# BladeSynth

## Damage Detection and Assessment in Aircraft Engines with Synthetic Data

Chengming Feng

# BladeSynth

## Damage Detection and Assessment in Aircraft Engines with Synthetic Data

by

Chengming Feng

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Monday August 29, 2022 at 1:30 PM.

| | | |
|---|---|---|
| Student Number: | 5290333 | |
| Project Duration: | 11, 2021 - 08, 2022 | |
| Thesis Committee: | Dr. J.C. van Gemert | TU Delft, Supervisor & Chair |
| | Dr.Ir. J. Dauwels | TU Delft, Core Member |
| | Dr. N.Töman | TU Delft, Core Member |

An electronic version of this thesis is available at *https://repository.tudelft.nl/*

TU Delft

# Preface

*This report presents my Master thesis project of Damage Detection and Assessment in Aircraft Engines with Synthetic Data to obtain the degree of Master of Science at the Delft University of Technology. This project is conducted in the Computer Vision Lab of TU Delft and Aiir Innovations under the supervision of Dr. Jan van Gemert and Dr. Yancong Lin from CV Lab, and Steve Nowee from Aiir Innovations. This report is composed of two parts. The first part is a scientific paper, which introduces a novel approach to generating large-scale synthetic data of damages in aircraft engines and demonstrates damage detection in aircraft engines benefits from synthetic data. The second part is the supplemental materials which provide the background of this thesis.*

*I have been enthusiastic about computer vision and deep learning since my junior year. With great passion and effort, I conducted my Bachelor thesis in the field of deep learning based object detection and won the Excellent Bachelor Thesis at Northwestern Polytechnical University. To further my research interest, during the two-year Master at TU Delft, I accomplished the Deep Learning course and finished the extra project and Master thesis project under the supervision of Dr. Jan van Gemert. From September 2022, I am so glad that I will continue my study as a PhD Candidate in the Computer Vision Lab and the BioMorphic Intelligence Lab under the supervision of Dr. Nergis Tömen.*

*Firstly, I would like to sincerely appreciate my supervisor, Dr. Jan van Gemert, for his guidance and assistance in my Master thesis project. Secondly, I would like to express my gratitude to my co-supervisor, Dr. Yancong Lin, for the professional advice and for remarkably helping me during the project. I enjoy working with him a lot. Thirdly, I would like to appreciate my other supervisor, Steve Nowee, for the proficient suggestions and beneficial guidance. Then, I would like to appreciate Dr.Ir. Justin Dauwels and Dr. Nergis Tömen for their interest in my Master thesis project and is the core member of my thesis committee. Eventually, I want to thank my parents and friends for their heartwarming emotional support in the journey.*

***Chengming Feng***
*Delft, August 2022*

# Contents

# 1

# Scientific Paper

# BladeSynth: Damage Detection and Assessment in Aircraft Engines with Synthetic Data

Chengming Feng[1], Steve Nowee[2], Yancong Lin[1], and Jan van Gemert[1]
Delft University of Technology[1]
Aiir Innovations[2]

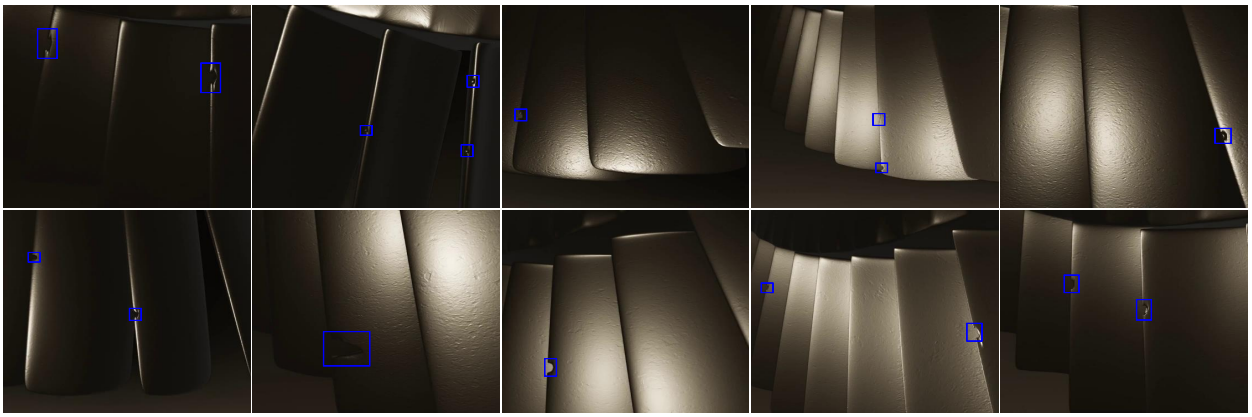


Figure 1. We introduced BladeSynth, a large-scale synthetic dataset for damage detection and assessment in aircraft engines. We show examples from the BladeSynth dataset, where each image contains one or several defects on the surface of blades. The random size, shape, and location of damages vary across examples. In addition, we also change the texture of blades, illumination, and camera poses. The synthetic damages are labeled with blue bounding boxes.

## Abstract

*Deep learning has been widely implemented in industrial inspection, such as damage detection from images. However, training deep networks requires massive data, which is hard to collect and laborious to annotate, especially in the aviation scenario of aircraft engines. To alleviate the demand for annotated data, we create BladeSynth - a large synthetic image dataset for detecting damage from aircraft engines, and empirically evaluate the transferability of state-of-the-art Scaled-YOLOV4 from synthetic to real world by pre-training on synthetic data and fine-tuning on real data. Our experiments show that pre-training on synthetic data improves the performance in damage detection in aircraft engine images.*

## 1. Introduction

Damage detection is crucial in various industrial applications. Conventional approaches often rely on manual inspection. The recent advancement in deep learning-based object detection has significantly facilitated damage detection with remarkable efficiency and accurate-detection rate [1, 22, 25, 30, 35]. However, learning-based approaches suffer from the lack of large-scale datasets, thus not applicable in certain applications, e.g. aircraft engine inspection where annotated data is rare.

To handle the scarcity of real-world datasets, there has been great interest in generating synthetic data. The works [8, 12, 14, 15, 19, 24] have shown that synthetic data are comparable to real data in terms of the performance of object detection, segmentation, and tracking. Furthermore, the synthetic data has the following advantages: $1$) low production cost and labor; $2$) marvelous controllability and abundant variability; $3$) remarkable efficiency and convenience in labeling. Although generating a large-scale synthetic dataset has been proved beneficial, the domain gap between synthetic and real data substantially hampers the transferability of deep networks.

In this paper, we are interested in not only generating

large-scale, high-quality synthetic datasets but also mitigating the domain gap by pretraining on synthetic data and fine-tuning a faction of real data. Our focus is on detecting damages from images that capture aircraft engines. We introduced BladeSynth, a synthetic dataset generator that can automatically, efficiently, and painlessly create labeled synthetic images of damaged turbine blades, as shown in Fig 1. BladeSynth takes domain-randomization [27,28] into consideration, where crucial rendering parameters: illuminations, roughness, textures, and camera pose, alter in a reasonable range comparing to the real data.

We evaluated BladeSynth with the state-of-the-art object detector Scaled-YOLOV4 [29] to test the transferability of the synthetic dataset. We first train a number of pre-trained models on synthetic data, and then we test their performance on real-world data by fine-tuning with a small amount of real data. We show empirically that pre-training on synthetic data improves the performance of damage detection in aircraft engines.

In summary, the contribution of this paper is: *1*) we introduce a novel method to automatically and efficiently model, generate, and label synthetic data of damages in aircraft engines; *2*) we demonstrate that damage detection in aircraft engines benefits from synthetic data.

## 2. Related work

**Synthetic Data.** Generative Adversarial Networks (GANs) [11, 23] are a common approach extensively used in generating synthetic data [3, 7, 9, 10, 31, 33, 34]. For instance, Siva *et al.* [20] used controllable GANs [18] to create synthetic datasets with diverse properties. However, GANs-based approaches suffer from tuning parameters and time-consuming training. Our approach is based on modeling software to avoid numerous hyperparameters in deep learning.

Game engines and 3D modeling software such as CAD and Blender are also viable solutions to generate synthetic data. Matteo *et al.* applied the game engine of Grand Theft Auto V to create a synthetic crowded pedestrian dataset MOTSynth [8]. Eung-Joo *et al.* utilized the Unity game engine to synthesize data taken from onboard Unmanned Aerial Vehicles [15]. Xu *et al.* [32] used untextured CAD models and few real images to create synthetic data. In [21], Planche *et al.* introduced a framework to generate realistic depth images from CAD models for 2.5D recognition. Antoine *et al.* utilized Blender to produce synthetic data for quality inspection [5]. Nevertheless, the game engine requires a specialized simulator which is not available for aircraft engine blades. CAD models are mainly designed for the purpose of mechanical engineering, which is not appropriate in our case. In this paper, we choose Blender to generate synthetic data because of the high quality of rendering, controllability over modeling, and simplification of manufacturability. With Blender Python script, we can automatically and efficiently generate a large-scale synthetic dataset with desired properties.

**Synthtic to Real**. The domain gap between synthetic and real data affects the performance of deep networks in challenging real-world scenarios. One possible solution to bridge this gap is domain randomization. Jonathan *et al.* adopted domain randomization by randomizing the number, position, and orientation of interest objects and non-interest objects, as well as textures [28]. Rendering is randomized in simulators to achieve domain randomization [2, 27]. Another approach to bridge the domain gap is domain adaptation, which contributes to the domain shift of the different distributions of the synthetic and real world [13]. In [4,6,13,17,26], domain adaptation is applied to improve the performance of segmentation in the medical field, vehicle, and driving as well as the performance of action recognition with synthetic data. In this paper, we focus on bridging the domain gap with Domain Randomization during generating synthetic data and fine-tuning synthetic pre-trained models on real data.

## 3. Method

BladeSynthe is a large-scale synthetic dataset for damage detection and assessment in an aircraft engine. The dataset generation process is illustrated in Section 3.1. The statistics of BladeSynthe are explained in Section 3.2.

### 3.1. Dataset Generation Process

To generate synthetic images of engine blades, we use Blender, an open-source 3D modeling software that supports manual modeling and auto-modeling with Python.

**Creating Blade Model**. Firstly, we manually modeled the Low-pressure Turbine (LPT) in Blender. In reality, most defects are caused by small obstacles (e.g. stones) hitting on the blade surface. We simulate this process to generate synthetic damages, as shown in Fig. 2. The size, shape, and position of synthetic damages are all randomly determined.

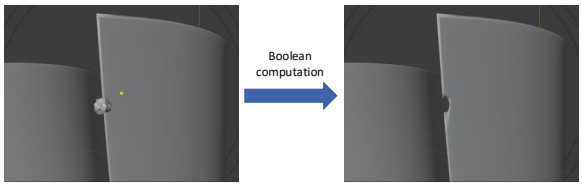



Figure 2. We randomly place obstacles on the surface of the blade. The Boolean computation in Blender is applied to create defects on the blade.

**Rendering Settings**. Rendering is critical for high-quality synthetic data. The challenge is to model the light

reflection on the blade surface, as the light source inside aircraft engines is a spotlight rather than natural sunlight. Therefore, we control a series of rendering parameters to achieve light reflection, including power, radius, and specular of the light source, and metallic, specular, roughness, and color of the blade surface.

The camera poses are also important in rendering. Because the receptive field of the borescope is limited (the free space inside an aircraft engine is constrained), we also control the receptive field of the camera in Blender to make the synthetic data more realistic.

We sample the rendering parameters and camera poses in reasonable ranges to accommodate diversity.

**Inspection Animation**. In the real-world inspection of an aircraft engine, the borescope is always motionless, while the blades are rotating. To simulate the trajectory of cameras, we also fixed the camera pose initially, and rotate the blade model. Additionally, we forced the camera to track damages by moving along the y-axis to acquire more synthetic images, where damages can be captured in diverse positions in the camera with variational illuminations.

**Labeling Defects**. As shown in Fig .3, we implement histogram backprojection to label defects with a pair of inspection animations: A and B. Animations A and B are the same except the colored obstacles are remained in animation A but removed in animation B. Animation A is used for providing the location of damages. Because colored obstacles are tightly affixed to corresponding damages, we acquire the location of damages from obstacles with some modifications. And animation B is the source of synthetic images.

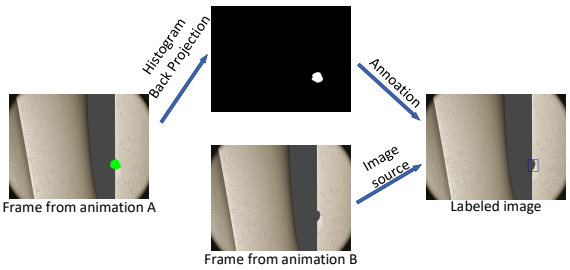



Figure 3. We use histogram backprojection to locate the colored obstacles in frame A. Then we translate the minimum bounding box that encloses the obstacle to frame B on the corresponding frame from the animation and marginally modified the bounding box.

### 3.2. Dataset Statistics

**BladeSynth**. We created 400 pairs of animations A and B as the source of BladeSynth. The FPS of each animation is 30, the resolution is 896×896 pixels, and the frames vary from 480 to 1200. To prevent repetition, in BladeSynth, we firstly separate 400 pairs of animations by train, val, and test as the images library of the train, val, and test set. Then, images are randomly selected from the library to compose the synthetic dataset. We select 20,000 images for training, 2500 image for validation, and 2500 images for testing to compose BladeSynth-20, while BladeSynth-10 is created with 10,000 images for training, 1250 images for validation, and 1250 images for testing. All images are also aimlessly chosen from the image library. BladeSynth contains both damaged blades and healthy blades. We consider all the defects in synthetic data as the same class - 'Dent'.

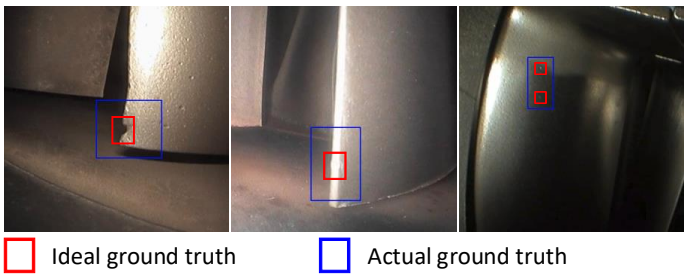
**Real Dataset**. The real dataset is provided by Aiir Innovations[1]. In the Real-Old dataset, all defects are labeled as one class - 'Dent'. In our work, all experiments are pretrained and fine-tuned with Real-Old. To distinguish different types of defects, a new dataset was upgraded from the Real-Old. In the new dataset, the defects are labeled as 'Dent' for small and tiny defects, and 'Missing materials' for large defects. We separate 'Dent' and 'Missing materials' into two datasets, which Real-MissM stands for the dataset of 'Missing materials' and Real-Dent stands for the dataset of 'Dent'. To explore how models trained on Real-Old perform in the two classes, we also evaluate all pretraining and fine-tuning with Real-MissM and Real-Dent. The detailed information of synthetic and real datasets is shown in Tab. 1.

Actually, two labeling problems exist in the real data, which diminish the quality of the real dataset. The first problem is most dents are labeled with large bounding boxes, but the dents are tiny in shape. In Fig. 4a, the ground truth of each defect is labeled with blue bounding boxes, while the ideal ground truth is labeled with red bounding boxes. The second problem is that a considerable amount of dents are miss labeled. In Fig. 4b, the black bounding boxes refer to the missing-labeled defects.

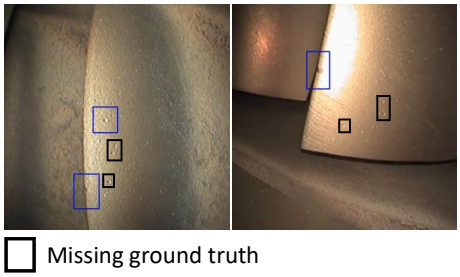
| Dataset | Class | Frames: Train | Val | Test |
|---|---|---|---|---|
| BladeSynth-20 | Dent | 20k | 2.5k | 2.5k |
| BladeSynth-10 | Dent | 10k | 1.25k | 1.25k |
| Real-Old | Dent | \ | \ | \ |
| Real-MissM | Missing materials | \ | \ | \ |
| Real-Dent | Dent | \ | \ | \ |

Table 1. Detailed information about BladeSynth-20, BladeSynth-10, Real-Old, Real-MissM, and Real-Dent. Due to confidentiality, the statistics of real Real-Old, Real-MissM, and Real-Dent are not available.

[1] Aiir Innovations: https://aiir.nl/




(a) Examples of large ground truth.




(b) Examples of missing ground truth.

Figure 4. The labeling error in the real-world dataset. A considerable number of defects are labeled with rather large bounding boxes or completely ignored.

## 4. Experiments

To test whether damage detection in aircraft engines benefits from BladeSynth, we conduct a number of experiments on various benchmarks using the Scaled-YOLOV4-p5 [**?**]. In Sec. 4.1, we first pre-train models on BladeSynth and then test their performance on both synthetic and real-world datasets. Sec. 4.2, we additionally fine-tuned these pre-trained models with real data and evaluate the performance gain on the real-world datasets. We choose Scaled-YOLOV4-p5 for damage detection in all experiments.

Given that the annotated bounding boxes are often much larger than the ground truth(Sec. 3.2), the Intersection over Union (IoU) is rather low, which results in misleading results during evaluation. We, therefore, drop the IoU threshold to 0.25 and calculate $AP_{25}$. We also vary the IoU interval from 0.25 to 0.5 and calculate AP@25:5:50 (AP).

### 4.1. Exp 1: Pre-training with Synthetic Data

We test two pre-training strategies: (1) training from scratch on BladeSynth-20 or BladeSynthe-10, and (2) initializing the models with the COCO pre-trained weights [16] and then training them on BladeSynthe-20 or BladeSynthe-10.

We evaluate the experiment results using AP, $AP_{25}$, numbers of true positives (TP), false positives (FP), and false negatives (FN). For comparison, we also show the performance of the model trained from scratch on Real-Old (the baseline in Tab. 2).

**Domain Gap.** As shown in Tab. 2, training on BladeSynth (including both *Training from scratch* and *fine-tuning COCO*) is able to achieve more than 90% AP on the BladeSynth test set, while the performance on real-world datasets (Real-Old and Real-Dent) is substantially lower (less than 10%), indicating that there is significant domain gap between the synthetic and real data. The main reason is that BladeSynth can not realistically model tiny defects, which are the majority in Real-Old and Real-Dent. Similarly, we find the baseline model also shows a dramatic difference when testing on synthetic and real-world datasets, verifying the existence of the domain gap. This is due to the large number of False Positives generated by the models trained on BladeSynth. For example, the number of false positives from training from scratch is 10× more than the baseline model when testing on the Real-Old dataset.

| | Train | Test | AP | $AP_{25}$ | TP | FP | FN |
|---|---|---|---|---|---|---|---|
| Training from scratch | BladeSynth-20 | BladeSynth-20 | 97.9% | 98.2% | 2541 | 509 | 69 |
| | | BladeSynth-10 | \ | \ | \ | \ | \ |
| | | Real-Old | 5.55% | 8.88% | 186 | 4506 | 554 |
| | | Real-MissM | 30.5% | 48.1% | 35 | 82 | 28 |
| | | Real-Dent | 8.12% | 13.8% | 158 | 1302 | 487 |
| | BladeSynth-10 | BladeSynth-20 | \ | \ | \ | \ | \ |
| | | BladeSynth-10 | 98.2% | 98.4% | 1251 | 148 | 35 |
| | | Real-Old | 3.47% | 5.25% | 192 | 4577 | 548 |
| | | Real-MissM | 27.4% | 43.9% | 39 | 94 | 24 |
| | | Real-Dent | 7.96% | 12.6% | 145 | 843 | 500 |
| Fine-tuning COCO | BladeSynth-20 | BladeSynth-20 | 99.3% | 99.3% | 2583 | 234 | 27 |
| | | BladeSynth-10 | \ | \ | \ | \ | \ |
| | | Real-Old | 3.36% | 4.46% | 168 | 6287 | 572 |
| | | Real-MissM | 32% | 59.4% | 44 | 59 | 19 |
| | | Real-Dent | 6.5% | 9.45% | 147 | 1335 | 498 |
| | BladeSynth-10 | BladeSynth-20 | \ | \ | \ | \ | \ |
| | | BladeSynth-10 | 99.1% | 99.1% | 1264 | 122 | 22 |
| | | Real-Old | 3.61% | 4.78% | 174 | 5730 | 566 |
| | | Real-MissM | 30.2% | 50.5% | 55 | 60 | 8 |
| | | Real-Dent | 6.35% | 9.36% | 145 | 969 | 500 |
| Baseline | Real-Old | BladeSynth-20 | 28.1% | 41.2% | 1008 | 817 | 1602 |
| | | BladeSynth-10 | 26.8% | 39.2% | 469 | 339 | 817 |
| | | Real-Old | 53.3% | 63.2% | 476 | 482 | 264 |
| | | Real-MissM | 44.8% | 70.1% | 41 | 22 | 22 |
| | | Real-Dent | 60.1% | 70.2% | 428 | 240 | 217 |

Table 2. We train all models either from scratch or with COCO pre-trained weights on BladeSynth-20 or BladeSynth-10. We test all pre-trained models on the test set of BladeSynth and real-world datasets. We also show the true positives, false positives, and false negatives with the confidence threshold being 0.1. The baseline model is trained from scratch on Real-Old rather than BladeSynth. There is a significant performance decrease when training on BladeSynth and testing on real-world datasets.

**Comparison Between Training Strategies.** When comparing training from scratch and fine-tuning COCO with BladeSynth, we find that COCO pre-trained model

| Fine-tune on Real-Old | | Real-Old | | | | | Real-MissM | | | | | Real-Dent | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Pre-train | AP | $AP_{25}$ | TP | FP | FN | AP | $AP_{25}$ | TP | FP | FN | AP | $AP_{25}$ | TP | FP | FN |
| | *B20* | 58.5% | 67.5% | 530 | 580 | 210 | 61.8% | 82% | 53 | 29 | 10 | 64.9% | 73.1% | 458 | 205 | 187 |
| | *B10* | 58.4% | 67.3% | 525 | 572 | 215 | 59.1% | **87.8%** | **55** | 30 | 8 | 63.7% | 73% | 457 | 272 | 188 |
| | *C+B20* | 63.9% | **74.4%** | **566** | 555 | 174 | 57.4% | 83.8% | 53 | 34 | 10 | **72.5%** | **81.4%** | **499** | 203 | 146 |
| | *C+B10* | **65.5%** | 74.2% | 557 | 490 | 193 | **62.5%** | 79% | 51 | 31 | 12 | 71.3% | 80% | 482 | 169 | 163 |
| | *C* | 62.7% | 73.7% | 555 | 424 | 185 | 59.1% | 83.1% | 51 | 33 | 12 | 67.6% | 78.8% | 491 | 214 | 154 |
| Baseline | | 53.5% | 63.2% | 476 | 482 | 264 | 42.6% | 64.1% | 40 | 32 | 23 | 60.1% | 70.2% | 428 | 240 | 217 |

Table 3. We fine-tune various pre-trained models on Real-Old and then test on Real-Old, Real-MissM, and Real-Dent individually. The baseline is the model trained from scratch on Real-Old. TP, FP, and TN are evaluated under a confidence threshold of 0.1. We see that pre-training on Synthetic datasets improves detection performance on all real-world datasets.

marginally improves the detection performance on BladeSynth and Real-MissM which contain mostly large defects in real data, but slightly shrinks in detecting tiny defects from Real-Old and Real-Dent.

### 4.2. Exp2: Transferring from Synthetic to Real

We verify the benefit of synthetic data in detecting real-world damages by fine-tuning models which are pre-trained on BladeSynth. The fine-tuning is conducted on the Real-Old dataset. We report the performance AP, $AP_{25}$, numbers of true positives (TP), false positives (FP), and false negatives (FN). We use *B20* and *B10* to represent the pre-trained model of training from scratch on BladeSynth-20 and BladeSynth-10 respectively. *C+B20* indicates that the pre-trained model is first initialized with the COCO weights and then trained on BladeSynth-20. *C+B10* indicates that the pre-trained model is first initialized with the COCO weights and then trained on BladeSynth-20, where *C* indicates COCO weights. Subsequently, we use the subscript of $_{FT}$ to represent fine-tuning pre-trained models on the Real-Old dataset, where $B20_{FT}$ stands for fine-tuning *B20* and *C+B20* stands for fine-tuning $C{+}B20_{FT}$.

Tab. 3 shows the performance of all experiments, which are fine-tuning all pre-trained models on Real-old as well as the baseline that is trained from scratch on Real-Old. When testing on Real-Old, we observe that fine-tuning leads to better performance over the baseline. $C{+}B10_{FT}$ improves over the baseline remarkably by +12% AP on Real-Old.

Additionally, we test the aforementioned models on Real-MissM and Real-Dent to study their ability in detecting large defects and tiny defects, respectively. The results are shown in Tab. 3, where $C{+}B10_{FT}$ outperforms the baseline by $+19.9\%$ AP on Real-MissM and $C{+}B20_{FT}$ outperforms the baseline $+12.4\%$ AP on Real-Dent, thus verifying the assumption that real-world damage detection benefits from pre-training on synthetic BladeSynth. However, when comparing $C{+}B20_{FT}$ and $C{+}B10_{FT}$, there is no more performance increase, indicating adding more synthetic data does not necessarily exemplify the advantage.

In each fine-tuning experiment, we find that the number of false positives from Real-Old is rather higher than

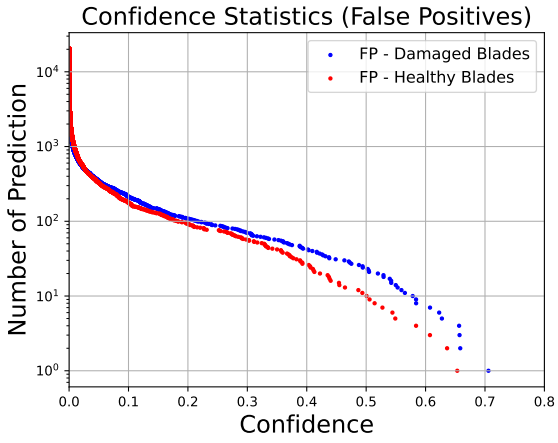
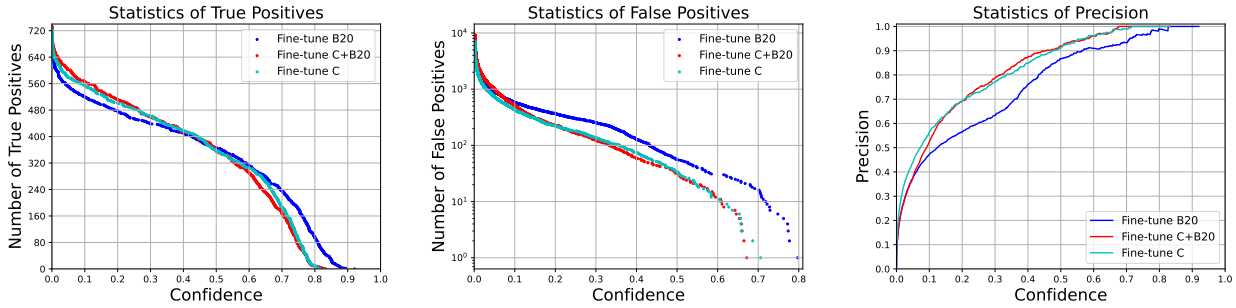



Figure 5. 33680 false positives are predicted by the best model from fine-tuning COCO on Real-Old, where 20327 false positives are from healthy blades and 13353 false positives from damaged blades. We find that fine-tuned models tend to predict a large number of false positives from healthy blades, which has a dramatic impact during evaluation. For example, when the confidence threshold is 0.1, 178 false positives are from healthy blades and 218 false positives are from damaged blades.

Real-MissM and Real-Dent. This is because Real-Old is composed of both healthy blades (without any damage) and damaged blades, while Real-MissM and Real-Dent contain only damagedblades. The excessive false positives in Real-Old originate from the healthy blades as shown in Fig. 5, where testing on healthy blades gives 178 false positives and testing on damaged blades outputs 218 false positives with the confidence threshold being 0.1. Similarly, we observe 58 false positives from healthy blades and 73 false positives from damaged blades when increasing the confidence threshold to 0.3. This is the primary reason why AP on Real-Old is lower than AP on Real-MissM and Real-Dent. And there are two factors that lead to numerous false positives on healthy blades: *1*) some texture on the blade surface resembles the tiny defects, which results in wrong detections; *2*) there are certain unlabeled defects on the "healthy" blades.




(a) Statistics of true positives. (b) Statistics of false positives. (c) Statistics of precision.

Figure 6. We summarized the true positives, false positives, and precision of evaluating $B20_{FT}$, $C{+}B20_{FT}$, and $C_{FT}$ on Real-Dent. We find that $B20_{FT}$ presents more true positives and false positives under a higher confidence threshold, which caused lower precision and AP.

Therefore, we encounter a class imbalance because our network can not correctly distinguish healthy blades from damaged blades. Unfortunately, we did not do further research regarding the class imbalance.

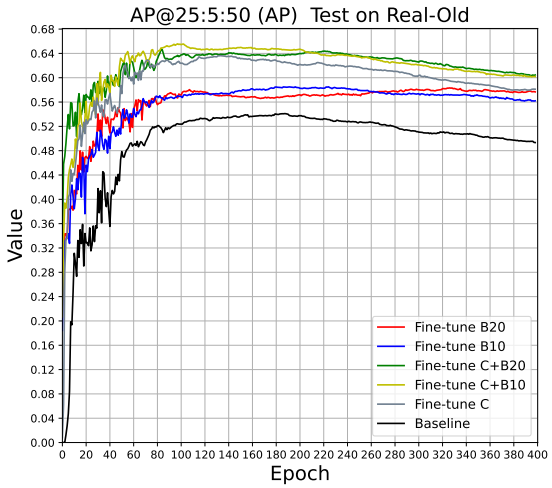



Figure 7. AP during Fine-tuning all synthetic pre-trained models with Real-Old. Fine-tuning *C+B20*, *C+B10*, and *C* outperform fine-tuning *B20* and *B10*, while fine-tuning *B20* and *B10* outperform the baseline.

Another interesting observation is that fine-tuning COCO weights not only improves the results in detecting real-world damages but also outperforms fine-tuning synthetic pre-trained models (*B20* and *B10*) on Real-Old, as shown in Tab. 3. However, COCO does not include any knowledge of defects on blades, indicating that COCO and BladeSynth enhance the performance of the fine-tuned models from different perspectives. And the AP of fine-tuning all pre-trained models on Real-Old is shown in Fig. 7. We observe that fine-tuning *C+B20*, *C+B10*, and *C* relatively outperform fine-tuning *B20* and *B10*, while fine-tuning *B20* and *B10* relatively outperform the baseline. Then we ask a question, why does implementing COCO pre-trained model work better than synthetic only?

To analyze the impact of pre-training on COCO and BladeSynth, we analyze the statistics of predictions from fine-tuning *C+B20*, *B20*, and COCO on Real-Old. In Fig. 6a, we see that $B20_{FT}$ predicts more true positives than $C_{FT}$ and $C{+}B20_{FT}$ when the confidence threshold is higher than 0.47. And $B20_{FT}$ presents 95 true positives, while $C_{FT}$ and $C{+}B20_{FT}$ only present 9 true positives under the confidence threshold of 0.8. However, $B20_{FT}$ predicts significantly more false positives than $C{+}B20_{FT}$ and $C_{FT}$. For example, when the confidence threshold is higher than 0.7, $B20_{FT}$ outputs 17 false positives, while $C_{FT}$ and $C{+}B20_{FT}$ produce 0 false positives, as shown in Fig. 6b. Although $B20_{FT}$ predicts more true positives with high confidence, which develops the recall, more false positives are presented as well even from a low confidence threshold, which substantially weakens the precision, as shown in Fig. 6c. We suspect the excessive false positives are the primary cause of the loss in AP, as the precision of $C{+}B20_{FT}$ decreases dramatically from a low confidence threshold, compared to $C_{FT}$ and $C{+}B20_{FT}$.

Consequently, we explain the effect of COCO and synthetic pre-trained models as follows: *1*) fine-tuning the pre-trained model, which is trained from scratch on synthetic data, improves the detection performance by detecting more true positives with high confidence. However, this approach yields more false positives as well due to the domain gap between synthetic and real-world; *2*) We reckon that fine-tuning COCO pre-trained model with real data improves the detection performance by providing a better start point for training than training from scratch since the COCO pre-trained model was trained for 300 epochs with normal COCO 80-classes object detection dataset and another 200 epochs with augmented data [29].

## 5. Conclusions

We present a large-scale synthetic dataset for industrial inspection, specifically for detecting defects on aircraft engine blades, and evaluate the benefit of pre-training on this synthetic data in detecting real-world damages using Scaled-YOLOV5-P5. Experimental results on various real-world datasets show that the proposed synthetic data is able to boost the performance when fine-tuning with a limited amount of real-world data.

One limitation of the proposed approach in generating data is synthesizing tiny defects, which requires high-quality rendering engines because of the light reflection on blades. Another limitation is the lack of diversity due to the absence of real-world damages.

Future work will focus on collecting more real-world samples and generating a sufficient diverse dataset. Testing different object detectors would also be of inter also help find the best algorithm for damage detection in aircraft engines. For detecting tiny defects, it is kind of a small object detection problem. Some modifications in the structure of the algorithm (e.g. feature pyramid) could help improve the detection performance of tiny defects.

## References


[1] Abdullah Alfarrarjeh, Dweep Trivedi, Seon Ho Kim, and Cyrus Shahabi. A deep learning approach for road damage detection from smartphone images. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 5201–5204. IEEE, 2018. 1

[2] Joao Borrego, Atabak Dehban, Rui Figueiredo, Plinio Moreno, Alexandre Bernardino, and José Santos-Victor. Applying domain randomization to synthetic data for object category detection. *arXiv preprint arXiv:1807.09834*, 2018. 2

[3] Christopher Bowles, Liang Chen, Ricardo Guerrero, Paul Bentley, Roger Gunn, Alexander Hammers, David Alexander Dickie, Maria Valdés Hernández, Joanna Wardlaw, and Daniel Rueckert. Gan augmentation: Augmenting training data using generative adversarial networks. *arXiv preprint arXiv:1810.10863*, 2018. 2

[4] Han Chen, Yifan Jiang, Murray Loew, and Hanseok Ko. Unsupervised domain adaptation based covid-19 ct infection segmentation network. *Applied Intelligence*, 52(6):6340–6353, 2022. 2

[5] Antoine Cordier, Pierre Gutierrez, and Victoire Plessis. Improving generalization with synthetic training data for deep learning based quality inspection. *arXiv preprint arXiv:2202.12818*, 2022. 2

[6] Gabriela Csurka. Domain adaptation for visual applications: A comprehensive survey. *arXiv preprint arXiv:1702.05374*, 2017. 2

[7] Christine Dewi, Rung-Ching Chen, Yan-Ting Liu, and Shao-Kuo Tai. Synthetic data generation using dcgan for improved traffic sign recognition. *Neural Computing and Applications*, pages 1–16, 2021. 2

[8] Matteo Fabbri, Guillem Brasó, Gianluca Maugeri, Orcun Cetintas, Riccardo Gasparini, Aljoša Ošep, Simone Calderara, Laura Leal-Taixé, and Rita Cucchiara. Motsynth: How can synthetic data help pedestrian detection and tracking? In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10849–10859, 2021. 1, 2

[9] Wei Fang, Feihong Zhang, Victor S Sheng, and Yewen Ding. A method for improving cnn-based image recognition using dcgan. *Computers, Materials and Continua*, 57(1):167–178, 2018. 2

[10] Maayan Frid-Adar, Idit Diamant, Eyal Klang, Michal Amitai, Jacob Goldberger, and Hayit Greenspan. Gan-based synthetic medical image augmentation for increased cnn performance in liver lesion classification. *Neurocomputing*, 321:321–331, 2018. 2

[11] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014. 2

[12] Stefan Hinterstoisser, Olivier Pauly, Hauke Heibel, Marek Martina, and Martin Bokeloh. An annotation saved is an annotation earned: Using fully synthetic training for object detection. In *Proceedings of the IEEE/CVF international conference on computer vision workshops*, pages 0–0, 2019. 1

[13] Hana Isoi, Atsuko Takefusa, Hidemoto Nakada, and Masato Oguchi. Performance of domain adaptation schemes in video action recognition using synthetic data. In *2022 4th International Conference on Image, Video and Signal Processing*, pages 70–79, 2022. 2

[14] Lea Laux, Sebastian Schirmer, Simon Schopferer, and Johann C Dauer. Build your own training data–synthetic data for object detection in aerial images. 2022. 1

[15] Eung-Joo Lee, Damon M Conover, Shuvra S Bhattacharyya, Heesung Kwon, Jason Hill, and Kenneth Evensen. Validation of object detection in uav-based images using synthetic data. In *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications III*, volume 11746, pages 584–601. SPIE, 2021. 1, 2

[16] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014. 4

[17] Qing Liu, Adam Kortylewski, Zhishuai Zhang, Zizhang Li, Mengqi Guo, Qihao Liu, Xiaoding Yuan, Jiteng Mu, Weichao Qiu, and Alan Yuille. Learning part segmentation through unsupervised domain adaptation from synthetic vehicles. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 19140–19151, 2022. 2

[18] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014. 2

[19] Samarth Mishra, Rameswar Panda, Cheng Perng Phoo, Chun-Fu Richard Chen, Leonid Karlinsky, Kate Saenko, Venkatesh Saligrama, and Rogerio S Feris. Task2sim: Towards effective pre-training and transfer from synthetic data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9194–9204, 2022. 1

[20] Siva Karthik Mustikovela, Shalini De Mello, Aayush Prakash, Umar Iqbal, Sifei Liu, Thu Nguyen-Phuoc, Carsten Rother, and Jan Kautz. Self-supervised object detection via generative image synthesis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8609–8618, 2021. 2

[21] Benjamin Planche, Ziyan Wu, Kai Ma, Shanhui Sun, Stefan Kluckner, Oliver Lehmann, Terrence Chen, Andreas Hutter, Sergey Zakharov, Harald Kosch, et al. Depthsynth: Real-time realistic synthetic data generation from cad models for 2.5 d recognition. In *2017 International Conference on 3D Vision (3DV)*, pages 1–10. IEEE, 2017. 2

[22] Wenting Qiao, Biao Ma, Qiangwei Liu, Xiaoguang Wu, and Gang Li. Computer vision-based bridge damage detection using deep convolutional networks with expectation maximum attention module. *Sensors*, 21(3):824, 2021. 1

[23] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015. 2

[24] Jacob Shermeyer, Thomas Hossler, Adam Van Etten, Daniel Hogan, Ryan Lewis, and Daeil Kim. Rareplanes: Synthetic data takes flight. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 207–217, 2021. 1

[25] ASM Shihavuddin, Xiao Chen, Vladimir Fedorov, Anders Nymark Christensen, Nicolai Andre Brogaard Riis, Kim Branner, Anders Bjorholm Dahl, and Rasmus Reinhold Paulsen. Wind turbine surface damage detection by deep learning aided drone inspection analysis. *Energies*, 12(4):676, 2019. 1

[26] Tao Sun, Mattia Segu, Janis Postels, Yuxuan Wang, Luc Van Gool, Bernt Schiele, Federico Tombari, and Fisher Yu. Shift: A synthetic driving dataset for continuous multi-task domain adaptation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21371–21382, 2022. 2

[27] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE, 2017. 2

[28] Jonathan Tremblay, Aayush Prakash, David Acuna, Mark Brophy, Varun Jampani, Cem Anil, Thang To, Eric Cameracci, Shaad Boochoon, and Stan Birchfield. Training deep networks with synthetic data: Bridging the reality gap by domain randomization. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 969–977, 2018. 2

[29] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Scaled-yolov4: Scaling cross stage partial network. In *Proceedings of the IEEE/cvf conference on computer vision and pattern recognition*, pages 13029–13038, 2021. 2, 6

[30] Niannian Wang, Xuefeng Zhao, Peng Zhao, Yang Zhang, Zheng Zou, and Jinping Ou. Automatic damage detection of historic masonry buildings based on mobile deep learning. *Automation in Construction*, 103:53–66, 2019. 1

[31] Qiufeng Wu, Yiping Chen, and Jun Meng. Dcgan-based data augmentation for tomato leaf disease identification. *IEEE Access*, 8:98716–98728, 2020. 2

[32] Xu Yang, Xiumin Fan, Jinge Wang, and Kunbo Lee. Image translation based synthetic data generation for industrial object detection and pose estimation. *IEEE Robotics and Automation Letters*, 2022. 2

[33] Huijuan Zhang, Zongrun Huang, and Zhongwei Lv. Medical image synthetic data augmentation using gan. In *Proceedings of the 4th International Conference on Computer Science and Application Engineering*, pages 1–6, 2020. 2

[34] Yan Zhang, Shiyun Wa, Pengshuo Sun, and Yaojun Wang. Pear defect detection method based on resnet and dcgan. *Information*, 12(10):397, 2021. 2

[35] Yu Zhou, Ronggang Cao, Ping Li, Xiao Ma, Xueyi Hu, and Fadong Li. A damage detection system for inner bore of electromagnetic railgun launcher based on deep learning and computer vision. *Expert Systems with Applications*, 202:117351, 2022. 1

# 2

# Deep Learning Based Object Detection

Object detection is a technology that is connected with computer vision and image processing. And object detection is widely applied in the field of security, medical, military, transportation, facial detection, and pedestrian detection. In recent years, with the development of deep learning, object detection based on deep learning has achieved significant milestones. And object detection has become a hot challenge in computer vision.
Deep learning based Object detection algorithms can be subdivided into anchor based and anchor free. In this chapter, the concept of deep learning based object detection, anchor based object detectors, and anchor free object detectors is introduced.

## 2.1. Basic Concept of Object Detection

The problem definition of object detection is to determine where objects are located in a given image and which category each object belongs to [51]. Therefore, in the process of object detection, the algorithm is supposed to correctly classify and localize the target of interest. As shown in Fig. 2.1, the person, dog, and ball are correctly localized and classified.

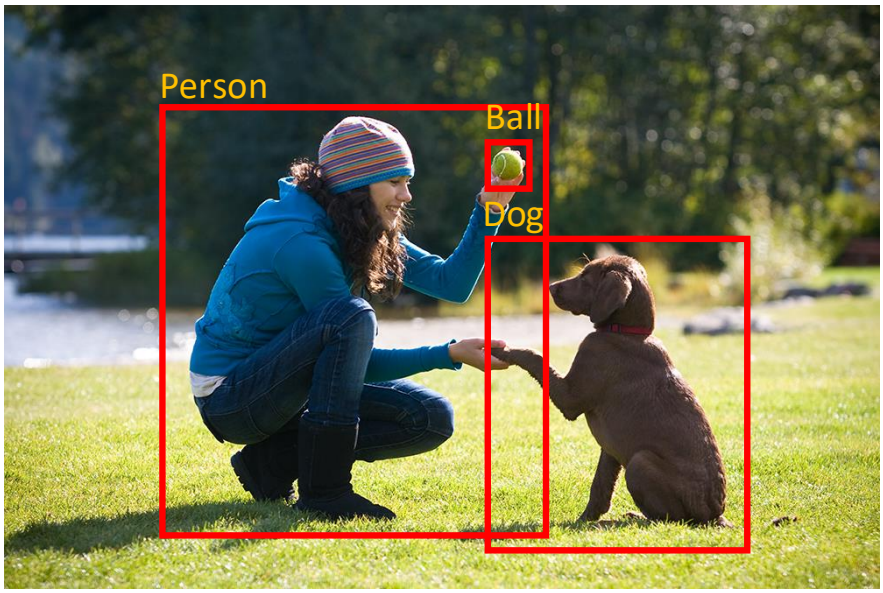



**Figure 2.1:** An example[1] of object detection. The person, ball, and dog are localized with red bounding boxes.

[1] Original image source: https://www.cdc.gov/healthypets/pets/dogs.html

In the traditional approaches to object detection, the sliding window is the main tool for extracting features. But moving sliding windows along an image suffers from heavy computation complexity and is time-consuming. And the feature extraction is not satisfying. A vital advancement in deep learning based object detection algorithms is the extraction of features. With the application of convolution neural networks, the performance of feature extraction has reached a considerable degree. And the computation burden is reduced with a variety of backbones and detection algorithms.

### 2.1.1. Components of Object Detection Algorithms

The structure of object detection algorithms can be subdivided into four Components, which are input, backbone, neck, and head [4], as shown in Fig. 2.2. The input is images, patches, etc. And there are many CNN that can be assigned as backbones, e.g., VGG [41], Densenet [25], ResNet [18], ResNeXt [47], Mobilenets [23], and CSPDarknet [45]. The neck is used for extracting features. Because the deepest feature map has rich semantic features for classification and layers which is slightly shallower than the deepest feature map has abundant spatial features for localization. Usually, the neck has multiple top-down and bottom-up paths through multi-level feature maps to share the context to improve the detection of multi-scale objects. FPN [28], ASPP [7], SPP [20], PAN [31], SFAM [50], ASFF [32], and BiFPN [42] are techniques for the neck. For heads, they can be separated into two-stage including: Faster R-CNN [39], Cascade R-CNN [6], Mask R-CNN [19], R-FCN [9]; one-stage including: YOLOv1 [38], YOLO9000 [36], YOLOv3 [37], YOLOv4 [4], Scaled-YOLOv4 [44], RetinaNet [29], SSD [33], FCOS [43], Cornernet [26], Centernet [11].

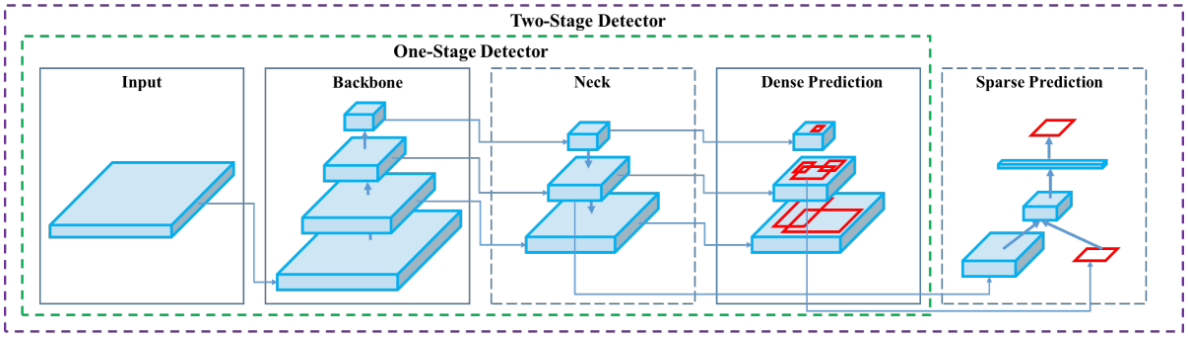



**Figure 2.2:** The structure of object detection algorithms [4]. In general, an object detection algorithm is constructed by input, backbone, neck, and head. The input can be images or patches. The backbone is a CNN for downsampling images. The neck is for extracting both semantic and spatial features. The head is applied to predict the bounding box and class.

### 2.1.2. Evaluation Metrics of Object Detection

In this section, the concept of evaluation metrics including true positives, false positives, false negatives, precision, recall, average precision, and mean average precision is introduced.

#### 2.1.2.1 TP, FP, FN

To introduce the evaluation metrics of object detection, some basic component is presented first. The prediction from an object detector can be divided into True Positives (TP), False Positives (FP), and False Negatives (FN).

A true positive is defined as the correct prediction for both classification and localization. And the correct prediction for localization is judged by the Intersection of Union (IoU), as shown in Fig. 2.3. The IoU is computed between the bounding box of prediction and the ground truth by diving the area of overlap by the area of union. And an IoU threshold is always set to judge whether the localization of prediction is correct. If the IoU is higher than the IoU threshold, then it is a correct localization. Otherwise, the prediction of localization is incorrect.

Then, the correct prediction for classification is judged by confidence. The network will predict a vector of the probability of each class. For instance, in Fig. 2.4a, the class person has the highest confidence, then the classification is a person. And practically, a confidence threshold is also applied

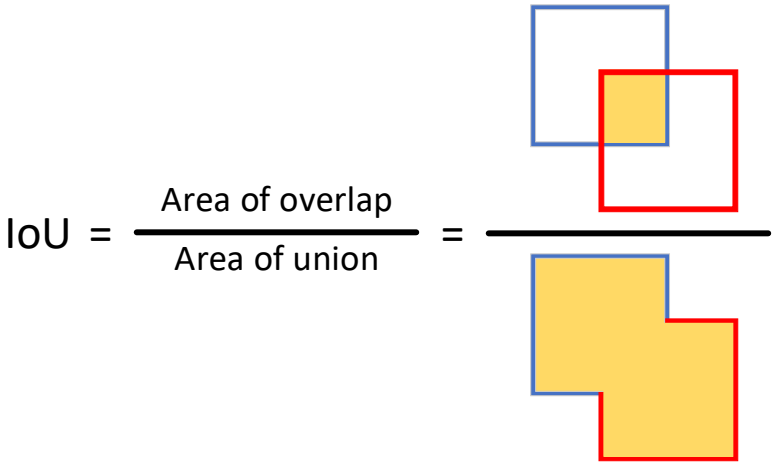
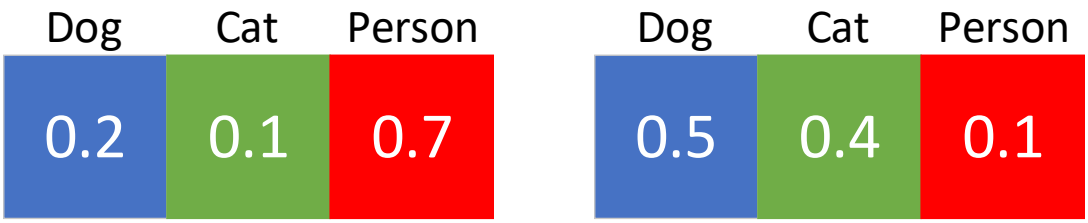



**Figure 2.3:** Intersection of Union (IoU) equals the ratio between the area of overlap and the area of union between the prediction and the ground truth

to filter the low-confidence predictions. In Fig. 2.4b, if we set the confidence threshold at 0.6, although the class dog has the highest confidence, the detection as a dog is wrong. Therefore, only if a prediction exceeds the IoU threshold and confidence threshold, the prediction is correct.




**(a)** A confidence vector.

**(b)** A confidence vector.

**Figure 2.4:** A confidence vector of three classes is shown. If we set a confidence threshold at 0.6, the vector in Fig. 2.4a gives the prediction as a person. And the vector in Fig. 2.4b is an incorrect prediction.

A false positive is defined as a prediction that is incorrect in classification or localization. And a false negative is defined as an ignored ground truth. In object detection, the number of true positives pluses the number of false negatives is the number of ground truth.

#### 2.1.2.2 Precision, Recall

Precision and recall, which are defined in Equ. 2.1 and 2.2, are crucial evaluation metrics in object detection.

$$Precision = \frac{true\ positives}{true\ positives + false\ positives} = \frac{true\ positives}{predictions} \tag{2.1}$$

$$Recall = \frac{true\ positives}{true\ positives + false\ negatives} = \frac{true\ positives}{ground\ truth} \tag{2.2}$$

As mentioned in Sec. 2.1.2.1, the confidence threshold determines the true positives significantly. If a higher confidence threshold is set, then the network will predict fewer true positives and false positives because the prediction with low confidence is filtered. In general, there are more true positives with high confidence than false positives. Therefore the precision will increase but the recall will decrease. And if a lower confidence threshold is applied, then the network will give more true positives and false positives. In this case, numerous false positives will be given, which results in a decreasing precision but increasing recall. To analyze the influence of the confidence threshold, the precision-recall curve is implemented. Fig. 2.5 is the plot of precision-recall curves. The precision-recall curve is plotted under all confidence thresholds. At the intersection of the recall-axis, the confidence threshold is 0. And from the bottom right point to the top left point, the confidence threshold increases. When the confidence threshold is beyond the highest confidence of false positives, then there are no false positives and the precision is 1.

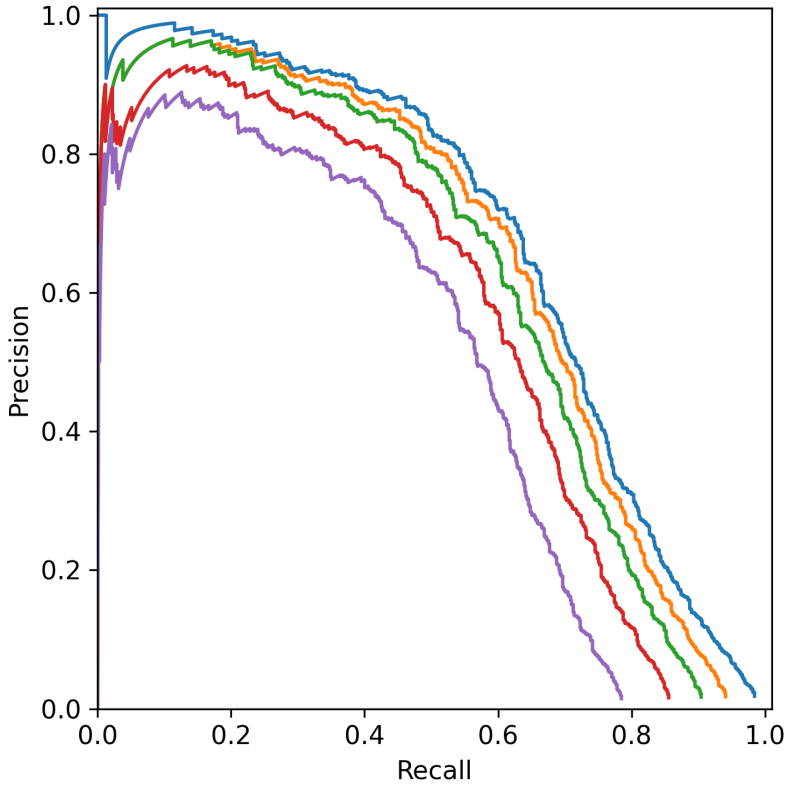



**Figure 2.5:** Precision-Recall curves (PR curves) are plotted under the interval of confidence threshold. Each point on a curve corresponds to a specific confidence threshold.

#### 2.1.2.3 Average Precision

The most common evaluation metric in object detection is average precision (AP) [34]. To compute average precision, the interpolated method introduced in PASCAL VOC [12] is introduced. The common 11-interpolate method is described as: dividing the interval of recall by 0.1, then we get a 11-value interval $[0,\ 0.1,\ 0.2,\ 0.3,\ 0.4,\ 0.5,\ 0.6,\ 0.7,\ 0.8,\ 0.9,\ 1]$. For each value in the interval of recall, we know the corresponding prevision $P(R)$. But in practical, $P(\tilde{R})$ is replaced by the maximum precision $P_{interp}(R)$, where $\tilde{R} \ll any\ recall$. Then the average of $P_{interp}(R)$ is over the 11-value recall interval is the average precision, as shown in Equ. 2.5. Also, more interpolation can be applied to calculate average precision. In COCO [30], 101-value interpolation is used.

$$AP_{11} = \frac{1}{11} \sum_{R\in[0,0.1,\ldots,1]} P_{interp}(R)$$
$$where \quad P_{interp}(R) = \max_{\tilde{r}\gg r} P(\tilde{R}) \tag{2.3}$$

As the IoU threshold will also influence the precision and recall, therefore, for the competition of COCO object detection, an interval of $[0.5,\ 0.55,\ 0.6,\ 0.65,\ 0.7,\ 0.75,\ 0.8,\ 0.85,\ 0.9,\ 0.95]$ is set to calculate average under different IoU threshold. Where the AP in COCO is:

$$AP = AP@[.5:0.95] = \frac{1}{10} \sum_{IoU\in[0.5,0.55,\ldots,0.95]} AP@IoU \tag{2.4}$$

The AP@.5 refers to the average precision under the IoU threshold of 0.5. And AP@.5 is the metric for PASCAL VOC.

AP is evaluated in a single class, for multiple classes, the mean average precision (mAP) is presented to calculate the average AP over all classes:

$$mAP = \frac{1}{N} \sum_{i}^{N} AP_i \tag{2.5}$$

## 2.2. Anchor Based Object Detection Algorithm

The sliding window is widely used in traditional object detection algorithms to search targets. Although the sliding window is simple to apply, this approach leads to abundant computation and complexity. To achieve better performance, the anchor is introduced as prior to predicting the location of objects. In general, the network will predict numerous anchors and corresponding offsets on one image as a candidate. To modify anchors approaching the ground truth, the process is treated as regression progress by adjusting the offsets.

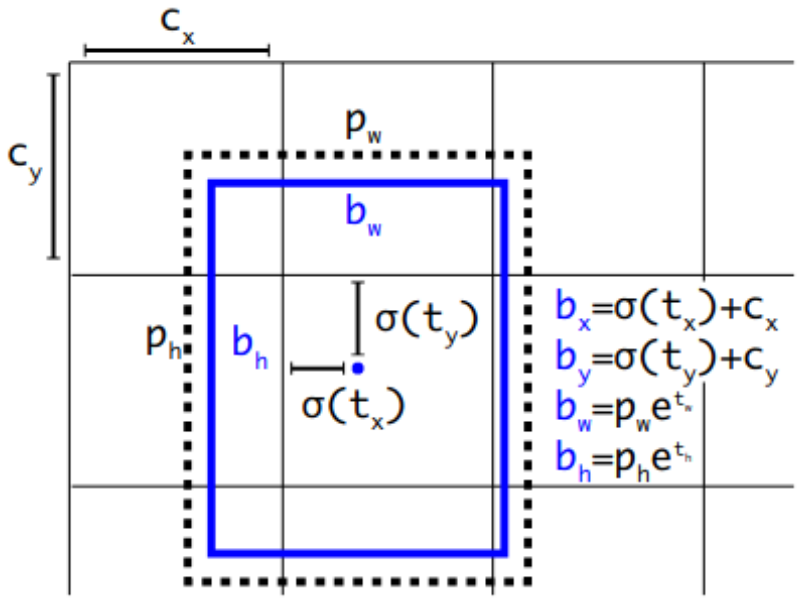



**Figure 2.6:** Anchor in YOLOv3 [37]. The input is $t_x$, $t_y$, $t_w$, $t_h$, and the corresponding coordinates of the actual bounding box is $b_x$, $b_y$, $b_w$, $b_h$. The detailed computation is shown in this figure. And $\sigma(t_x)$ is pass the $t_x$ into a sigmoid function.

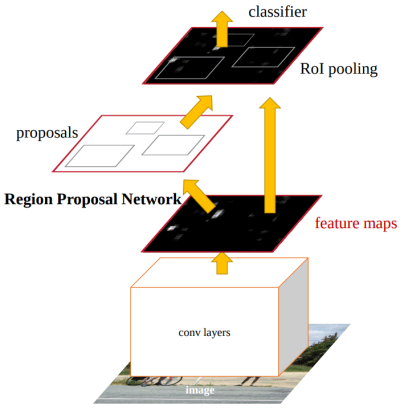
In this section, anchor based two-stage detectors including: Faster R-CNN [39], Feature Pyramid Network [28]; and anchor based one-stage detectors including: SSD [33], YOLOv3 and [37], Scaled-YOLOv4 [44] are introduced in short.

### 2.2.1. Two-stage Detector

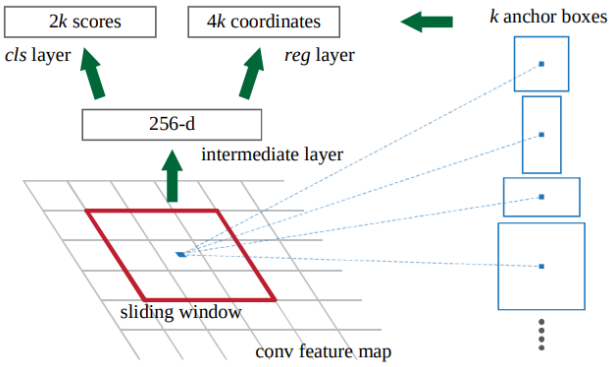
The process of anchor based, two-stage object detection algorithm can be defined as firstly generating region proposals from images, then regressing the region proposals to acquire the bounding box of the target.

#### 2.2.1.1 Faster R-CNN

He et.al. introduced Faster R-CNN [39] which is an end-to-end, two-stage, and anchor based object detection algorithm. In Faster R-CNN, the pipeline of Fast R-CNN [16] is utilized. Instead of selective search, Faster R-CNN applied Region Proposal Network (RPN) (see Fig. 2.7a) to generate region proposals. RPN slides a small CNN over the last shared convolutional feature map from conv layers. As shown in Fig. 2.7b, at each position of the sliding window, K region proposals (anchors) are predicted. And for each proposal, 4 outputs are for encoding coordinates, and 2 outputs are for estimating the probability of a binary label, which is an object or not object. Therefore, the embedded coordinates and scores are sent to RoI pooling for the following bounding box regression and object classification.




**(a)** The pipeline of Faster R-CNN.




**(b)** The structure of Region Proposal Network. In Fig. 2.7a, Faster R-CNN applied Region Proposal Network to generate region proposal. In Fig. 2.7b, Region Proposal Network predicts k anchors, 2k scores for classification, and 4k coordinates for localization.

**Figure 2.7:** Pipeline and Region Proposal Network of Faster R-CNN [39].

#### 2.2.1.2 Feature Pyramid Networks

In the deep convolution neural network, the feature map in the deeper layer has abundant semantic information but sickly spatial information, while the feature map in the shallower layer has better spatial information but weaker semantic information. This character decides that deeper feature maps are better for object classification and shallower deeper feature maps are better for object localization. To improve both object classification and localization, Lin et.al. presented Feature Pyramid Networks (FPN) [28] in 2017. In Fig. 2.8, Feature Pyramid Network is a top-down architecture with lateral connections, which is developed for building high-level semantic feature maps at all scales [28]. And at all levels, the architecture will predict individually. FPN significantly improved the performance of object detection, especially for large-scale object detection. And FPN is extensively applied in the following works of object detection [37, 4, 39, 44].

### 2.2.2. One-stage Detector

Differing from two-stage detectors, one-stage detectors do not need the progress of predicting region proposals. Instead, one-stage detectors predict the object coordinate and the probability of object classification. Generally, one-stage detectors have faster detecting speed.

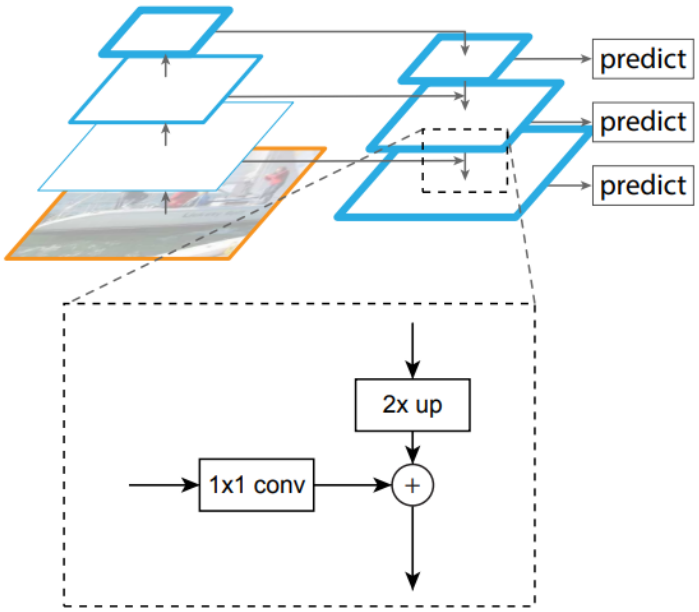



**Figure 2.8:** The structure of Feature Pyramid Network (FPN) [28]. FPN is an architecture composed of a top-bottom path with lateral connections, which is developed for building high-level semantic feature maps at all scales. And at all levels, the architecture will predict individually.

#### 2.2.2.1 SSD

Wei et.al. presented Single Shot Detector (SSD) [33] in 2016. The previous object detectors before 2016 only used the deep branch of feature maps for detection. SSD introduced a novel method to detect with multi branches of convolution layers to achieve detecting on multi-resolution as shown in Fig. [33]. VGG-16 is utilized as the backbone of SSD, With the following extracting-feature layers in multi-resolution, SSD can detect objects in multi-scales. And SSD has an advanced performance on small object detection.

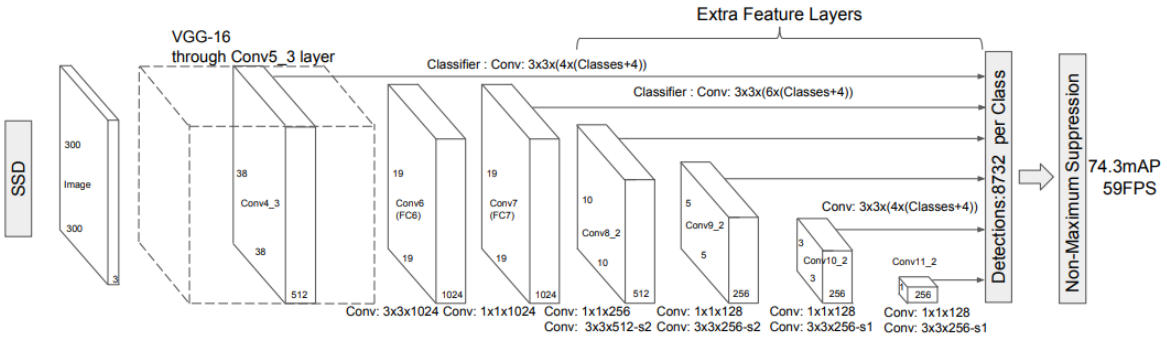



**Figure 2.9:** The pipeline of Single Shot Detector (SSD) [33]. SSD has multiple detection branches. Therefore, SSD can achieve multi-resolution detection.

#### 2.2.2.2 YOLOv3

YOLOv3 [37] is famous for its detection speed and comparable detection precision. To improve the shortage of multi-scale context on YOLO9000 (YOLOv2)[36], YOLOv3 implemented three detection branches with the idea of Feature Pyramid Network to achieve multi-scale object detection. Also, YOLOv3 applied a new backbone - Darknet-53 to extract features. In Fig. 2.10 is the structure of

Darknet-53, where the $3 \times 3$, $1 \times 1$ convolutional layers, and residual layer [18] are used.

During prediction, YOLOv3 predicts an objectness score for each bounding box using logistic regression. Then, the biggest IoU between ground truth and prediction box is found and the prediction box with biggset IoU is classified as a positive prediction. While inference, a threshold of confidence is set and a Non-Maximum Suppression [21] is applied to give the final prediction result. The loss of YOLOv3 is composed of the classification loss, the object loss (confidence loss), and the IoU loss.

| | Type | Filters | Size | Output |
|---|---|---|---|---|
| | Convolutional | 32 | 3 × 3 | 256 × 256 |
| | Convolutional | 64 | 3 × 3 / 2 | 128 × 128 |
| 1× | Convolutional | 32 | 1 × 1 | |
| | Convolutional | 64 | 3 × 3 | |
| | Residual | | | 128 × 128 |
| | Convolutional | 128 | 3 × 3 / 2 | 64 × 64 |
| 2× | Convolutional | 64 | 1 × 1 | |
| | Convolutional | 128 | 3 × 3 | |
| | Residual | | | 64 × 64 |
| | Convolutional | 256 | 3 × 3 / 2 | 32 × 32 |
| 8× | Convolutional | 128 | 1 × 1 | |
| | Convolutional | 256 | 3 × 3 | |
| | Residual | | | 32 × 32 |
| | Convolutional | 512 | 3 × 3 / 2 | 16 × 16 |
| 8× | Convolutional | 256 | 1 × 1 | |
| | Convolutional | 512 | 3 × 3 | |
| | Residual | | | 16 × 16 |
| | Convolutional | 1024 | 3 × 3 / 2 | 8 × 8 |
| 4× | Convolutional | 512 | 1 × 1 | |
| | Convolutional | 1024 | 3 × 3 | |
| | Residual | | | 8 × 8 |
| | Avgpool | | Global | |
| | Connected | | 1000 | |
| | Softmax | | | |

**Figure 2.10:** The structure of Darknet-53 [37]. Darknet-53 is a fully convolutional neural network. See in this figure that there are only convolutional and residual layers.

#### 2.2.2.3 YOLOv4

YOLOv4 [4] implemented abundant novel technologies in deep learning to advance detection performance. Firstly, YOLOv4 applied CSPDarknet53 [45] as the backbone. Then, for the feature extraction, YOLOv4 implemented the structure of SPP [20] and PAN [31]. YOlOv4 took the same strategy of YOLOv3 to predict the class and bounding box of the object. YOLOv4 also applied novel technologies includeing: CIoU loss [52], Cross mini-Batch Normalization (CmBN) [4], Mish activation [4], etc.

#### 2.2.2.4 Scaled-YOLOv4

In Scaled-YOLOv4 [44], Chien-Yao et.al. focused on scaling the depth, width, resolution, and structure of the network to improve the performance of YOLOv4 [4]. Scaled-YOLOv4 has multi-scales including: YOLOv4-tiny, YOLOv4-large, and YOLOv4-CSP as shown in Fig. 2.11. In our work, to peruse a quality mAP and the potential demand of real-time detection, we chose YOLOv4-P5 in YOLOv4-large as the pipeline.

### 2.2.3. The disadvantage of Anchor Based Object Detection Algorithm

Although anchor is widely implemented in object detection, the drawback and limitation of using anchor are non-negligible. Firstly, the detector is sensitive to the size, length-width ratio, and quantity of anchors, because, for different objects, the best setting of anchors varies. Also, the setting of anchors is an additional hyperparameter, which increases the complexity of the algorithm. Secondly, numerous anchors will be generated during training to match the ground truth, but most anchors are predicted as false positives, which causes an imbalance between false and true positives.

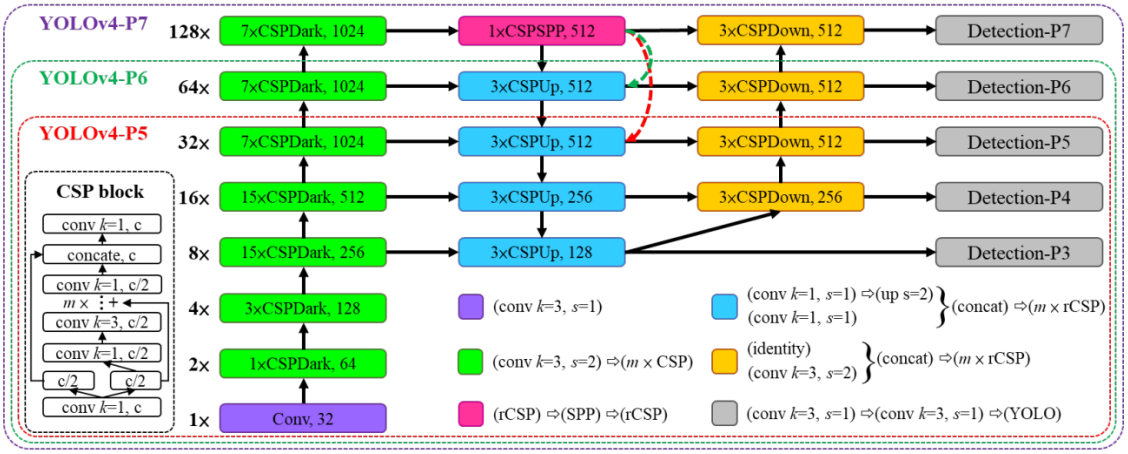



**Figure 2.11:** The multi-scales of Scaled-YOLOv4 [37]. Scaled-YOLOv4 focused on scaling the depth, width, resolution, and structure of the network.

## 2.3. Anchor Free Object Detection Algorithm

To overcome the disadvantage of the heavy computation complexity and abundant hyperparameters in anchor-based object detection algorithm, anchor free object detection algorithm is presented by dividing the images into grids or detecting with key points. In this section, anchor free one-stage object detection algorithms including: YOLOv1 [38], Cornernet [26], and Centernet [11] are shortly introduced.

### 2.3.1. YOLOv1

YOlOv1 [38] is an anchor-free one-stage object detector. In Fig. 2.12, YOlOv1 firstly divides the images into $7$ grids. Then, for each grid, it will predict 2 bounding boxes, the confidence of bounding boxes, and the classification probability. A grid will predict an object if the center of the object falls in the grid. The disadvantage of YOLOv1 is obvious, because for each grid, it can only predict 2 objects. If centers of three or more objects fall into the same grid, then YOLOv1 can not predict all objects. Also, the input size and output size of YOLOv1 are fixed. Although YOLOv1 has a faster detection speed than two-stage detection algorithms, the mAP is weaker.

### 2.3.2. Cornernet

Cornernet [26] implemented key points to achieve anchor free. In Fig. 2.13, Cornernet predicts the top-left and bottom-right corner of an object. The backbone - Hourglass will predict the heatmaps of all top-left and bottom-right corners, as well as the embedding vector and offsets for each detected corner.

Cornernet introduced a novel computation for corners, named corner pooling as shown in Fig. 2.14. For the top-left corner, corner pooling applied the max-pooling horizontally from left to right and vertically from bottom to top on two feature maps respectively. And the two feature maps are added to get the blue point.

### 2.3.3. Centernet

With the foundation of Cornetnet, Centernet [11] was designed with two customized modules, center pooling and cascade corner pooling to improve Cornernet, as shown in Fig. 2.16. Cornernet applied center pooling to output the heatmap of center key points and cascade corner pooling to output the heatmap of corner keypoint. The potential bounding boxes are predicted by corner key points and the center key points decide the final bounding box as a prediction.

See Fig. 2.16a, the aim of center pooling is searching the maximum value vertically and horizontally on a feature map, which is generated by the backbone, to find out if one pixel on the feature is the keypoint or not. As Cornernet is sensitive to the information of edges, cascade corner pooling is introduced to improve by forcing corners to extract the feature inside center regions, as shown in Fig. 2.16b.

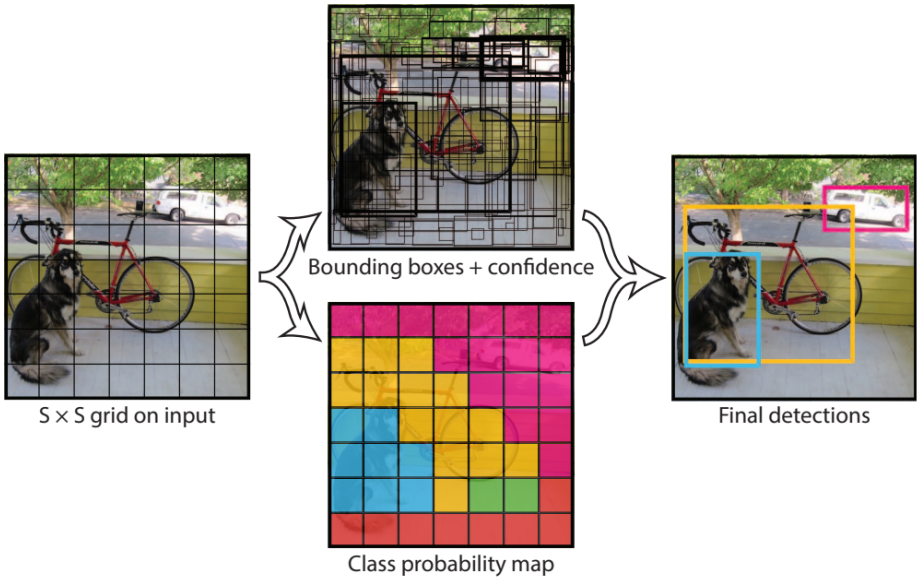



**Figure 2.12:** The process of YOLOv1 [38]. YOLOv1 divides an image into $S \times S$ grids. For each grid, it will predict 2 bounding boxes, as well as the classification probability and the confidence of bounding boxes.

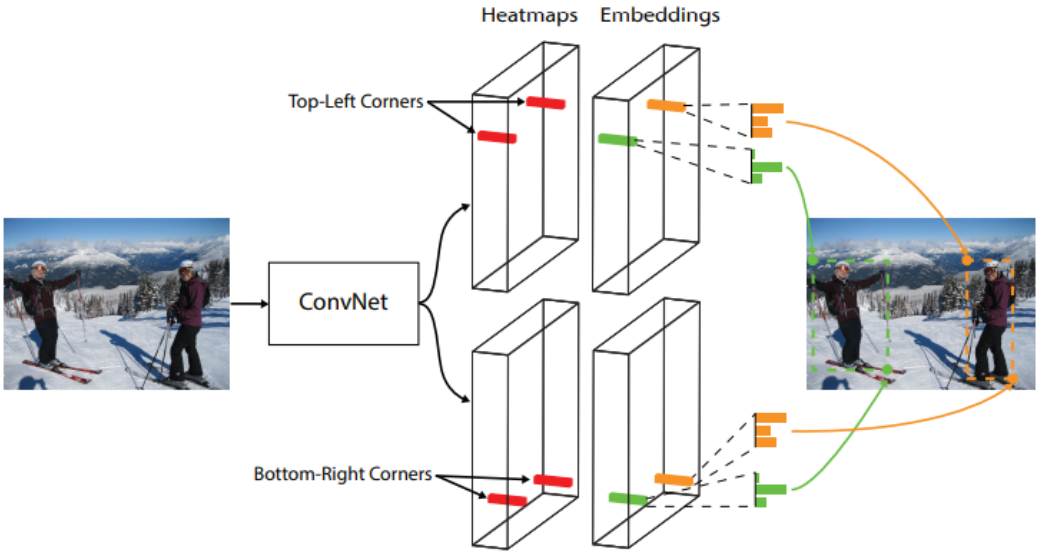



**Figure 2.13:** The process of Cornernet [26]. Cornernet discarded predicting by anchors but searching the top-left corners and bottom-right corners.

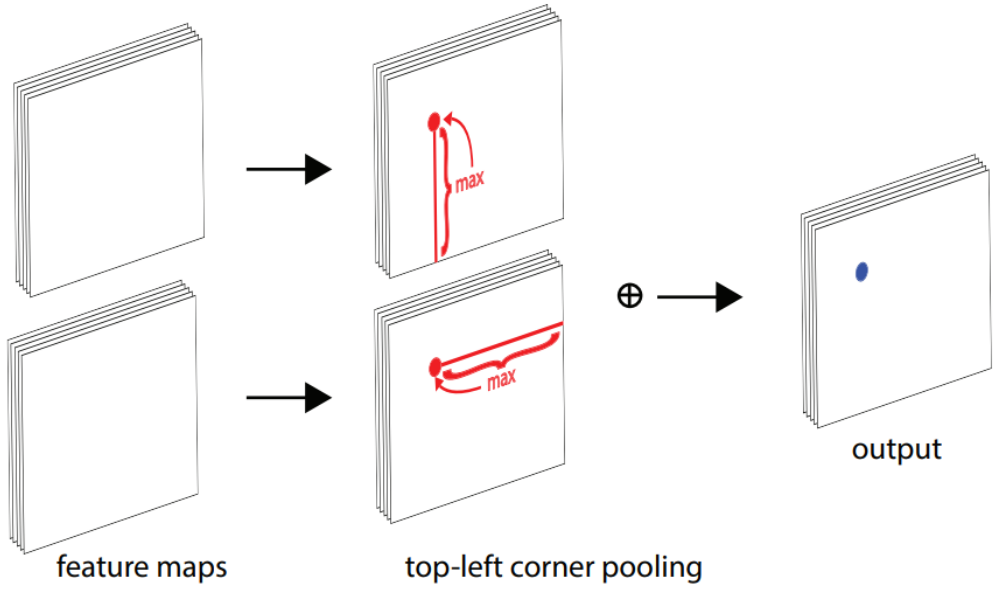



**Figure 2.14:** Corner pooling. [26]. For the top-left corner, corner pooling applied the max-pooling horizontally from left to right and vertically from bottom to top on two feature maps respectively. Then the two feature maps are added to get the blue point on the output.

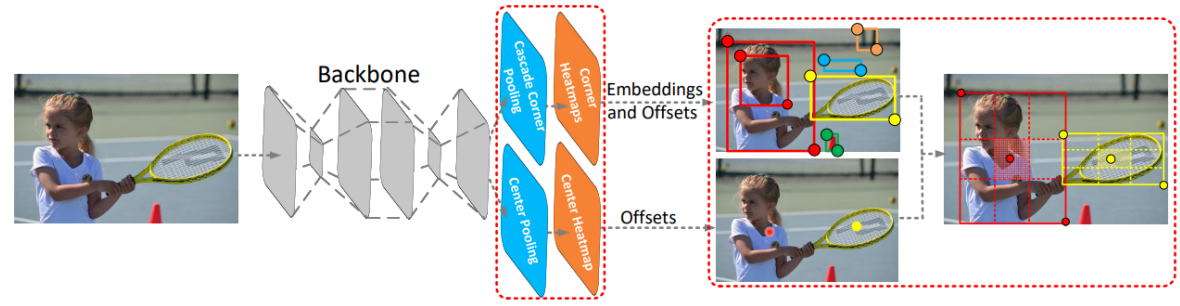
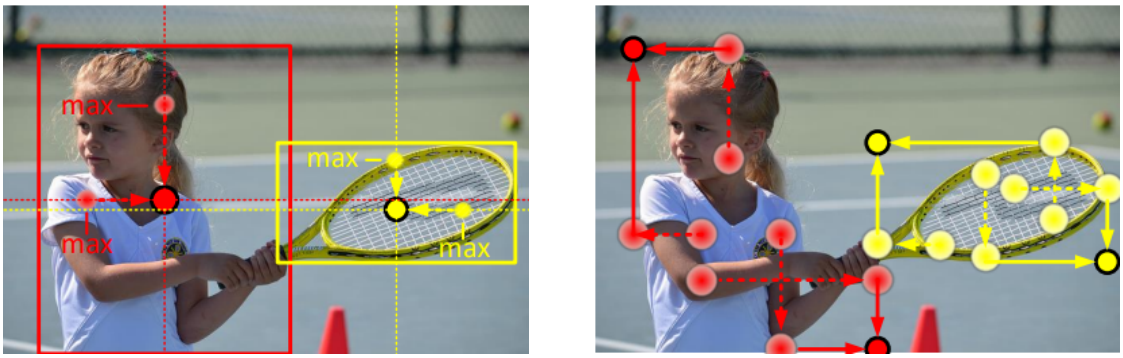



**Figure 2.15:** The pipeline of Centernet [11]. Centernet improved Cornernet by implementing center pooling and cascade corner pooling on two series of feature maps individually to extract more features inside the bounding box of the object.




**(a)** Center pooling.

**(b)** Cascade corner pooling

**Figure 2.16:** The center pooling and cascade corner pooling in Centernet [39]. In Fig. 2.16a, center pooling aims at searching the maximum value vertically and horizontally on a feature map. In Fig. 2.16b, cascade corner pooling focus on forcing corners to extract the feature inside center regions.

# 3

# Generating Synthetic Data

During the development of object detection, large-scale datasets are widely utilized to improve detection performance. However, the burden of labeling data is extremely heavy for creating a large-scale dataset. Also, in some specific and unusual application scenarios like inspection of aircraft engines and medical images of rare and orphan diseases, object detection suffers from the lack of data. To overcome these challenges, synthetic data is introduced. Nowadays, synthetic data are extensively implemented in object detection, segmentation, tracking, and pose estimation due to the efficiency on generate a large-scale dataset and labeling data. In this section, several approaches to generating synthetic data include game engines, generative adversarial nets (GANs), and 3D modeling software - Blender.

## 3.1. Game engine

With the improvement of the game engine, the environment and objects in the game world are more realistic with excellent light reflection, illuminations, and textures. And the game engine has reliable controllability to change the property of environments and objects. Therefore, the game engine has become a useful tool for generating synthetic data on pedestrians, vehicles, etc.

### 3.1.1. Grand Theft Auto V

Grand Theft Auto V (GTA V) provides a large-scale and 3-dimension open world of a city, where the objects are modeled with authentic textures and the background contains abundant and various features. GTA V simulates sorts of weather and the light conditions of the real world from daytime to night, which contributes to the domain adaptation of illuminations.

Fabbri et.al created a synthetic dataset of pedestrians for detection and tracking named MOTSynth[13]. In 3.1, MOTSynth provides several labels including bounding boxes, pose, masks, and depth. And MOTSynth is demonstrated to be a substitute for a real pedestrian dataset for tasks of pedestrians of detection, segmentation, re-identification, and tracking.

**Figure 3.1:** The labels in MOTSynth [13]. MOTSynth contains labels including bounding boxes, poses, segmentation masks, and depth maps of pedestrians.

GTA V is also applied to generate synthetic data of pedestrians in [8, 1, 49]

In [40], Richter et.al. used GTA V to generate synthetic data and label the semantic masks, as shown in 3.2. Hu et.al. applied GTA V to generate synthetic data for monocular 3D vehicle detection and tracking [24], as shown in 3.3.

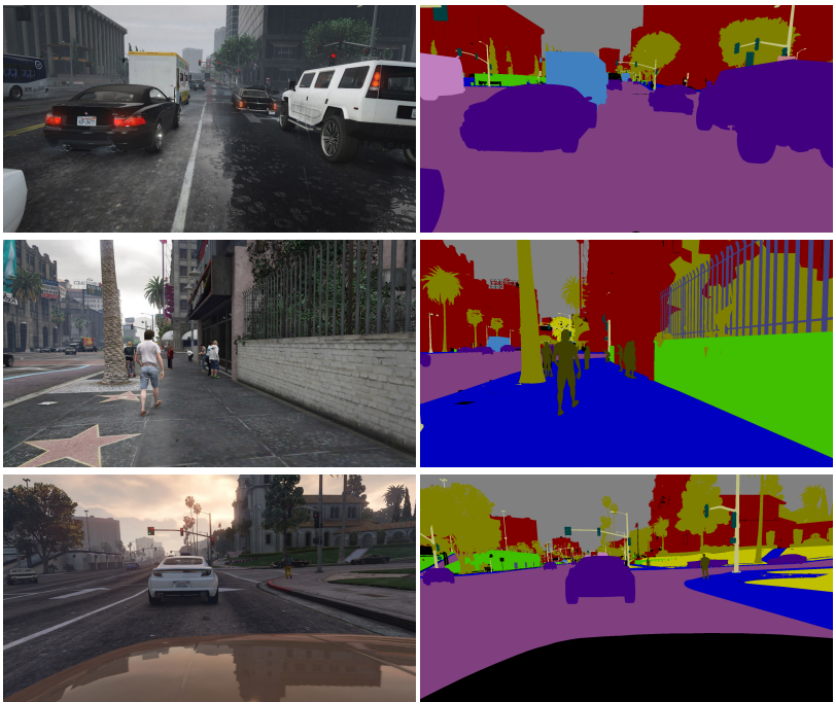


**Figure 3.2:** The left is the synthetic data from GTA V, and The right is the label for segmentation [40].

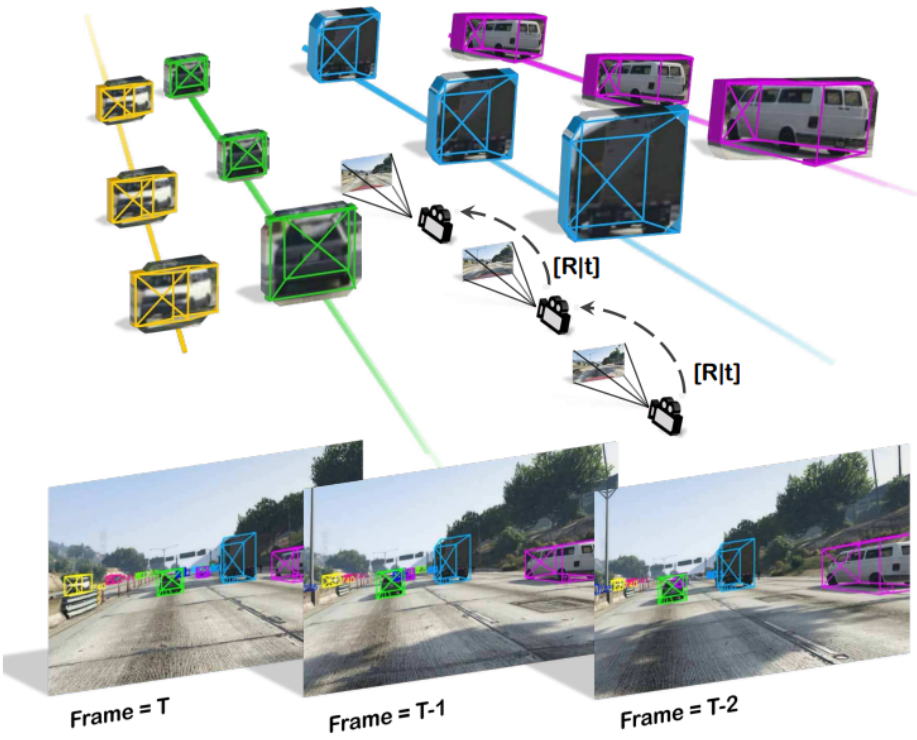



**Figure 3.3:** The pipeline of [24], which achieves the dynamic 3D vehicle tracking.

### 3.1.2. Unity

Unity is another game engine, which is feasible to create synthetic data. In [27], Lee et.al. created people from the view of drones, as shown in Fig. 3.4. [3, 22] also implemented unity to generate synthetic data of pedestrians.

**Figure 3.4:** The synthetic data of people in Unity [27].

The game engine is ideal for generating the synthetic data of pedestrians and vehicles because they can be simply invoked from existing packages. But when generating the synthetic data of the object which does not exist in the game engine, we need to model the new object with the game engine, which is a time-consuming and difficult task.

## 3.2. Generative adversarial nets

Generative Adversarial Networks (GANs) [17, 35] are widely implemented to generate synthetic data [5, 10, 14, 15, 46, 48]. Differ from non-deep-learning approaches, although GANS performs well in generating synthetic data, training GANs suffers from parameter tuning and optimization.

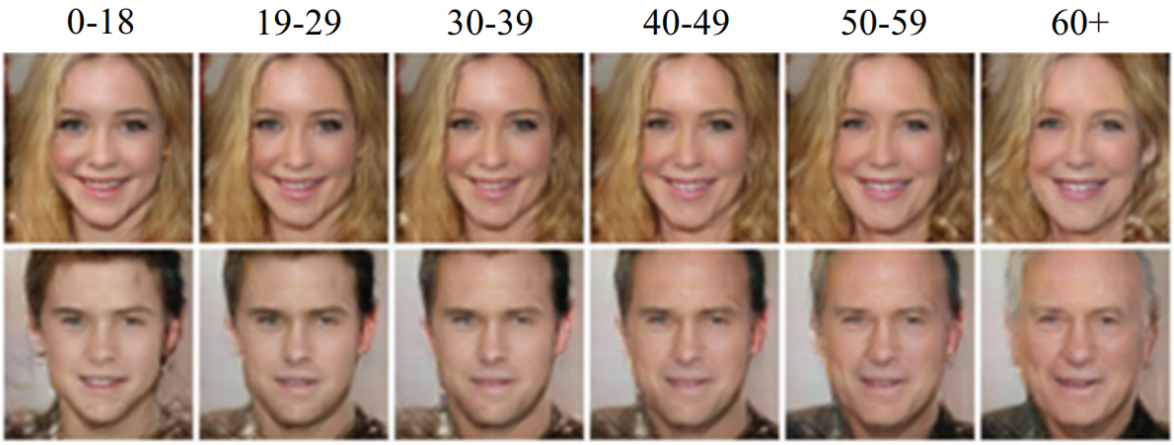



**Figure 3.5:** Synthetic faces with different ages generated by Age-cGAN [2].

## 3.3. Blender

Blender is an open-source 3D modeling software. Blender provides complete controllability on modeling and high-quality rendering. Also, Blender supports Python to automatically model objects. Moreover, Blender can output many formats of images, videos, and depth, which supports several purposes. However, since the quality of synthetic data is decided by the quality of models and rendering, the operator should be proficient in playing Blender. So it will take some time to learn the operation in Blender. But once the operator knows Blender well, it would be easy to model objects and modify the texture.

# References


[1] Giuseppe Amato et al. “Learning pedestrian detection from virtual worlds”. In: *International Conference on Image Analysis and Processing*. Springer. 2019, pp. 302–312.

[2] Grigory Antipov, Moez Baccouche, and Jean-Luc Dugelay. “Face aging with conditional generative adversarial networks”. In: *2017 IEEE international conference on image processing (ICIP)*. IEEE. 2017, pp. 2089–2093.

[3] Leyre Artal-Villa, Ahmed Hussein, and Cristina Olaverri-Monreal. “Extension of the 3DCoAutoSim to Simulate Vehicle and Pedestrian Interaction based on SUMO and Unity 3D”. In: *2019 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2019, pp. 885–890.

[4] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. “Yolov4: Optimal speed and accuracy of object detection”. In: *arXiv preprint arXiv:2004.10934* (2020).

[5] Christopher Bowles et al. “Gan augmentation: Augmenting training data using generative adversarial networks”. In: *arXiv preprint arXiv:1810.10863* (2018).

[6] Zhaowei Cai and Nuno Vasconcelos. “Cascade r-cnn: Delving into high quality object detection”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 6154–6162.

[7] Liang-Chieh Chen et al. “Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs”. In: *IEEE transactions on pattern analysis and machine intelligence* 40.4 (2017), pp. 834–848.

[8] Luca Ciampi et al. “Virtual to real adaptation of pedestrian detectors”. In: *sensors* 20.18 (2020), p. 5250.

[9] Jifeng Dai et al. “R-fcn: Object detection via region-based fully convolutional networks”. In: *Advances in neural information processing systems* 29 (2016).

[10] Christine Dewi et al. “Synthetic Data generation using DCGAN for improved traffic sign recognition”. In: *Neural Computing and Applications* (2021), pp. 1–16.

[11] Kaiwen Duan et al. “Centernet: Keypoint triplets for object detection”. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 6569–6578.

[12] Mark Everingham et al. “The pascal visual object classes (voc) challenge”. In: *International journal of computer vision* 88.2 (2010), pp. 303–338.

[13] Matteo Fabbri et al. “Motsynth: How can synthetic data help pedestrian detection and tracking?” In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 10849–10859.

[14] Wei Fang et al. “A method for improving CNN-based image recognition using DCGAN”. In: *Computers, Materials and Continua* 57.1 (2018), pp. 167–178.

[15] Maayan Frid-Adar et al. “GAN-based synthetic medical image augmentation for increased CNN performance in liver lesion classification”. In: *Neurocomputing* 321 (2018), pp. 321–331.

[16] Ross Girshick. “Fast r-cnn”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1440–1448.

[17] Ian Goodfellow et al. “Generative adversarial nets”. In: *Advances in neural information processing systems* 27 (2014).

[18] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[19] Kaiming He et al. “Mask r-cnn”. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2961–2969.

[20] Kaiming He et al. “Spatial pyramid pooling in deep convolutional networks for visual recognition”. In: *IEEE transactions on pattern analysis and machine intelligence* 37.9 (2015), pp. 1904–1916.

[21] Jan Hosang, Rodrigo Benenson, and Bernt Schiele. “Learning non-maximum suppression”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 4507–4515.

[22] Yunzhong Hou, Liang Zheng, and Stephen Gould. “Multiview detection with feature perspective transformation”. In: *European Conference on Computer Vision*. Springer. 2020, pp. 1–18.

[23] Andrew G Howard et al. “Mobilenets: Efficient convolutional neural networks for mobile vision applications”. In: *arXiv preprint arXiv:1704.04861* (2017).

[24] Hou-Ning Hu et al. “Joint monocular 3D vehicle detection and tracking”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 5390–5399.

[25] Gao Huang et al. “Densely connected convolutional networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 4700–4708.

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

[27] Eung-Joo Lee et al. “Validation of object detection in UAV-based images using synthetic data”. In: *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications III*. Vol. 11746. SPIE. 2021, pp. 584–601.

[28] Tsung-Yi Lin et al. “Feature pyramid networks for object detection”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 2117–2125.

[29] Tsung-Yi Lin et al. “Focal loss for dense object detection”. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2980–2988.

[30] Tsung-Yi Lin et al. “Microsoft coco: Common objects in context”. In: *European conference on computer vision*. Springer. 2014, pp. 740–755.

[31] Shu Liu et al. “Path aggregation network for instance segmentation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 8759–8768.

[32] Songtao Liu, Di Huang, and Yunhong Wang. “Learning spatial fusion for single-shot object detection”. In: *arXiv preprint arXiv:1911.09516* (2019).

[33] Wei Liu et al. “Ssd: Single shot multibox detector”. In: *European conference on computer vision*. Springer. 2016, pp. 21–37.

[34] Rafael Padilla, Sergio L Netto, and Eduardo AB Da Silva. “A survey on performance metrics for object-detection algorithms”. In: *2020 international conference on systems, signals and image processing (IWSSIP)*. IEEE. 2020, pp. 237–242.

[35] Alec Radford, Luke Metz, and Soumith Chintala. “Unsupervised representation learning with deep convolutional generative adversarial networks”. In: *arXiv preprint arXiv:1511.06434* (2015).

[36] Joseph Redmon and Ali Farhadi. “YOLO9000: better, faster, stronger”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 7263–7271.

[37] Joseph Redmon and Ali Farhadi. “Yolov3: An incremental improvement”. In: *arXiv preprint arXiv:1804.02767* (2018).

[38] Joseph Redmon et al. “You only look once: Unified, real-time object detection”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.

[39] Shaoqing Ren et al. “Faster r-cnn: Towards real-time object detection with region proposal networks”. In: *Advances in neural information processing systems* 28 (2015).

[40] Stephan R Richter et al. “Playing for data: Ground truth from computer games”. In: *European conference on computer vision*. Springer. 2016, pp. 102–118.

[41] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* (2014).

[42] Mingxing Tan, Ruoming Pang, and Quoc V Le. “Efficientdet: Scalable and efficient object detection”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 10781–10790.

[43] Zhi Tian et al. “Fcos: Fully convolutional one-stage object detection”. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 9627–9636.

[44] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. “Scaled-yolov4: Scaling cross stage partial network”. In: *Proceedings of the IEEE/cvf conference on computer vision and pattern recognition*. 2021, pp. 13029–13038.

[45] Chien-Yao Wang et al. “CSPNet: A new backbone that can enhance learning capability of CNN”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*. 2020, pp. 390–391.

[46] Niannian Wang et al. “Automatic damage detection of historic masonry buildings based on mobile deep learning”. In: *Automation in Construction* 103 (2019), pp. 53–66.

[47] Saining Xie et al. “Aggregated residual transformations for deep neural networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1492–1500.

[48] Huijuan Zhang, Zongrun Huang, and Zhongwei Lv. “Medical image synthetic data augmentation using GAN”. In: *Proceedings of the 4th International Conference on Computer Science and Application Engineering*. 2020, pp. 1–6.

[49] Xingxuan Zhang et al. “Towards Domain Generalization in Object Detection”. In: *arXiv preprint arXiv:2203.14387* (2022).

[50] Qijie Zhao et al. “M2det: A single-shot object detector based on multi-level feature pyramid network”. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 33. 01. 2019, pp. 9259–9266.

[51] Zhong-Qiu Zhao et al. “Object detection with deep learning: A review”. In: *IEEE transactions on neural networks and learning systems* 30.11 (2019), pp. 3212–3232.

[52] Zhaohui Zheng et al. “Distance-IoU loss: Faster and better learning for bounding box regression”. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 34. 07. 2020, pp. 12993–13000.