Research Internship Thesis

Automated Detection in Archaeological Geophysical Surveys

Leiden University

ESIEA

Martin Olivier

June 24, 2020

Contents

1 Foreword **6**

2 Introduction **7**

I Context **8**

3 Organisation **9**

II Preliminaries **10**

4 General context **11**

5 Elementary concepts **12**
5.1 LiDAR . . . 12
5.2 Archaeology Structures . . . 12
5.3 Deep Learning . . . 12

6 Metrics **15**
6.1 Type of Errors . . . 15
6.2 Precision and Recall . . . 15
6.3 Intersection over Union (IoU) . . . 16
6.4 Mean Average Precision (mAP) and F1-Score . . . 17
6.5 Evaluating a model . . . 17

7 Challenges **18**
7.1 Detection of objects in LiDAR: challenges . . . 18
7.2 Dataset . . . 19

8 State of the Art **20**
8.1 You Only Look Twice[18] . . . 20
8.1.1 Network Architecture . . . 20
8.1.2 Scale Mitigation . . . 21
8.1.3 Training . . . 21
8.1.4 Results . . . 21
8.2 A Simple and Efficient Network for Small Target Detection[23] . . . 22
8.2.1 Proposed Modules . . . 22
8.2.2 General Architecture . . . 23
8.2.3 Results . . . 24

8.3 Object Detection in Remote Sensing Images Based on Improved Bounding Box Regression and Multi-Level Features Fusion[27] . . . 26
8.3.1 Architecture . . . 26
8.3.2 Generalized Intersection over Union . . . 27
8.3.3 Bounding Box Regression based on Improved GIoU Loss (IGIoU) 27
8.3.4 Results . . . 28
8.4 A Single Shot Framework with Multi-Scale Feature Fusion for Geospatial Object Detection[30] . . . 29
8.4.1 Architecture . . . 29
8.4.2 Results . . . 33

9 State of the Art **35**
9.1 Automatic Detection in Magnetostatic Surveys using Convolutional Neural Networks . . . 35
9.2 Learning To Look At LiDAR . . . 35

10 The YOLO Framework **36**
10.1 General Principle . . . 36
10.2 Network design and Architecture . . . 36
10.3 YOLOv4 . . . 38
10.3.1 General Architecture of a Object Detector . . . 38
10.3.2 Bag of Freebies . . . 39
10.3.3 Bag Of Specials . . . 40
10.3.4 Additional Improvements . . . 41
10.3.5 YOLOv4 Architecture . . . 43
10.3.6 Results . . . 44
10.3.7 Influence of different backbones and pre-trained weights on the detector . . . 46

III Implementation and Results **48**

11 Dataset **49**
11.1 Dataset creation . . . 49
11.1.1 Cropping Images . . . 49
11.1.2 Annotations . . . 50

12 Architecture **52**
12.1 Feature Fusion . . . 52
12.2 Backbone and activation . . . 52
12.3 Activation Functions . . . 53
12.3.1 Scaled Exponential Linear Unit . . . 53
12.3.2 Swish . . . 54
12.3.3 Mish . . . 55
12.4 Dual Scale Detectors . . . 55
12.5 Multi-scale Feature Fusion . . . 55
12.6 Receptive Field Improvements . . . 57
12.7 Attention Modules . . . 58
12.8 Post-processing . . . 58
12.9 List of all the possible testing arrangements . . . 58

12.10YOLOv4 59

13 Training **60**
13.1 Data Augmentations 60
13.1.1 Image Modification 60
13.1.2 Regularization and Normalization 61
13.2 Self Adversarial Training 61
13.3 Loss Systems 61
13.3.1 GIoU, DIoU and CIoU 62
13.3.2 Loss flooding 63
13.4 Training methodology 64

14 Results **65**

15 Discussion **66**

IV Conclusions **67**

16 Further Improvements **68**

17 Conclusion **69**

Acronymes **71**

Glossary **73**

Bibliography **74**

List of Figures

5.1 Example of overfitting on a classification task . . . 13
5.2 Gradient Descent . . . 14

6.2 Intersection over Union . . . 17

7.1 A natural image *versus* a LiDAR survey . . . 18

8.1 Dilated Convolution Example . . . 22
8.2 General Architecture for the Simple and Efficient Network for Small Target Detection . . . 23
8.3 General Architecture of the ODRSI . . . 26
8.4 Architecture of the Darknet 53, used as a base network for features extraction in the Single-Shot Object Detection Framework . . . 29
8.7 Anchors and location predictions . . . 32

10.1 YOLO model . . . 37
10.2 Architecture of the backbone of the YOLO Framework . . . 37
10.3 General Architecture of an Object Detector . . . 38
10.4 Some data augmentation techniques . . . 40
10.7 SAM and its YOLO modification . . . 43
10.8 PAN and its YOLO modification . . . 43

11.1 Functionnal Diagram of the createAnnotation script . . . 50

12.2 Scaled Exponential Linear Unit (SELU) Activation Function . . . 54
12.3 The Swish activation function, with different values of β . . . 54
12.4 The mish activation function . . . 55
12.5 Feature Fusion in Zhuang *et al.* . . . 56
12.6 Architecture of Qian *et al.* . . . 57
12.7 PAN and its YOLO modification . . . 57
12.9 SAM and its YOLO modification . . . 58

13.1 Regularisation by dropping: random drops *v.s.* DropBlock . . . 61
13.2 Bounding boxes overlap cases . . . 62
13.3 Loss flooding . . . 64

List of Tables

8.1 YOLT Detection performance on all classes 21
8.2 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. 24
8.3 Comparison of the ODRSI against four existing detection framework on the NWPU VHR-10 28

10.1 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 44
10.2 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 44
10.3 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. 45
10.4 Comparison study of different classifier pre trained weights for detector trainings. 46
10.5 Comparison study of different mini-batch size for detector training. . . . 46

Chapter 1

Foreword

This internship was not as traditional as it could have been: both because of the COVID-19 Situation, which profoundly modified the course of my work, but also because it was at its heart a research position, and not a more traditional developper/manager type of work. While this type of work is not typically assumed by engineers in their final year of study, this experience has proven to me to be unvaluable, as it has allowed me to delve deep into research I have been doing in my free time; and gave me the means and the time to create what was only a dream for me a few years ago: a dedicated object detection model optimized for Archaelogical Geophysical Surveys.

As such, this document is also not traditional, and will take both the place of an internship report, and a form of thesis, where I will detail the work I have been undertaken during the past two years in the field of automated detection in geophysical surveys.

Chapter 2

Introduction

This document is served as the final assignment of my graduate engineering degree which concludes my 5 year as an ESIEA Student. This document details my work during my 6 month stay as an intern and research student at Leiden University.

I can only give my deepest and most sincere thanks to the Leiden University and both Professor Lambers and Mr. V. D. Schoof for their help. This institution accepted my offer and gave me an opportunity to push myself, which I will stay forever grateful for.

Part I

Context

Chapter 3

Organisation

Part II

Preliminaries

Chapter 4

General context

Chapter 5

Elementary concepts

The aim of this short section is to present the various domains of studies seen in this work and explain concepts that are necessary for comprehension.

5.1 LiDAR

LiDAR is a method for accurately measuring distances and stands for "*light detection and ranging*". LiDAR is used to make high resolution depth maps, and has seen uses in geography, seismology, forestry ... Along with Archaeology. LiDAR has also been in use in some autonomous vehicles. LiDAR is able to map out large regions of terrain and produces very high resolution data, while being relatively cheap. In essence, LiDAR accurately captures the distance between the sensor and an object, similar in how a sonar works.

In Archaeology, LiDAR is used to create DEM, that are then integrated into GIS for analysis. By carefully choosing the wavelength used, LiDAR is able to see through vegetation and show structure that would be otherwise hidden. This makes it a very valuable tool for Archaeology.

LiDAR works by emitting short laser pulses and measuring the time of flight between its emission and the time at which it hits an object.

5.2 Archaeology Structures

5.3 Deep Learning

Deep Learning is a ensemble of techniques which are based on Artificial Neural Networks, with multiple layers stacked upon each other. Those layers will progressively extract higher level features from the input.

One of the most successful innovation of Deep Learning would be Convolutional Neural Networks, or CNN, first introduced by Yann LeCun[1]. Convolution Neurons applies a convolution operation on its input. The matrix that is convoluted is learned through

training. CNNs are commonly used with images, but have also seen uses in audio signal processing.

CNNs had massive success during the 2010's, where the use of gpu allowed for a massive increase in computation speed which rendered the use of deeper and more powerful network possible. One of the first of such network was AlexNet[2] which was trained on a commercial gpu and outperformed competitors by more than 10 points. Other quickly followed suit, with deeper and more performant nets being developed each year, like GoogleNet[3], VGG[4] and resNet[5] which introduced the idea of feature fusion via residual unit. Today, deep convolutional networks with residual units are a staple in computer vision, but are also used in audio analysis[6][7][8][9][10] among various other field with great success.

The work here centers around the task of object detection. This task is actually two fold: given an input image, localise an object then give a classification of this object. A good object detector must be able to do both, but they are evaluated differently.

A Deep Learning model is trained on a dataset. Deep Learning requires very large amounts of data to correctly train as they are very prone to overfitting. In statistics, a model is said to overfit when it "describes features that arises from noise or variance in the data, rather than the underlying distribution from which the data is drawn"[11]. In other words, this means that the model has "memorised" the data, and will perform badly on data which it has never seen before. In Figure 5.1, we see a classification task where a curve has to be drawn to differentiate two class and the green line is overfitted to the data, while the black line is a more reasonable fit.

Overfitting is an issue that affects all Deep Learning models, but a lot of effort has been done to address this, either by creating a more diverse and robust dataset with data augmentation, or by modifying the network itself via regularisation and normalisation.

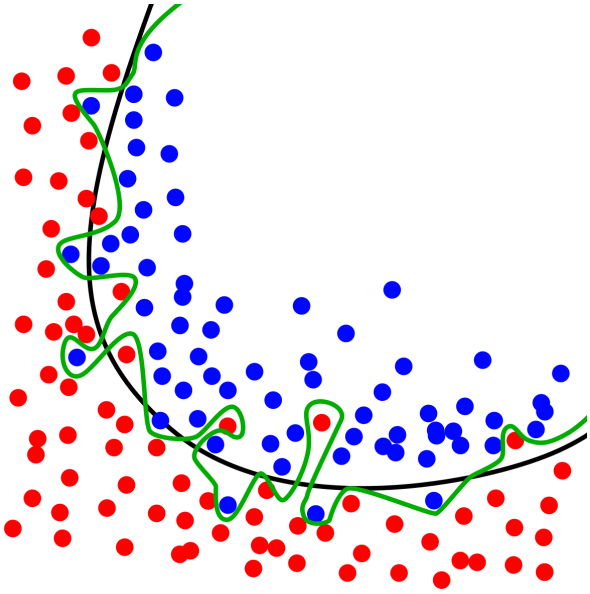


Figure 5.1: Example of overfitting on a classification task. The black line represents a good fit to the data, with an acceptable error rate or, misclassified data points. The green represents an overfit: while the error rate is lower than the black line, the fit is too complicated, and will badly generalize on unseen data. Doc. Wikipedia

There are two main parts of training a model on a particular dataset. First the dataset

is divided in two sub dataset. One is used for training, and the other for evaluation. For simplicity, we will use an example of classification of an input image. During training, the model will produce a inference based on the input data *i.e.* a label. We will compare the inferred label to the ground truth using a particular metric, like Categorical Cross-Entropy or MSE and obtain a distance between the two labels. This distance is known as the loss. We will try to reduce the loss by optimising the weights of the model using gradient descent, as shown on Figure 5.2.

Given a multi variable differentiable function $F(x)$ that is defined in the neighborhood of a point $a \in R^n$, the gradient descent is defined as follows:

$$a_{n+1} = a_n - \gamma \nabla F(a_n) \tag{5.1}$$

With a small $\gamma \in R_+$, we have $F(a_{n+1}) \leq F(a_n)$, meaning that $\gamma F(a)$ is subtracted from a, moving toward a minimum. The parameter γ is called the learning rate. A large learning rate will mean that the descent might converge faster, but might "overshoot" and miss the minimum, while a small learning rate means a more precise convergence, but will dramatically increase the number of steps necessary. On Figure 5.2, the learning represent the length of each arrow during the descent. It should be noted that nowadays, gradient descent is rarely used with modern networks, as most uses other optimizers, such as ADAM[12] or ADAGRAD[13].

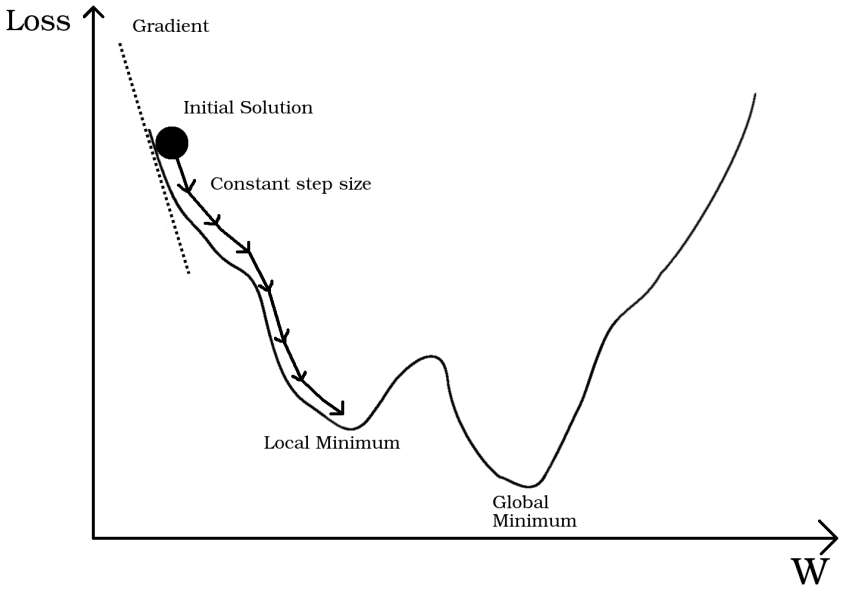


Figure 5.2: Gradient Descent on a loss function with one parameter. Since the loss function of neural network is not convex, there are possibly multiple minimum, which the gradient descent algorithm can get "stuck" into. As the dimensionality of the problem increase, the complexity of the dynamics of the gradient descent increase.

Since the gradient gives the direction of steepest incline of a differentiable function $F(x)$, this means that the function $F(x)$ declines the fastest in the direction of negative gradient. In Deep Learning backpropagation is used, an recursive and efficient process that is able to compute the gradient with respect to the weight of the network.

Chapter 6

Metrics

One of the hardest part of creating and training ML models is accurately evaluating their performance. How can you known if your model is giving out accurate predictions ? What is accuracy, and how can we precisely define it ? Understanding the type of errors a detection network can make is a crucial part in enhancing their performance in real world situation. In this section, we will present metrics used in detection and classification tasks.

6.1 Type of Errors

In a classification task, where a model is ask to classify an item into a fixed number of categories based on input characteristics. A model will predict the class of a given item, for example if an input image contains a cat or a dog. In doing so, the model can make mistakes. When predicting a particular class, there are 2 type of predictions that can be made: positive or negative, and those predictions can be true or false. When a network correctly predicts that an image contains a cat, it is a true positive. When it predicts that there is no dog when there is one, this is a false negative.

All in all, there are 4 categories of predictions: True Positive (TP), False Positive (FP), True Negative (TN) and False Negative (TN).

6.2 Precision and Recall

Precision is defined as "the fraction of detections reported by model that were correct" and recall is "the fraction of detections reported by the model which were correct"[14, p.411]. We can see those concepts visualised in Figure 6.1. In that Figure, the precision would be equal to the number True Positives in green divided by number of selected elements *i.e.* the green and red sets combined. The recall would be equal to the number of true positives over the total number of relevant elements or the dark gray set and the green set combined.

Precision and recall allows us to quantify with more accuracy the quality of a model than just the number of errors that it makes. For example, in the case of a model trying to detect a rare event like a disease that would infect only a small fraction of the population,

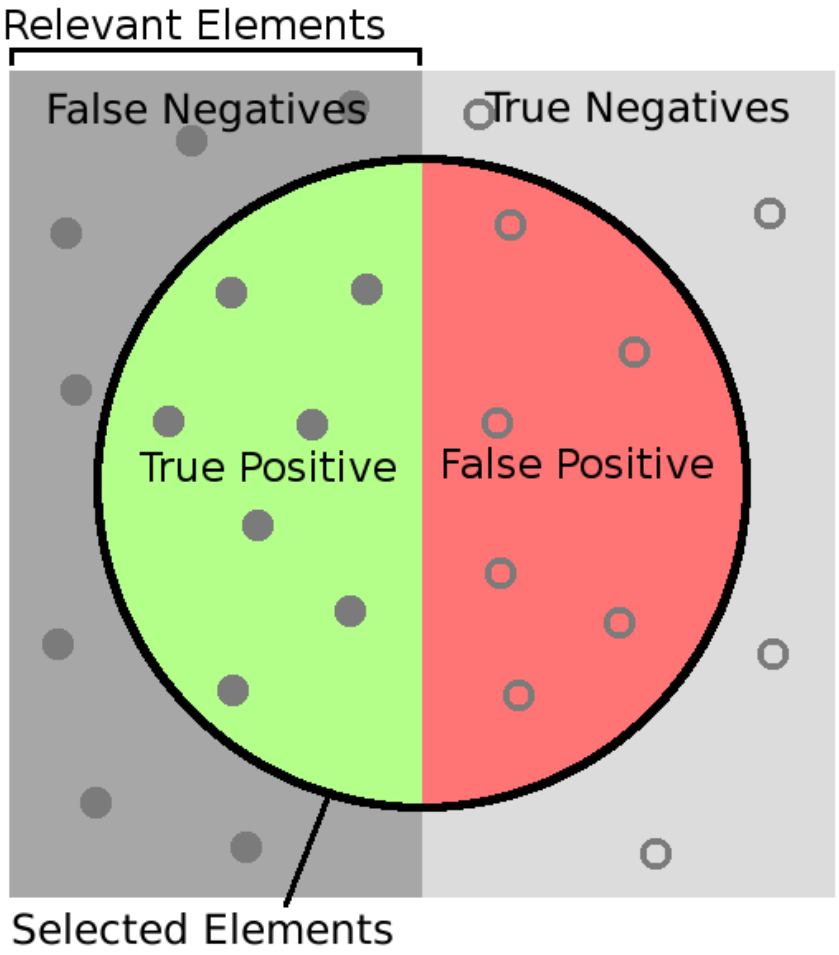


Figure 6.1: Precision and recall on a classification task.

say one in a million, **a detector that would always infer that a person is not sick would be able to attain a** 99.9999% **accuracy.** However, while this detector would achieve a very high precision, it would have a very bad recall, which would indicate that it is not a very good model after all.

6.3 Intersection over Union (IoU)

The intersection over union, or Jaccard Index quantifies the accuracy of the bounding box prediction. IoU measure the quality of a bounding box : if the predicted bounding box overlaps significantly with the ground truth bounding box, then it should have a low score, and inversely if the prediction does not overlap with the ground truth. The IoU is given by the following equation:

$$IoU = \frac{\text{area}(BBox_{truth} \cap BBox_{pred})}{\text{area}(BBox_{truth} \cup BBox_{pred})} \tag{6.1}$$

Generally a detection is considered correct if the IoU ≥ 0.5; this threshold is also used when computing the mAP. It should also be noted that the IoU is not the only way to measure how good a bounding box is. Improvements on this metric has been done that

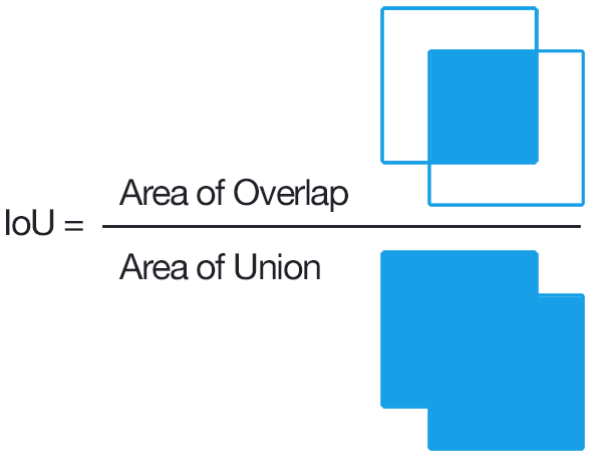


Figure 6.2: Intersection over Union
Source: https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/

gives out a better measure on the quality of the box, such as the GIoU[15], dIoU[16] and cIoU[16]. Those metrics are used in our model, and are presented in Section **??**.

6.4 Mean Average Precision (mAP) and F1-Score

The F1-Score is a measure of test accuracy. It is the harmonic mean of precision and recall :

$$2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{6.2}$$

The mean average precision is the metric used by the COCO dataset[17]. As said above, when the IoU between the prediction and the ground truth is over 0.5, the prediction is considered a valid detection.

6.5 Evaluating a model

Computing those metrics and analyzing them allows a researcher to better understand the behavior of its model, and preventing degenerate behavior. It should however be noted that those metrics do not always give out accurate or useful information. In that case, only knowledge and experience of the usual issues can help.

For example, it can happen that a model will always infer one class other another, for any kind of object and still outputs good accuracy [1]. It can also be the case that a metric is very sensible at a some scale of errors but not at others: the L2 distance for example grows quadratically and outputs small distances when the L1 distance is below 1, but very high values that might be hard to optimize for when the L1 distance is over 1.

The only way to diagnose those issues will be to manually inspect the prediction of the model, and carefully choosing the metrics used.

[1] This often happens in the case of an unbalanced dataset, where one class appear more often than any other.

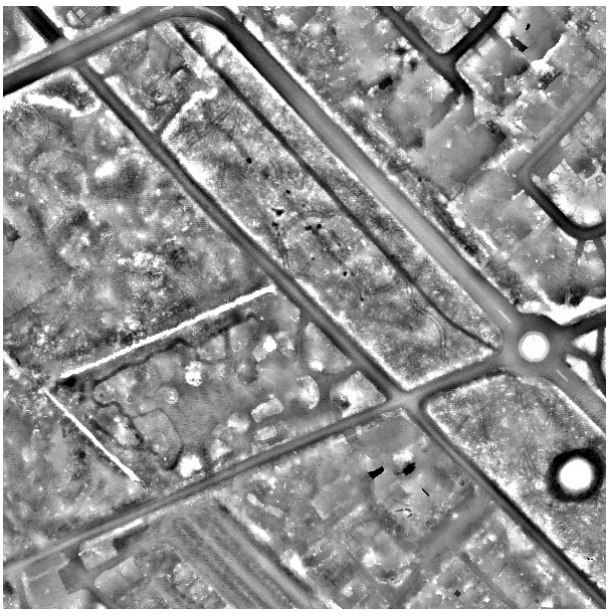
Chapter 7

Challenges

7.1 Detection of objects in LiDAR: challenges

Traditional work in object detection focusses on what we will call "natural images", which are photographs of scenes seen in normal settings. For examples, people sitting at a table, or children in front of horses. They are similar to images we would see with our own eyes. This means that, if there is an object in a photograph, there is a high probability that this object also exist and is the same category in the real scene that has been photographed. In other words, there is a good correspondence between the real object, its classification, and the kind of signal it emits in the visual field. A photograph of a camel will contain a camel, and we are able to very confidently classify the image as containing a camel, even though we are only classifying a *visual representation*, not the object itself.

(a) A natural image, taken from the COCO[17] dataset



(b) An example from the LiDAR dataset. On the bottom right, we can see an example of a barrow. However, this barrow is very similar to the roundabount seen slightly above.

Figure 7.1: A natural image *versus* a LiDAR survey

In geophysical surveys, or more generally with any kind of image that is not made of

purely visible light, the difference between the representation and the real object is more pronounced, and while it might be easy to detect and classify objects in the representation, to guarantee that those detected objects always match up with a real object is much more tricky. How can we say that the white mass seen in an X-Ray scanner that we detect and classify as a tumor is not only indeed a tumor in its representation, but also match up with a real tumor in a patient? The problem is even more prevalent with geophysical data, with noise and geological details adding yet another layer of complexity.

Other issues that are specific and prevalent in geophysical data are the size and resolution of the images. Whereas in natural images might be around a thousand pixels in width, geophysical surveys cover hundreds of kilometer square of area, and are often tens of thousands of pixels wide. Moreover, the size of the object to detect is tiny in comparison to the size of the image, which is not usually the case in natural images. Traditional object detection techniques take advantage of this by heavily downscaling the image through the layers of the network, greatly reducing the computational cost. This is not possible to do with geophysical surveys: a barrow, which might be a few meters wide in reality will be only a few pixels wide, and by downscaling we might lose all informations concerning those small objects.

When working with geophysical data, one must always keep in mind those issues. In our case, the only reliable way of classifying archaelogical objects is by prospection and physical inspection, something that **cannot be done using only geophysical surveys**. However, this create another issue, one of scale. In the dataset that I had at my disposition, more than hundreds of kilometers squared were surveyed and contains more than 3000 objects of different classes. It is almost impossible to verify for every object if they are of the correct classification by way of prospecting: it is too costly and too much time consuming.

In short, even if the annotations in the dataset are entirely correct, something that is very hard to confirm, and even if the network architecture is fit for the task and trained well, something that is even harder to do, we will still have an incertainty as to wether the detected object are not only actually real, but of the correct class.

7.2 Dataset

In of itself, it is quite challenging to obtain large segments of LiDAR surveys, or for that matter any kind of geophysical surveys. Deep Learning techniques usually requires vast amount of data to train properly. However, here data can be prohibitively expensive to produce: a geophysical survey of an area is itself a challenge, with funding to secure and plannification to be done.

In Archaelogy, the practice of Open Data, the act of freely sharing the data produced, is not yet widely adopted for a multitude of reasons.

Chapter 8

State of the Art in Detection in Satellite Imagery

A detailled SotA have already been done, and is available as its own document. For the sake of brievity, we will not include its entirety in this section, but we will only highlight the major contribution of each paper.

8.1 You Only Look Twice[18]

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[19] 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.

8.1.1 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[5] and was first used in YOLOv9000[20].

Each layer uses Batch Normalization[21] and uses the leaky-ReLU activation[22], except for the last layer, which uses a linear activation. This final layer provides the predictions

for the bounding boxes and class.

8.1.2 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.

8.1.3 Training

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.

8.1.4 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 8.1 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 8.1: 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.

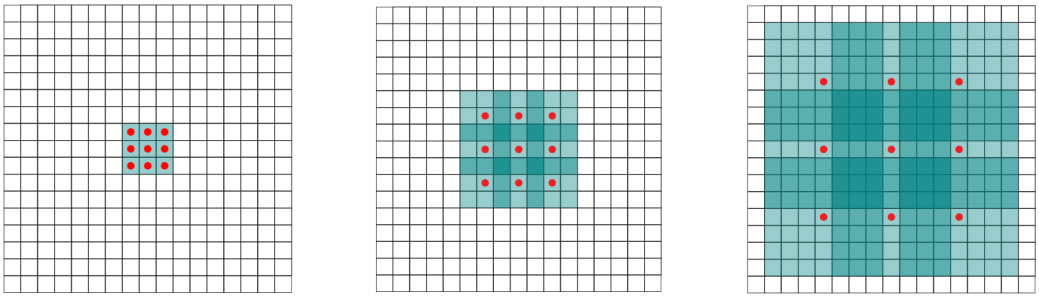
8.2 A Simple and Efficient Network for Small Target Detection[23]

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[18], described in section 8.1 and in ResNet[5] 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[24] dataset along with the DOTA[25] dataset, and obtains state of the art results, with FPS performance comparable to a tiny YOLOv3 network[19] but with average precision comparable to a "full size" YOLOv3 network.

8.2.1 Proposed Modules

Dilated Modules



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

Figure 8.1: 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 8.1 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"[26].

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.

Passthrough module

In a detection network, earlier layer contains more fine grained information which can be useful to detect and accurately 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 $N \times N \times 4C$. 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.

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. 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. Since the feature maps undergoes downsampling, their dimensions changes. The paper propose to unify their dimension by using another passthrough layer and upsampling.

8.2.2 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.

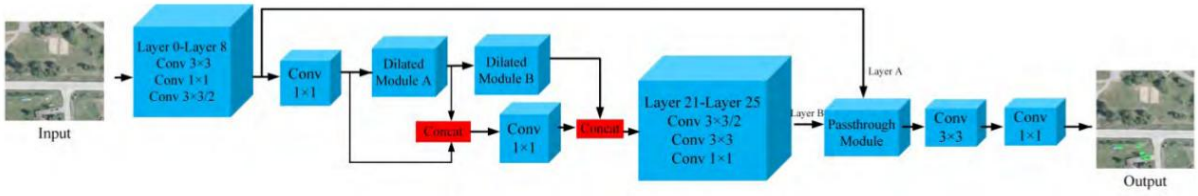


Figure 8.2: General Architecture

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 information 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.

8.2.3 Results

The models were trained and tested on the VEDAI[24] and DOTA datasets[25]. 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.

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.

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 8.2). 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 (Higher is better)	58.3	**76.4**	14.7	75.4
BFLOPS (Lower is better)	44.417	**8.243**	101.784	9.692
AP (Higher is better)	54.33	58.17	**85.37**	80.16

Table 8.2: 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.

In short, this model combine the precision of YOLOv3 with the fast and efficient computation of the tiny model.

8.3 Object Detection in Remote Sensing Images Based on Improved Bounding Box Regression and Multi-Level Features Fusion[27]

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 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 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[28].

8.3.1 Architecture

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[5] across multiple levels and concatenates them along the channel dimension. The fused features are utilized for bounding box regression and classification. A novel GIoU loss is proposed, instead of the smooth L1 loss.

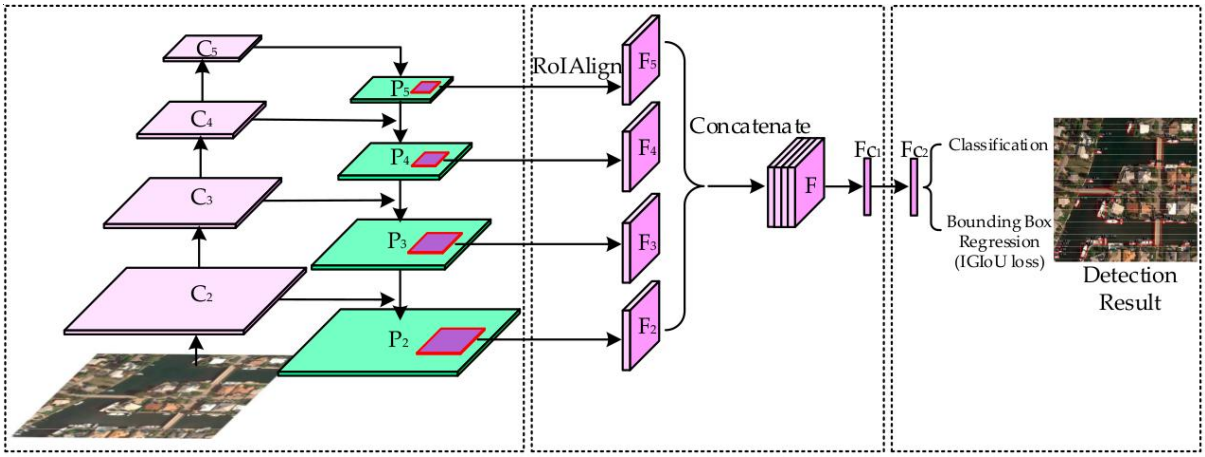


Figure 8.3: Architecture of the proposed framework. The left part shows the feature pyramid network. Multilevel features fusion is shown in the middle, and classification and bounding box regression based on the IGIoU loss is shown in the rightmost part.

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 8.3 using RoiAlign[5]. 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.

8.3.2 Generalized Intersection over Union

A novel metric, the Generalized IoU (GIoU) is proposed to enhance the evaluation of proximity between two bounding boxes. The traditional IoU is insensitive to the scales of bounding boxes, and can be calculated using formula 8.1. Let B_{GT} be the ground truth bounding box and B_{PT} be the predicted bounding box.

$$IoU = \frac{area(B_{GT} \cap B_{PT})}{area(B_{GT} \cup B_{PT})} \tag{8.1}$$

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 IoU is as follows:

$$GIoU = IoU + \frac{area(B_{GT} \cup B_{PT})}{area(B_{EC})} - 1 \tag{8.2}$$

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.

8.3.3 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 8.3 from Rezatofighi et al. [15].

$$L_{GIoU} = 1 - GIoU \tag{8.3}$$

This 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:

$$L_{IGIoU} = 2 \times log_2 - 2 \times log(1 + GIoU) \tag{8.4}$$

8.3.4 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[28]. Those results are listed in table 8.3

Table **??** shows results of the proposed method on the DIOR[29] dataset. The DIOR dataset is a large scale benchmark, of size comparable to the DOTA dataset[25]. 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 8.3.

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

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

8.4 A Single Shot Framework with Multi-Scale Feature Fusion for Geospatial Object Detection[30]

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[18] 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[28] dataset.

8.4.1 Architecture

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[21] instead of dropout is used to control overfitting and convergence during training. The network uses leaky ReLU activations on all convolutional layers.

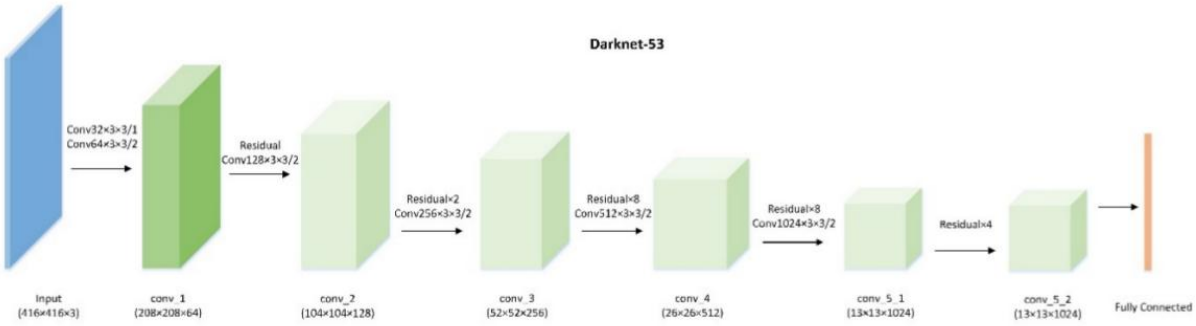


Figure 8.4: Architecture of the Darknet 53, used as a base network for feature extraction

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

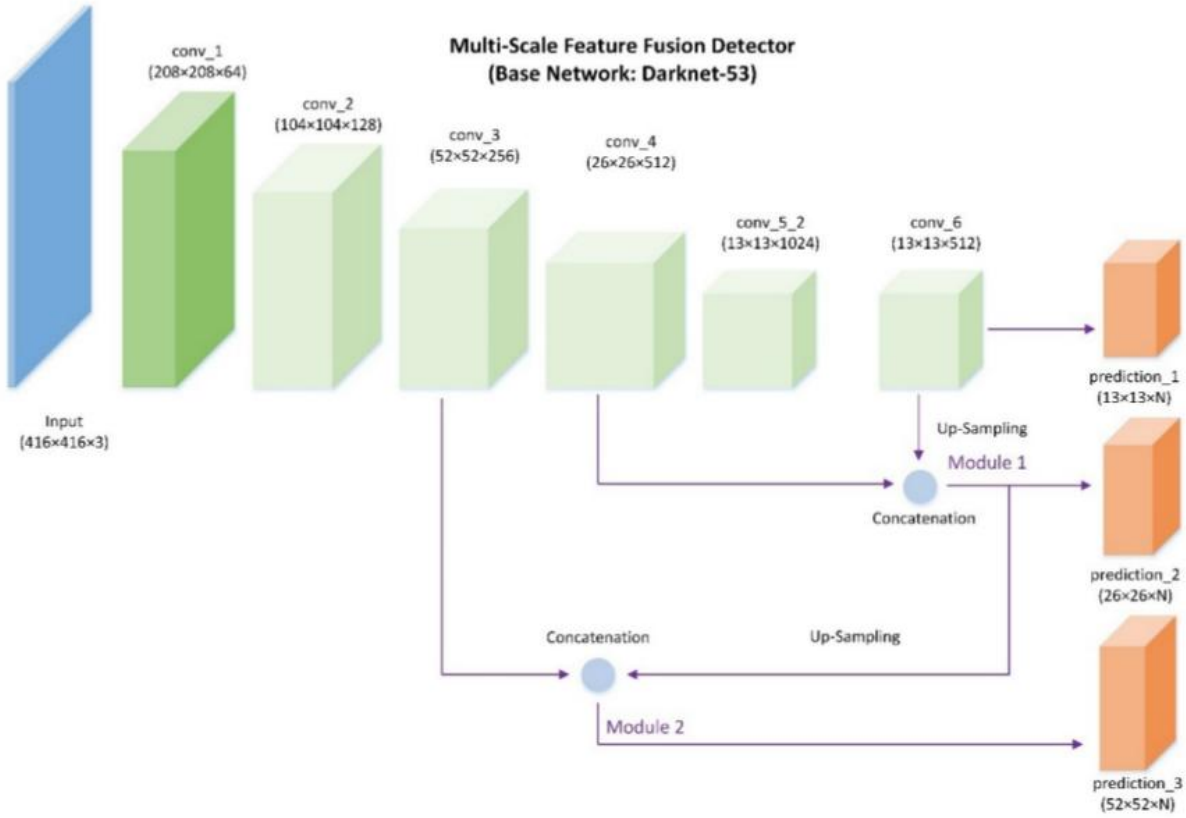


Figure 8.5: General architecture of the multi-scale feature fusion detector. The model uses Darknet-53 as the base feature extractor. Three predictions are generated at three different scales.

prediction of the `conv_6` layer, upsamples it 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.

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 8.6 show details of the feature fusion module.

Anchors and predictions

Since the model is unstable during early training iterations, anchors are used, similar to the ones used in Faster R-CNN [31]. The designed network outputs three kind of feature

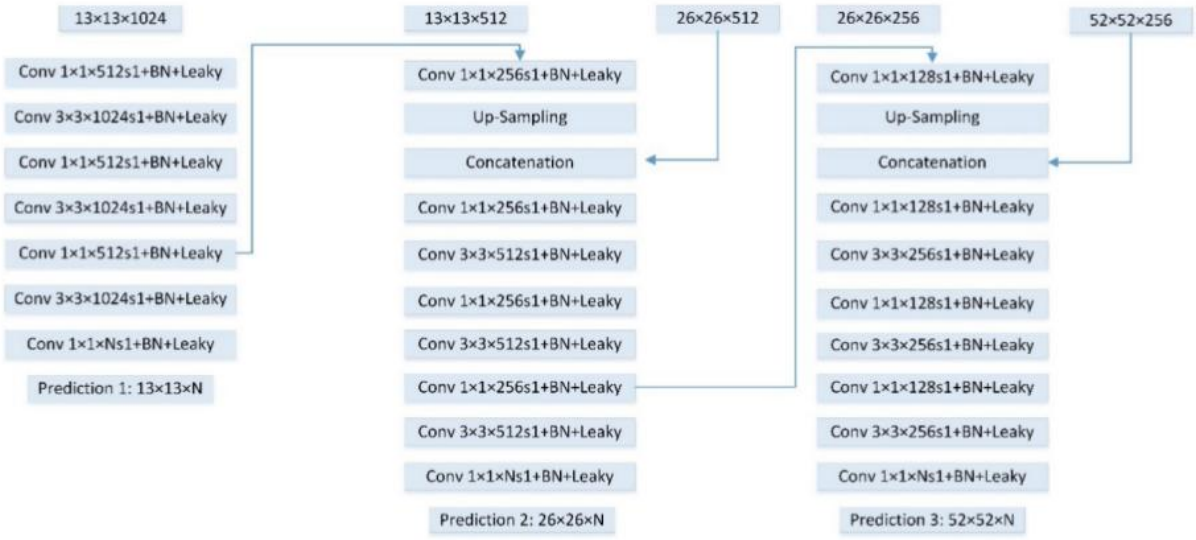


Figure 8.6: Multi-scale feature fusion module. The Feature maps from different layers are merged through up-sampling and concatenation. Each convolutional layer is batch-normalized, uses leaky ReLU activations, and have a stride of 1.

maps with different size : 13×13, 26×26, 52×52. B anchors are generated and the corresponding B 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:

$$b_x = \sigma(t_x) + c_x \tag{8.5}$$

$$b_y = \sigma(t_y) + c_y \tag{8.6}$$

$$b_w = p_w e^{t_w} \tag{8.7}$$

$$b_h = p_h e^{t_h} \tag{8.8}$$

$$p_o = \sigma(t_o) \tag{8.9}$$

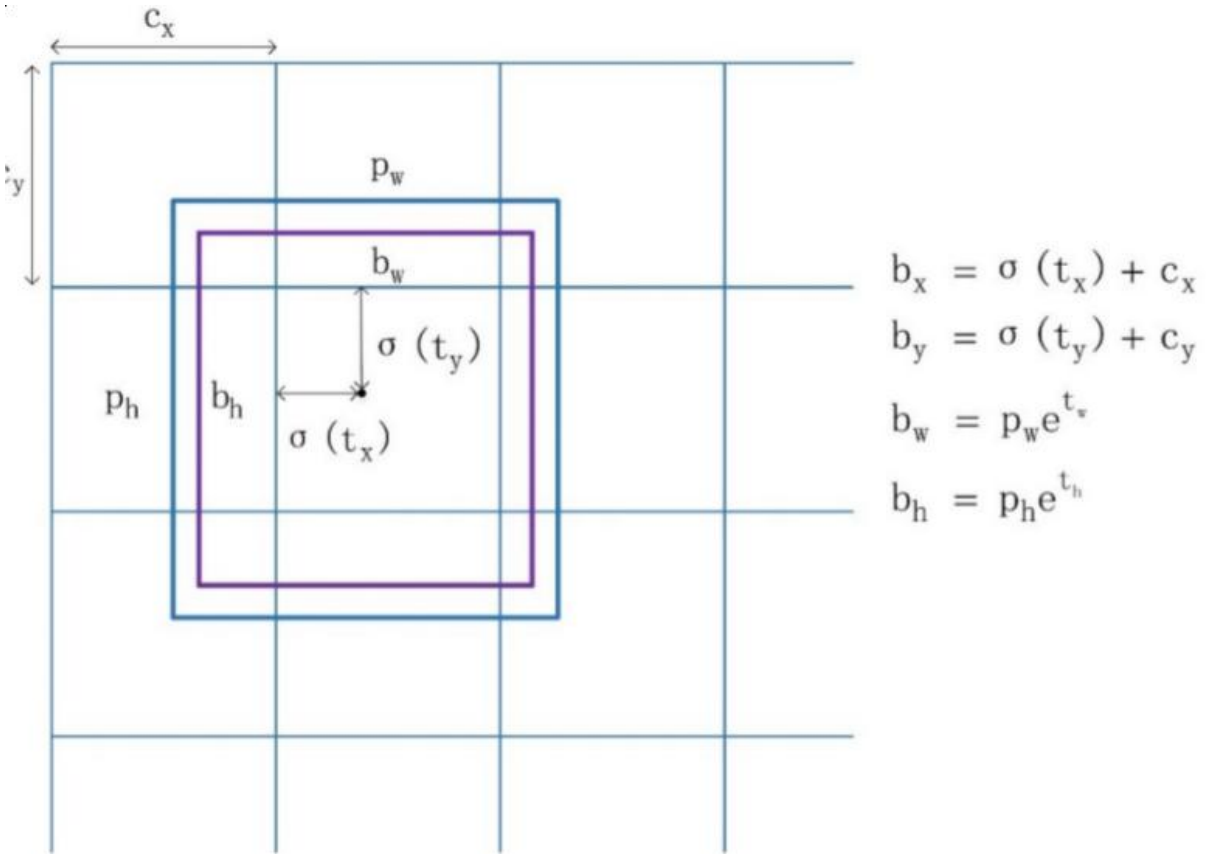


Figure 8.7: The model generates 4 coordinates: t_x, t_y, t_w and t_h. The center location of the final bounding box (b_x, b_y) is relative to the grid cell offsets (c_x, c_y) and the sigmoid activation function value of location coordinates (t_x, t_y). (c_x, c_y) denotes the offsets from the top left corner of the original image to the current grid cell

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.

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 $N = (5 + C) \times B$

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 8.11 and 8.12). The class loss is computed using the categorical crossentropy loss and is given in equation 8.13.

$$L_{overall} = L_{loc} + L_{conf} + L_{cla} \tag{8.10}$$

$$L_{loc} = \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} [(x_i \hat{x_i})^2 + (y_i - \hat{y_i})^2 + (w_i - \hat{w_i})^2 + (h_i - \hat{h_i})^2] \tag{8.11}$$

$$L_{conf} = \lambda_{obj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} P^{obj}(c_i - \hat{c_i})^2 + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} (1 - P^{obj})(c_i - \hat{c_i})^2 \tag{8.12}$$

$$L_{cla} = -\lambda_{cla} \sum_{i=0}^{S^2} P^{obj} log(\hat{p_i}) \tag{8.13}$$

Here, $\lambda_{coord}, \lambda_{noobj}$ 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: $\lambda_{coord} = 1, \lambda_{obj} = 5, \lambda_{noobj} = 1, \lambda_{cla} = 1$

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 8.14

$$s_i = \begin{cases} s_i & iou(b_i, b_M) < T; i \neq M \\ s_i * f(iou(b_i, b_M)) & iou(b_i, b_M) \geq T; i \neq M \end{cases} \tag{8.14}$$

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:

$$f(iou(b_i, b_M) = exp[\frac{-(iou(b_i, b_M))^2}{\rho}] \tag{8.15}$$

8.4.2 Results

The authors tested their method on the RSD-GOD dataset and compared it to other detection framework: Faster R-CNN[31], SSD[32], YOLOv2[20]. 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 table**??**.

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[28] dataset, along with a collection of part detectors (COPD)[33], rotation-invariant CNN (RICNN)[34] and a R-P-Faster R-CNN[35] 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.

Using lessons taken from those paper, we can hope to create a network that is fit for detection in LiDAR surveys. However, we should emphasize that there are notable differences between satellite imagery and LiDAR surveys. LiDAR surveys suffer more from the problem of occlusion, where objects are partially obscured, which renders detection and classification harder. As stated in Section 7, geomagnetic surveys in general are harder to analyze, and object within can be harder to detect because they don't emit the same kind of response as with visible light.

The main lessons we need to take from the satellite imagery detection methods are their way of addressing the issue of scale. Surveys are often very high resolution, cover large swaths of terrain, but the object we want to detect are often very small both in absolute size and relative size to the image[1], and those issues arise both in satellite imagery and Geophysics. Most of the improvements here can be categorised into 3 main categories: an increase in raw compute performance, an increase in bounding box accuracy and improvements in earlier layer feature reuse.

[1]A barrow, a type of object in our dataset is around 50×50 pixels and the raw input image is 13140×10290. A barrow is therefore about 2×10^{-4} of the original image.

Chapter 9

State of the Art in Detection in Archaeological Remote Sensing

9.1 Automatic Detection in Magnetostatic Surveys using Convolutional Neural Networks

I started work in the field of automated detection in geophysical surveys in 2017, where I first developped, with the help of my colleague Fabien GOGLIO, a simple CNN that could categorise low resolution images of magnetostatic surveys.

Those surveys were centered mainly on the archaelogical dig of Marsal, France. The project proved to be a success, with high precision being achieved rapidly. However, this was just a proof of concept; showcasing that this kind of techniques could work for this type of data which is very challenging for traditionnal computer vision.

9.2 Learning To Look At LiDAR

Present wouter works :)

Chapter 10

The YOLO Framework

The first version of the YOLO detection framework was published in 2015[**yolo**]. It was, and still is, an hallmark in object detection; combining an mAP comparable with other detection method at the time, but with a very impressive inference time. YOLO was one of the first detection method being able to infer in real time. YOLO is considered a "one stage" detector, meaning that the object localisation and classification is a part of the same method. It passes the image only one time through its CNN, and directly outputs predictions; e.g it looks only once at the input. This dramatically increase the speed of the model. Since then, improvements[20][19] have been made to the model but the basic mechanism behind YOLO stays the same. For a review of the latest version, YOLOv4[36], please refer to Section 10.3.7.

In this section we will give a short explanation of the workings behind this framework.

10.1 General Principle

The main idea behind YOLO is to take the task of object classification and localisation as a regression task. The bounding boxes and class prediction are outputted directly as a tensor, without using a RPN. This idea makes the network perform extremely fast.

YOLO can take as input image of various size, but it will first downscale them to a fixed resolution, which is a hyperparameter. The input image is then divided into an $S \times S$ grid. Each grid cell predicts B bounding boxes and confidence scores for these boxes. Each bounding box is comprised of 5 predictions : x, y, width, height and confidence. The (x, y) are the position of the center of the bounding box relative to the grid cell. The confidence prediction represents the IoU between the predicted bounding box and the ground truth. Each grid cell also predict C conditional class probabilities: $P(\text{Class}_i|\text{Object})$, which are conditioned on the grid cell containing an object.

S, B and C are also hyperparameter, with C being the number of classes.

10.2 Network design and Architecture

The backbone behind YOLO is called Darknet53 containing 53 layers. YOLOv3[19] introduced the idea of prediction at different scale. In the YOLOv3 network, 3 prediction

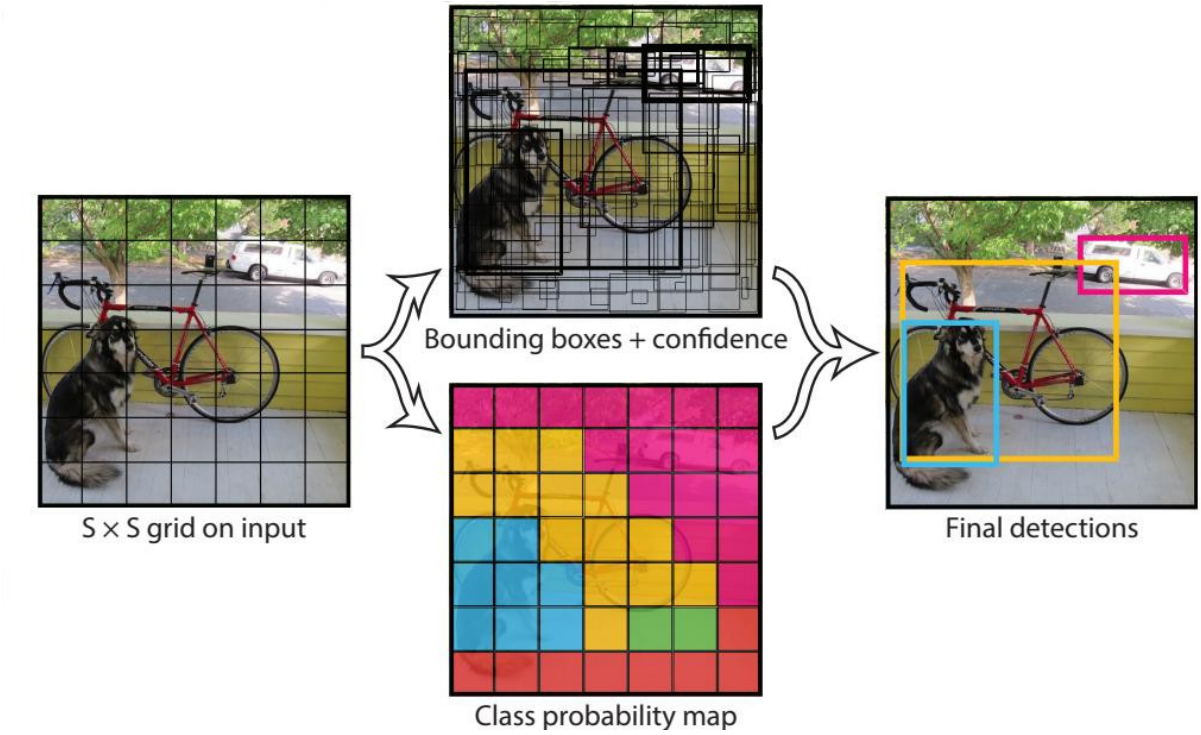


Figure 10.1: YOLO Model. First the image is divided into a $S \times S$, and for each grid cell predicts B bounding boxes, confidence, and C class scores, which are encoded into an $S \times S \times (B * 5 + C)$ tensor

are made at different scales, a technique known as multi-scale prediction which is improved in several papers of detection in satellite imagery, Multi-scale prediction allows the network to find small objects that are often missed otherwise. Figure 10.2 shows the architecture of the model.

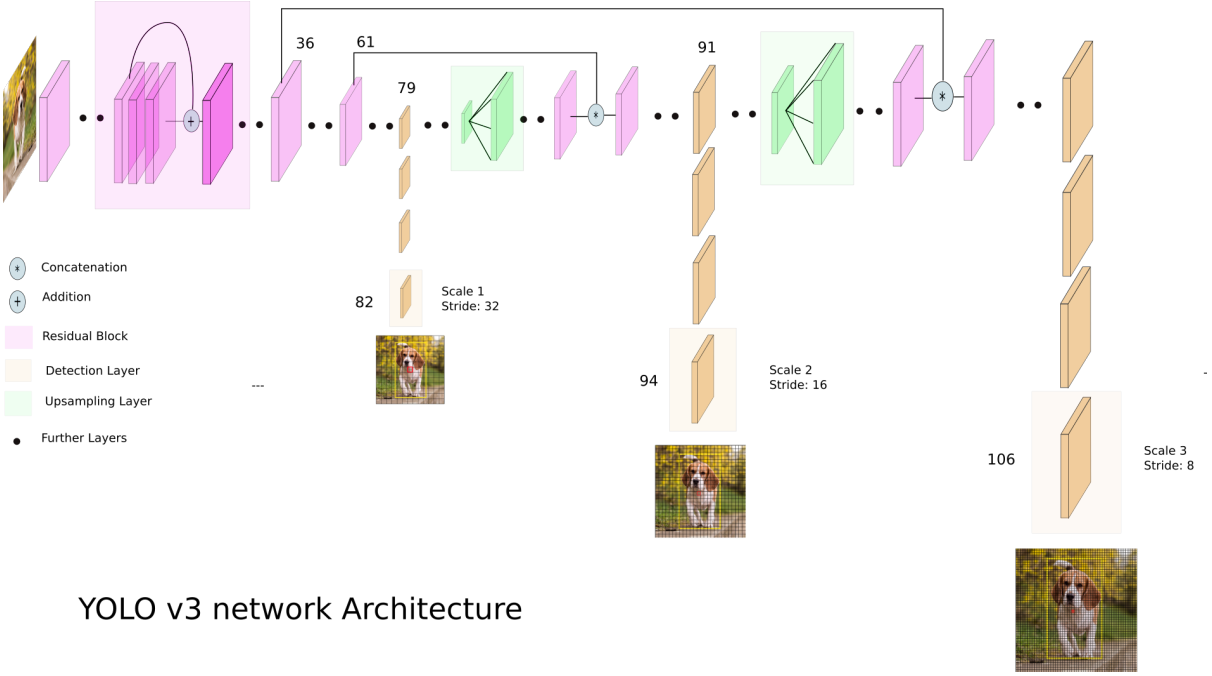


Figure 10.2: Architecture of the backbone of the YOLO Framework

10.3 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 Section 10.3.2. 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 10.3.3.

10.3.1 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[37] 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[4], resNet[5], resNeXt[38] or DenseNet[39]. We have already seen a few of those backbones in previously reviewed articles, especially resNet.

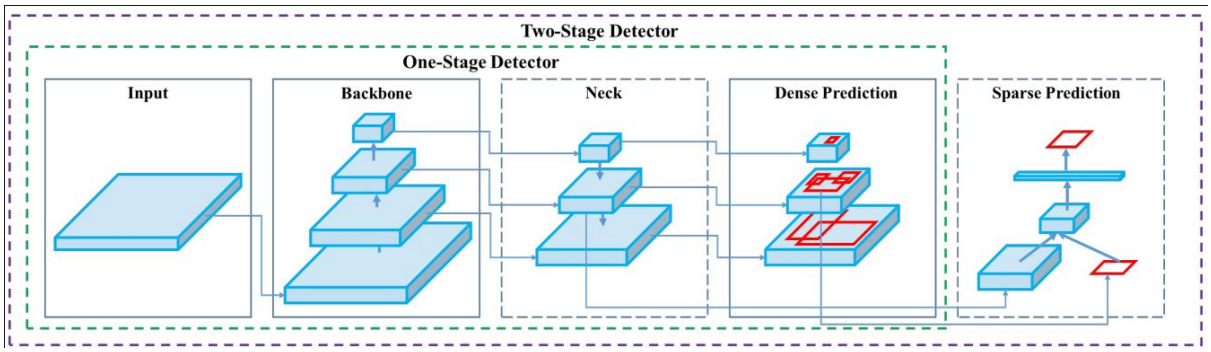


Figure 10.3: General Architecture of an Object Detector

The head part is divided in two categories: one-stage object detectors such as YOLO[20, 19], SSD[32] or RetinaNet[40]. Examples of two stages detectors are the R-CNNs[41], like Fast R-CNN [42], Faster R-CNN[31] or R-FCN[43].

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)[44], Path Aggregation Network (PAN)[45], BiPFN[46] and NAS-FPN[47]

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

- **Backbones**: VGG[4], ResNet-50[5], EfficientNet[48], CSPResNeXt50[38], CSP-Darknet53[49].

- **Neck**:
  - **Additional Blocks**: SPP[50], ASPP[51], RFB[52], SAM[53]
  - **Path-aggregation blocks**: FPN[44],NAS-FPN[47] PAN[45], BiFPN[46], ASFF[54], SFAM[55]
- **Heads**:
  - **Dense Prediction (One-Stage)**
    - Anchor-based: RPN[31], SSD[32], YOLO[19], RetinaNet[40]
    - Anchor-free: CornerNet[56], CenterNet[57], MatrixNet[58], FCOS[59]
  - **Sparse Prediction (Two-stage)**
    - Anchor-based: Faster R-CNN[31], R-FCN[43], Mask R-CNN[60]
    - Anchor Free : RepPoints[61]

10.3.2 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.

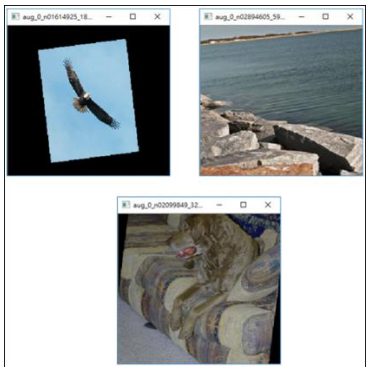
Those techniques are pixel-wise adjustments, meaning that the original pixel information in the adjusted area is retained. Novel data augmentation techniques have an emphasis put in object occlusion issues. Random Erase[62] or CutOut[63] randomly selects a rectangle region in an image and fills with random values. Hide and seek[64] and grid mask[65] randomly selects multiple rectangle regions in an image and replaces them to all zeros. Similar techniques are applied to feature maps, for example, DropOut[66], DropConnect[67] and DropBlock[68].

Other methods uses multiple image together. MixUP[69] uses two images to multiply and superimpose them with different coefficient ratios, and adjusts the labels with these ratios. CutMix[70] crops an image and cover another images with this cropped region, adjusting the labels according to the size of the covered area.

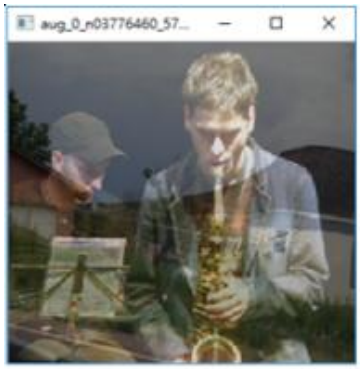
Even StyleTransfer GANs[71] have been used for data augmentation, reducing the texture bias learned by CNN.

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[72] or online hard example mining[73] in two-stage object detectors. However these example mining methods are not applicable in one-stage object detectors. For those detectors, focal loss[74] 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[75] convert hard label into soft label, which can make



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



(b) MixUp

(c) CutMix

Figure 10.4: Some data augmentation techniques

the model more robust. Knowledge distillation have also been used to design a label refinement network.[76].

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[77] puts the coverage of the predicted bounding box area and the ground truth area into consideration. GIoU loss[15] is an improvement on this loss, and also include the shape and orientation of the object in addition to the coverage area[1]. Finally, DIoU loss[16] also considers the distance of the center of an object and CIoU[16] considers the overlapping area, the distance between center points and the aspect ratio.

10.3.3 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[50], ASPP[51] and RFB[52]. The SPP originates from Spatial Pyramid Matching (SPM) [78]. SPM method is to split the feature maps into several $d \times d$ blocks of equal size, where d can be $1, 2, 3, ...$ 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[19], the SPP module is improved by concatenating the max-pooling outputs with kernel size $k \times k$ with $k = \{1, 5, 9, 13\}$ and

[1] An improvement to this has been done for automated detection in satellite imagery by Qian and Al and is covered in section 8.3

stride of 1. This modules improves the YOLOv3 AP_{50} by 2.7% on the MS COCO[17] 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)[79] and point-wise attention with Spatial Attention Module (SAM) [53]. SE modules can improve the accuracy of ResNet by 1% in the ImageNet[37] dataset, however this comes at in increase of 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[80] or hyper-column[81] integrate low-level physical and geometric features to high-level semantic feature. Now, lightweights modules that integrates different feature pyramids are used, like SFAM[55], ASFF[54] and BiFPN[46]. 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[82] was proposed as a solution to vanishing gradient problem. Offshoots, such as LeakyRelu[22], PReLU[83], ReLU6[84], Scale Exponential Linear Unit (SELU) [85], Swish [86], hard-Swish[87] and Mish[88] 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.*[89] added classification confidence scores in R-CNN as a reference. Soft NMS[90] considers the problem that occlusion of an object may cause the degradation of confidence score in greedy NMS with IoU score. DIoU NMS[16] adds the information of the center point distance to bounding box.

10.3.4 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 10.5: 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 10.5. 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.

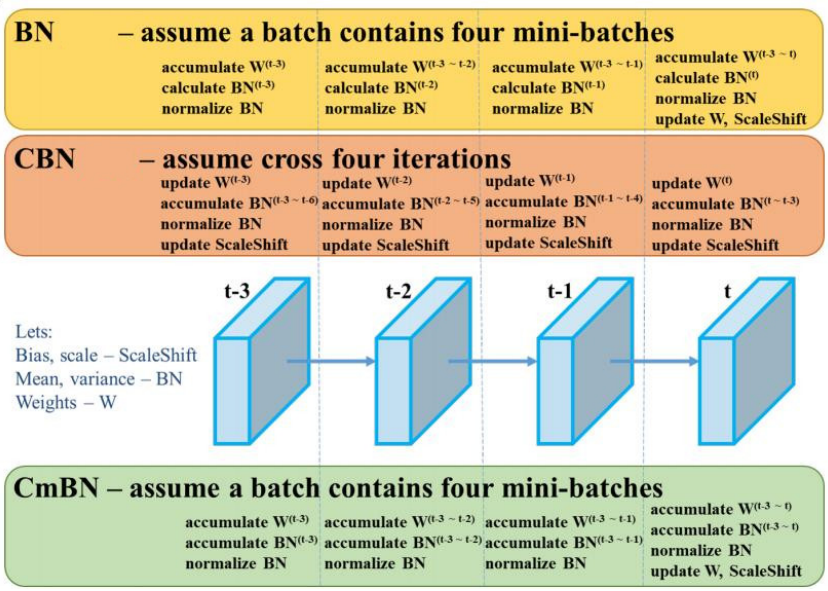
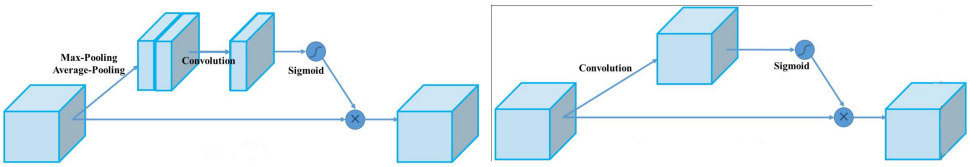


Figure 10.6: Cross Mini Batch Normalization

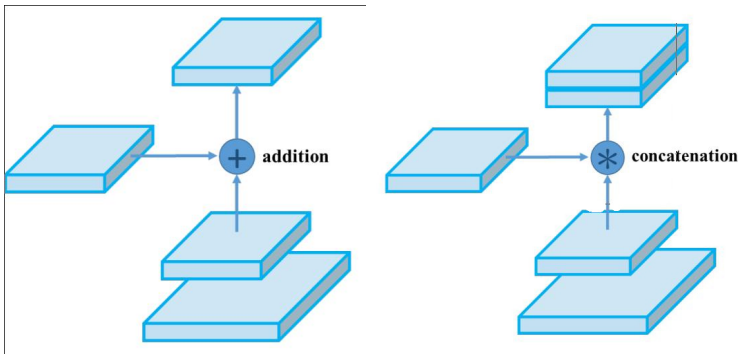
Cross mini Batch Normalisation is used. It is a modified Cross Batch Normalization (CmBN). CmBN collects statistics only between mini-batches within a single batch.

SAM is modified from spatial-wise attention to point-wise attention (figure 12.9), and the shortcut connection in PAN are replaced to concatenation, as can be seen in figure 12.7



(a) SAM (b) Modified SAM

Figure 10.7: SAM and its YOLO modification



(a) PAN (b) Modified PAN

Figure 10.8: PAN and its YOLO modification

10.3.5 YOLOv4 Architecture

YOLOv4 basic architecture is as follows:

- **Backbone**CSPDarknet53[49]
- **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 [91], 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.

10.3.6 Results

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 10.1 shows the results of those tests for the CSPResNext-50 backbone, and Table 10.2 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 10.1: 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 10.2: 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

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.

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 10.3: 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.

- **M**: Mosaic data augmentation - using the 4-image mosaic during training instead of a single image
- **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**: Optimized 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 10.3

10.3.7 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 10.4. 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 10.4: Comparison study of different classifier pre trained weights for detector trainings.

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 10.5

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 10.5: 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.

Part III

Implementation and Results

Chapter 11

Dataset

11.1 Dataset creation

The original dataset from Wouter[**wouter2019**] was constructed using LiDAR surveys from a small region of the Netherlands. The data I was given was constructed in the form of a GIS project, with the position of the different objects being encoded in .shp files.

A dataset for the training of a Mask-RCNN[60] model was also present. It was made from more of a dozen of very high resolution and details tif files, cropped to 600×600 pixels, and annotated using the COCO[17] format.

I wanted to create a new dataset that was of a higher resolution, but if each object was annotated and given a position in a given reference system, it was not indicated in which image the object was located. To create a new dataset, I first needed to associate each object with a LiDAR survey image.

11.1.1 Cropping Images

In order to do this, we first converted the .shp files, which are GIS specific to CSV using the `ogr2ogr` script. In our case the .shp files contained the position of the bounding boxes of the different objects in the same reference as the image. Each class of objects had its own .shp file. To find out in which image each object where, we needed to also find the coordinates of the images. Luckily, those were encoded in the .tif files themselves. Using `glalinfo`, we are able to retrieve the position of the image. Finally, we check if all of the vertex of the bounding box of an object are inside of the position of an image to associate it. Using this newly acquired information, we build a new database containing the position of the bounding box, the type of object and in which image this object is in.

We are now in possession of all the necessary information to create a new dataset. This dataset will be created by cropping 1000×1000 images from the original LiDAR images, which will be centered around an object. This is done for every object in the dataset (around 2K). Luckily, those objects are often cluttered together, meaning that we will often see multiple examples of one class in one image. To create a cropped image, we first need to transform the coordinates of the object, which are from a world reference into a image reference, i.e. between 0 and the length/width of the image. This is done using

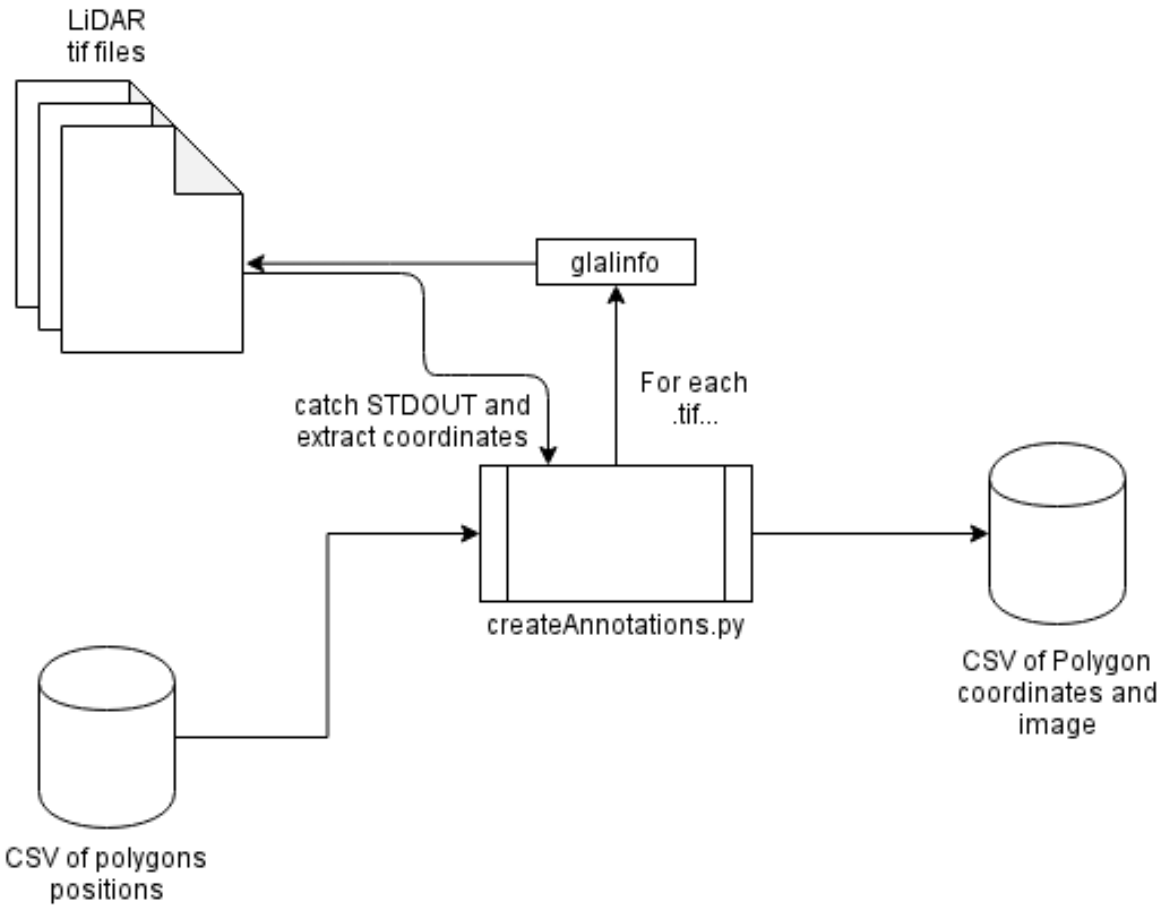


Figure 11.1: Functionnal Diagram of the createAnnotation script. For each objects in the CSV, it cycles for each of the .tif files, extract information concerning its position using the `glalinfo` program, then if all of the vertex of the bounding box are inside of the image, it adds it to the row of the object, and append it to a new CSV

simple scaling and translating formula, given in Equation 11.1. This gives us values that are equivalent to pixels in the image rather than a position in the GIS world.

$$f[x] = \frac{x - x_{min}}{x_{max} - x_{min}} * (y_{max} - y_{max}) + y_{min}, \tag{11.1}$$

Where x_{min}, y_{min} are the minimum value of the original range and the destination range respectively, and where x_{min}, y_{min} are the maximum value of the original range and the destination range respectively

We center the 1000×1000 image around those newly transformed values, adding a random jitter value, taken uniformly between $(-100; 100)$. This guarantees us that the image won't be perfectly centered around an object, creating a more difficult image to analyze. Finally, we need to find which objects are in the cropped image. To do so, we iterate through every object in the database, and if they are in the same image, and if all of their vertex are in the cropped section, we add it to the annotation of the image.

11.1.2 Annotations

Using the scripts, we create a dataset using the already annotated GIS project. This dataset contains about 7500 examples, split into a 80/20 train/test dataset. The anno-

tation is done using the COCO format[17], where each image is accompanied with a .txt file containing the position of each bounding box, along with its class. The bounding box is position is encoded as $(x, y, width, height)$, where x, y is the position of the top left corner of the bounding box. Those coordinates are also relative to the image, meaning that $(x, y) \in [0, 1] \times [0, 1]$. To obtain those bounding box, we again took their position from the .shp file and converted their position from a world reference to a image reference using the same formula as above. We then normalised to obtain values between 0 and 1.

Chapter 12

Architecture

The decision was made to try and adapt the YOLOv4[36] architecture, based on a CSV-Darknet53 backbone, to archaeological object detection by applying modifications inspired from the State of the Art object detection methods in satellite imagery detection. Those modifications mostly concerned feature fusion, and receptive field improvements by way of dilated convolution.

12.1 Feature Fusion

Feature Fusion is a technique used in most CNNs. The main issue with Deep Networks is that, as the input is convoluted as goes further into the network, the feature maps contains more and more semantic and less geometric information. In other words, deeper feature maps incorporates information as to *what is it* and less about *where is it/what does it looks like*. Moreover, since the feature maps gradually become smaller and smaller, by way of max-pooling, stride of convolution or otherwise, this means that the receptive field or *what the layer is actually sensing* becomes coarser and coarser. Small objects can be missed, since they disappear during the downscaling.

Feature Fusion aims at mitigating this loss by adding information coming from earlier feature maps. Traditional networks usually uses "Skip Connections" to achieve this. ResNet[5] simply uses vector addition to add the weight matrices in its Residual Modules. DenseNet[39] concatenates the weight matrices from earlier layer to later ones along the channel dimensions.

12.2 Backbone and activation

The backbone is CSPDarknet53[49], most notably used in YOLOv4[36]. This backbone obtained better result than the CSPResNeXt-50[38] in the YOLO paper, which is why it is used. Inspiration could also be taken from Zhuang *et al.*[30], with passthrough and concatenation from earlier layers.

Layer	Filters size	Repeat	Output size
Image			416 × 416
Conv	32 3 × 3/1	1	416 × 416
Conv	64 3 × 3/2	1	208 × 208
Conv Conv Residual	32 1 × 1/1 64 3 × 3/1	Conv Conv Residual × 1	208 × 208 208 × 208 208 × 208
Conv	128 3 × 3/2	1	104 × 104
Conv Conv Residual	64 1 × 1/1 128 3 × 3/1	Conv Conv Residual × 2	104 × 104 104 × 104 104 × 104
Conv	256 3 × 3/2	1	52 × 52
Conv Conv Residual	128 1 × 1/1 256 3 × 3/1	Conv Conv Residual × 8	52 × 52 52 × 52 52 × 52
Conv	512 3 × 3/2	1	26 × 26
Conv Conv Residual	256 1 × 1/1 512 3 × 3/1	Conv Conv Residual × 8	26 × 26 26 × 26 26 × 26
Conv	1024 3 × 3/2	1	13 × 13
Conv Conv Residual	512 1 × 1/1 1024 3 × 3/1	Conv Conv Residual × 4	13 × 13 13 × 13 13 × 13

Figure 12.1: Architecture of the Darknet 53 backbone

12.3 Activation Functions

The performance of different activation functions will be tested. Mish[88](Figure 12.4) and Swish (Figure 12.3) or even Scaled Exponential Linear Unit (SELU) (Figure 12.2) are good candidates.

12.3.1 Scaled Exponential Linear Unit

?? is one of the first activation developed to address the issue of normalisation in FNN: as those networks become deeper and deeper, they become more and more sensitive to gradient issues. Usually, Batch Normalisation is used to mitigate this, but this makes FNN sensitive to normalisation during training. **SELUs has been developed to incorporate normalisation inside the activation function.**

SELU is defined by the following equation:

$$\sigma[x] = \lambda \begin{cases} x \\ \alpha * exp[x] - \lambda \end{cases} \tag{12.1}$$

With α and λ being fixed parameters who are derived from the inputs. In the Figure 12.2,

the values used are for standard scaled inputs with a 0 mean and a 1 standard deviation: $\alpha = 1.6732$ and $\lambda = 1.0507$.

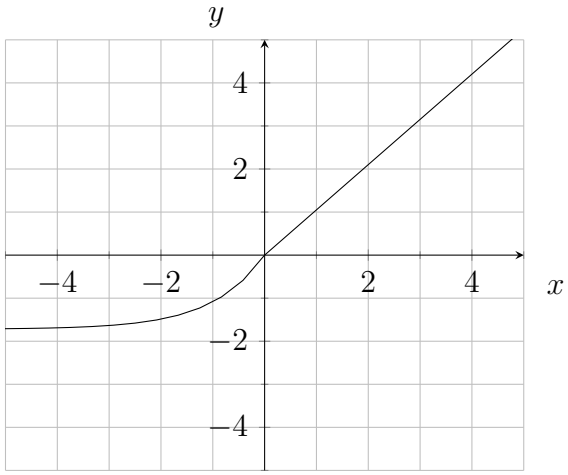


Figure 12.2: Scaled Exponential Linear Unit (SELU) Activation Function

12.3.2 Swish

Swish[86] is a part of a recent push in research toward a better activation function to replace ReLU. While ReLU offers very good performance, it is not smooth due to the non differentiable point at $x = 0$, and is always zero for all $x < 0$ which can imper learning. Swish has been developped by Google Brain to address those issues, and can be defined as follow:

$$\sigma[x, \beta] = x \cdot (1 + exp[-\beta x])^{-1} \tag{12.2}$$

Where $(1 + exp[-\beta x])^{-1}$ is the sigmoid function and where β is either a constant or a training hyperparameter. If $\beta = 1$, then Swish is equivalent to the weighed Sigmoid-Weighted Linear Unit (SiL). If $\beta = 0$, then Swish becomes the scaled linear function $f[x] = 0.5 \cdot x$. As $\beta \leftarrow \infty$, Swish approaches a $0 - 1$ function, like the ReLU. In other words, Swish can be interpreted as **a non linear interpolation between the linear function and the ReLU function, which degree of interpolation can be controlled by the model if β is set as a trainable parameter**

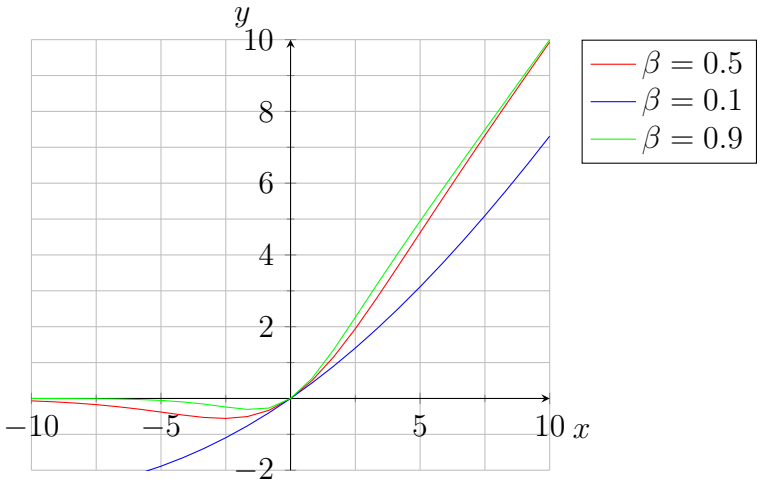


Figure 12.3: The Swish activation function, with different values of β

12.3.3 Mish

Mish is a novel activation function which has been proposed by Misra[88] to act as a replacement for the immensely popular ReLU function. Mish is built as a successor for ReLU and is inspired by the Swish function, described above. Mish is very simple, and defined as below:

$$\sigma[x] = x \cdot \tanh[1 + exp[x]] \tag{12.3}$$

Similar to Swish, Mish is unbounded above, preventing saturation, but bounded below, creating a self-regularisation. Mish has an order of continuity C^{∞}, a preferable property over ReLU having a C^0, which can cause problem in gradient-based optimisation. Finally, Mish is non monotonic, which causes small negative inputs to be preserved as negative outputs. Mish obtains improve performance by up to 1.6% on dataset such as the MNIST[92] or CIFAR-10[93]

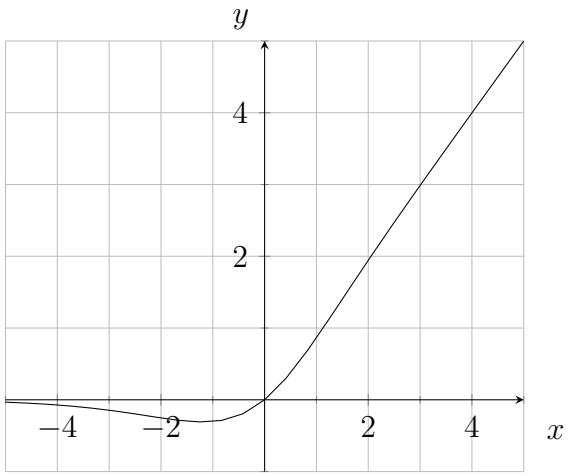


Figure 12.4: The mish activation function

12.4 Dual Scale Detectors

An idea introduced in YOLT[18] is to use two different detectors running simultaneously: one trained on small scale images to detect small objects, the other trained on larger images to detect bigger objects. The small scale network would be fed small chips of images, while the large scale network would be fed downscaled large swathes of terrain. This meant that the large scale network would run much less often than the small scale network, and mitigate the loss of performance of running two network at the same time.

An ensemble method similar to this could be used, where two networks try to find objects at different scales.

12.5 Multi-scale Feature Fusion

To be able to detect the small objects often seen in LiDAR data, some kind of MLFF module needs to be used. There are a few existing modules and techniques that can boost

the detection rates of a model by aggregating low level feature maps from earlier layers with ones from higher layers.

The MLFF module from Zhuang *et al.*[30] takes 3 feature maps from different layers, upscales and concatenates them, then applies a series of convolution operations to obtain 3 predictions at different scale. This is very similar to the MLFF used in YOLOv3[19], where 3 predictions are made at different scale level and which reuses feature maps from deeper layers to build the predictions.

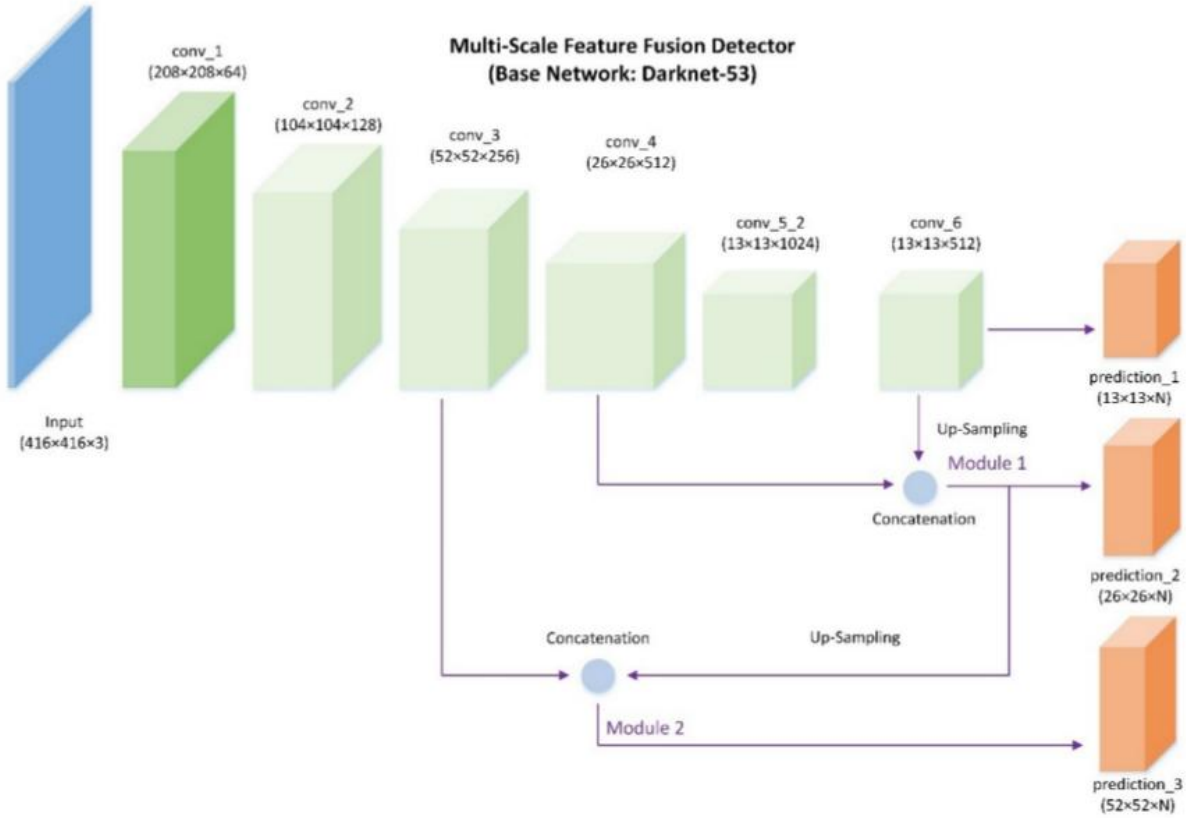


Figure 12.5: Feature Fusion implementation of Zhuang *et al.*. We are only interested in the bottom part of the figure, with feature maps from different levels being concatenated.

The MLFF module from Qian *et al.*[27] shown in Figure 12.6 takes each proposal generated by a RPN, and maps its position to all level of feature map generated by the FPN. From there it obtains N regions of the feature maps (N being the number of levels), which it transforms into 7×7 feature maps through the RoiAlign[60] operation. Finally, it concatenates these 4 regions along the channel dimensions, and applies two convolution operations along with a Fully Connected Layer for bounding box regression and classification.

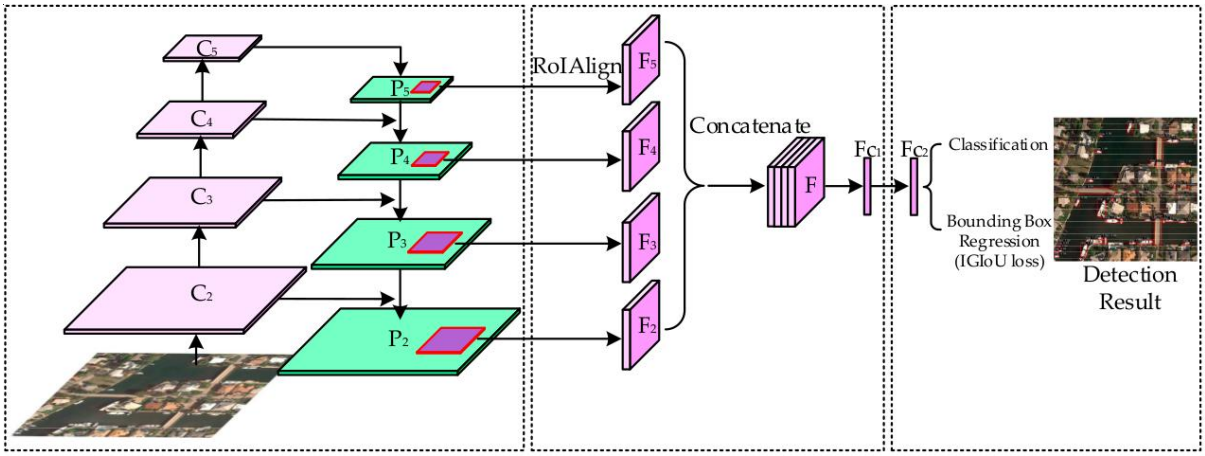
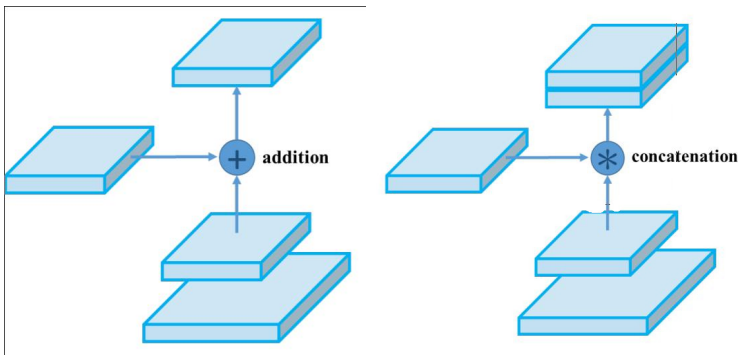


Figure 12.6: Architecture of the model from Qian et al. We are interested in the module shown in (b), where particular regions of different feature maps are concatenated together

The PAN is a sort of multi layer feature fusion module. Following the same principle as the MLFF from Qian *et al.*, PAN takes the features maps generated by a FPN. It first downscales the lower level layer (denoted P_i with a 3×3 convolution operation with a stride of 2. It then adds element wise this downscaled layer with the layer P_{i+1}. This is done iteratively until the up most layer is attained. **In the YOLOv4 paper, instead of adding the layers, a concatenation is done**.



(a) PAN (b) Modified PAN

Figure 12.7: PAN and its YOLO modification

12.6 Receptive Field Improvements

To improve the size of the receptive field, the modified SPP module from YOLOv3[19] would be used. Experimentations could be done using dilated convolutions, as seen in Yu *et al.*[26] and Ju *et al.*. Dilated convolution improve the receptive field by increasing the kernel size without increasing the number of parameters in the kernel. This can help reducing the number of parameters, improving performance of the network in terms of ??. We can replace some of the classical convolution blocks from the CSPDarknet53 backbone with dilated convolution modules, as can be seen in Figure 12.8.

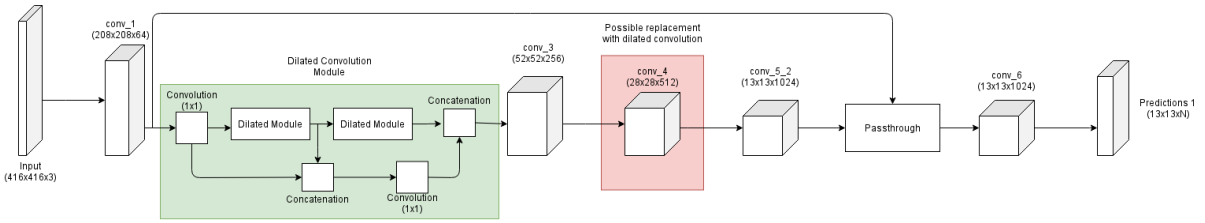
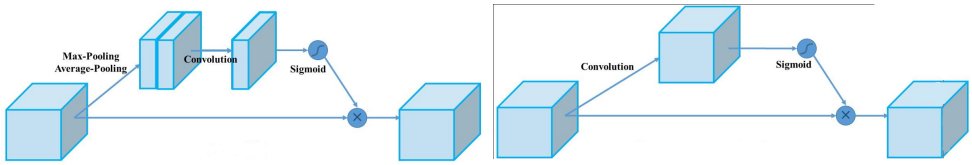


Figure 12.8: General Architecture of the Darknet Backbone with a dilated convolution module replacing a convolution block

12.7 Attention Modules

Attention modules are used to improve the accuracy of detections network by increasing the importance of particular regions or channels. The SAM[53] would be used, as it increases the accuracy without significantly impacting inference speed. Testing would have to be done to evaluate whether the SAM improvement done by Bochkovsky *et al.* in the YOLOv4 paper[36] gives out a performance increase.



(a) SAM (b) Modified SAM

Figure 12.9: SAM and its YOLO modification

12.8 Post-processing

NMS needs to be used to remove the superfluous bounding boxes generated by the network. Soft Non maximum suppression, as introduced by Qian *et al.* [27] would be used to achieve such a task, while trying to not miss the smaller objects.

12.9 List of all the possible testing arrangements

We will need to test a multiple of different modifications on the base CSPDarknet-53 and test out if those modifications results in performances improvements.

- Double YOLOv4 trained on large and small scale (similar to YOLT[18])
- CSPDarknet with one or two dilated convolution modules
- MLFF module from Zhuang *et al.*: see Figure 12.5
- MLFF module from Qian *et al.*: see Figure 12.6
- Modified PAN module: see Figure 12.7
- Modified SAM Module: see Figure 12.9

- Different activations: ReLU, Leaky ReLU, SELU, Mish or Swish
- Different activations: Different IoUs: GIoU, cIoU, dIoU
- Various input resolutions

Now State of the Art methods uses more dedicated techniques to fuses information from different depths. Usually multiple predictions are done at different depths levels, and incorporates information from all layers. YOLOv3[19] also uses multi-scale predictions to improve performance. **Leveraging the power of those techniques is key in enhancing the performance on deep learning models in high resolution - small scale object detection.**

12.10 YOLOv4

The latest version of the YOLO architecture is used as a base to modify. YOLO is a state of the art detector, obtaining extremely good results on a variety of datasets, and with the benefits of still being extremely fast: it is capable of running on real time will requiring only a modest amount of processing power. YOLOv4 is the latest iteration of this architecture, incorporating the latest and best modifications and improvement on the always growing class of object detectors. For a review on those improvements, please refer to section 10.3.7.

Chapter 13

Data augmentation techniques and training regimes

This section will present techniques that improve the accuracy of the detector but are not part of the architecture in itself. Those techniques applies during training but not during inference, so they do not increase the inference computing cost.

13.1 Data Augmentations

Deep Learning is known for requiring vast amount of data in order to train it properly and avoid ??[**shorten2019**]. However, most domains which requires Deep Learning do not possess the amount of data that is required. This is particularly true for Medical Image Analysis, but the problem is also posed here: see section 7.2.

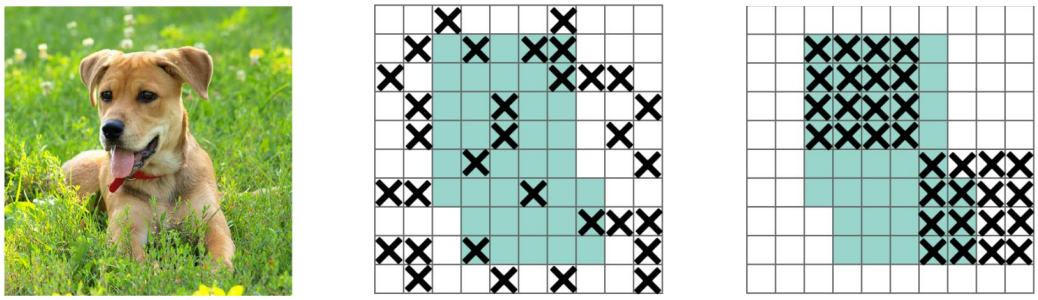
Data Augmentation is a solution to this issue and tries to enhance both the size and quality of the dataset so that a more robust model can be trained. This is done by modifying the image in various way, with color modification or geometric transformations. We will describe here the augmentations that are used in our model.

13.1.1 Image Modification

An issue seen in LiDAR is the fact that object are often occluded, fragmented and do not appear with the same clarity as examples seen in satellite imagery. This means that we need to construct and train a network that is robust to changes in context and occlusion. Fortunately, research has been focussed in constructing better data augmentation techniques that attempt to solve those issues. Those techniques "mixes" different images from the dataset, covering parts of one another to make it harder for the network to correctly infer. CutMix[70] is one of those techniques. Mosaic, introduced by Bochkovsky, Wang and Liao in the YOLOv4 paper[36] creates a new images out of 4, by creating a "mosaic" of sort, where the 4 images can take varying portions of the new image.

13.1.2 Regularization and Normalization

Regularization allows to reduce the complexity of a network and prevent overfitting. This is usually done by "dropping" random connections in a network, and training without those connections, a technique known as DropOut[66]. DropBlock[68] relies on a similar method, but is more suitable for convolutional networks. DropBlock works by first choosing random seed points in a mask, and dropping a continuous region around those points. This is more effective than removing purely random activation as close activations contain closely related information.



(a) Base image (b) Random Drop in activations (c) DropBlock

Figure 13.1: Regularisation by dropping: random drops *v.s.* DropBlock
The blue regions represents neurons that contains semantic information on the base image. (b) shows the effect of removing activations at random, which is not effective as neurons close to each other contains closely related semantic information. (c) shows the DropBlock method, which have a better chance of entirely removing important semantic information on the base image, such as the head or the feet of the dog, forcing the remaining neurons to learn useful features

Label smoothing [75] convert hard labels, like one hot labeling into soft labels. This works by introducing a noise distribution in the labeling of the data, and converting the original label in relation to this noise distribution. **However, this label smoothing requires a rewriting of the loss system.**

13.2 Self Adversarial Training

In the YOLOv4[36] paper, the author mention the use of **self adversarial training**. In this training regime, the network first tries to alter the original image instead of its weights. This is an adversarial attack on itself, modifying the original image to fool itself into a wrong inference. Then, the network is trained to detect an object on this modified image in the normal way.

13.3 Loss Systems

There are multiple ways of measuring the quality of the bounding box that is produced by the model, the most used is of course the IoU or Jaccard Index. However, the IoU is

more akin to a measure of *how good* a bounding box is : an excellent bounding box will have an IoU close to 1. However, the IoU does not indicate *how bad* a bounding box is: if there is no intersection between the bounding box and the ground truth, the IoU will of course be 0. But this means that bad bounding box that is close to the ground truth will have the same score as a *very bad* bounding box that is far away from the ground truth. Recent work have been done to address this issue, and is presented here.

13.3.1 GIoU, DIoU and CIoU

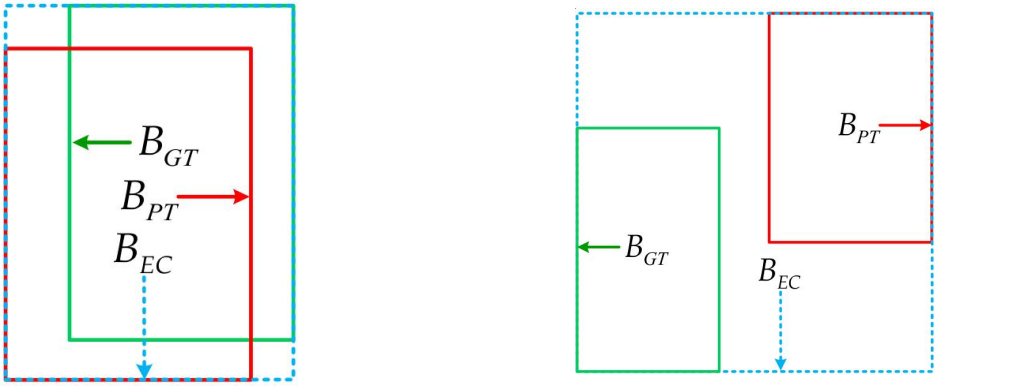
Generalized Intersection over Union Loss: an improvement on the traditional Intersection over Union performance metric would be used to more accurately measure the accuracy of the bounding boxes. The GIoU is defined as follows:

$$GIoU = 1 - IoU + \frac{area(B_{GT} \cup B_{PT})}{area(B_{EC})} \tag{13.1}$$

With IoU being defined in Equation 13.2

$$IoU = \frac{area(B_{GT} \cap B_{PT})}{area(B_{GT} \cup B_{PT})} \tag{13.2}$$

In those equations, B_{GT} represents the bounding box ground truth, B_{PT} represents the predicted bounding box and B_{EC} represents the smallest enclosing box of B_{GT} and B_{PT}.



(a) Two intersecting bounding boxes

(b) Two non-overlapping bounding boxes

Figure 13.2: 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.

The GIoU loss is a better measure of the quality of the predicted bounding box. Qian *et al.* also presents a new loss system that has a stronger gradient when the predicted bounding box is farther away to the truth. This loss system is shown in Equation 13.3

$$\mathcal{L}_{GIoU} = 2 \times log_2 - 2 \times log(1 + GIoU) \tag{13.3}$$

In Zheng *et al.*[16] the general IoU loss is defined as follows:

$$\mathcal{L} = 1 - IoU + \mathcal{R}(B_{GT}, B_{PT}) \tag{13.4}$$

Where $\mathcal{R}(B_{GT}, B_{PT})$ is a penalty term for the predicted box and ground truth. The authors then design a specialised penalty based on this formula.

The Distance IoU Loss is defined as follows:

$$\mathcal{L} = 1 - IoU + \frac{\rho^2(B_{GT}, B_{PT})}{c^2} \tag{13.5}$$

Here, b_{GT}, b_{PT} represents the center point of the ground truth bounding box and predicted box respectively, $\rho(\cdot)$ is the Euclidean distance and c is the diagonal length of the smallest enclosing box covering the two boxes *i.e.* B_{EC}.

The Complete IoU Loss incorporate the dIoU and add another term which forces aspect ratio consistency. The cIoU is defined as follows:

$$\mathcal{L} = 1 - IoU + \frac{\rho^2(B_{GT}, B_{PT})}{c^2} + \alpha v \tag{13.6}$$

Where α is a positive trade-off parameters. v measures the consistency of aspect ratios and is defined as follows.

$$v = \frac{4}{\pi}(\arctan[\frac{w_{GT}}{h_{GT}}] - \arctan[\frac{w_{PT}}{h_{PT}}])^2 \tag{13.7}$$

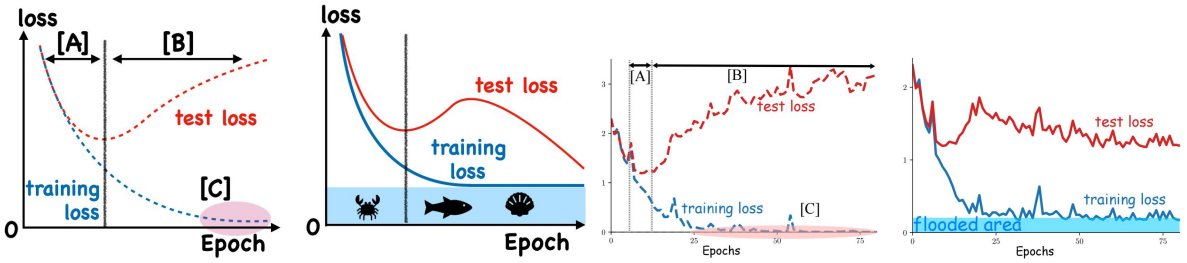
Finally the trade-off parameter α is defined as follows:

$$\alpha = \frac{v}{1 - IoU + v} \tag{13.8}$$

The cIoU loss and the dIoU both obtain AP improvements up to 3.14% on a FasterR-CNN[31] model trained on the COCO[17] dataset, up to 2.92% on a SSD[32] model also trained on COCO.

13.3.2 Loss flooding

In a recent paper by Ishida *et al.*[94], a new kind of regularization is presented, where **the model is prevented from reaching a zero training loss, even though it has zero training error**. This technique, called *flooding* intentionally prevents further reduction of he training loss when it reaches a certain low value: the *flooding level*. This approach makes the loss move around the flooding level by doing gradient descent if the loss is above the flooding level, but gradient ascent if it is below. This technique allows the network to obtain better generalization results and is very easily implementable.



(a) Without Flooding (b) With Flooding (c) CIFAR-10 without flooding (d) CIFAR-10 with flooding

Figure 13.3: Subfigure (a) shows the different regime during which a network first learns, then overfit. During [A], the network learns correctly, and both the training loss and the test loss decreases. During [B] the training loss continues to decreases, but the test loss increases; this is overfitting. In [C] the training loss is nearly zero. The authors show a way to avoid [C] by flooding the bottom area, forcing the loss to stay around a constant, which leads to a decreasing test loss. This is seen in (b) and (c) on the CIFAR-10 dataset

By letting the loss constant, the model hopefully drift towards a subspace where the loss landscape is flat, leading to better generalization. The authors tested this approach with various datasets, namely MNIST[92], Fashion-MNIST[95], CIFAR-10[93] and often obtained better results and in nearly every datasets, suggesting this approach is effective. Considering the ease of implementation, the lack of performance degradation and the possible improvements on performance, this technique could be very worthwhile.

13.4 Training methodology

First, a non modified YOLOv4 model, using the original architecture has been trained on the dataset, and will serve as a baseline, along with the MaskRCNN model trained by Wouter*et al.*. This will allow us to effectively quantify the improvements or deteriorations in performance between the model themselves.

We also computed the anchors for the dataset, in the hopes that this would bring us better results.

We trained each model on the same dataset, for 10000 epochs. For every model, the best version of the weights were selected, based on the mAP.

Chapter 14

Results

Chapter 15

Discussion

Part IV

Conclusions

Chapter 16

Further Improvements

Chapter 17

Conclusion

Annexes

Acronymes

AP Average Precision 61

BoF Bag of Freebies 44, 45

BoS Bag of Specials 45

cIoU Complete Intersection Over Union 15, 57, 61

DEM Digital Elevation Model 10

dIoU Distance Intersection Over Union 15, 57, 61

FNN Feedforward Neural Network 51

FPN Feature Pyramid Network 24–26, 54, 55

GIoU Generalized Intersection Over Union 15, 24–26, 57, 60

GPU Graphics Processing Unit 45

IoU Intersection Over Union 14, 24, 25, 34, 57, 59, 60

LiDAR LIght Detection And Ranging 10, 32, 47, 53, 58

mAP Mean Average Precision 14, 34

ML Machine Learning 13

MLFF Multi Layer Feature Fusion 24–26, 53–56

MSE Mean Squared Error 12

NMS Non-Maximum Suppression 31, 56

PAN Path Aggregation Network 55

RPN Region Proposal Network 34, 54

SAM Spatial Attention Module 56

SELU Scaled Exponential Linear Unit 51, 57

SPP Spatial Pyramidal Pooling 55

SVM Support Vector Machine 32

YOLO You Only Look Once 34, 50, 57

Glossary

CNN Convolutional Neural Network. Type of neural network, who learns weights for matrices who will be convoluted and produces feature maps. Generaly, those feature maps will be convoluted upon by other convolutional layers. 10, 11, 33, 34, 50

GIS Geographical Information System. Type of specialised software created to deal with maps and geographical data 10, 47, 48

hyperparameter An hyperparameter is a parameter whose value is used to control the model behavior. As opposed to weights, those values are not learned during the training process.[14, p.117] 34, 52

weights A weight is a value that is learned during the training process.[14, p.14] 12

Bibliography

[1] Yann Lecun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE*. 1998, pp. 2278–2324.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[3] Christian Szegedy et al. “Going Deeper with Convolutions”. In: *Computer Vision and Pattern Recognition (CVPR)*. 2015. URL: http://arxiv.org/abs/1409.4842.

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

[5] 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.

[6] Karol J Piczak. “Environmental sound classification with convolutional neural networks”. In: *2015 IEEE 25th International Workshop on Machine Learning for Signal Processing (MLSP)*. IEEE. 2015, pp. 1–6.

[7] S. Adavanne, P. Pertilä, and T. Virtanen. “Sound Event Detection Using Spatial Features and Convolutional Recurrent Neural Network”. In: *arXiv e-prints* (June 2017). arXiv: 1706.02291 [cs.SD].

[8] C.-C. Kao et al. “R-CRNN: Region-based Convolutional Recurrent Neural Network for Audio Event Detection”. In: *arXiv e-prints* (Aug. 2018). arXiv: 1808.06627 [cs.SD].

[9] Emre Cakir and Tuomas Virtanen. *Convolutional Recurrent Neural Networks for Rare Sound Event Detection*. Tech. rep. DCASE2017 Challenge, Sept. 2017.

[10] H. Phan et al. “DNN and CNN with Weighted and Multi-task Loss Functions for Audio Event Detection”. In: *arXiv e-prints* (Aug. 2017). arXiv: 1708.03211 [cs.SD].

[11] Geoffrey I. Webb. “Overfitting”. In: *Encyclopedia of Machine Learning*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, 2010, pp. 744–744. ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_623. URL: https://doi.org/10.1007/978-0-387-30164-8_623.

[12] D. P. Kingma and J. Ba. “Adam: A Method for Stochastic Optimization”. In: *arXiv e-prints* (Dec. 2014). arXiv: 1412.6980.

[13] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization". In: *Journal of Machine Learning Research* 12.Jul (2011), pp. 2121–2159.

[14] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[15] 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.

[16] 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].

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

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

[24] 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.

[25] 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.

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

[27] 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.

[28] 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.

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

[30] 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.

[31] 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].

[32] 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.

[33] 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).

[34] 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.

[35] 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.

[36] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. "YOLOv4: Optimal Speed and Accuracy of Object Detection". In: *arXiv e-prints*, arXiv:2004.10934 (Apr. 2020), arXiv:2004.10934. arXiv: 2004.10934 [cs.CV].

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

[38] 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.

[39] 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.

[40] 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.

[41] 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.

[42] 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.

[43] 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.

[44] 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.

[45] 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.

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

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

[48] 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.

[49] 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].

[50] 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.

[51] 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.

[52] 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.

[53] 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.

[54] 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].

[55] 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.

[56] 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.

[57] 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.

[58] 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].

[59] 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.

[60] 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.

[61] 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].

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

[63] 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.

[64] 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.

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

[66] 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.

[67] 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.

[68] 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].

[69] 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.

[70] 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.

[71] 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].

[72] 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.

[73] 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.

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

[75] 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.

[76] 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.

[77] 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.

[78] 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.

[79] 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.

[80] 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.

[81] 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.

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

[83] 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.

[84] 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.

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

[86] 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.

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

[88] 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].

[89] 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.

[90] 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.

[91] 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.

[92] Yann LeCun and Corinna Cortes. “MNIST handwritten digit database”. In: (2010). URL: http://yann.lecun.com/exdb/mnist/.

[93] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. “CIFAR-10 (Canadian Institute for Advanced Research)”. In: (). URL: http://www.cs.toronto.edu/~kriz/cifar.html.

[94] Takashi Ishida et al. “Do We Need Zero Training Loss After Achieving Zero Training Error?” In: *arXiv e-prints*, arXiv:2002.08709 (Feb. 2020), arXiv:2002.08709. arXiv: 2002.08709 [cs.LG].

[95] Han Xiao, Kashif Rasul, and Roland Vollgraf. “Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms”. In: *CoRR* abs/1708.07747 (2017). arXiv: 1708.07747. URL: http://arxiv.org/abs/1708.07747.