same size of input (512×512) was used on each model.

Results can be seen in table 3 for the VEDAI datasets, and in table 4 for the DOTA datasets. Table 5 shows a performance comparison between all tested algorithms.

Detection Algorithm	Input	TP	FP	FN	P	R	AP
YOLOv2	512×512	1472	438	439	77	77	72.74
Tiny YOLOv3	512×512	1557	536	354	74.4	81.5	73.2
YOLOv3	512×512	1750	**228**	161	**88.47**	91.6	88.31
Proposed Model	512×512	**1753**	278	**158**	86.5	**91.7**	**88.63**

Table 4: Results of all tested algorithm on the DOTA dataset.

We should note that while YOLOv3 tends to obtain better scores in AP, the computing cost associated with running this algorithm in much higher, as it requires ten times more BFLOPs than the proposed model (see Table 5. This complexity is reflected in the number of Frames Per Seconds that is able to be computed. In short, it seems that the proposed method is able to obtain results similar, if slightly inferior than YOLOv3 but is much faster and has much less parameters than both YOLOv3 and tiny YOLOv3.

Object Detection Algorithm	YOLOv2	Tiny YOLOv3	YOLOv3	Proposed Model
Input	512×512	512×512	512×512	512×512
Model Size	202.3M	34.7M	236.3M	2.8M
FPS	58.3	76.4	14.7	75.4
BFLOPs	44.417	8.243	101.784	9.692

Table 5: Comparative results of FPS, BFLOPs and Model Size with all tested algorithms. BFLOPS refer to the number of billions of floating points operations needed to calculate the prediction.

5 Object Detection in Remote Sensing Images Based on Improved Bounding Box Regression and Multi-Level Features Fusion[21]

This article by Qian et al. aims to solve two issues prevalent in anchor-based detection methods: First the loss of low level information when using only the highest level feature maps for the feature extraction of region proposal. Secondly, existing metrics, such as IoU, are not able to measure the distance between two non overlapping bounding boxes. During training, the bounding box loss is not able to directly optimize this metric.

The authors implements a new metric, the Generalized IoU (GIoU), which is able to measure the distance between non-overlapping bounding boxes, along with a bounding box loss system that is able to directly optimize the new metric. A new multi-level feature module (MLFF), is proposed, and incorporated into an existing network.

This allows the authors to reach state of the art performance on the NWPU VHR-10 dataset[4].

5.1 Architecture

5.1.1 General Network Architecture

The network can use an arbitrary size image as an input. This image is fed into a FPN, which acts as the backbone of the network. This FPN outputs multi-scale feature maps at different levels. Those multi-scale feature maps are used by the MLFF, which pools features using RoIAlign[9] across multiple levels and concatenates them along the channel dimension. The fused features are utilized for bounding box regression and classification. The novel generalized IoU is used, instead of the smooth L1 loss.

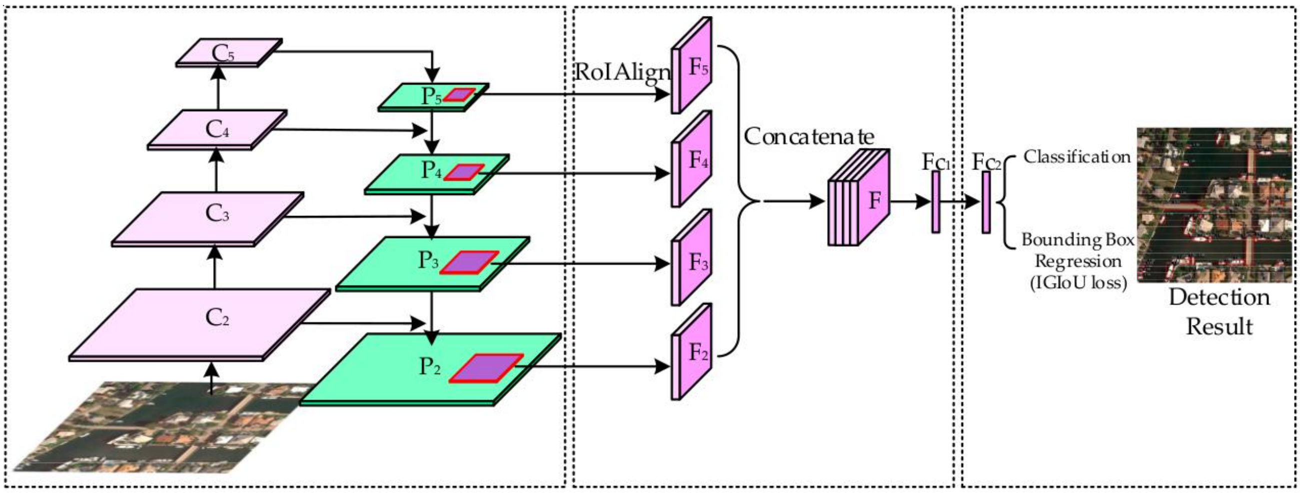


Figure 7: Architecture of the proposed framework. The left part shows the feature pyramid network. Multilevel features fusion is shown in the middle, and classification and bounding box regression based on the IGIoU loss is shown in the rightmost part.

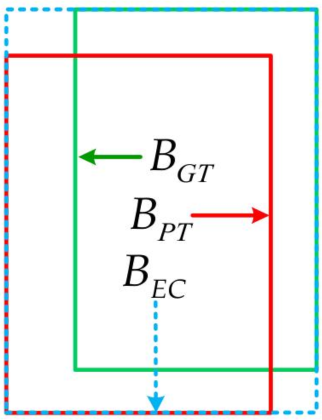
5.1.2 MLFF

A novel MLFF module is proposed. The feature maps of all levels are used by a MLFF module for feature extraction. Each proposal generated by the FPN are mapped to the feature maps of all levels. The size and location of the proposed region in the feature maps can be calculated based on the size ration between the proposal and the feature maps.

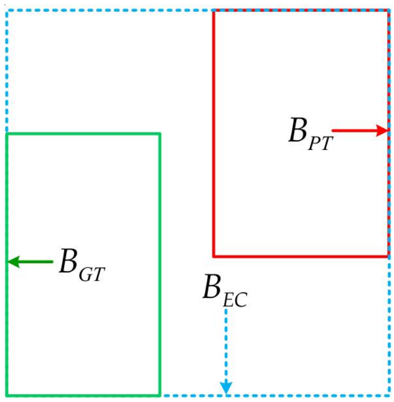
Four regions of each proposal are transformed into four groups of 7×7 feature maps, denoted F_2, F_3, F_4 and F_5 in figure 7 using RoiAlign[9]. The features are then concatenated along the channel dimension into a fused feature map called F.

Finally, a convolutional layer with a 7×7 kernel is used on F to obtain F_{C1} which is then passed to a fully connected layer.

5.2 Generalized Intersection over Union



(a) Two intersecting bounding boxes



(b) Two non-overlapping bounding boxes

Figure 8: Illustration showing the two cases of bounding box position: intersecting and non-overlapping. The rectangle enclosed by a green solid line denotes the ground truth B_{GT}; the predicted box B_{PT} is denoted by a red solid line, and the smallest enclosing box B_{EC} is denoted by a blue dashed line.

A novel metric, the Generalized IoU (GIoU) is proposed to enhance the evaluation of proximity between two bounding boxes. Figure 8 shows the difference between IoU and GIoU. The traditional IoU is insensitive to the scales of bounding boxes, and can be calculated using formula 2. Let B_{GT} be the ground truth bounding box and B_{PT} be the predicted bounding box.

$$IoU = \frac{area(B_{GT} \cap B_{PT})}{area(B_{GT} \cup B_{PT})} \tag{2}$$

The IoU is essentially the fraction of the intersection of the area of the predicted bounding box and the ground truth over the union of both bounding box. The IoU is not capable of measuring the distance when two bounding boxes are not overlapping. The introduced metric, address this issue.

The formula for the GIoU is as follows:

$$GIoU = IoU + \frac{area(B_{GT} \cup B_{PT})}{area(B_{EC})} - 1 \tag{3}$$

Where B_{EC} represents the smallest enclosing box of B_{GT} and B_{PT}. The IoU is inversely proportional to he distance between B_{GT} and B_{PT} where they are overlapping, but stays at 0 when they were not overlapping. The GIoU is proportional to the distance of the two bounding boxes, and decreases with the distance between B_{GT} and B_{PT}, whether or not the bounding boxes were overlapping.

5.3 Bounding Box Regression based on Improved GIoU Loss (IGIoU)

The bounding box regression loss used in traditional object detection methods is usually adopted to smooth the L1 or L2 loss. However, those two loss functions do not directly optimize the IoU metric. The smooth L1 or L2 loss are used to optimize the four parameters of the predicted bounding box, and the IoU is used to give more importance to the overlapping degree between the two bounding boxes.

Integrating the value of the GIoU into the loss can be done using formula 4 from Rezatofighi et al. [22].

$$L_{GIoU} = 1 - GIoU \tag{4}$$

The GIoU loss has a constant gradient during the training process, which restricts the effect of bounding box regression. The authors note that strength of the training should be enhanced when the predicted bounding box is far away from the ground truth, i.e. the absolute value of the gradient should be higher when the GIoU is small. Moreover, the value of the bounding box regression loss should decrease with the GIoU.

The improved GIoU Loss (IGIoU) is used to address those issues, and is given in the following formula:

$$L_{IGIoU} = 2 \times log_2 - 2 \times log(1 + GIoU) \tag{5}$$

5.4 Results

To validate the IGIoU loss and the MLFF module, quantitative comparisons were made between the proposed methods and five others methods on the NWPU VHR-10 dataset[4]. Those results are listed in table 6

Table 7 shows results of the proposed method on the DIOR[23] dataset. The DIOR dataset is a large scale benchmark, of size comparable to the DOTA dataset[5]. We see that the proposed method, with FPN+MLFF+IGIoU is superior to the baseline FPN in all of the evaluation metrics. It should be noted that the performance of FPN+MLFF+IGIoU is better than that of FPN+IGIoU and FPN+MLFF, which indicates that the MLFF in combination with IGIoU loss is effective.

The proposed method is also evaluated against four state of the art methods on the NWPU VHR-10 datasets, and are listed in table 6.

The method obtains state of the art results and better precision scores than all of the other tested methods, except in one case.

Method	GIoU			IoU		
	mAP (%)	AP50(%)	AP75(%)	mAP(%)	AP50(%)	AP75(%)
Faster R-CNN	53.5	86.8	61.0	54.6	87.1	62.6
Mask R-CNN	54.7	88.8	62.6	55.8	89.4	64.2
FPN	55.3	88.8	64.0	56.5	89.3	65.9
PANet	56.3	**90.5**	63.9	57.8	**91.8**	65.8
Proposed Method	**58.0**	**90.5**	**67.5**	**59.2**	91.4	**69.6**

Table 6: Comparison of the ODRSI against four existing detection framework on the NWPU VHR-10

Method	GIoU			IoU		
	mAP (%)	AP50(%)	AP75(%)	mAP(%)	AP50(%)	AP75(%)
FPN(baseline)	42.6	66.5	46.3	43.6	67.9	47.6
FPN + MLFF	43.3	67.8	46.9	44.2	68.9	48.1
FPN + GIoU	43.3	66.7	47.5	44.2	67.9	48.4
FPN + IGIoU	44.0	67.0	48.2	44.8	68.2	49.3
FPN+MLFF+GIoU	43.8	67.2	47.6	44.6	68.5	48.7
FPN+MLFF+IGIoU	**44.8**	**67.9**	**49.2**	**45.7**	**69.2**	**50.3**

Table 7: Comparison with the baseline method on the DIOR datasets

6 A Single Shot Framework with Multi-Scale Feature Fusion for Geospatial Object Detection[24]

This paper presents a novel architecture along with a new loss system, allowing for more precise bounding boxes along detected objects. The new architectures incorporates ideas similar to YOLT[8] where multiple detectors are trained and ran at different scales. This time the different scales detection is directly incorporated into the architecture itself: the feature maps from different layers are concatenated together. This approach allows the model to fully use the low level feature map with high resolution along with the high level feature maps incorporating more semantic information.

The model is trained and tested on two different datasets: the RSD-GOD dataset, a new dataset comprising of 5 different categories and 18K annotated images introduced by this paper, and the NWPU VHR-10[4] dataset.

6.1 Architecture

6.1.1 Base Feature Extractor

The proposed network is heavily based on the Darknet-53 architecture and reuses most of it features. 53 Convolutional layers are used, without pooling layers. The network reduce the features dimension by 2 by applying a stride. The Network also uses residual blocks containing 1×1 and 3×3 convolutional filters and 23 residual blocks. Batch normalisation[11] instead of dropout is used to control overfitting and convergence during training. The network uses leaky ReLU activations on all convolutional layers.

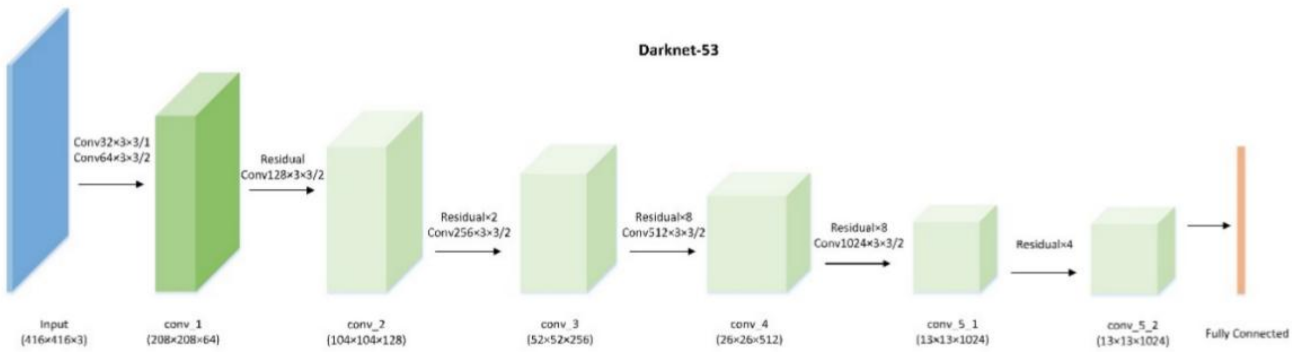


Figure 9: Architecture of the Darknet 53, used as a base network for feature extraction

6.1.2 Multi-scale feature fusion detector

To allow the detector to fully exploit both low level high resolution with fine detail feature maps and high level semantic features, multi-scale feature are used.

Three convolutional layers at different scales of the base feature extractor are used to make predictions. The first-scale predictions are made using an added convolutional layer on top of the last convolutional layer of the base feature detector. Following the article definitions, we will call this convolutional prediction layer `conv_6`. Two feature fusion modules are used to combine shallow feature. The first fusion module takes the prediction of the `conv_6` layer, upsamples it

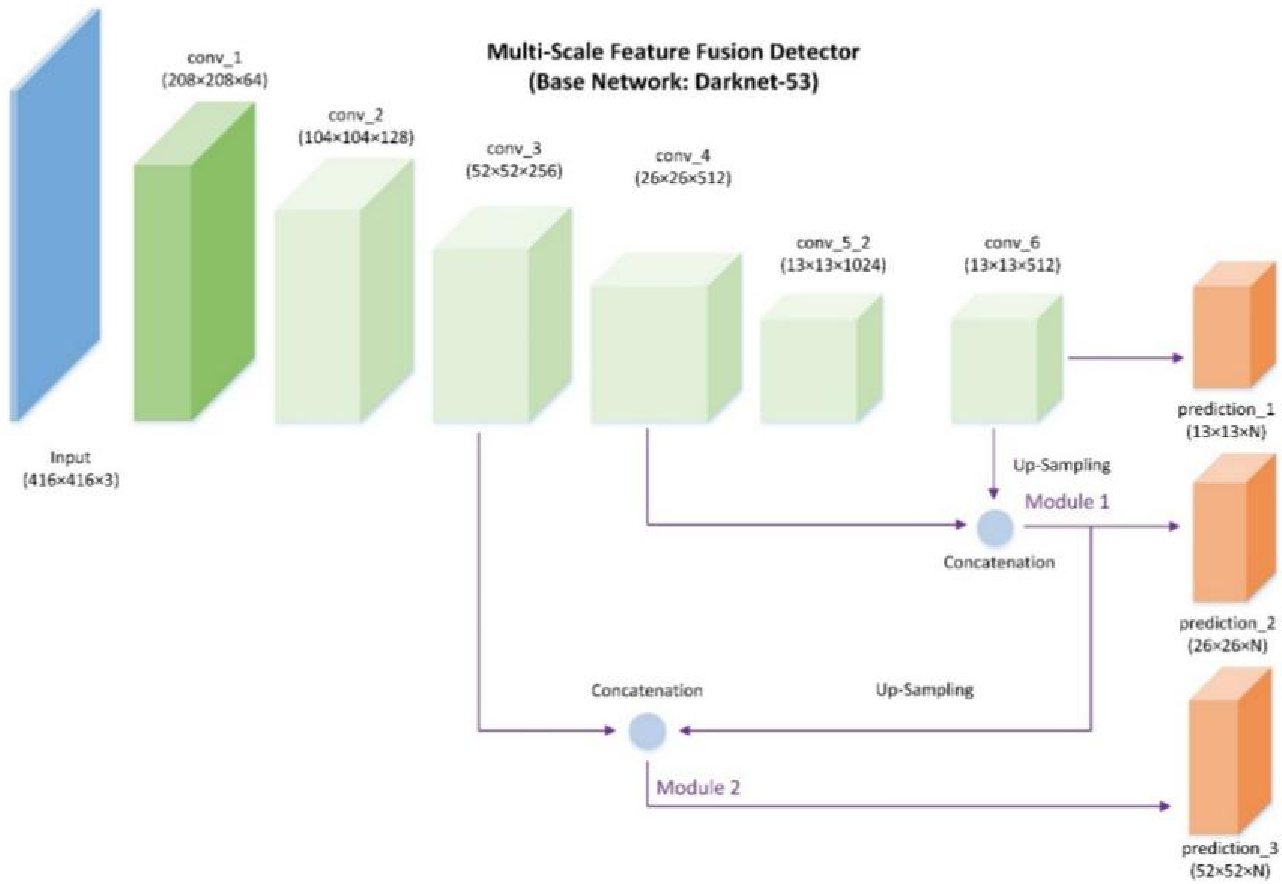


Figure 10: General architecture of the multi-scale feature fusion detector. The model uses Darknet-53 as the base feature extractor. Three predictions are generated at three different scales.

and concatenate it to the feature maps yielded by the `conv_4` layer. This gives out the second scale predictions. Finally, the second fusion modules takes the feature maps of the `conv_3` layer and concatenate it to the output of the first fusion modules, after up-sampling. This yield the third and final prediction.

6.1.3 Multi-scale feature fusion module

Three multi-scale feature fusion module are used to create 3 different scale prediction.

In a multi-scale feature fusion module, the dimension of the input feature maps are first reduced through the use of 1×1 convolutional kernel. High level feature maps are up-sampled after the 1×1 convolution to be same size as the lower level feature maps. Then, the high level feature maps are concatenated with the lower level feature maps. Alternate 1×1 and 3×3 convolutional layer are then used to progressively reduce the dimensions of the feature maps and make predictions. Figure 11 show details of the feature fusion module.

6.1.4 Anchors and predictions

Since the model is unstable during early training iterations, anchors are used, similar to the ones used in Faster R-CNN [7]. The designed network outputs three kind of feature maps with different size : 13×13, 26×26, 52×52. B anchors are generated and the corresponding B

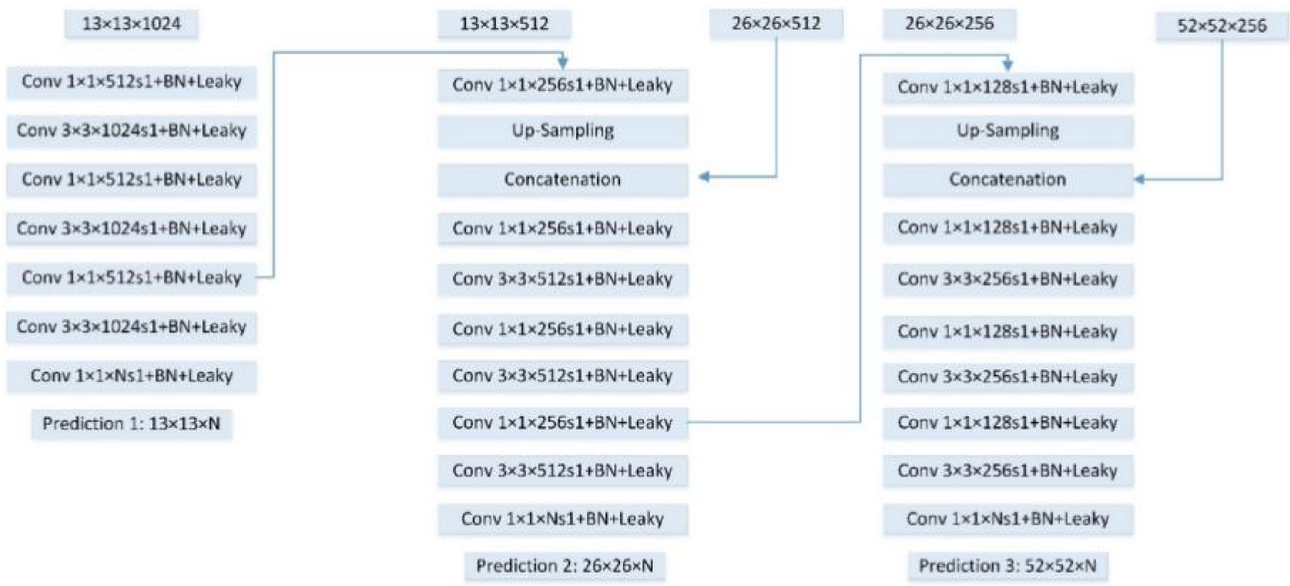


Figure 11: Multi-scale feature fusion module. The Feature maps from different layers are merged through up-sampling and concatenation. Each convolutional layer is batch-normalized, uses leaky ReLU activations, and have a stride of 1.

bounding boxes are predicted for each grid cell. During the training, the network outputs 5 coordinate values t_x, t_y, t_w, t_h, t_o; the final location of the predicted bounding box is obtained through the anchor size and the network outputs.

The location of the center of the bounding boxes (b_x, b_y) is relative to the grid cell offset (c_x, c_y) and the sigmoid activation function value of the location coordinates (t_x, t_y). Here (c_x, c_y) denotes the offsets from the top left corner of the original image to the current grid cell. The width and height of anchors are denoted as (p_w, p_h). p_o denotes the confidence score of object probability. The σ denotes the sigmoid function. Applying the sigmoid function on the predicted t_x, t_y, t_o normalize their value and stabilize the model during training.

We compute b_x, b_y, b_w, b_h and p_o using the following formulas:

$$b_x = \sigma(t_x) + c_x \tag{6}$$

$$b_y = \sigma(t_y) + c_y \tag{7}$$

$$b_w = p_w e^{t_w} \tag{8}$$

$$b_h = p_h e^{t_h} \tag{9}$$

$$p_o = \sigma(t_o) \tag{10}$$

For each predictions module, 3 anchor priors with different scales are used $(B = 3)$. K-means clustering have been applied on the annotated bounding boxes in the training data in order to obtain suitable priors.

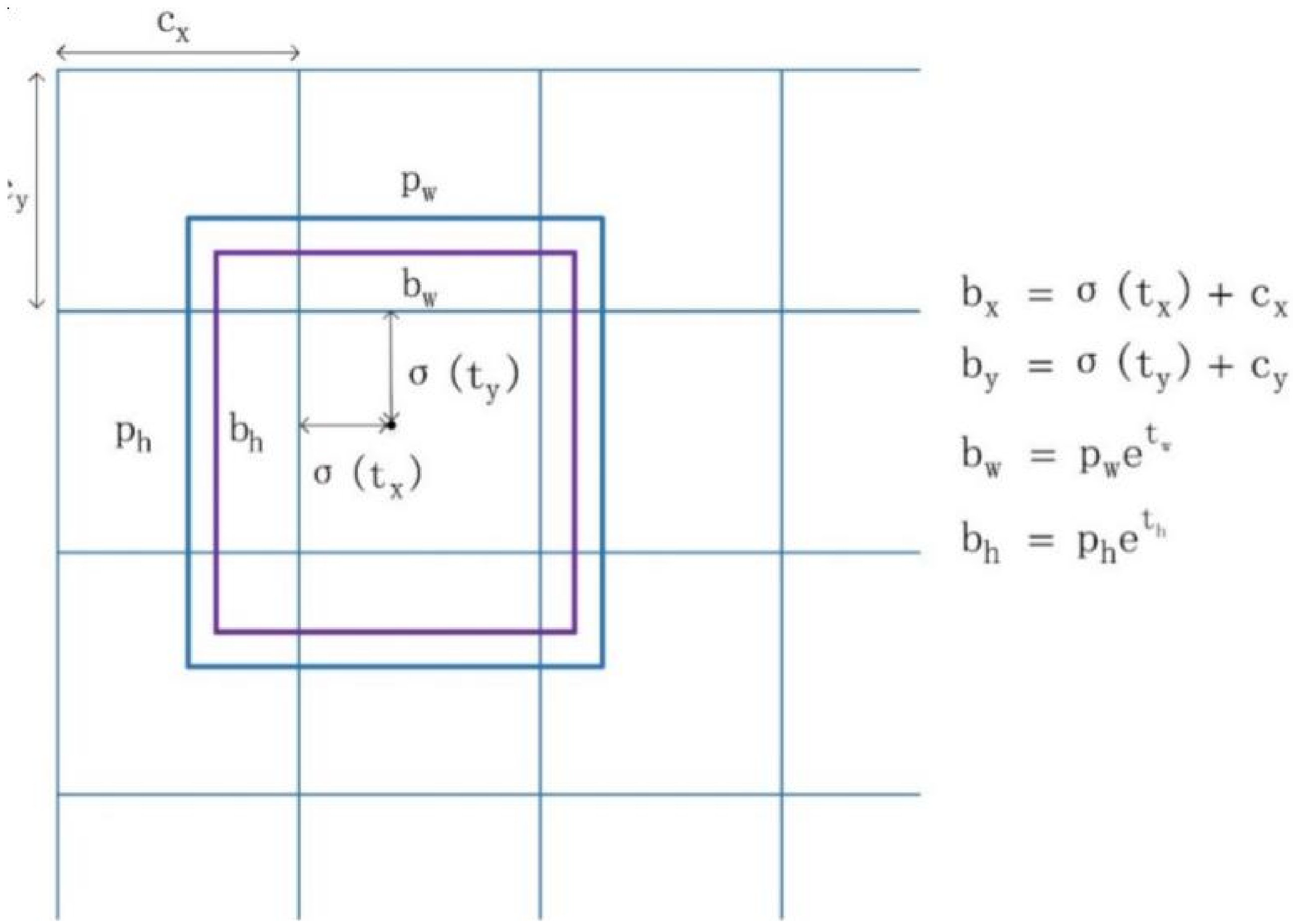


Figure 12: The model generates 4 coordinates: t_x, t_y, t_w and t_h. The center location of the final bounding box (b_x, b_y) is relative to the grid cell offsets (c_x, c_y) and the sigmoid activation function value of location coordinates (t_x, t_y). (c_x, c_y) denotes the offsets from the top left corner of the original image to the current grid cell

The network outputs four coordinates, one object confidence information and C class probabilities for each bounding box. The dimension of the network output tensor is then $S \times S \times N$, where S is the grid size, and $N = (5 + C) \times B$

6.1.5 Loss Function

The training objective loss is defined as the sum of a localization loss (L_{loc}), a confidence loss (L_{conf}) and a classification loss (L_{cla}).

The localization and confidence are computed using the squared error loss (see equations 12 and 13). The class loss is computed using the categorical crossentropy loss and is given in equation 14.

$$L_{overall} = L_{loc} + L_{conf} + L_{cla} \tag{11}$$

$$L_{loc} = \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} [(x_i \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 + (w_i - \hat{w}_i)^2 + (h_i - \hat{h}_i)^2] \tag{12}$$

$$L_{conf} = \lambda_{obj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} P^{obj} (c_i - \hat{c}_i)^2 + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} (1 - P^{obj})(c_i - \hat{c}_i)^2 \tag{13}$$

$$L_{cla} = -\lambda_{cla} \sum_{i=0}^{S^2} P^{obj} log(\hat{p}_i) \tag{14}$$

Here, λ_{coord}, λ_{noobj} and λ_{cla} represents scaling factors for the weight localization loss, the confidence loss and classification loss. P^{obj} is probability that there is an object in the box. Predicted bounding boxes without objects are more penalized.

The authors used the following values of λ: $\lambda_{coord} = 1, \lambda_{obj} = 5, \lambda_{noobj} = 1, \lambda_{cla} = 1$

6.1.6 Soft Non-Maximum Suppression

The proposed detection method generates a large number of cluttered bounding boxes. In traditional one stage detection pipelines, Non Maximum Suppression (NMS) is used to remove repetitive bounding boxes. NMS ranks location candidates according to their classification, and removes overlapping bounding boxes with the lowest scores.

Here, this method might cause the framework to miss part of neighboring detections, whose classification scores are lower. Instead of removing the location candidates, the Soft Non Maximal Suppression assign a new classification score to the bounding boxes, following equation 15

$$s_i = \begin{cases} s_i & iou(b_i, b_M) < T; i \neq M \\ s_i * f(iou(b_i, b_M)) & iou(b_i, b_M) \geq T; i \neq M \end{cases} \tag{15}$$

Where b_i denotes the ith bounding box in the location candidates and b_M is the bounding box with the maximum score. If the IoU between b_i and b_M is larger than a specified threshold T, a decayed score will be given to b_i using the Gaussian penalty function:

$$f(iou(b_i, b_M) = exp[\frac{-(iou(b_i, b_M))^2}{\rho}] \tag{16}$$

6.2 Results

The authors tested their method on the RSD-GOD dataset and compared it to other detection framework: Faster R-CNN[7], SSD[14], YOLO2[10]. The authors also tested their method with and without the Soft-NMS. The table with results are fairly large, and so are replicated in the appendices. Results for the RSD-GOD datasets are shown in table16.

Faster R-CNN obtains the best precision score for the airport class. However, the proposed methods with the soft-NMS obtain the best score for all other classes. The detector was also trained and tested on the NWPU VHR-10[4] dataset, along with a collection of part detectors (COPD)[25], rotation-invariant CNN (RICNN)[26] and a R-P-Faster R-CNN[27] and the detectors used for the evaluation of RSD-GOD. COPD and RICNN are rotation-invariant frameworks with SVM classifier for geospatial object detection. COPD uses hand-crafted features while RICNN applies learned features from the CNN.

As can be seen in table 17, SSD obtains the best precision score for the airplane and ship class, along with the baseball diamond. Faster R-CNN obtains the best score for basketball court and YOLOv2 the best score for ground track field. For all other classes, the proposed method with soft-NMS obtains the best score.

Methods	COPD	R-P-Faster R-CNN	RICNN	SSD	Faster R-CNN	YOLO2
Backbone	-	VGG16	-	VGG-16	ResNet50	Darknet5
Average Running Time (s)	1.070	0.150	8.770	0.027	0.430	0.026

Table 8: Average running time of the tested methods

7 Conclusion

Throughout this document, we have seen a variety of solutions to the problem's inherent with automated detection in remote sensing imagery. Most of the networks presented here uses some kind of feature fusion module. Qian et al, seen in section 5 also uses a novel Generalized Intersection Over Union formula to obtain a better positioning of the bounding boxes.

A model designed for automated detection in LiDAR imagery would take inspiration from such architectures. A feature fusion module, along with a finer grid and less downsampling would be used to lessen the information loss throughout the model, and help detect small and cluttered object. The GIoU loss could be used to improve the bounding box position.

8 Annexes

8.1 YOLOv4

This sections describes the latest iteration of the YOLO detection pipeline. While this network is not specifically designed for detection in remote sensing, it consistently obtains extremely great scores in all datasets. Understanding how this version is able to improve both speed and precsion can help us designing a better network. The article details and experiments with a large number of features, be it training wise or architecture wise to help configure a better YOLO model. In short, this article presents the **best practices in CNN design**. First the authors describes two different approaches: the "**Bag of Freebies**" (BoF), where researchers develop better training method to make the detector more accurate without increasing the inference cost. We will summarize those in section8.3. Another approaches is the one using special modules, that increases the inference cost by a small fractions, but gives out great improvement to the accuracy. This is what the authors calls the "**Bag of Special**" (BoS), which are described in section 8.4.

8.2 General Architecture of a Object Detector

The authors describes the composition of a modern detector. Those detectors are usually comprised of two parts. First a backbone, usually trained on ImageNet[28] which creates the feature maps from the input image, and secondly a head, which infer bounding boxes and classes. The backbone we are usually interested in, meaning the ones running on GPUs and not CPUs are VGG[29], resNet[9], resNeXt[30] or DenseNet[31]. We have already seen a few of those backbones in previously reviewed articles, especially resNet.

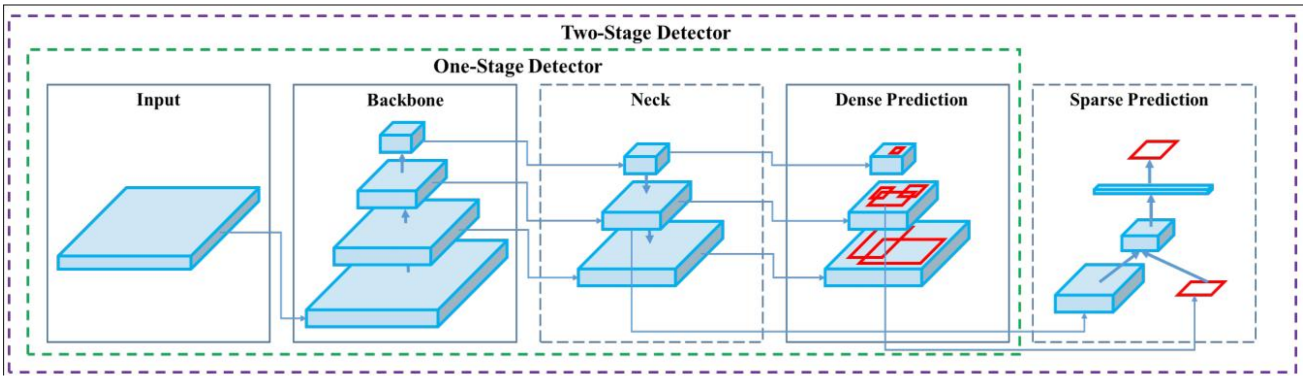


Figure 13: General Architecture of an Object Detector

The head part is divided in two categories: one-stage object detectors such as YOLO[10, 6], SSD[14] or RetinaNet[32]. Examples of two stages detectors are the R-CNNs[33], like Fast R-CNN [34], Faster R-CNN[7] or R-FCN[15].

Modern object detectors usually have a layer collecting feature maps from different stages, such as feature fusion modules. The author call this layer the 'neck'. Networks using this type of mechanisms include Feature Pyramid Network (FPN)[35], Path Aggregation Network (PAN)[36], BiPFN[37] and NAS-FPN[38]

To summarize, an object detector is made up of the following parts:

- **Backbones**: VGG[29], ResNet-50[9], EfficientNet[39], CSPResNeXt50[30], CSPDarknet53[40].

- **Neck**:
  - **Additional Blocks**: SPP[41], ASPP[42], RFB[43], SAM[44]
  - **Path-aggregation blocks**: FPN[35],NAS-FPN[38] PAN[36], BiFPN[37], ASFF[45], SFAM[46]
- **Heads**:
  - **Dense Prediction (One-Stage)**
    * Anchor-based: RPN[7], SSD[14], YOLO[6], RetinaNet[32]
    * Anchor-free: CornerNet[47], CenterNet[48], MatrixNet[49], FCOS[50]
  - **Sparse Prediction (Two-stage)**
    * Anchor-based: Faster R-CNN[7], R-FCN[15], Mask R-CNN[51]
    * Anchor Free : RepPoints[52]

8.3 Bag of Freebies

This section presents training techniques to improve detection rates without increasing the inference costs. Most of those techniques that falls under this definition are data augmentation methods. Data augmentation aims to increase the variability of the training images, so that the model will be more robust. Commonly used techniques are photometric distortions, where the brightness, contrast, hue and saturation of an image is adjusted, or geometric distortion, with random rescaling, cropping, flipping and rotating.

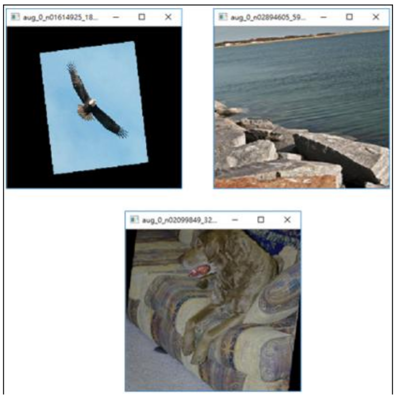
Those techniques are pixel-wise adjustments, meaning that the original pixel information in the adjusted area is retained. Novel data augmentation techniques have an emphasis put in object occlusion issues. Random Erase[53] or CutOut[54] randomly selects a rectangle region in an image and fills with random values. Hide and seek[55] and grid mask[56] randomly selects multiple rectangle regions in an image and replaces them to all zeros. Similar techniques are applied to feature maps, for example, DropOut[57], DropConnect[58] and DropBlock[59].

Other methods uses multiple image together. MixUP[60] uses two images to multiply and superimpose them with different coefficient ratios, and adjusts the labels with these ratios. CutMix[61] crops an image and cover another images with this cropped region, adjusting the labels according to the size of the covered area.

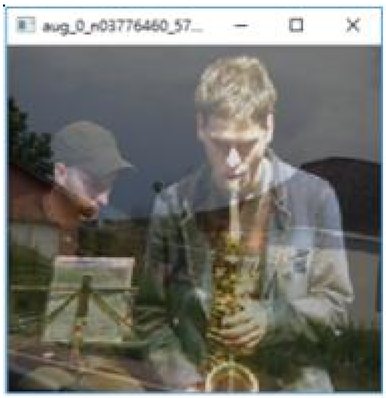
Even StyleTransfer GANs[62] have been used for data augmentation, reducing the texture bias learned by CNN.

Other data augmentation techniques have been focussed in reducing the amount of bias present in the semantic distribution of the dataset. For example, there might be a problem of data imbalance between different classes, which can be solved by hard negative example mining[63] or online hard example mining[64] in two-stage object detectors. However these example mining methods are not applicable in one-stage object detectors. For those detectors, focal loss[65] have been proposed to deal with the problem of data imbalance.

Another issue is that it is difficult to express the relationship of the degree of association between different categories with the one-hot representation, often used when executing labeling. Label smoothing[66] convert hard label into soft label, which can make the model more robust. Knowledge distillation have also been used to design a label refinement network.[67].



(a) Classic image data augmentation: crop, rotation, flip, hue, saturation, brightness



(b) MixUp

(c) CutMix

Figure 14: Some data augmentation techniques

Finally, the objective function for the bounding box have also been the focus of improvements. Usually, Mean Square Error (MSE) have been used to perform regression on the center point coordinates, height and width of the Bounding Box. However, to directly estimate the coordinates of the bounding box is to treat those points as independent variables, which **does not consider the integrity of the object itself**. The IoU loss[68] puts the coverage of the predicted bounding box area and the ground truth area into consideration. GIoU loss[22] is an improvement on this loss, and also include the shape and orientation of the object in addition to the coverage area[2]. Finally, DIoU loss[69] also considers the distance of the center of an object and CIoU[69] considers the overlapping area, the distance between center points and the aspect ratio.

8.4 Bag Of Specials

Other methods that improve the detection rate at the expense of a small increase in inference cost are what the authors refers to as "Bag of Specials" (BoS). Those methods are generally plugin modules that enhance certain attributes of the model. For example, they can enlarge the receptive field, introduce an attention mechanism or strengthen the feature integration capability.

Common modules uses to improve the receptive field are SPP[41], ASPP[42] and RFB[43]. The SPP originates from Spatial Pyramid Matching (SPM) [70]. SPM method is to split the feature maps into several $d \times d$ blocks of equal size, where d can be $1, 2, 3, \ldots$. This forms spatial pyramids, where we can then extract bag-of-words features. SPP integrate SPM into a CNN, and uses the max-pooling operation instead of bag-of-word. However, this SPP module output one dimensional feature vectors, which makes it inapplicable for Fully Convolutionnal Network (FCN). In YOLOv3[6], the SPP module is improved by concatenating the max-pooling outputs with kernel size $k \times k$ with $k = \{1, 5, 9, 13\}$ and stride of 1. This modules improves the YOLOv3 AP_{50} by 2.7% on the MS COCO[71] datasets for only 0.5% extra computation.

Attention is also used to improve the capabilities of detection networks. Attention modules are divided in two categories: channel-wise attention, with the Squeeze-and-Excitation (SE)[72] and point-wise attention with Spatial Attention Module (SAM) [44]. SE modules can improve the accuracy of ResNet by 1% in the ImageNet[28] dataset, however this comes at in increase of

[2]An improvement to this has been done for automated detection in satellite imagery by Qian and Al and is covered in section 5

inference time of 10% on a GPU. In comparison, SAM only demands 0.1% extra calculation but improve the ResNet50 top-1 accuracy by 0.5%. This module does not affect the inference speed.

Feature integration modules allows the model to take into account features computed from earlier layers. Early examples are skip connections[73] or hyper-column[74] integrate low-level physical and geometric features to high-level semantic feature. Now, lightweights modules that integrates different feature pyramids are used, like SFAM[46], ASFF[45] and BiFPN[37]. The idea behind SFAM is to use SE modules to execute channel-wise re-weighting on multi-scale concatenated feature maps. ASFF uses softmax as point-wise level re-weighting and adds feature maps of different scales. In BiFPN multi-input weighted residual connections is used to execute scale-wise level re-weighting, and then add feature maps of different scales.

A good activation function is crucial to correctly train a network, and much work has been put into trying to design a better one. In 2010, ReLU[75] was proposed as a solution to vanishing gradient problem. Offshoots, such as LeakyRelu[12], PReLU[76], ReLU6[18], Scale Exponential Linear Unit (SELU) [77], Swish [78], hard-Swish[79] and Mish[80] have been proposed. LReLU and PReLU serves to solve the problem that the gradient of ReLU is zero when the output is less than 0. ReLU6 and hard-swish are specially designed for quantization networks. The SELU activation function tries to self-normalize a neural network. Swish and Mish are continuously differentiable.

Post-processing with NMS is usually done with detection network to remove superfluous bounding boxes and only retain the ones with the highest response. However NMS does not consider contextual information, so Girshick *et al.*[81] added classification confidence scores in R-CNN as a reference. Soft NMS[82] considers the problem that occlusion of an object may cause the degradation of confidence score in greedy NMS with IoU score. DIoU NMS[69] adds the information of the center point distance to bounding box.

8.5 Additional Improvements

The main goal of this architecture is to make the detector suitable for training on a single consumer-grade GPU so additional design improvements were made. First, new methods of data augmentation are used: Mosaic and Self-Adversarial Training. Optimal hyper-parameters were chosen using genetic algorithms. Finally, modifications were made to existing methods to make them more efficient for training and detection, namely SAM, PAN and Cross mini-Batch normalization.

Figure 15: The Mosaic data augmentation technique

Mosaic is a new data augmentation method that mixes 4 training images so that 4 different context are mixed, and is shown in Figure 15. This allows detection of objects outside their normal context. Batch normalisation calculates activation statistics from 4 different images on each layer, which significantly reduces the need for a large mini-batch size.

Self-Adversarial Training operates in 2 forward-backward stages. In the 1st stage, the neural network alters the original image instead of the network weights. **This way, the network executes an adversarial attack on itself, altering the original image and making it look like there is no object to detect**. In the 2nd stage, the network is trained to detect an object on this modified image in the normal way.

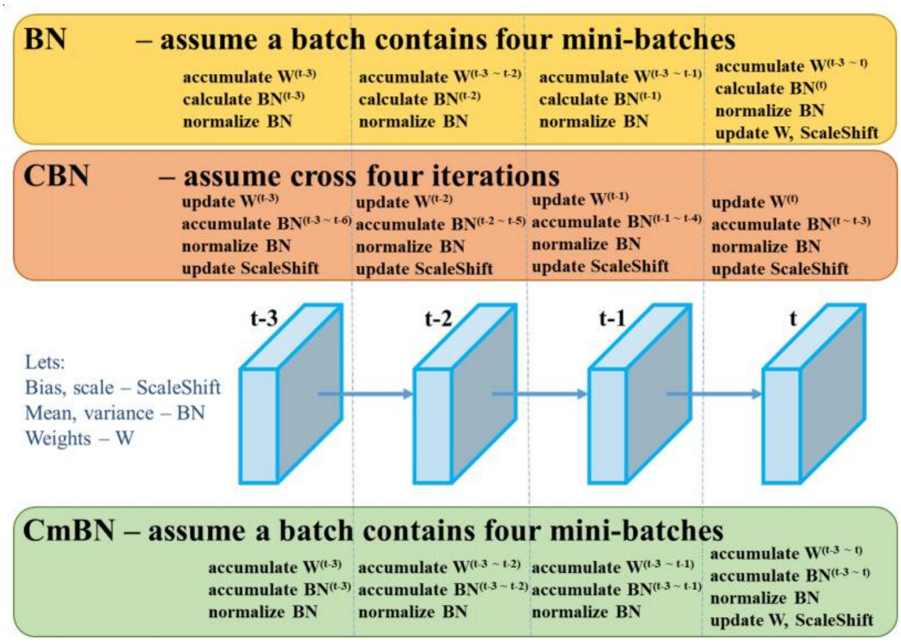
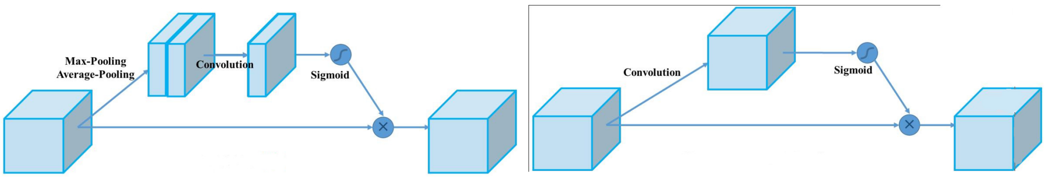


Figure 16: Cross Mini Batch Normalization

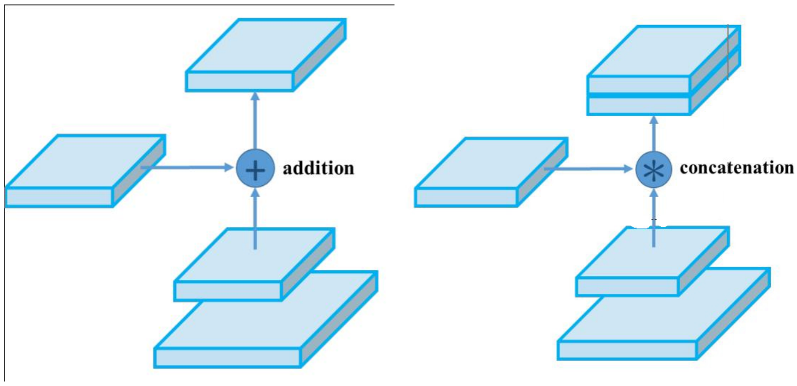
Cross mini Batch Normalisation is used. It is a modified Cross Batch Normalization (CmBN). CmBN collects statistics only between mini-batches within a single batch.

SAM is modified from spatial-wise attention to point-wise attention (figure 17), and the shortcut connection in PAN are replaced to concatenation, as can be seen in figure 18



(a) SAM (b) Modified SAM

Figure 17: SAM and its YOLO modification



(a) PAN (b) Modified PAN

Figure 18: PAN and its YOLO modification

8.6 YOLOv4 Architecture

YOLOv4 basic architecture is as follows:

- **Backbone**CSPDarknet53[40]
- **Neck**: SPP, PAN
- **Head**: YOLOv4

For the backbone, YOLOv4 uses the following "Bag of Freebies": CutMix and Mosaic Data Augmentation, DropBlock regularization and Class label smoothing. The bag of Specials are Mish Activation, Cross-stage partial connections (CSP) and multi-input weighted residual connections (MiWRC).

For the detector, YOLOv4 uses these "Bag of Freebies": CIoU-loss, CmBn, DropBlock Regularization, Mosaic Data augmentation, Self-Adversarial Training, elimination of grid sensitivity, usage of multiple anchors for a single ground truth, a cosine annealing scheduler [83], optimal hyper-parameters and random training shapes. These Bag of Specials are used: Mish Activation, SPP-block, SAM-block, PAN Path-aggragation block, DIoU-NMS.

8.7 Results

8.7.1 Influence of features on Classifier training

The authors studies the influence of different data augmentation techniques, such as bilateral blurring, MixUp, CutMix and Mosaic, along with the influence of different activations like Leaky-ReLU, Swish and Mish.

Table 9 shows the results of those tests for the CSPResNext-50 backbone, and Table 10 for the CSPDarknet-53 backbone. The classifier accuracy is improved by introducing the CutMix and Mosaic data augmentation, Class label smoothing and the Mish activation.

MixUp	CutMix	Mosaic	Bluring	Label Smoothing	Swish	Mish	Top-1	Top-5
							77.9%	94.0%
✓							77.2%	**94.0%**
	✓						**78.0%**	**94.3%**
		✓					**78.1%**	**94.5%**
			✓				77.5%	93.8%
				✓			**78.1%**	**94.4%**
					✓		64.5%	86.0%
						✓	**78.9%**	**94.5%**
	✓	✓		✓			**78.5%**	**94.8%**
	✓	✓		✓		✓	**79.8%**	**95.2%**

Table 9: Impact of Bag of Freebies and different activation on the CSPResNext-50 Classifier. Baseline is shown on the first row. Results which are better than the baseline are shown in bold

MixUp	CutMix	Mosaic	Bluring	Label Smoothing	Swish	Mish	Top-1	Top-5
							77.2%	93.6%
	✓	✓		✓			**77.8%**	**94.4%**
	✓	✓		✓			**78.7%**	**94.8%**

Table 10: Impact of Bag of Freebies and Mish on the CSPDarknet-53 Classifier. Baseline is shown on the first row. Results which are better than the baseline are shown in bold

8.7.2 Influence of features on the Detector training

In order to better understand the impact of different Bag-of-Freebies on the detector training accuracy, the authors studies different features that increase the detector accuracy without affecting the FPS.

- **S** - Elimination of grid sensitivity. In YOLOv3, the equation $b_x = \sigma(t_x)+c_x, b_y = \sigma(t_y)+c_y$, with c_x, c_y being whole numbers is used to evaluate the object coordinates. Very high absolute values of t_x are needed for the b_x value to approach c_x or $c_x + 1$. This problem is solved by multiplying the sigmoid by a factor exceeding 1.0, eliminating the effect of grid on which the object is undetectable.
- **M**: Mosaic data augmentation - using the 4-image mosaic during training instead of a single image

S	M	IT	GA	LS	CBN	CA	DM	OA	Loss	AP	AP_50	AP_75
									MSE	38.0%	60.0%	40.8%
✓									MSE	37.7%	59.9%	40.5%
	✓								MSE	**39.1%**	**61.8%**	**42.0%**
		✓							MSE	36.9%	59.7%	39.4%
			✓						MSE	**38.9%**	**61.7%**	**41.9%**
				✓					MSE	33.0%	55.4%	35.4%
					✓				MSE	**38.4%**	**60.7%**	**41.3%**
						✓			MSE	**38.7%**	**60.7%**	**41.9%**
							✓		MSE	35.3%	57.2%	38.0%
✓									GIoU	**39.4%**	59.4%	**42.5%**
✓									DIoU	**39.1%**	64.0%	**44.8%**
✓									CIoU	**39.6%**	59.2%	**42.6%**
✓	✓	✓	✓						CIoU	**41.5%**	**64.0%**	**44.8%**
	✓		✓					✓	CIoU	36.1%	56.5%	38.4%
✓	✓	✓	✓					✓	MSE	**40.3%**	**64.0%**	**43.1%**
✓	✓	✓	✓					✓	GIoU	**42.4%**	**64.4%**	**45.9%**
✓	✓	✓	✓					✓	CIoU	**42.4%**	**64.4%**	**45.9%**

Table 11: Ablation Studies of Bag-of-Freebies using CSPResNeXt50-PANet-SPP with 512×512 input. Baseline is shown on the first row. Results better than baseline are shown in bold.

- **IT**: IoU thresholding - Using multiple anchors for a single ground truth with $IoU(truth, anchor) > IoU_{threshold}$
- **GA**: Genetic Algorithms - Using genetic algorithms to select the optimal hyperparameters during network training
- **LS**: Class label smoothing - using class label smoothing for sigmoid activation
- **CBN**: CmBN - using Cross mini-Batch Normalization for collecting statistics inside the entire batch instead of inside a single mini-batch
- **CA**: Cosine annealing scheduler - altering the learning rate following a cosine
- **DM**: Dynamic mini-batch size - automatic increase of mini-batch size during small resolution training by using random training shape
- **OA**: Optimzied Anchors - using the optimized anchors for training with 512×512 network resolution
- **GIoU, CIoU, DIoU, MSE**: Using different loss algorithms for bounded box regression

An ablation studies using those bag of freebies is shown in table 11

8.8 Influence of different backbones and pre-trained weights on the detector

The authors conducted another study, where they measured the performance of different backbones with the same training parameters, which is reproduced on table 12. It is apparent that **the model with the best classification accuracy is not always the best in terms**

of detector accuracy. CSPResNeXt-50 models trained with different features obtain better classification accuracy, CSPDarknet53 obtains a higher accuracy in the object detection task.

Using BoF and Mish activations for the CSPResNeXt50 classifier increases its classification accuracy but further application of the pre trained weights reduces the detector accuracy. This is not true for CSPDarknet53, with the applications of BoF and Mish increasing both the detector and classifier accuracy. Considering those results, it would seem that **CSPDarknet53 is more suitable for the detector.**

Model (with optimal settings)	**Input Size**	**AP**	AP_50	AP_75
CSPResNeXt50-PANet-SPP	512 × 512	42.4	64.4	45.9
CSPResNeXt50-PANet-SPP (BoF-backbone)	512 × 512	42.3	64.3	45.7
CSPResNeXt50-PANet-SPP (BoF-backbone + Mish)	512 × 512	42.3	64.2	45.8
CSPDarknet53-PANet-SPP (BoF-backbone)	512 × 512	42.4	64.5	46.0
CSPDarknet53-PANet-SPP (BoF-backbone + Mish)	512 × 512	43.0	64.9	46.5

Table 12: Comparison study of different classifier pre trained weights for detector trainings.

8.8.1 Influence of mini-batch size on detector training

Finally, the author studied the impact of the mini-batch size on training, which have been reproduced in table 13

Model	**Input Size**	**AP**	AP_50	AP_75
CSPResNeXt50-PANet-SPP (without BoF/BoS, mini-batch 4)	608 × 608	37.1	59.2	39.9
CSPResNeXt50-PANet-SPP (with BoF/BoS, mini-batch 8)	608 × 608	38.4	60.6	41.6
CSPDarknet53-PANet-SPP (without BoF/BoS, mini-batch 4)	512 × 512	41.6	64.1	45.0
CSPDarknet53-PANet-SPP (with BoF/BoS, mini-batch 8)	512 × 512	41.7	64.2	45.2

Table 13: Comparison study of different mini-batch size for detector training.

It seems that that after adding BoF and BoS training strategies, **the mini-batch size has almost no impact on the detector performance**. This would allow the use of consumer grade GPU to be used to train those detectors, as they would not require as much VRAM for the training.

Layer	Type	Filters	Size/Stride	Output Size
0	Convolutional	32	$3 \times 3/1$	$416 \times 416 \times 32$
1	Maxpool		$2 \times 2/2$	$208 \times 208 \times 32$
2	Convolutional	64	$3 \times 3/1$	$208 \times 208 \times 64$
3	Maxpool		$2 \times 2/2$	$104 \times 104 \times 64$
4	Convolutional	128	$3 \times 3/1$	$104 \times 104 \times 12$
5	Convolutional	64	$1 \times 1/1$	$104 \times 104 \times 64$
6	Convolutional	128	$3 \times 3/1$	$104 \times 104 \times 12$
7	Maxpool		$2 \times 2/2$	$52 \times 52 \times 128$
8	Convolutional	256	$3 \times 3/1$	$52 \times 52 \times 256$
9	Convolutional	128	$1 \times 1/1$	$52 \times 52 \times 128$
10	Convolutional	256	$3 \times 3/1$	$52 \times 52 \times 256$
11	Maxpool		$2 \times 2/2$	$26 \times 26 \times 256$
12	Convolutional	512	$3 \times 3/1$	$26 \times 26 \times 256$
13	Convolutional	256	$1 \times 1/1$	$26 \times 26 \times 256$
14	Convolutional	512	$3 \times 3/1$	$26 \times 26 \times 512$
15	Convolutional	256	$1 \times 1/1$	$26 \times 26 \times 256$
16	Convolutional	512	$3 \times 3/1$	$26 \times 26 \times 512$
17	Convolutional	1024	$3 \times 3/1$	$26 \times 26 \times 1024$
18	Convolutional	1024	$3 \times 3/1$	$26 \times 26 \times 1024$
19	Passtrough		$10 \to 20$	$26 \times 26 \times 1024$
20	Convolutional	1024	$3 \times 3/1$	$26 \times 26 \times 1024$
21	Convolutional	N_f	$1 \times 1/1$	$26 \times 26 \times N_f$

Table 14: YOLT Network Architecture

8.9 Tables

Layer	Type	Filters	Size/Stride/Dilation Rate	Output
0	Convolutional	16	3/1	512
1	Convolutional	32	3/2	256
2	Convolutional	16	1/1	256
3	Convolutional	32	3/1	256
4	Convolutional	64	3/2	128
5	Convolutional	32	1/1	128
6	Convolutional	64	3/1	128
7	Convolutional	32	1/1	128
8	Convolutional	64	3/1	128
9	Convolutional	32	1/1	128
10	Dilated Convolution	64	3/1/2	128
11	Convolutional	32	1/1	128
12	Concatenation		Layer 11 + Layer 9	128
13	Convolutional	32	1/1	128
14	Dilated Convolutional	64	3/1/4	128
15	Convolutional	32	1/1	128
16	Concatenation		Layer 15 + Layer 13	128
17	Convolutional	32	1/1	128
18	Concatenation		Layer 9 + Layer 13	128
19	Convolutional	32	1/1	128
20	Concatenation		Layer 19 + Layer 17	128
21	Convolutional	128	$3 \times 3/2$	64
22	Convolutional	64	1/1	64
23	Convolutional	128	3/1	64
24	Convolutional	64	1/1	64
25	Convolutional	128	3/1	64
26	Route		Layer 8	64
27	Convolutional	16	1/1	64
28	Passtrough		/2	64
29	Concatenation		Layer 25 + Layer 28	64
30	Convolutional	128	1/1	64
31	Convolutional	256	3/1	64
32	Convolutional	N	1/1	64

Table 15: Architecture of the Single Shot Detection model described in section 6

Method	Faster R-CNN	SSD	YOLOv2	Proposed	Proposed (Soft-NMS)
Pretrained Backbone	ResNet50	VGG16	Darknet-19	Darknet-53	Darknet-53
Airport	**0.911**	0.788	0.598	0.839	0.847
Helicopter	0.876	0.893	0.917	0.946	**0.946**
Plane	0.673	0.819	0.813	0.897	**0.904**
Oiltank	0.645	0.898	0.909	0.920	**0.922**
Warship	0.759	0.755	0.695	0.793	**0.826**
Mean AP	0.773	0.831	0.786	0.879	**0.890**

Table 16: Average Precision values for each class of the RSD-GOD dataset of the different detection methods along with the Single Shot Framework model from section 6

Method	COPD	R-P-Faster R-CNN	RICNN	SSD	Faster R-CNN	YOLOv2	SSD	SSD (Soft-NMS)
Airplane	0.623	0.904	0.884	**0.957**	0.946	0.733	0.929	0.934
Ship	0.689	0.750	0.773	**0.829**	0.823	0.749	0.765	0.771
Storage Tank	0.637	0.444	0.853	0.856	0.653	0.344	0.849	0.875
Baseball diamon	0.833	0.899	0.881	**0.966**	0.955	0.889	0.930	0.930
Tennis court	0.321	0.797	0.408	0.821	0.819	0.291	0.824	0.827
Basketball court	0.363	0.776	0.585	0.860	**0.897**	0.276	0.815	0.838
Ground track field	0.853	0.877	0.867	0.582	0.924	**0.988**	0.837	0.837
Harbor	0.553	0.791	0.686	0.548	0.724	0.754	0.816	**0.825**
Bridge	0.148	0.682	0.615	0.419	0.575	0.518	0.702	**0.725**
Vehicle	0.440	0.732	0.711	0.756	0.778	0.513	0.819	**0.823**
Mean AP	0.546	0.765	0.726	0.759	0.809	0.607	0.829	**0.838**

Table 17: Average Precision values for each class of the NWPU VHR-10 dataset for each tested method and the Single Shot Framework described in section 6

References

[1] Alex Yang et al. "Remote Sensing Object Localization with Deep Heterogeneous Superpixel Features". In: *2019 IEEE International Conference on Big Data (Big Data)* (2019), pp. 5453–5461.

[2] Zhong Dong and Baojun Lin. "BMF-CNN: an object detection method based on multiscale feature fusion in VHR remote sensing images". In: *Remote Sensing Letters* 11 (Mar. 2020), pp. 215–224. DOI: 10.1080/2150704X.2019.1706007.

[3] Sébastien Razakarivony and Frédéric Jurie. "Vehicle Detection in Aerial Imagery : A small target detection benchmark". In: *Elsevier* (2015). URL: https://hal.archives-ouvertes.fr/hal-01122605v2/document.

[4] Gong Cheng, Junwei Han, and Xiaoqiang Lu. "Remote Sensing Image Scene Classification: Benchmark and State of the Art". In: *CoRR* abs/1703.00121 (2017). arXiv: 1703.00121. URL: http://arxiv.org/abs/1703.00121.

[5] Gui-Song Xia et al. "DOTA: A Large-scale Dataset for Object Detection in Aerial Images". In: *CoRR* abs/1711.10398 (2017). arXiv: 1711.10398. URL: http://arxiv.org/abs/1711.10398.

[6] Joseph Redmon and Ali Farhadi. "YOLOv3: An Incremental Improvement". In: *CoRR* abs/1804.02767 (2018). arXiv: 1804.02767. URL: http://arxiv.org/abs/1804.02767.

[7] S. Ren et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *arXiv e-prints* (June 2015). arXiv: 1506.01497 [cs.CV].

[8] Adam Van Etten. "You Only Look Twice: Rapid Multi-Scale Object Detection In Satellite Imagery". In: *CoRR* abs/1805.09512 (2018). arXiv: 1805.09512. URL: http://arxiv.org/abs/1805.09512.

[9] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[10] Joseph Redmon and Ali Farhadi. "YOLO9000: Better, Faster, Stronger". In: *CoRR* abs/1612.08242 (2016). arXiv: 1612.08242. URL: http://arxiv.org/abs/1612.08242.

[11] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *CoRR* abs/1502.03167 (2015). arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167.

[12] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. "Rectifier nonlinearities improve neural network acoustic models". In: *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*. 2013.

[13] Adam Van Etten. "Satellite Imagery Multiscale Rapid Detection with Windowed Networks". In: *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)* (2019), pp. 735–743.

[14] Wei Liu et al. "SSD: Single Shot MultiBox Detector". In: *CoRR* abs/1512.02325 (2015). arXiv: 1512.02325. URL: http://arxiv.org/abs/1512.02325.

[15] Jifeng Dai et al. "R-FCN: Object Detection via Region-based Fully Convolutional Networks". In: *CoRR* abs/1605.06409 (2016). arXiv: 1605.06409. URL: http://arxiv.org/abs/1605.06409.

[16] Jonathan Huang et al. "Speed/accuracy trade-offs for modern convolutional object detectors". In: *CoRR* abs/1611.10012 (2016). arXiv: 1611.10012. URL: http://arxiv.org/abs/1611.10012.

[17] Christian Szegedy et al. “Rethinking the Inception Architecture for Computer Vision”. In: *CoRR* abs/1512.00567 (2015). arXiv: 1512.00567. URL: http://arxiv.org/abs/1512.00567.

[18] Andrew G. Howard et al. “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications”. In: *CoRR* abs/1704.04861 (2017). arXiv: 1704.04861. URL: http://arxiv.org/abs/1704.04861.

[19] M. Ju et al. “A Simple and Efficient Network for Small Target Detection”. In: *IEEE Access* 7 (2019), pp. 85771–85781.

[20] Fisher Yu and Vladlen Koltun. “Multi-Scale Context Aggregation by Dilated Convolutions”. In: *International Conference on Learning Representations (ICLR)*. May 2016.

[21] Xiaoliang Qian et al. “Object Detection in Remote Sensing Images Based on Improved Bounding Box Regression and Multi-Level Features Fusion”. In: *Remote Sensing* 12 (Jan. 2020), p. 143. DOI: 10.3390/rs12010143.

[22] Seyed Hamid Rezatofighi et al. “Generalized Intersection over Union: A Metric and A Loss for Bounding Box Regression”. In: *CoRR* abs/1902.09630 (2019). arXiv: 1902.09630. URL: http://arxiv.org/abs/1902.09630.

[23] Ke Li et al. “Object Detection in Optical Remote Sensing Images: A Survey and A New Benchmark”. In: *ArXiv* abs/1909.00133 (2020).

[24] Shuo Zhuang et al. “A Single Shot Framework with Multi-Scale Feature Fusion for Geospatial Object Detection”. In: *Remote Sensing* 11 (Mar. 2019). DOI: 10.3390/rs11050594.

[25] Dalal AL-Alimi et al. “Multi-Scale Geospatial Object Detection Based on Shallow-Deep Feature Extraction. Remote Sens. 2019”. In: *Remote Sens 11* 21 (2019).

[26] W. Zhang et al. “Object Detection in High-Resolution Remote Sensing Images Using Rotation Invariant Parts Based Model”. In: *IEEE Geoscience and Remote Sensing Letters* 11.1 (2014), pp. 74–78.

[27] Xiaobing Han, Yanfei Zhong, and Liangpei Zhang. “An Efficient and Robust Integrated Geospatial Object Detection Framework for High Spatial Resolution Remote Sensing Imagery”. In: *Remote Sensing* 9 (June 2017), p. 666. DOI: 10.3390/rs9070666.

[28] J. Deng et al. “ImageNet: A Large-Scale Hierarchical Image Database”. In: *CVPR09*. 2009.

[29] Karen Simonyan and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *International Conference on Learning Representations*. 2015.

[30] Saining Xie et al. “Aggregated Residual Transformations for Deep Neural Networks”. In: *CoRR* abs/1611.05431 (2016). arXiv: 1611.05431. URL: http://arxiv.org/abs/1611.05431.

[31] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. “Densely Connected Convolutional Networks”. In: *CoRR* abs/1608.06993 (2016). arXiv: 1608.06993. URL: http://arxiv.org/abs/1608.06993.

[32] Tsung-Yi Lin et al. “Focal Loss for Dense Object Detection”. In: *CoRR* abs/1708.02002 (2017). arXiv: 1708.02002. URL: http://arxiv.org/abs/1708.02002.

[33] Ross Girshick. “Fast R-CNN”. In: *CoRR* abs/1504.08083 (2015). URL: http://www.cv-foundation.org/openaccess/content_iccv_2015/papers/Girshick_Fast_R-CNN_ICCV_2015_paper.pdf.

[34] Ross Girshick. “Fast R-CNN”. In: *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*. ICCV 15. USA: IEEE Computer Society, 2015, pp. 1440–1448. ISBN: 9781467383912. DOI: 10.1109/ICCV.2015.169. URL: https://doi.org/10.1109/ICCV.2015.169.

[35] Tsung-Yi Lin et al. "Feature Pyramid Networks for Object Detection". In: *CoRR* abs/1612.03144 (2016). arXiv: 1612.03144. URL: http://arxiv.org/abs/1612.03144.

[36] Shu Liu et al. "Path Aggregation Network for Instance Segmentation". In: *CoRR* abs/1803.01534 (2018). arXiv: 1803.01534. URL: http://arxiv.org/abs/1803.01534.

[37] Mingxing Tan, Ruoming Pang, and Quoc V. Le. "EfficientDet: Scalable and Efficient Object Detection". In: 2020. URL: https://arxiv.org/abs/1911.09070.

[38] Mingxing Tan, Ruoming Pang, and Quoc V. Le. "EfficientDet: Scalable and Efficient Object Detection". In: 2020. URL: https://arxiv.org/abs/1911.09070.

[39] Mingxing Tan and Quoc V. Le. "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks". In: *CoRR* abs/1905.11946 (2019). arXiv: 1905.11946. URL: http://arxiv.org/abs/1905.11946.

[40] Chien-Yao Wang et al. "CSPNet: A New Backbone that can Enhance Learning Capability of CNN". In: *arXiv e-prints*, arXiv:1911.11929 (Nov. 2019), arXiv:1911.11929. arXiv: 1911.11929 [cs.CV].

[41] Kaiming He et al. "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition". In: *CoRR* abs/1406.4729 (2014). arXiv: 1406.4729. URL: http://arxiv.org/abs/1406.4729.

[42] Liang-Chieh Chen et al. "DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs". In: *CoRR* abs/1606.00915 (2016). arXiv: 1606.00915. URL: http://arxiv.org/abs/1606.00915.

[43] Songtao Liu, Di Huang, and Yunhong Wang. "Receptive Field Block Net for Accurate and Fast Object Detection". In: *CoRR* abs/1711.07767 (2017). arXiv: 1711.07767. URL: http://arxiv.org/abs/1711.07767.

[44] Sanghyun Woo et al. "CBAM: Convolutional Block Attention Module". In: *CoRR* abs/1807.06521 (2018). arXiv: 1807.06521. URL: http://arxiv.org/abs/1807.06521.

[45] Songtao Liu, Di Huang, and Yunhong Wang. "Learning Spatial Fusion for Single-Shot Object Detection". In: *arXiv e-prints*, arXiv:1911.09516 (Nov. 2019), arXiv:1911.09516. arXiv: 1911.09516 [cs.CV].

[46] Qijie Zhao et al. "M2Det: A Single-Shot Object Detector based on Multi-Level Feature Pyramid Network". In: *CoRR* abs/1811.04533 (2018). arXiv: 1811.04533. URL: http://arxiv.org/abs/1811.04533.

[47] Hei Law and Jia Deng. "CornerNet: Detecting Objects as Paired Keypoints". In: *CoRR* abs/1808.01244 (2018). arXiv: 1808.01244. URL: http://arxiv.org/abs/1808.01244.

[48] Kaiwen Duan et al. "CenterNet: Keypoint Triplets for Object Detection". In: *CoRR* abs/1904.08189 (2019). arXiv: 1904.08189. URL: http://arxiv.org/abs/1904.08189.

[49] Abdullah Rashwan, Agastya Kalra, and Pascal Poupart. "Matrix Nets: A New Deep Architecture for Object Detection". In: *arXiv e-prints*, arXiv:1908.04646 (Aug. 2019), arXiv:1908.04646. arXiv: 1908.04646 [cs.CV].

[50] Zhi Tian et al. "FCOS: Fully Convolutional One-Stage Object Detection". In: *CoRR* abs/1904.01355 (2019). arXiv: 1904.01355. URL: http://arxiv.org/abs/1904.01355.

[51] Kaiming He et al. *Mask R-CNN*. cite arxiv:1703.06870Comment: open source; appendix on more results. 2017. URL: http://arxiv.org/abs/1703.06870.

[52] Ze Yang et al. "RepPoints: Point Set Representation for Object Detection". In: *arXiv e-prints*, arXiv:1904.11490 (Apr. 2019), arXiv:1904.11490. arXiv: 1904.11490 [cs.CV].

[53] Zhun Zhong et al. "Random Erasing Data Augmentation". In: *CoRR* abs/1708.04896 (2017). arXiv: 1708.04896. URL: http://arxiv.org/abs/1708.04896.

[54] Terrance Devries and Graham W. Taylor. "Improved Regularization of Convolutional Neural Networks with Cutout". In: *CoRR* abs/1708.04552 (2017). arXiv: 1708.04552. URL: http://arxiv.org/abs/1708.04552.

[55] Krishna Kumar Singh et al. "Hide-and-Seek: A Data Augmentation Technique for Weakly-Supervised Localization and Beyond". In: *CoRR* abs/1811.02545 (2018). arXiv: 1811.02545. URL: http://arxiv.org/abs/1811.02545.

[56] Pengguang Chen et al. "GridMask Data Augmentation". In: *arXiv e-prints*, arXiv:2001.04086 (Jan. 2020), arXiv:2001.04086. arXiv: 2001.04086 [cs.CV].

[57] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *J. Mach. Learn. Res.* 15.1 (Jan. 2014), pp. 1929–1958. ISSN: 1532-4435.

[58] Li Wan et al. "Regularization of Neural Networks using DropConnect". In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 1058–1066. URL: http://proceedings.mlr.press/v28/wan13.html.

[59] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V. Le. "DropBlock: A regularization method for convolutional networks". In: *arXiv e-prints*, arXiv:1810.12890 (Oct. 2018), arXiv:1810.12890. arXiv: 1810.12890 [cs.CV].

[60] Hongyi Zhang et al. "mixup: Beyond Empirical Risk Minimization". In: *CoRR* abs/1710.09412 (2017). arXiv: 1710.09412. URL: http://arxiv.org/abs/1710.09412.

[61] Sangdoo Yun et al. "CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features". In: *CoRR* abs/1905.04899 (2019). arXiv: 1905.04899. URL: http://arxiv.org/abs/1905.04899.

[62] Xu Zheng et al. "STaDA: Style Transfer as Data Augmentation". In: *arXiv e-prints*, arXiv:1909.01056 (Sept. 2019), arXiv:1909.01056. arXiv: 1909.01056 [cs.CV].

[63] Kah-Kay Sung and Tomaso Poggio. "Example-Based Learning for View-Based Human Face Detection". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 20.1 (Jan. 1998), pp. 39–51. ISSN: 0162-8828. DOI: 10.1109/34.655648. URL: https://doi.org/10.1109/34.655648.

[64] Abhinav Shrivastava, Abhinav Gupta, and Ross B. Girshick. "Training Region-based Object Detectors with Online Hard Example Mining". In: *CoRR* abs/1604.03540 (2016). arXiv: 1604.03540. URL: http://arxiv.org/abs/1604.03540.

[65] Tsung-Yi Lin et al. "Focal Loss for Dense Object Detection". In: *The IEEE International Conference on Computer Vision (ICCV)*. Oct. 2017.

[66] Christian Szegedy et al. "Rethinking the Inception Architecture for Computer Vision". In: *CoRR* abs/1512.00567 (2015). arXiv: 1512.00567. URL: http://arxiv.org/abs/1512.00567.

[67] Md. Amirul Islam et al. "Label Refinement Network for Coarse-to-Fine Semantic Segmentation". In: *CoRR* abs/1703.00551 (2017). arXiv: 1703.00551. URL: http://arxiv.org/abs/1703.00551.

[68] Jiahui Yu et al. "UnitBox: An Advanced Object Detection Network". In: *CoRR* abs/1608.01471 (2016). arXiv: 1608.01471. URL: http://arxiv.org/abs/1608.01471.

[69] Zhaohui Zheng et al. "Distance-IoU Loss: Faster and Better Learning for Bounding Box Regression". In: *arXiv e-prints*, arXiv:1911.08287 (Nov. 2019), arXiv:1911.08287. arXiv: 1911.08287 [cs.CV].

[70] S. Lazebnik, C. Schmid, and J. Ponce. “Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories”. In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’06)*. Vol. 2. 2006, pp. 2169–2178.

[71] Tsung-Yi Lin et al. “Microsoft COCO: Common Objects in Context”. In: *CoRR* abs/1405.0312 (2014). arXiv: 1405.0312. URL: http://arxiv.org/abs/1405.0312.

[72] Jie Hu, Li Shen, and Gang Sun. “Squeeze-and-Excitation Networks”. In: *CoRR* abs/1709.01507 (2017). arXiv: 1709.01507. URL: http://arxiv.org/abs/1709.01507.

[73] Jonathan Long, Evan Shelhamer, and Trevor Darrell. “Fully Convolutional Networks for Semantic Segmentation”. In: *CoRR* abs/1411.4038 (2014). arXiv: 1411.4038. URL: http://arxiv.org/abs/1411.4038.

[74] Bharath Hariharan et al. “Hypercolumns for Object Segmentation and Fine-grained Localization”. In: *CoRR* abs/1411.5752 (2014). arXiv: 1411.5752. URL: http://arxiv.org/abs/1411.5752.

[75] Geoffrey Hinton. “Rectified Linear Units Improve Restricted Boltzmann Machines Vinod Nair”. In: vol. 27. June 2010, pp. 807–814.

[76] Kaiming He et al. “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”. In: *CoRR* abs/1502.01852 (2015). arXiv: 1502.01852. URL: http://arxiv.org/abs/1502.01852.

[77] Günter Klambauer et al. “Self-Normalizing Neural Networks”. In: *CoRR* abs/1706.02515 (2017). arXiv: 1706.02515. URL: http://arxiv.org/abs/1706.02515.

[78] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. “Searching for Activation Functions”. In: *CoRR* abs/1710.05941 (2017). arXiv: 1710.05941. URL: http://arxiv.org/abs/1710.05941.

[79] Andrew Howard et al. “Searching for MobileNetV3”. In: *CoRR* abs/1905.02244 (2019). arXiv: 1905.02244. URL: http://arxiv.org/abs/1905.02244.

[80] Diganta Misra. “Mish: A Self Regularized Non-Monotonic Neural Activation Function”. In: *arXiv e-prints*, arXiv:1908.08681 (Aug. 2019), arXiv:1908.08681. arXiv: 1908.08681 [cs.LG].

[81] Ross B. Girshick et al. “Rich feature hierarchies for accurate object detection and semantic segmentation”. In: *CoRR* abs/1311.2524 (2013). arXiv: 1311.2524. URL: http://arxiv.org/abs/1311.2524.

[82] Navaneeth Bodla et al. “Improving Object Detection With One Line of Code”. In: *CoRR* abs/1704.04503 (2017). arXiv: 1704.04503. URL: http://arxiv.org/abs/1704.04503.

[83] Ilya Loshchilov and Frank Hutter. “SGDR: Stochastic Gradient Descent with Restarts”. In: *CoRR* abs/1608.03983 (2016). arXiv: 1608.03983. URL: http://arxiv.org/abs/1608.03983.