RAMAIAH
Institute of Technology

Drone Detection using Deep learning

Submitted in Partial fulfillment of Requirement for the award of the Degree of

BACHELOR OF ENGINEERING

in

Electronics and Communication

Sinchana S R	USN: 1MS17EC106
Soma Rohith	USN: 1MS17EC109
Sushmith Thuluva	USN: 1MS17EC119
Chiranthan K	USN: 1MS17EC135

Under the guidance of

Dr. Raghuram S

Associate Professor, Department of E & C

Department of Electronics and Communication

RAMAIAH INSTITUTE OF TECHNOLOGY

(Autonomous Institute, Affiliated to VTU)
Accredited by National Board of Accreditation & NAAC with 'A+' Grade
MSR Nagar, MSRIT Post, Bangalore-560054
www.msrit.edu

2021

RAMAIAH
Institute of Technology

CERTIFICATE

This is to certify that the dessertation work entitled **"Drone Detection using deep learning"** is carried out by **Sinchana S R, (1MS17EC106), Soma Rohith, (1MS17EC109), Sushmith Thuluva, (1MS17EC119), and Chiranthan K, (1MS17EC135)**, bonafide students of Ramaiah Institute of Technology, Bangalore, in partial fulfillment for the award of Bachelor of Engineering in **Electronics and Communication** of the Visvesvaraya Technological University, Belgaum,during the year 2020 -2021. It is certified that all corrections / suggestions indicated for Internal Assessment have been incorporated in the thesis. The thesis has been approved as it satisfies the academic requirements in respect to dessertation work prescribed for the said degree.

Guide	**HOD**	**Principal**
Dr. Raghuram S	**Dr. Maya V Karki**	**Dr. N V R Naidu**
Associate Professor	Professor and HOD,	Principal,
Department of E & C	Department of E & C	RIT, Bangalore
RIT, Bangalore	RIT, Bangalore	

Name & Signature of Examiners with Date: -

1)

2)

RAMAIAH
Institute of Technology

DECLARATION

We hereby declare that the Project entitled "Drone Detection using deep learning" has been carried out independently at Ramaiah Institute of Technology under the guidance of **Dr. Raghuram S, Associate Professor, Department of Electronics and Communication, RIT, Bangalore.**

We hereby declare that work submitted in this thesis is our own, except where acknowledged in the text and has not been previously submitted for the award of the degree of Visvesvaraya Technological University, Belgaum or any other institute or University

Signature of Students:

1. Sinchana S R (USN: 1MS17EC106)
2. Soma Rohith (USN:1MS17EC109)
3. Sushmith Thuluva (USN:1MS17EC119)
4. Chiranthan K (USN:1MS17EC135)

Place:

Date:

RAMAIAH
Institute of Technology

ACKNOWLEDGEMENT

The immense satisfaction that accompanies the successful completion of the project would be incomplete without the mention of the people who made it possible. I consider it my honor to express my deepest gratitude and respect to the following people who always guided and inspired me during the course of the Project.

I am deeply indebted to **Dr. N. V. R. Naidu**, Principal, RIT, Bangalore for providing me with a rejuvenating master course under a very creative learning environment.

I am much obliged to **Dr. Maya V Karki**, Professor &HOD, Department of Electronics and Communication Engineering, RIT, Bangalore for her constant support and motivation.

I sincerely thank my guide **Dr. Raghuram S, Associate Professor,** Department of Electronics and Communication Engineering, RIT, Bangalore and express my humble gratitude for **his** valuable guidance, inspiration, encouragement and immense help which made this project work a success.

I sincerely thank **Dr. S Sethu Selvi** and **Sara Mohan George** for reviewing my project work and providing valuable suggestions. I also thank all the faculty members of Department of E&C, RIT for their kind support to carry out this project successfully.

Last but not the least I would like to express my heartfelt gratitude to my parents, relatives and friends for their constant support, motivation and encouragement.

RAMAIAH
Institute of Technology

ABSTRACT

There has been a rapid increase in the usage of drones in the last few years, raising security concerns as these devices can invade sensitive spaces through the air. Computer vision is extensively used compared to other available solutions due to the small footprint of drones. Among these computer vision-based approaches, deep learning algorithms have proved maximally effective. This project deals with the design and development of a Deep Learning solution for drone detection, along with implementation on an embedded system. Various implementations are tested such as the Yolo object detection using OpenCV which distinguished birds and drones, Masked RCNN which places bounding boxes, and Object detection using Image Library which detects many drones individually. A 10 layered CNN along with Mobilenet V2 has been implemented with an accuracy of 0.9994, Validation accuracy – 0.9929. The precision, recall, f1-score for Bird and Drone being (1.00,0.99,0..99) and (0.99,1.00,0.99) respectively. These models were tested on the Jetson Nano hardware module. The real-time object detection has been achieved by implementing SSD- MobileNet V2 model on Jetson Nano interfaced with a Raspberry Pi Camera module, with the dataset manually prepared by sourcing images from Google, Kaggle and Pinterest. Future scope includes faster detection, increased range, differentiation between various kinds of drones, and counter-drone technologies.

RAMAIAH
Institute of Technology

ACRONYMS

NASA	National Aeronautics and Space Administration
CNN	Convolutional Neural Networks
R-CNN	Region Based Convolutional Neural Networks
CCTV	Closed-Circuit Television
UAV	Unmanned Aerial Vehicle
LIDAR	Light Detection and Ranging
UPS	Uninterrupted Power Supply
AUV	Autonomous Underwater Vehicle
FBI	Federal Bureau of Investigation
AI	Artificial Intelligence
RF	Radio Frequency
MAC	Media Access Control Address
YOLO	You Only Look Once
TFCS	TensorFlow Compression
DCN	Dynamic Circuit Network
COCO	Common Objects in Context

RAMAIAH
Institute of Technology

Contents

RAMAIAH
Institute of Technology

List of figures

RAMAIAH
Institute of Technology

List of tables

CHAPTER 1

INTRODUCTION

1.1. DRONES

In aviation and in space, a drone refers to an unpiloted aircraft or spacecraft. Another term for it is an "unmanned aerial vehicle," or UAV. Originally developed for the military and aerospace industries, drones have found their way into the mainstream because of the enhanced levels of safety and efficiency they bring. These robotic UAVs operate without a pilot on board and with different levels of autonomy. A drone's autonomy level can range from remotely piloted (a human controls its movements) to advanced autonomy, which means that it relies on a system of sensors and LIDAR detectors to calculate its movement. In addition, drones don't require rest, enabling them to fly as long as there is fuel in the craft and there are no mechanical difficulties.

Fig 1.1. Various kinds of drones

Different drones are capable of traveling varying heights and distances. Very close-range drones usually have the ability to travel up to three miles and are mostly used by hobbyists. Close-range UAVs have a range of around 30 miles. Short-range drones travel up to 90 miles and are used primarily for espionage and intelligence gathering. Mid-range UAVs have a 400-mile distance range and could be used for intelligence gathering, scientific studies and meteorological research. The longest-range drones are called "endurance" UAVs and have the ability to go beyond the 400-mile range and up to 3,000 feet in the air.

Because drones can be controlled remotely and can be flown at varying distances and heights, they make perfect candidates to take on some of the toughest jobs in the world. They can be found assisting in a search for survivors after a hurricane, giving law enforcement and military an eye-in-the-sky during terrorist situations and advancing scientific research in some of the most extreme climates on the planet. Drones have even made their way into our homes and serve as entertainment for hobbyists and a vital tool for photographers.

1.2. APPLICATIONS OF DRONES

1.2.1 MILITARY

Probably the oldest, most well-known and controversial use of drones is in the military. The British and U.S. militaries started using very basic forms of drones in the early 1940's to spy on the Axis powers. Today's drones are much more advanced than the UAVs of yesteryear, equipped with thermal imaging, laser range finders and even tools to perform airstrikes. The most prominent military drone in use today is the MQ-9 Reaper. The aircraft measures 36 feet long, can fly 50,000 feet in the air undetected and is equipped with a combination of missiles and intelligence gathering tools.

1.2.2 DELIVERY

Delivery drones are usually autonomous UAVs that are used to transport food, packages or goods to your front doorstep. These flying vehicles are known as "last mile" delivery drones because they are used to make deliveries from stores or warehouses close by. Retailers and grocery chains all over the country are turning to drones as more efficient delivery alternative, instead of relying on delivery drivers with inefficient trucks. These drones can carry an impressive 55 pounds of goods to your front door without you ever having to leave the house. Amazon, Walmart, Google, FedEx, UPS and many other big brands are all currently testing out different versions of delivery drones.

1.2.3 EMERGENCY RESCUE

An Autonomous Underwater Vehicle (AUV) into the water to assist in the rescue. If there's an avalanche, drones are deployed to look for those caught in the snow. Aircraft maker, Kaman, has even developed a pilotless helicopter, called the K-MAX, designed to carry more than 6,000 pounds of cargo. The K-MAX has already been used in China and Australia to assist in fighting fires.

1.2.4 OUTER SPACE

NASA and the U.S. Air Force have been secretly testing out unmanned aircraft geared towards space travel. The X-37B UAV is the Air Force's ultra-secretive drone that looks like a miniature space shuttle. It has been quietly circling the Earth for the last two years, setting a record for longest flight from an unmanned aircraft (more than 719 days). Although vague, the Air Force has said "the primary objectives to the X-37B are twofold: reusable spacecraft technologies for America's future in space and operating experiments which can be returned to, and examined, on Earth." It seems

those drones have been made a priority when it comes to the future of space exploration and innovation.

1.2.5 WILDLIFE AND HISTORICAL CONSERVATION

Drones are a cheaper and more efficient alternative to wildlife conservation. Tracking wildlife populations is nearly impossible with humans on the ground. Having an eye-in-the-sky allows wildlife conservationists to track roaming groups of animals, ranging from Orangutans in Borneo to Bison on the Great Plains, to get a better idea of the health of their species and ecosystems. Conservation drones also make perfect tools in the fight against poaching efforts in Asia and Africa.

Drones are also being used for reforestation efforts all over the world. These drones scour the forest floors of forests decimated by fires and drop seed vessels filled with seeds, fertilizers and nutrients that will help a tree rise from the ashes. There have been around 300 million acres of deforested land since the early 1990's. What would take humans around 300 years to reforest can be more efficiently completed via seed-planting drone technology.

Finally, UAVs are becoming instrumental in historical conservation efforts. Drones are being used to map out 3D renderings of historical sites like Chernobyl, the ancient Greek sites of Ephesus, Turkey and Jewish cemeteries all over Europe. The vantage point gives historical preservationists the ability to find clues about culture and architecture, while using 3D imagery to recreate lost sites.

1.2.6 MEDICINE

Drones are also being tapped to deliver donated organs to transplant patients. Just recently, history was made when a kidney was transported by a specially-made drone from one hospital in Maryland to the next in just under five minutes. This could cut down on the alarmingly slow rate at which donations usually arrive (if they arrive at all). Usually, organs are delivered via chartered or commercial flights. Delays and lapses in judgement cause dangerous delays of two hours or more for 4% of all organ deliveries. Drones can cut the time down tremendously, while offering a safer and secure method of organ transportation.

1.2.7 PHOTOGRAPHY

Drones have been a boon for photographers, who use the UAVs to take expansive aerial photos. Ever wonder what it's like to get a bird's eye view of your favorite city, beach or building? There are drones made specifically for photography that provide a new way to photograph some of your favorite destinations from above.

Fig 1.2. Applications of drones

1.3. NEED FOR DRONE DETECTION

Drones pose a major threat to real world, if misused for the wrong reasons. The main issue, privacy of people can be compromised. The survey conducted by Pew Research Center gave us clarification on the same. More than half of Americans agree on the fact that drones should not be allowed near people's homes.

Firstly, the pictures that are captured by drones are transmitted back to the cloud. But not every transmission might be a secure one, as we know that hackers are always waiting to intercept and steal data. To add to the list, hackers or adversaries can use drones to watch people and collect their data too. To do so, criminals can fit cameras on drones and carry out their illicit activities. Another ugly truth about drones are the attacks, the crashes, and the disasters. FBI Chief, Christopher A. Wray, has burning reviews on drones. According to him, terrorists will use drones to attack the U.S. There have been instances of arming the drones and using them against the people and security forces.

Other horrible possibilities include drones crashing onto an airplane, tangling with a tree, or simply going missing. Even though high-quality obstacle avoidance sensors are embedded on drones, there are always risks around sensors not working correctly. A lot of drone mishaps have already occurred, giving a clear indication that drones have a horrible side too.

Whatever may be the dilemma around them, drones were invented with the right intention. Flaws are common with every technological invention. The success of technologies depends on how well the government, organizations, and commoners accept and tackle these flaws. And the case is similar for drones. Identifying the problems, drone-makers should find appropriate solutions and produce robust drones. Even when drones are

improved with highly advanced technologies like AI and components like sensors and cameras, they should be regularly checked for technical glitches. This way, instances of drone crashing and drone missing will be reduced to a great extent.

1.4 SOLUTION

1.4.1 DRONE MONITORING EQUIPMENT

Drone monitoring equipment can be passive (simply looking or listening) or active (sending a signal out and analyzing what comes back) and can perform several functions, including: Detection, Classification or Identification, Locating and Tracking & Alerting. Detection means the technology is able to detect drones. Detection alone usually isn't enough though. A radar that detects drones may also detect birds, for example.

That's why classification is useful. Technology that classifies drones will usually be able to separate drones from other types of objects - like planes, trains, and automobiles, for example.

Being alerted that a drone is present somewhere in the vicinity is already useful. But your situational awareness, and ability to deploy countermeasures is greatly enhanced if you know the drone's (and/or the controller's) exact location. Some equipment will even allow you to track the drone location in real-time.

There are four main types of drone monitoring equipment:

1) Radio Frequency (RF) Analyzers

2) Acoustic Sensors (Microphones)

3) Optical Sensors (Cameras)

4) Radar

1.4.1.1 RADIO FREQUENCY (RF) ANALYZERS

Counter-Drone Radio Frequency (Rf) Analysers

RF Analyzers consist of one or more antennas to receive radio waves and a processor to analyze the RF spectrum. They're used to try to detect radio communication between a drone and its controller.

Some systems are able to identify the more common drone makes and models, and some can even identify the MAC addresses of the drone and controller (if the drone uses Wi-Fi for communication). This is especially useful for prosecution purposes – proving that a particular drone and controller were active.

Some high-end systems can also triangulate the drone and its controller when using multiple radio units spread far apart.

Pros: Can be low cost, detects (and sometimes identifies) multiple drones and controllers, passive so no license required, some can triangulate drone and controller position.

Cons: Doesn't always locate and track drones, can't detect autonomous drones, less effective in crowded RF areas, typically short range.

Fig 1.3. RF Analyzing Equipment

1.4.1.2 ACOUSTIC SENSORS (MICROPHONES)

Counter-Drone Acoustic Sensors (Microphones)

Usually, a microphone, or microphone array (lots of microphones), which detects the sound made by a drone and calculates a direction. More sets of microphone arrays can be used for rough triangulation.

Pros: Detects all drones within the near-field, including those operating autonomously (without (RF-emissions). Detects drones in the ground clutter where other technologies can struggle. Great gap-filler in areas outside line-of-sight of other sensors. Highly mobile and quickly deployable. Completely passive.

Cons: Doesn't work as well in noisy environments, very short range (max. 300-500m)

Fig 1.4. Audio Surveillance module

1.4.1.3 OPTICAL SENSORS (CAMERAS)

Counter Drone Optical Sensors (Cameras)

Essentially a video camera. As well as standard daylight cameras, optical sensors can be infrared or thermal imaging.

Pros: Provides visuals on the drone and it's (potential) payload, can record images as forensic evidence for use in eventual prosecution.

Cons: Difficult to use for detection by itself, high false-alarm rates, mostly poor performance in dark, fog, etc.

Fig 1.5. Optical camera system (CCTV)

1.4.1.4 RADAR

Counter Drone Radar

A device using radio energy to detect an object. Drone detection radar sends out a signal and receives the reflection, measuring direction and distance (position). Most radars send their radio signal as a burst, then listen for the 'echo'. Almost all radars are designed to NOT pick up small targets. They are designed for large object tracking, like passenger aircraft.

Pros: Long range, constant tracking, highly accurate localisation, can handle hundreds of targets simultaneously, can track all drones regardless of autonomous flight, independent of visual conditions (day, night, fog, etc.)

Cons: Detection range dependent on drone size, most do not distinguish birds from drones, requires transmission license and frequency check to prevent interference.

Doppler radars are able to track moving objects and discard static objects so you won't see them on your screen. Nothing new there.

Micro-doppler radar, however, is able to detect movement - specifically, speed differences - within moving objects. And drones tend to have propellers which create a large spectrum of speed differences. Part of the propeller is moving towards you and part is moving away.

By using this technique our radar can identify drones easily, and, most importantly, distinguish drones from birds.

Fig 1.6. Military grade Radar

1.5 DEEP LEARNING FOR DETECTION

Of all the mentioned above, Object detection is most effective for the detection purpose. Since, it is able to detect and classify from many other objects similar to drones. Image classification involves assigning a class label to an image, whereas object localization involves drawing a bounding box around one or more objects in an image. Object detection is more challenging and combines these two tasks and draws a bounding box around each object of interest in the image and assigns them a class label. Together, all of these problems are referred to as object recognition. Object recognition is referring to a collection of related tasks for identifying objects in digital photographs.

Region-Based Convolutional Neural Networks, or R-CNNs, are a family of techniques for addressing object localization and recognition tasks, designed for model performance. You Only Look Once, or YOLO, is a second family of techniques for object recognition designed for speed and real-time use.

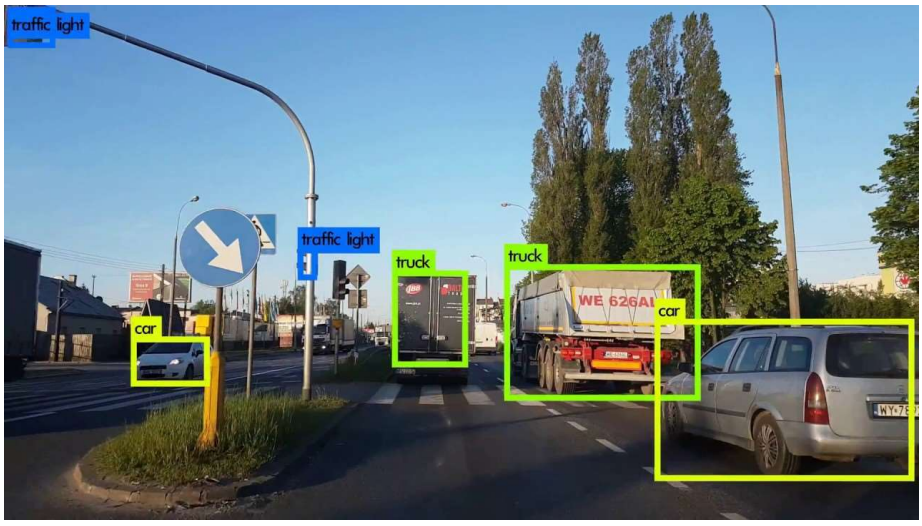

Fig 1.7. YOLO Object detection

CHAPTER 2

LITERATURE SURVEY

2.1. VIDEO OBJECT DETECTION BASED ON THE SPATIAL TEMPORAL CONVOLUTION FEATURE MEMORY MODEL

2.1.1 ABSTRACT

Video object detection technology can improve the battlefield object search capability of tank fire control system. However, complex battlefield environment and faster speeds of tank and objects bring great challenge to video object detection. A video object detection method is proposed in this paper for the tank fire control system. Given the rich spatial temporal information in the video and the large position deviation of the target in the adjacent video frames, a spatial temporal convolutional feature memory model consisted of spatial-temporal convolution feature alignment mechanism and convolution gated recurrent unit is proposed to transmit and fuse the information of adjacent frames. Moreover, the feature extraction network and the detection sub-network are improved by the deformable convolution networks to increase the detection accuracy of deformed objects. To evaluate the proposed method, a database named TFCS VID including 1396 videos labelled for seven types of typical objects in the battlefield was developed. Compared to several other video object detection methods, the proposed method achieved excellent detection results on TFCS VID and could better meet the actual application requirements of equipment

2.1.2 INTRODUCTION

Object detection is a fundamental problem in computer vision. In recent years, due to the application of convolutional neural networks (CNNs), great achievements have been made in the object detection in static images. However, object detection in video remains a challenge. Video frames are usually affected by motion blur, diffraction blur or defocus, which bring great challenges to object detection in videos. Moreover, video object detectors could accurately detect the object in each video frame and ensure the timing consistency of detection results. Despite these difficulties, if the abundant spatial-temporal object information contained in video frames can be fully utilized, object detection in videos can achieve better detection results than object detection in static images.

2.1.3 ARCHITECTURE

Overall Framework The overall framework of the proposed detector is shown in Figure 1. The whole network structure is composed of three parts: (i) feature extraction network, (ii) STCFMM and (iii) detection sub-network. We adopt Resnet-101 improved by DCN as the feature extraction network and we pre-train it in ImageNet VID to extract the convolutional feature of frames. STCFMM is proposed to transfer and aggregate the convolutional feature of multiple adjacent frames and generate the spatial-temporal convolutional feature map. The detection sub-network is applied in the feature map to detect objects. The detection sub-network of our method is similar to that of Light Head R-CNN, but the Position Sensitive Region of Interest pooling (PS ROI pooling) of the detection subnetwork is improved by DCN.

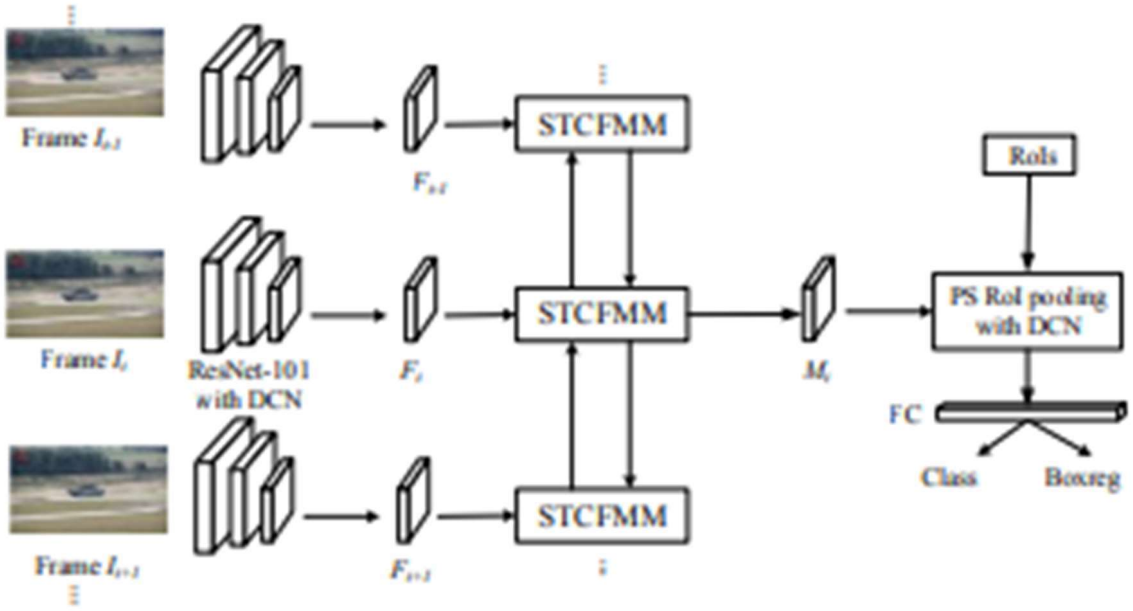

Fig 2.1. Framework of network

2.1.4 SPATIAL-TEMPORAL CONVOLUTIONAL FEATURE MEMORY MODEL

STCFMM is composed of ConvGRU and spatial-temporal convolutional feature alignment mechanism. ConvGRU can transmit the information between video frames, but the fast moving leads to significant variations in the spatial position of the object in the frame. If the spatial-temporal convolution feature is not aligned, the position in the fused spatial temporal convolution feature may be offset or even lost, thus resulting in wrong destination and reducing the object detection accuracy

2.1.5 CONCLUSION

According to the application requirements of TFCS, we analysed the main detection objects of tanks, established a battlefield video object detection dataset namely TFCS VID, and proposed a video object detection method via STCFMM. STCFMM is

composed of ConvGRU and spatial-temporal convolution feature alignment mechanism and can effectively transfer and

aggregate the convolutional features in the multi frames to obtain the spatial-temporal convolution feature of the current frame. An end-to-end video object detection model for TFCS video object detection task constructed by feature extraction network, detection sub-network and STCFMM realized the accurate detection of battlefield video object. In addition, in order to improve the detection effect of the video object detection model on the deformed object, DCN was also combined with the feature extraction network and the detection sub-network. Compared with other video object detection methods, our method achieved the excellent performance on TFCS VID and could meet the application requirements of tanks

Object types	Light Head R-CNN	FGFA	D&T	MANet	Ours
Tank	71.2	74.2	72.7	76.9	78.6
IFV	71.8	78.5	75.6	79.1	79.5
artillery	58.7	65.2	62.3	67.2	69.7
car	75.2	81.2	78.6	83.0	83.1
person	77.9	79.8	77.9	81.6	80.5
helicopter	73.8	77.6	78.3	79.1	81.2
UAV	62.9	63.8	60.8	68.7	67.4
mAP/%	70.2	74.3	72.2	76.5	77.1

Table 2.1. Comparison of various Network results

2.2 USING DEEP NETWORKS FOR DRONE DETECTION

2.2.1 ABSTRACT

Drone detection is the problem of finding the smallest rectangle that encloses the drone(s) in a video sequence. In this study, we propose a solution using an end-to-end object detection model based on convolutional neural networks. To solve the scarce data problem for training the network, we propose an algorithm for creating an extensive artificial dataset by combining background-subtracted real images. With this approach, we can achieve precision and recall values both of which are high at the same time

2.2.2 INTRODUCTION

In this study we have used an end-to-end object detection method based on CNNs to predict the location of the drone in the video frames. In order to be able to train the network, we created an artificial dataset by combining real drone and bird images with different background videos. The results show that the variance and the scale of the dataset make it possible to perform well on drone detection problem. With this method, we have participated in the Drone-vs-Bird Detection Challenge1 organized within the

International Workshop on Small-drone Surveillance, Detection and Counter action Techniques, and our trained network ranked third in terms of lowest prediction penalty

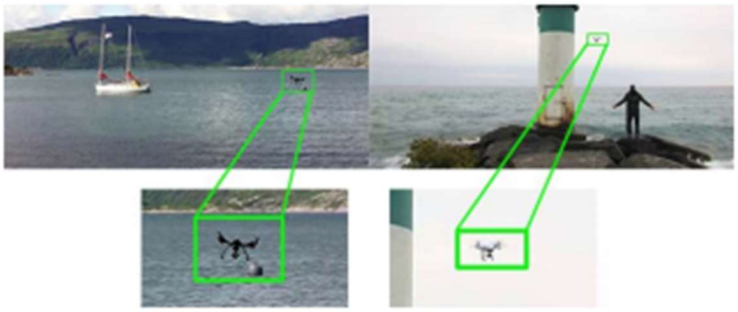

Fig 2.2. Sample Dataset Image

2.2.3 METHODOLOGY

Solution is based on a single shot object detection model, YOLOv2, which is the follow-up study of YOLO. We adapt and fine-tune this model to detect objects of two classes (i.e., drone and bird). Although the problem is detecting drones in the scene, we have included the bird class so that the network can learn robust features to distinguish them too. In order to achieve high accuracy with such deep models, one needs a large scale dataset that includes many scenarios of the problem, to get better generalization. To this end, we created an artificial dataset including real drones, real birds and real backgrounds.

2.2.4 DETECTION USING COMPUTER VISION

Although the problem of detecting UAVs is not a well studied subject, there are some attempts to mention. Mejias et al. utilized morphological pre-processing and Hidden Markov Model filters to detect and track micro unmanned planes . Gokce et al. used cascaded boosted classifiers along with some local feature descriptors.In addition to this pure spatial information based methods, spatiotemporal approaches exist. Rozantsev et al. propose a method that first creates spatio-temporal cubes using sliding window method at different scales, applies motion compensation to stabilize spatio-temporal cubes, and finally utilizes boosted tree and CNN.

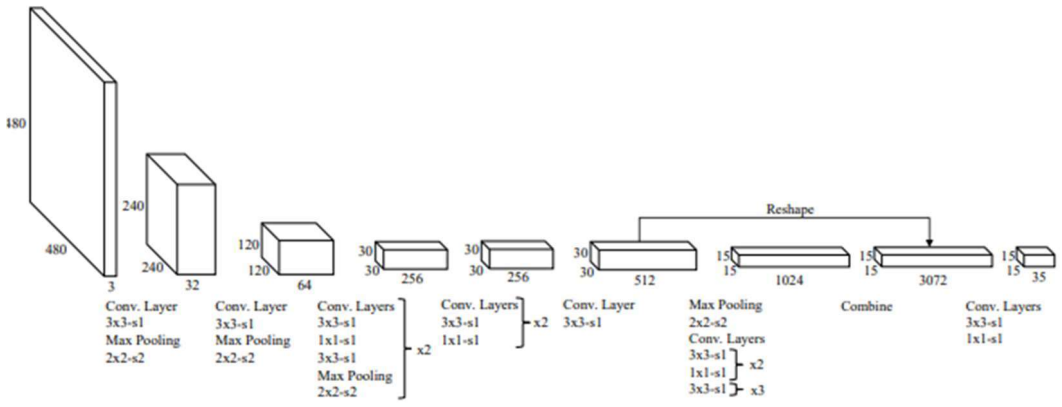

Fig 2.2. Layers fine-tuned according to dataset

2.2.5 RESULTS

Figure 2.3 presents the performance of the method with different detection thresholds in the range [0,1]. The closer the Precision-Recall (PR) curve to the top right corner the better the performance of the method. We can understand from the curve that precision and recall can be achieved to be approximately 0.9 at the same time. This shows that the approach performs well in detecting the correct bounding boxes. Figure 2.4 shows the change of the average penalty with respect to detection threshold. The reason for higher penalties is that when the threshold increases detection rate decreases. When a drone cannot be found, the top-left pixel is reported as prediction, which results in a huge penalty. Hence, we have chosen the smallest possible threshold (which is zero) for quantitative evaluation on the test video of the challenge. Although this threshold hurts precision in the artificial dataset, it works well in the provided test video except its detecting the bird as drone when the bird is closer to the camera and in specific poses that cannot be easily distinguished from a drone by human eye. Another observation is that when the drone and the bird are too close to each other, the network supposes that the bird is a part of the drone, and outputs a bounding box enclosing both of them. The predictions are provided online2 as a video rendered in 15 fps.

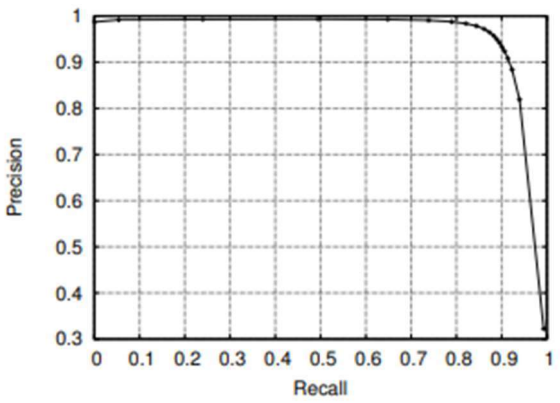

Fig 2.3. Precision-Recall Curve

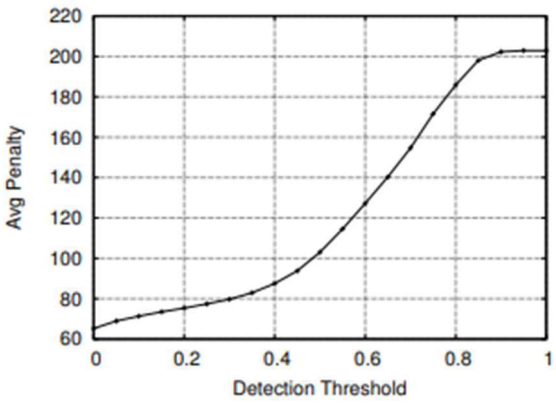

Fig 2.4.Prediction Penalty Curve

2.2.6 CONCLUSION

In this study, we showed that drones can be detected and distinguished from birds using an object detection model based on a CNN. The trained network generalizes well as it can achieve high precision and recall values at the same time. For future work we plan to consider time domain to improve the performance even further. Since collecting such data is not easy, we plan to devise an algorithm that generates random flight videos instead of randomly generated images.

CHAPTER 3

METHODOLOGY

The shortcomings of manual capabilities and time constraints in detecting objects as drones from footages or images have inspired the intentions of our project to develop a simple computer algorithm that could locate the drones in a matter of milliseconds embodying the power of object detection algorithms. Its applications span multiple and diverse industries, from round-the-clock surveillance to real-time vehicle detection in smart cities. In short, these are powerful deep learning algorithms.

3.1 HARDWARE MODEL FOR EDGE-BASED DETECTION:

Edge detection includes a variety of mathematical methods that aim at identifying points in a digital image at which the image brightness changes sharply or, more formally, has discontinuities. The points at which image brightness changes sharply are typically organized into a set of curved line segments termed edges. Edge detection is a fundamental tool in image processing, machine vision, and computer vision, particularly in the areas of feature detection. This algorithm when implemented with suitable hardware equipment can give us efficient results for drone detection.

The real-time implementation of drone detection and classification can be done by implementing an object detector on a single board computer. Here the object detector system will consist of a Raspberry Pi camera interfaced with Raspberry Pi 3 B+, which is capable of maintaining real-time frame rate keeping high precision. The Pi camera module is capable of taking both high-definition videos as well as still photographs. Raspberry Pi is a small single-board computer capable of doing everything along with a great ability to interact with the outside world for real-time application of many projects. Among the different models, we would be using Raspberry Pi 3 B+ for the real-time detection of drones. This is the most recent version of Pi boards with CPU, GPU, USB ports, I/O pins, WiFi, Bluetooth, USB, and network boot onboard.

3.1.1 RASPBERRY PI

A Raspberry Pi camera module interfaced with this Pi 3 B+ board will help in obtaining video of the area under surveillance. The captured video has to be processed by separating different frames of it and by eliminating the background, these processed frames will be given to the deep learning model to identify the presence or absence of drones among each.

Due to some installations, over heating issues. We planned on to use jetson nano processor the working.

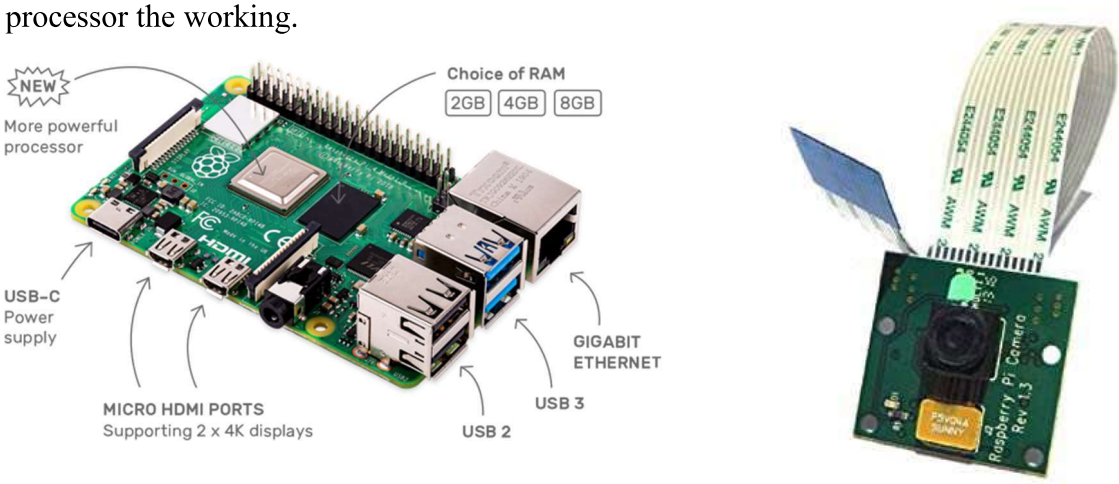

Fig 3.1.1 Raspberry Pi Module & Camera module

5MP Raspberry Pi camera module V2 is a Camera Serial Interface (CSI) camera was employed in the project for real time detection.

3.1.2 JETSON NANO

Jetson Nano delivers 472 GFLOPS of compute performance with a quad-core 64-bit ARM CPU and a 128-core integrated NVIDIA GPU. It also includes 4GB LPDDR4 memory in an efficient, low-power package with 5W/10W power modes and 5V DC input

The newly released JetPack 4.2 SDK provides a complete desktop Linux environment for Jetson Nano based on Ubuntu 18.04 with accelerated graphics, support for NVIDIA CUDA Toolkit 10.0, and libraries such as cuDNN 7.3 and TensorRT 5. The SDK also includes the ability to natively install popular open-source Machine Learning (ML) frameworks such as TensorFlow, PyTorch, Caffe, Keras, and MXNet, along with frameworks for computer vision and robotics development like OpenCV and ROS.

Full compatibility with these frameworks and NVIDIA's leading AI platform makes it easier than ever to deploy AI-based inference workloads to Jetson. Jetson Nano brings real-time computer vision and inferencing across a wide variety of complex Deep Neural Network (DNN) models. These capabilities enable multi-sensor autonomous robots, IoT devices with intelligent edge analytics, and advanced AI systems. Even transfer learning is possible for re-training networks locally onboard Jetson Nano using the ML frameworks.

The Jetson Nano Developer Kit fits in a footprint of just 80x100mm and features four high-speed USB 3.0 ports, MIPI CSI-2 camera connector, HDMI 2.0 and DisplayPort 1.3, Gigabit Ethernet, M.2 Key-E module, MicroSD card slot, and 40-pin GPIO header. The ports and GPIO header work out-of-the-box with a variety of popular peripherals, sensors, and ready-to-use projects, such as the 3D-printable deep learning JetBot that NVIDIA has open-sourced on GitHub. The devkit boots from a removable MicroSD card which can be formatted and imaged from any PC with an SD card adapter. The devkit can be conveniently powered via either the Micro USB port or a 5V DC barrel jack adapter. The camera connector is

compatible with affordable MIPI CSI sensors including modules based on the 8MP IMX219, available from Jetson ecosystem partners. Also supported is the Raspberry Pi Camera Module v2, which includes driver support in JetPack.

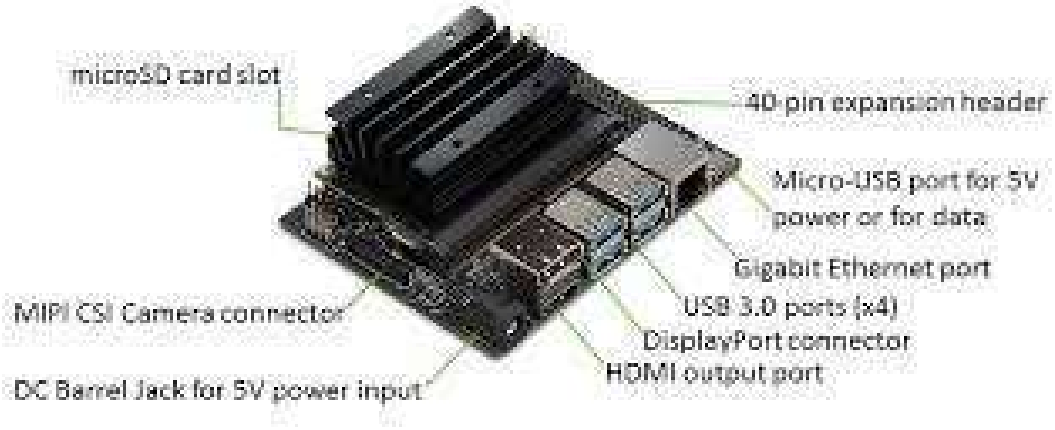

Fig 3.2. Jetson Nano Module

3.2 Dataset

Two datasets have been utilized in the implementation of the project. The dataset1 consists of birds and drones taken from real-time video, which has been split into frames to produce the images required. The drawback of this dataset was the objects in the images were too small and similar to each other. The detection results were not effective enough. So, new dataset has been created for further implementation.

Dataset2 consists of various drones and bird images separated into Train and test folders for classification. The dataset has been prepared from collecting images from various sources such as Google, Kaggle, Pinterest.

It is split into 80-20 ratio for train and test respectively.

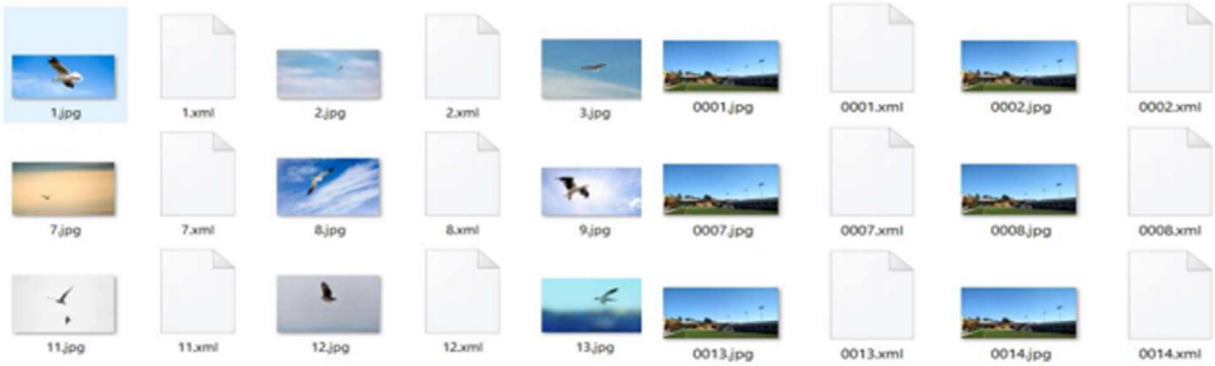

Fig 3.3. Old Dataset

Fig 3.4. New Dataset

CHAPTER 4

RESULTS AND DISCUSSION

4.1. YOLO OBJECT DETECTION USING OPENCV

YOLO (You Only Look Once) is a method / way to do object detection. It is the algorithm /strategy behind how the code is going to detect objects in the image.

The official implementation of this idea is available through DarkNet (neural net implementation from the ground up in C from the author). It is available on GitHub for people to use. Earlier detection frameworks, looked at different parts of the image multiple times at different scales and repurposed image classification technique to detect objects. This approach is slow and inefficient.

YOLO takes entirely different approach. It looks at the entire image only once and goes through the network once and detects objects. Hence the name. It is very fast. That's the reason it has got so popular. There are other popular object detection frameworks like Faster R-CNN and SSD that are also widely used.

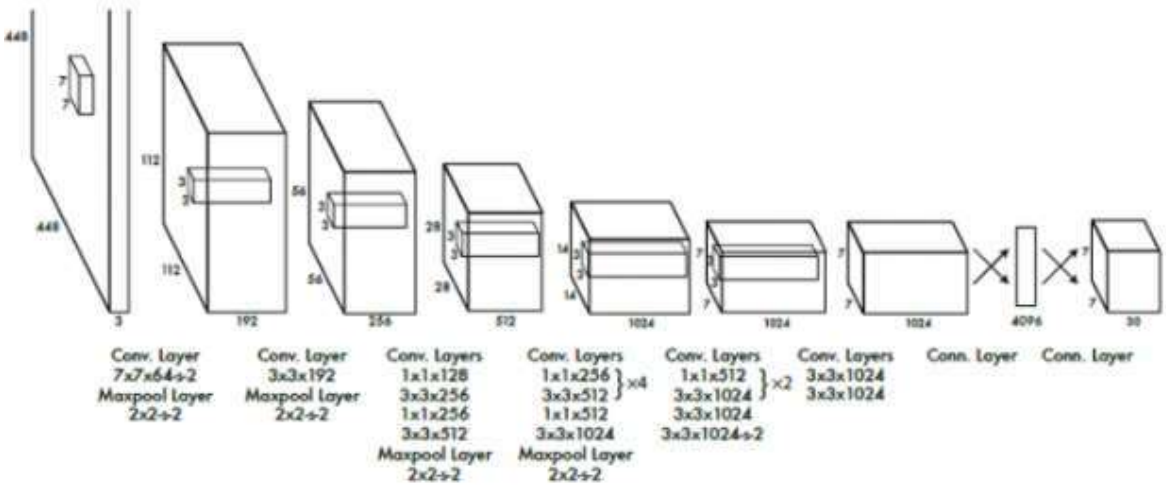

Fig 4.1.CNN Object Detection Architecture

There are various implementations of YOLO algorithm and perhaps most popular of them is the Darknet. But here we are going to use OpenCV to implement YOLO algorithm as it is really simple. To get started you need to install OpenCV on your Pc using this command in you command prompt.

pip install OpenCV-python

To use YOLO via OpenCV, we need three files viz -'yoloV3.weights', 'yoloV3.cfg' and "coco.names" (contain all the names of the labels on which this model has been trained on). Click on them to download and then save the files in a single folder. Now open a python script in this folder and start coding:

First, we are going to load the model using the function "cv2.dnn.ReadNet()". This function loads the network into memory and automatically detects configuration and framework based on file name specified.

```
import cv2

import numpy as np

# Load Yolo

print("LOADING YOLO")

net = cv2.dnn.readNet("yolov3.weights", "yolov31.cfg")

#save all the names in file o the list classes

classes = []

with open("coco.names", "r") as f:

classes = [line.strip() for line in f.readlines()]


#get layers of the network

layer_names = net.getLayerNames()

#Determine the output layer names from the YOLO model

output_layers = [layer_names[i[0] - 1] for i in net.getUnconnectedOutLayers()]

print("YOLO LOADED")
```

After loading the model now either we can use it detects objects in an image or you can even use it for real-time object detection for which you are going to need a PC with good processing speed.

```
 # Capture frame-by-frame

    img = cv2.imread("test_img.jpg")

#     img = cv2.resize(img, None, fx=0.4, fy=0.4)

    height, width, channels = img.shape
```

```
    # USing blob function of opencv to preprocess image
    blob = cv2.dnn.blobFromImage(img, 1 / 255.0, (416, 416),
     swapRB=True, crop=False)
    #Detecting objects
    net.setInput(blob)
    outs = net.forward(output_layers)
# Showing informations on the screen
    class_ids = []

    confidences = []
    boxes = []
    for out in outs:
 for detection in out:
            scores = detection[5:]
            class_id = np.argmax(scores)
            confidence = scores[class_id]
            if confidence > 0.5:
                # Object detected
                center_x = int(detection[0] * width)
                center_y = int(detection[1] * height)
                w = int(detection[2] * width)
                h = int(detection[3] * height)
                # Rectangle coordinates
                x = int(center_x - w / 2)
                y = int(center_y - h / 2)
                boxes.append([x, y, w, h])
                confidences.append(float(confidence))
  class_ids.append(class_id)
        #We use NMS function in opencv to perform Non-maximum Suppression
```

```
    #we give it score threshold and nms threshold as arguments.
    indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)
    colors = np.random.uniform(0, 255, size=(len(classes), 3))
    for i in range(len(boxes)):
if i in indexes:
        x, y, w, h = boxes[i]
        label = str(classes[class_ids[i]])
        color = colors[class_ids[i]]
        cv2.rectangle(img, (x, y), (x + w, y + h), color, 2)
        cv2.putText(img, label, (x, y -5),cv2.FONT_HERSHEY_SIMPLEX,
                        1/2, color, 2)
    cv2.imshow("Image",img)
    cv2.waitKey(0)
```

Source code: https://pysource.com/2019/06/27/yolo-object-detection-using-opencv-with-python/

Fig 4.1.1 Results obtained

4.2 MASKED RCNN USING OPENCV

The Mask R-CNN algorithm was introduced by He et al. in their 2017 paper, *Mask R-CNN*.

Mask R-CNN builds on the previous object detection work of *R-CNN* (2013), *Fast R-CNN* (2015), and *Faster R-CNN* (2015), all by Girshick et al.

In order to understand Mask R-CNN let's briefly review the R-CNN variants, starting with the original R-CNN:

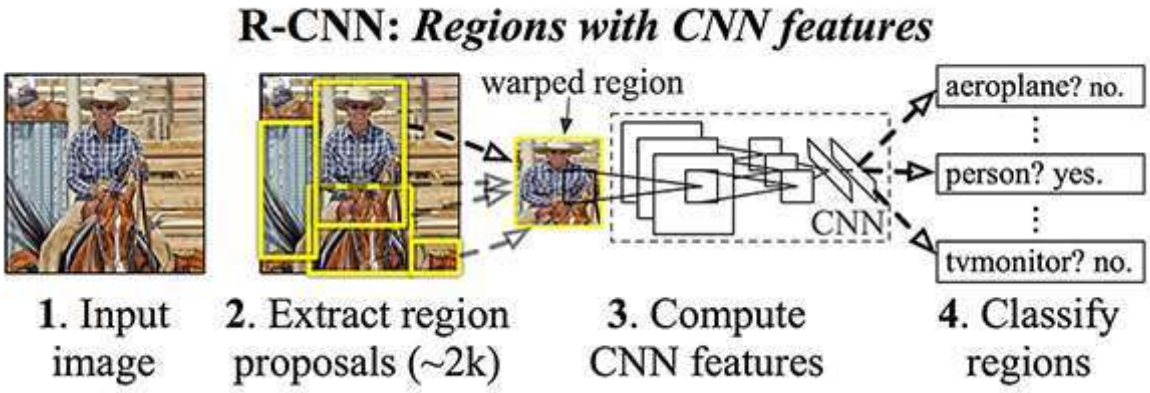

Fig 4.2.1 Masked-CNN Architecture

The original R-CNN architecture

The original R-CNN algorithm is a four-step process:

- Step #1: Input an image to the network.
- Step #2: Extract region proposals (i.e., regions of an image that potentially contain objects) using an algorithm such as Selective Search.
- Step #3: Use transfer learning, specifically feature extraction, to compute features for each proposal (which is effectively an ROI) using the pre-trained CNN.
- Step #4: Classify each proposal using the extracted features with a Support Vector Machine (SVM).

The reason this method works is due to the robust, discriminative features learned by the CNN.

However, the problem with the R-CNN method is it's incredibly slow. And furthermore, we're not actually learning to localize via a deep neural network, we're effectively just building a more advanced HOG + Linear SVM detector.

To improve upon the original R-CNN, Girshick et al. published the *Fast R-CNN* algorithm:

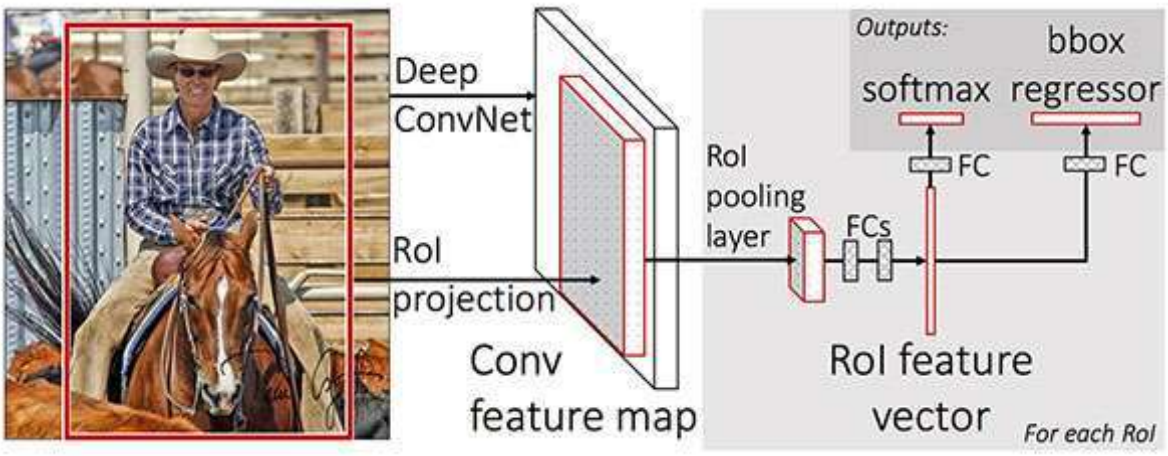

Fig 4.2.2 Fast R-CNN Architecture

Similar to the original R-CNN, Fast R-CNN still utilizes Selective Search to obtain region proposals; however, the novel contribution from the paper was Region of Interest (ROI) Pooling module.

ROI Pooling works by extracting a fixed-size window from the feature map and using these features to obtain the final class label and bounding box. The primary benefit here is that the network is now, effectively, end-to-end trainable:

1. We input an image and associated ground-truth bounding boxes
2. Extract the feature map
3. Apply ROI pooling and obtain the ROI feature vector
4. And finally, use the two sets of fully-connected layers to obtain (1) the class label predictions and (2) the bounding box locations for each proposal.

While the network is now end-to-end trainable, performance suffered dramatically at inference (i.e., prediction) by being dependent on Selective Search.

To make the R-CNN architecture even *faster* we need to incorporate the region proposal *directly* into the R-CNN:

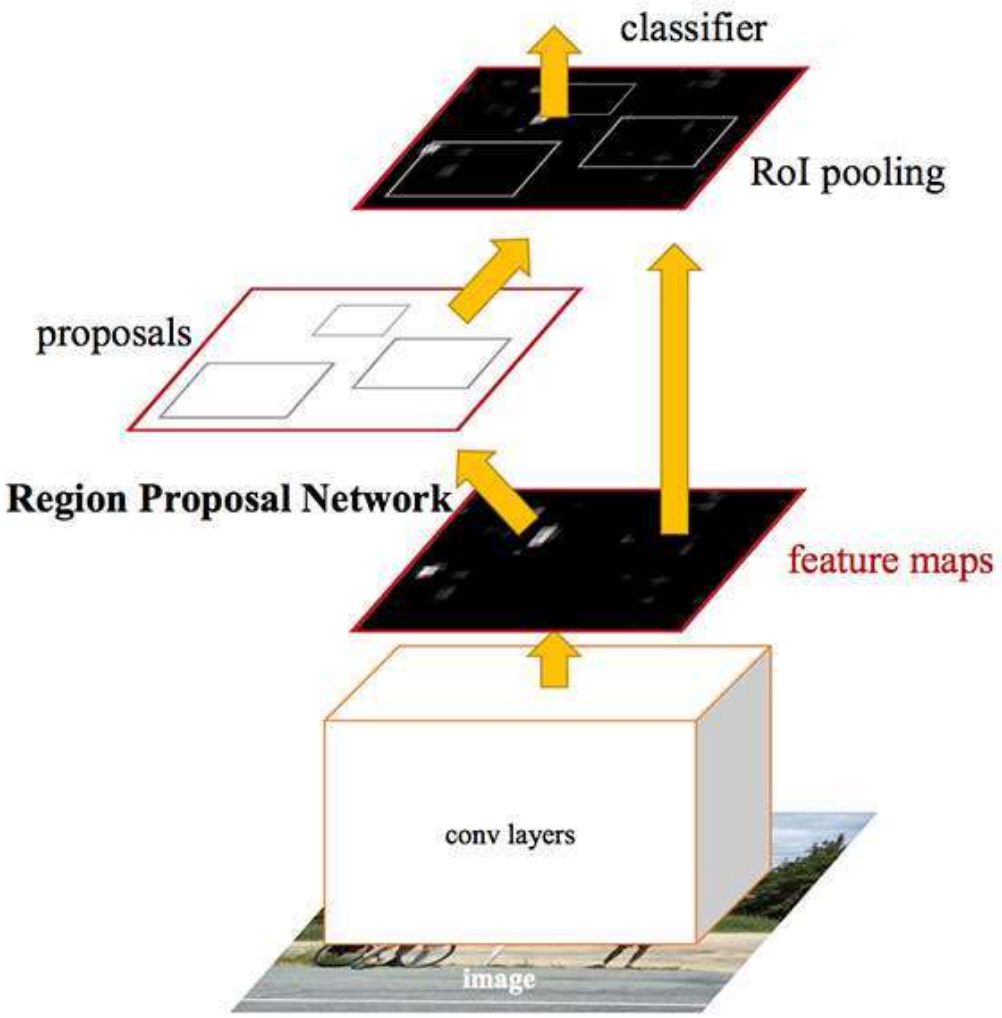

Fig 4.2.3 Fast R-CNN architecture working

The Faster R-CNN paper by Girshick et al. introduced the Region Proposal Network (RPN) that bakes region proposal directly into the architecture, alleviating the need for the Selective Search algorithm.

As a whole, the Faster R-CNN architecture is capable of running at approximately 7-10 FPS, a huge step towards making real-time object detection with deep learning a reality.

The Mask R-CNN algorithm builds on the Faster R-CNN architecture with two major contributions:

1. Replacing the ROI Pooling module with a more accurate ROI Align module
2. Inserting an additional branch out of the ROI Align module

This additional branch accepts the output of the ROI Align and then feeds it into two CONV layers.

The output of the CONV layers is the mask itself.

We can visualize the Mask R-CNN architecture in the following figure:

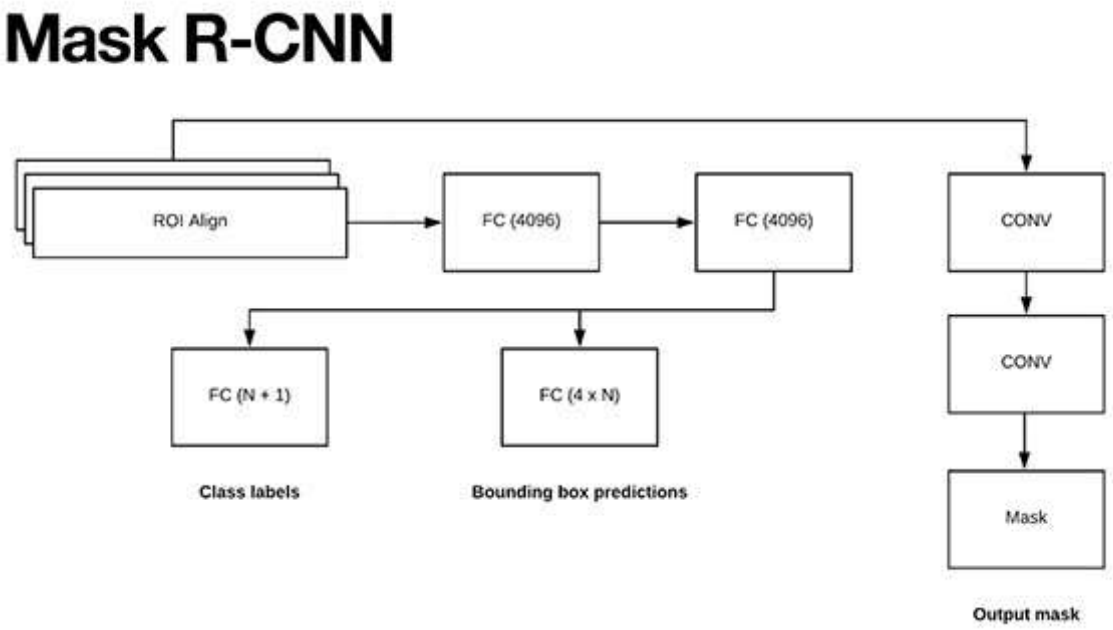

Fig 4.2.4 Fast R-CNN Block diagram

The Mask R-CNN work by He et al. replaces the ROI Polling module with a more accurate ROI Align module. The output of the ROI module is then fed into two CONV layers. The output of the CONV layers is the mask itself. Notice the branch of two CONV layers coming out of the ROI Align module — this is where our mask is actually generated.

As we know, the Faster R-CNN/Mask R-CNN architectures leverage a Region Proposal Network (RPN) to generate regions of an image that *potentially* contain an object.

Each of these regions is ranked based on their "objectness score" (i.e., how likely it is that a given region could potentially contain an object) and then the top *N* most confident objectness regions are kept.

In the original Faster R-CNN publication Girshick et al. set *N=2,000*, but in practice, we can get away with a much smaller N, such as *N={10, 100, 200, 300}* and still obtain good results.

He et al. set *N=300* in their publication which is the value we'll use here as well.

Each of the 300 selected ROIs go through three parallel branches of the network:

1. Label prediction
2. Bounding box prediction
3. Mask prediction

Figure 5 above visualizes these branches.

During prediction, each of the 300 ROIs go through non-maxima suppression and the top 100 detection boxes are kept, resulting in a 4D tensor of *100 x L x 15 x 15* where *L* is the number of class labels in the dataset and *15 x 15* is the size of each of the *L* masks.

The Mask R-CNN we're using here today was trained on the COCO dataset, which has *L=90* classes, thus the resulting volume size from the mask module of the Mask R CNN is *100 x 90 x 15 x 15*.

To visualize the Mask R-CNN process take a look at the figure below:

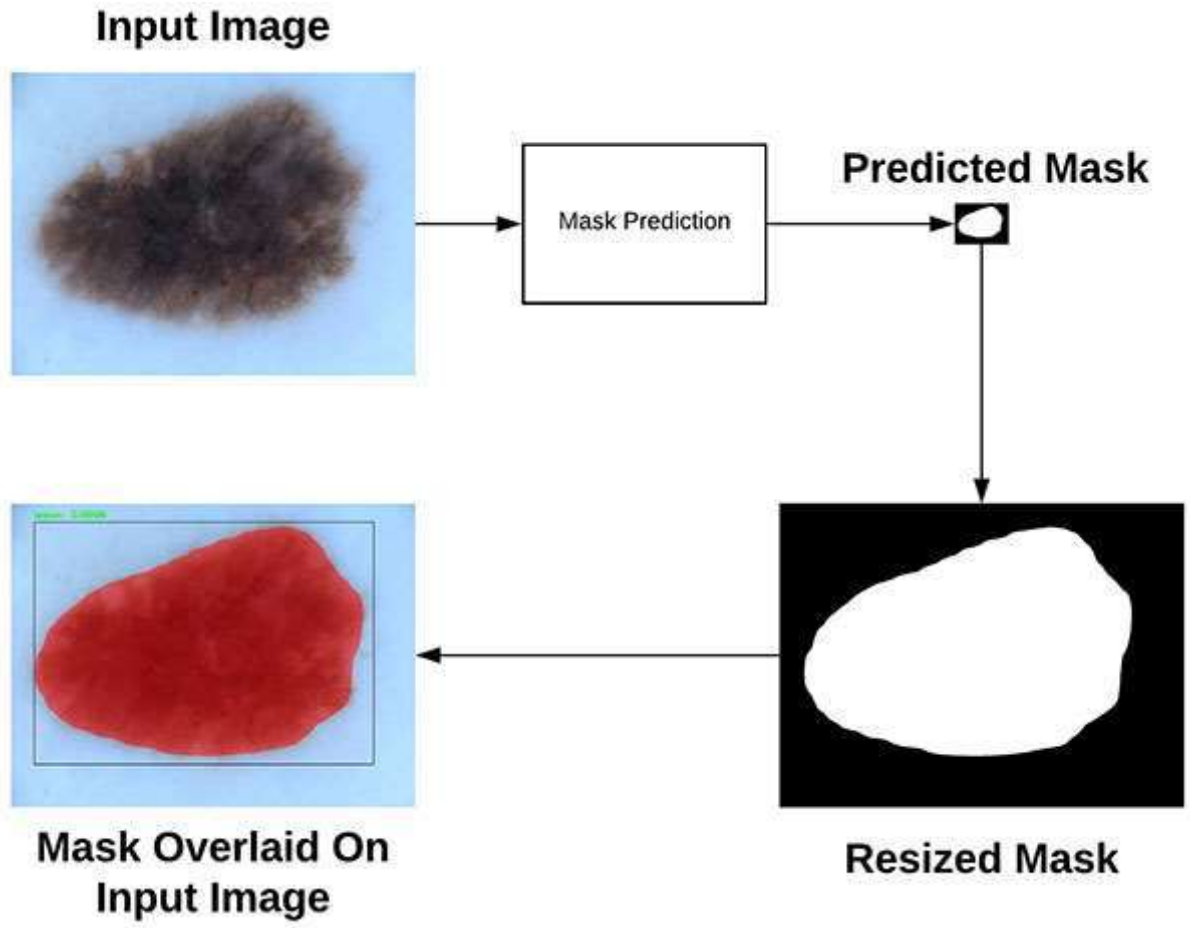

Fig 4.2.5 Visualization of Fast R-CNN

A visualization of Mask R-CNN producing a *15 x 15* mask, the mask resized to the original dimensions of the image, and then finally overlaying the mask on the original image. (source: *Deep Learning for Computer Vision with Python*, ImageNet Bundle)

Here you can see that we start with our input image and feed it through our Mask R-CNN network to obtain our mask prediction.

The predicted mask is only *15 x 15* pixels so we resize the mask back to the original input image dimensions.

Finally, the resized mask can be overlaid on the original input image.

Source code: https://www.pyimagesearch.com/2018/11/19/mask-r-cnn-with-opencv/

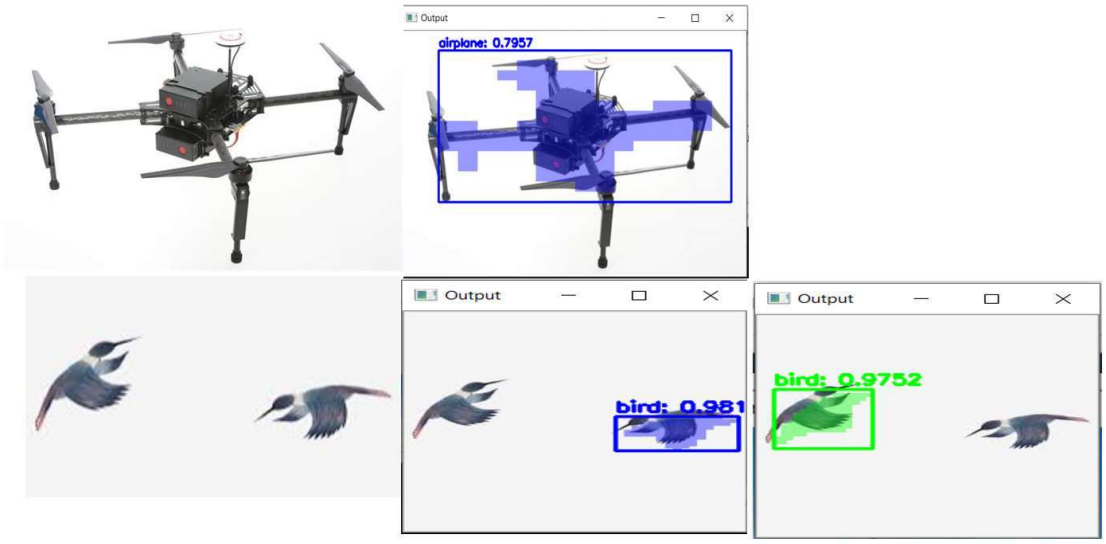

Fig 4.2.6 Results obtained

4.3. OBJECT DETECTION USING IMAGEAI LIBRARY

The breakthrough and rapid adoption of deep learning in 2012 brought into existence modern and highly accurate object detection algorithms and methods such as R-CNN, Fast-RCNN, Faster-RCNN, RetinaNet and fast yet highly accurate ones like SSD and YOLO. Using these methods and algorithms, based on deep learning which is also based on machine learning require lots of mathematical and deep learning frameworks understanding. There are millions of expert computer programmers and software developers that want to integrate and create new products that uses object detection. But this technology is kept out of their reach due to the extra and complicated path to understanding and making practical use of it.

John Olafenwa built ImageAI , a python library that lets programmers and software developers easily integrate state-of-the-art computer vision technologies into their existing and new applications, using just few lines of code.

ImageAI supports many powerful customizations of the object detection process. One of it is the ability to extract the image of each object detected in the image. By simply parsing the extra parameter extract_detected_objects=True into the detectObjectsFromImage function as seen below, the object detection class will create a folder for the image objects, extract each image, save each to the new folder created and return an extra array that contains the path to each of the images.

ImageAI provides many more features useful for customization and production capable deployments for object detection tasks. Some of the features supported are:

- Adjusting Minimum Probability: By default, objects detected with a probability percentage of less than 50 will not be shown or reported. You can increase this value for high certainty cases or reduce the value for cases where all possible objects are needed to be detected.

- Custom Objects Detection: Using a provided CustomObject class, you can tell the detection class to report detections on one or a few numbers of unique objects.

- Detection Speeds: You can reduce the time it takes to detect an image by setting the speed of detection speed to “fast”, “faster” and “fastest”.

-Input Types: You can specify and parse in file path to an image, Numpy array or file stream of an image as the input image

- Output Types: You can specify that the detectObjectsFromImage function should return the image in the form of a file or Numpy array

You can find all the details and documentation of how to make use of the above features, as well as other computer vision features contained in ImageAI on the official GitHub repository. ImageAI is an open-source project by DeepQuest AI.

To perform object detection using ImageAI,

1.Install Python on your computer system

2.Install ImageAI and its dependencies

3. Download the Object Detection model file

4. Run the sample codes (which is as few as 10 lines)

Fig 4.3.1 Detection using ImageAI Library

Source codes: https://github.com/OlafenwaMoses/ImageAI

https://towardsdatascience.com/object-detection-with-10-lines-of-code d6cb4d86f606

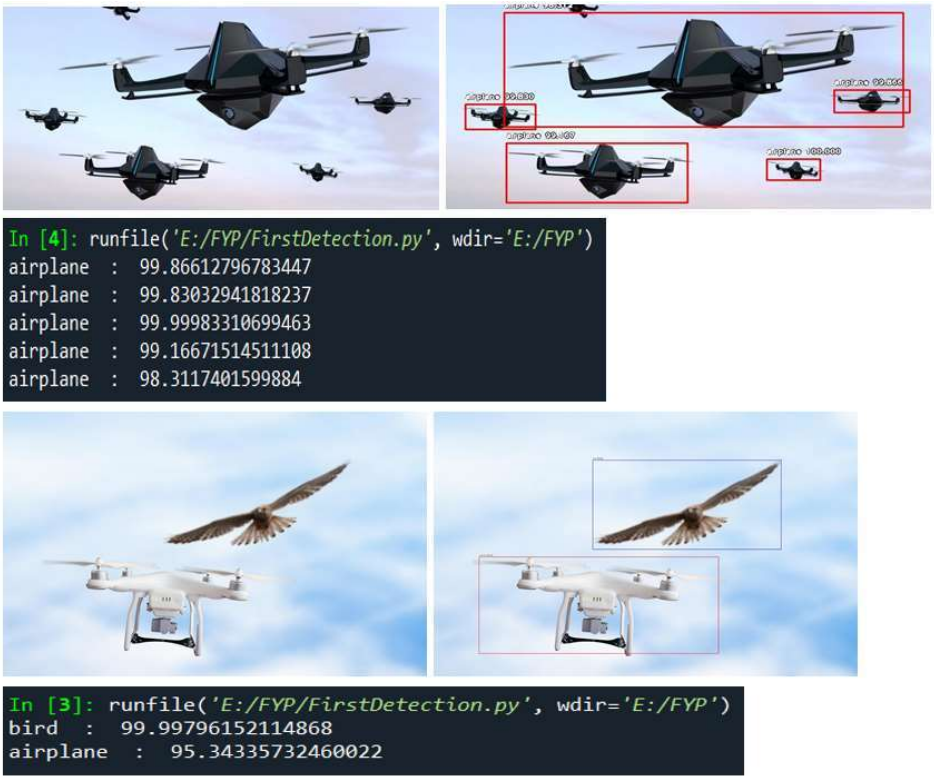

Fig 4.3.2 Results obtained

4.4. Yolo Object Detection using Darknet

Darknet prints out the objects it detected, its confidence, and how long it took to find them. Darknet is mainly for Object Detection, and have different architecture, features than other deep learning frameworks. It is faster than many other NN architectures and approaches like FasterRCNN etc. You have to be in C if you need speed, and most of the DNN frameworks are written in c. I would say TensorFlow has a broader scope, but Darknet architecture & YOLO is a specialized framework, and they are

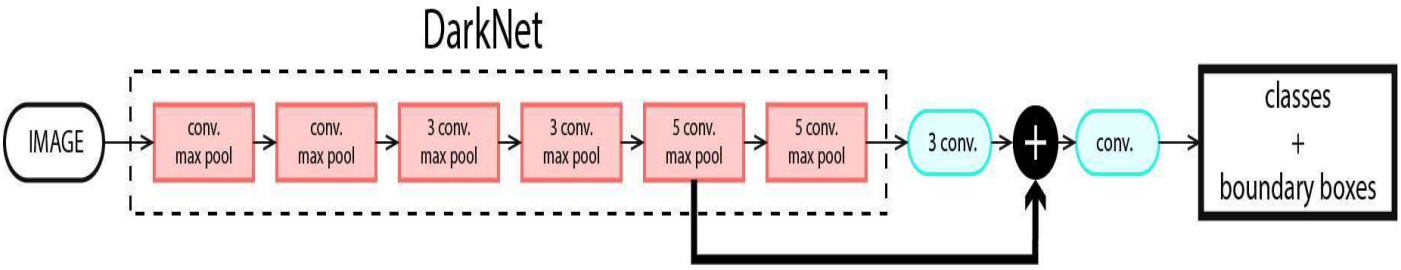

Fig 4.4.1 Detection using Darknet Methodology

on top of their game in speed and accuracy. YOLO can run on CPU but you get 500 times more speed on GPU as it leverages CUDA and cuDNN.

Training YOLO on VOC

You can train YOLO from scratch if you want to play with different training regimes, hyper-parameters, or datasets. Here's how to get it working on the Pascal VOC dataset.

Get The Pascal VOC Data

To train YOLO you will need all of the VOC data from 2007 to 2012.

Generate Labels for VOC

Now we need to generate the label files that Darknet uses. Darknet wants a .txt file for each image with a line for each ground truth object in the image that looks like:

<object-class> <x> <y> <width> <height>

Where x, y, width, and height are relative to the image's width and height. To generate these file we will run the voc_label.py script in Darknet's scripts/ directory. Let's just download it again because we are lazy.

wget https://pjreddie.com/media/files/voc_label.py

python voc_label.py

After a few minutes, this script will generate all of the requisite files. Mostly it generates a lot of label files in VOCdevkit/VOC2007/labels/ and VOCdevkit/VOC2012/labels/. In your directory you should see:

ls

2007_test.txt VOCdevkit

2007_train.txt voc_label.py

2007_val.txt VOCtest_06-Nov-2007.tar

2012_train.txt VOCtrainval_06-Nov-2007.tar

2012_val.txt VOCtrainval_11-May-2012.tar

The text files like 2007_train.txt list the image files for that year and image set. Darknet needs one text file with all of the images you want to train on. In this example, let's train with everything except the 2007 test set so that we can test our model. Run:

cat 2007_train.txt 2007_val.txt 2012_*.txt > train.txt

Now we have all the 2007 trainval and the 2012 trainval set in one big list. That's all we have to do for data setup!

Modify Cfg for Pascal Data

Now go to your Darknet directory. We have to change the cfg/voc.data config file to point to your data:

```
1 classes= 20
2 train  = <path-to-voc>/train.txt
3 valid  = <path-to-voc>2007_test.txt
4 names = data/voc.names
5 backup = backup
```

You should replace <path-to-voc> with the directory where you put the VOC data.

Download Pretrained Convolutional Weights

For training we use convolutional weights that are pre-trained on Imagenet. We use weights from the darknet53 model.

wget https://pjreddie.com/media/files/darknet53.conv.74

Train The Model

Now we can train! Run the command:

./darknet detector train cfg/voc.data cfg/yolov3-voc.cfg darknet53.conv.74

Training YOLO on COCO

You can train YOLO from scratch if you want to play with different training regimes, hyper-parameters, or datasets. Here's how to get it working on the COCO dataset.

Get The COCO Data

To train YOLO you will need all of the COCO data and labels. The script scripts/get_coco_dataset.sh will do this for you. Figure out where you want to put the COCO data and download it, for example:

cp scripts/get_coco_dataset.sh data

cd data

bash get_coco_dataset.sh

Now you should have all the data and the labels generated for Darknet.

Modify cfg for COCO

Now go to your Darknet directory. We have to change the cfg/coco.data config file to point to your data:

1 classes= 80

2 train = <path-to-coco>/trainvalno5k.txt

3 valid = <path-to-coco>/5k.txt

4 names = data/coco.names

5 backup = backup

You should replace <path-to-coco> with the directory where you put the COCO data.

You should also modify your model cfg for training instead of testing. cfg/yolo.cfg should look like this:

[net]

Testing

batch=1

subdivisions=1

Training

batch=64

subdivisions=8

....

Train The Model

Now we can train! Run the command:

./darknet detector train cfg/coco.data cfg/yolov3.cfg darknet53.conv.74

If you want to use multiple gpus run:

./darknet detector train cfg/coco.data cfg/yolov3.cfg darknet53.conv.74 -gpus 0,1,2,3

If you want to stop and restart training from a checkpoint:

./darknet detector train cfg/coco.data cfg/yolov3.cfg backup/yolov3.backup -gpus 0,1,2,3

YOLOv3 on the Open Images dataset

wget https://pjreddie.com/media/files/yolov3-openimages.weights

./darknet detector test cfg/openimages.data cfg/yolov3-openimages.cfg yolov3-openimages.weights

Source code: https://pjreddie.com/darknet/yolo/

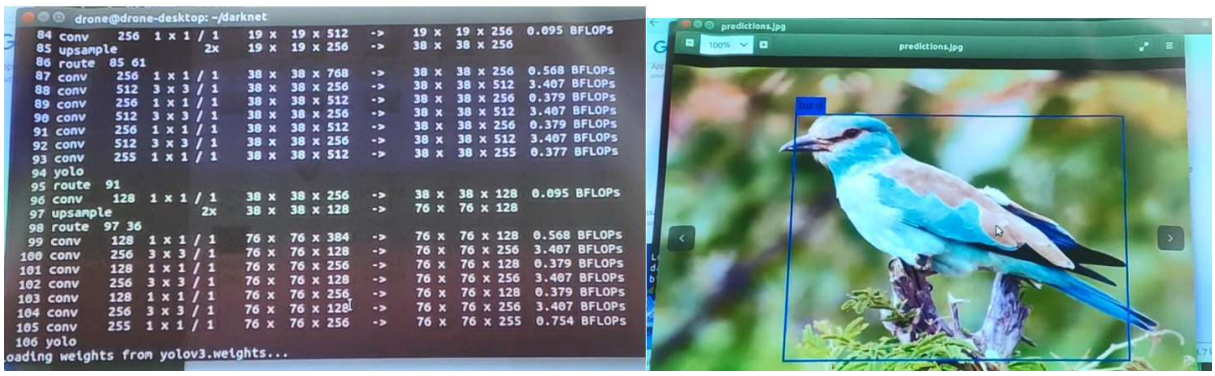

Fig 4.4.2 Results obtained

4.5. CLASSIFICATION USING MOBILENETV2

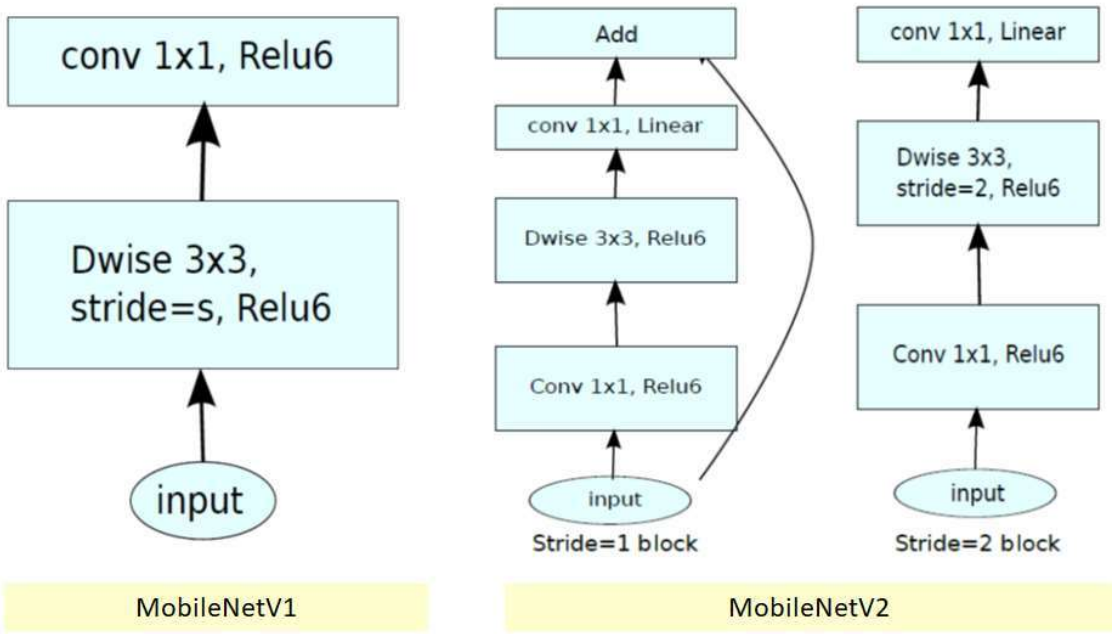

Fig 4.5.1 Classification using MobilenetV2 Methodology

MobileNetV1

- In MobileNetV1, there are 2 layers.
- The **first layer** is called a **depthwise convolution**, it performs lightweight filtering by applying a single convolutional filter per input channel.
- The **second layer** is a **1×1 convolution**, called a **pointwise convolution**, which is responsible for building new features through computing linear combinations of the input channels.
- **ReLU6** is used here for comparison. (Actually, in MobileNetV1 tech report, I cannot find any hints that they use ReLU6… Maybe we

need to check the codes in GitHub…), i.e. **min(max(*x*, 0), 6)** as follows:

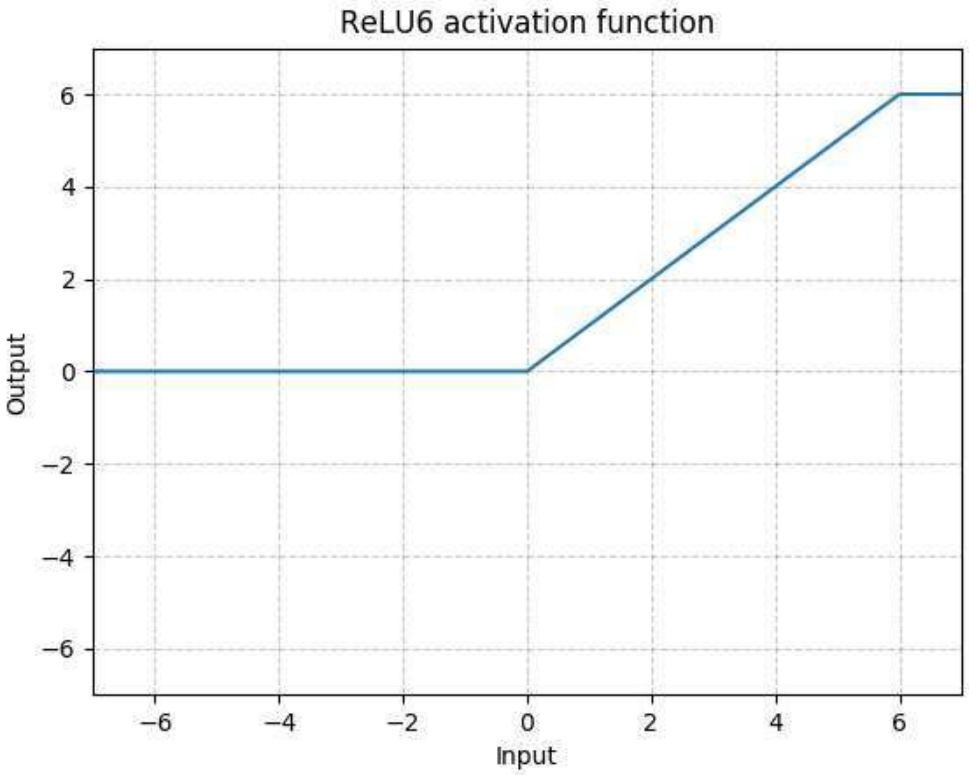

Fig 4.5.2 ReLU6 Activation function

ReLU6

- ReLU6 is used due to its robustness when used with low-precision computation, based on [27] MobileNetV1.

MobileNetV2

- In MobileNetV2, there are two types of blocks. One is residual block with stride of 1. Another one is block with stride of 2 for downsizing.
- There are 3 layers for both types of blocks.
- This time, the first layer is 1×1 convolution with ReLU6.
- The second layer is the depthwise convolution.
- The third layer is another 1×1 convolution but without any non-linearity. It is claimed that if ReLU is used again, the deep networks only have the power of a linear classifier on the non-zero volume part of the output domain.

Input	Operator	Output
$h \times w \times k$	1x1 conv2d , ReLU6	$h \times w \times (tk)$
$h \times w \times tk$	3x3 dwise s=s, ReLU6	$\frac{h}{s} \times \frac{w}{s} \times (tk)$
$\frac{h}{s} \times \frac{w}{s} \times tk$	linear 1x1 conv2d	$\frac{h}{s} \times \frac{w}{s} \times k'$

Table 4.5.1 CNN Layer working

- And there is an expansion factor *t*. And *t*=6 for all main experiments.
- If the input got 64 channels, the internal output would get 64×*t*=64×6=384 channels.

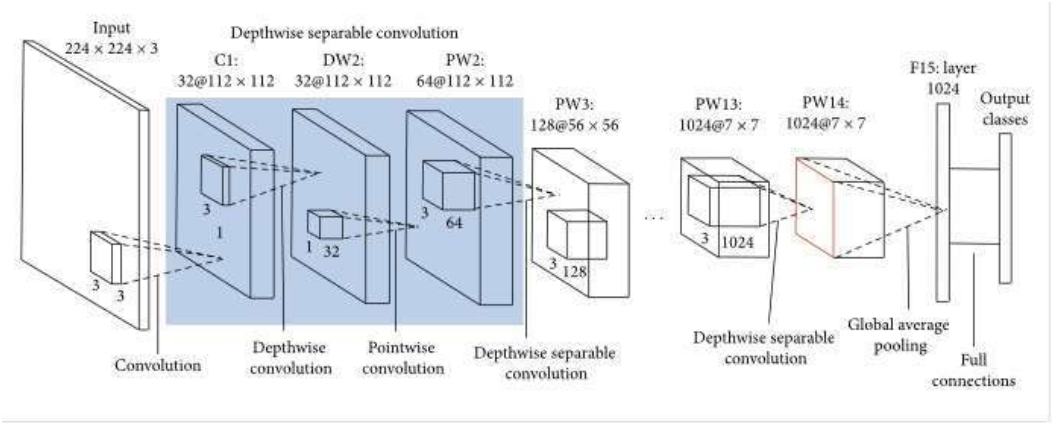

Fig 4.5.3 Architecture of MobilenetV2

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Table 4.5.2 MobileNet Operators

MobileNetV2 Overall Architecture

- where *t*: expansion factor, *c*: number of output channels, *n*: repeating number, s: stride. 3×3 kernels are used for spatial convolution.
- In typical, the primary network (width multiplier 1, 224×224), has a computational cost of 300 million multiply-adds and uses 3.4 million parameters. (Width multiplier is introduced in MobileNetV1.)
- The performance trade offs are further explored, for input resolutions from 96 to 224, and width multipliers of 0.35 to 1.4.
- The network computational cost up to 585M MAdds, while the model size vary between 1.7M and 6.9M parameters.
- To train the network, 16 GPU is used with batch size of 96.

Size	MobileNetV1	MobileNetV2	ShuffleNet (2x,g=3)
112x112	64/1600	16/400	32/800
56x56	128/800	32/200	48/300
28x28	256/400	64/100	400/600K
14x14	512/200	160/62	800/310
7x7	1024/199	320/32	1600/156
1x1	1024/2	1280/2	1600/3
max	1600K	**400K**	600K

Table 4.5.3 Comparison between Mobilenet versions

Number of Maximum Channels/Memory in Kb) at Each Spatial Resolution for Different Architecture with 16-bit floats for activation

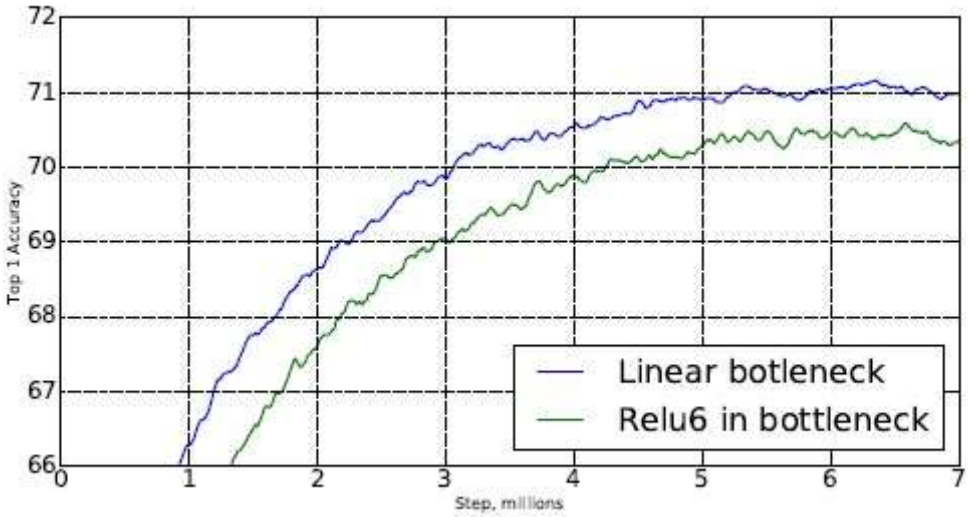

Fig 4.5.4 Linear vs Relu6 Bottleneck

- With the removal of ReLU6 at the output of each bottleneck module, accuracy is improved.

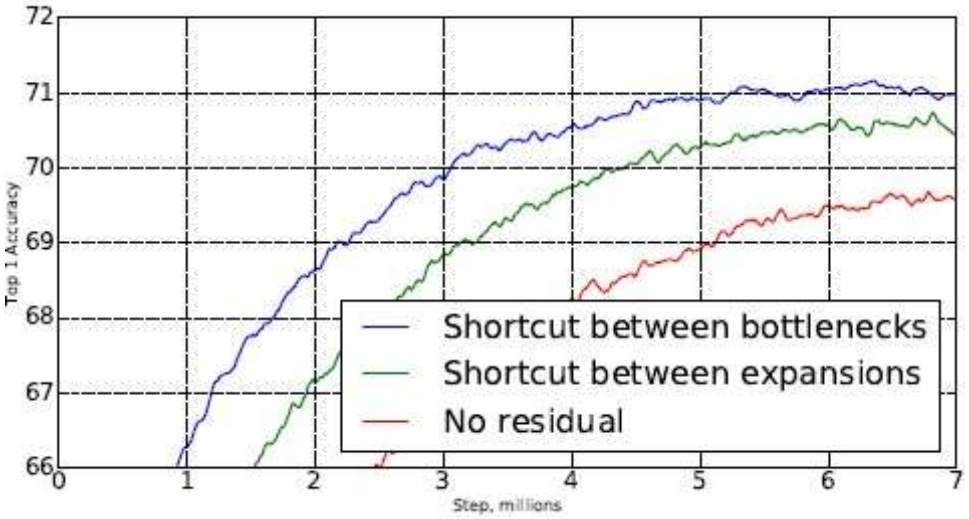

Fig 4.5.5 Impact of shortcut

- With shortcut between bottlenecks, it outperforms shortcut between expansions and the one without any residual connections.

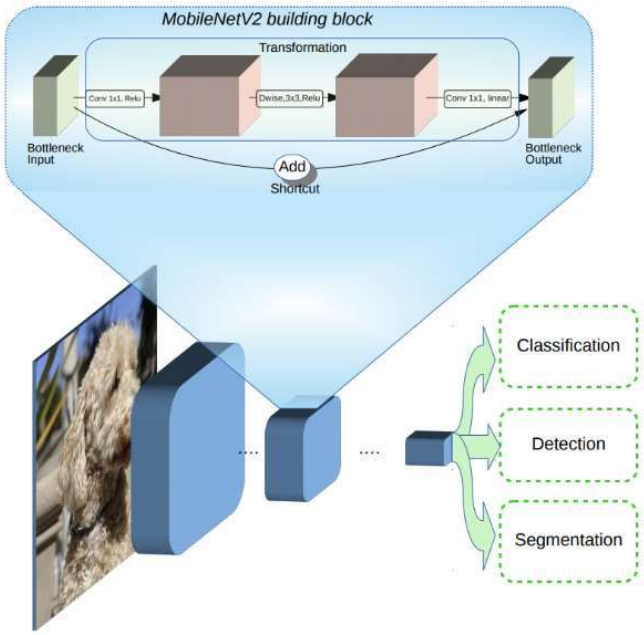

Fig 4.5.6 Mobilenet Blocks

Experimental Results

MobileNetV2 for Classification, Detection and Segmentation
(From https://ai.googleblog.com/2018/04/mobilenetv2-next-generation-of-on.html)

```
Epoch 27/30
69/69 [==============================] - 78s 1s/step - loss: 0.0085 - accuracy: 0.9995 - val_loss: 0.0189 - val_accuracy: 0.9929
Epoch 28/30
69/69 [==============================] - 77s 1s/step - loss: 0.0089 - accuracy: 0.9984 - val_loss: 0.0183 - val_accuracy: 0.9929
Epoch 29/30
69/69 [==============================] - 76s 1s/step - loss: 0.0105 - accuracy: 0.9978 - val_loss: 0.0176 - val_accuracy: 0.9929
Epoch 30/30
69/69 [==============================] - 78s 1s/step - loss: 0.0087 - accuracy: 0.9994 - val_loss: 0.0180 - val_accuracy: 0.9929
```

Fig 4.5.7 Epoch summary

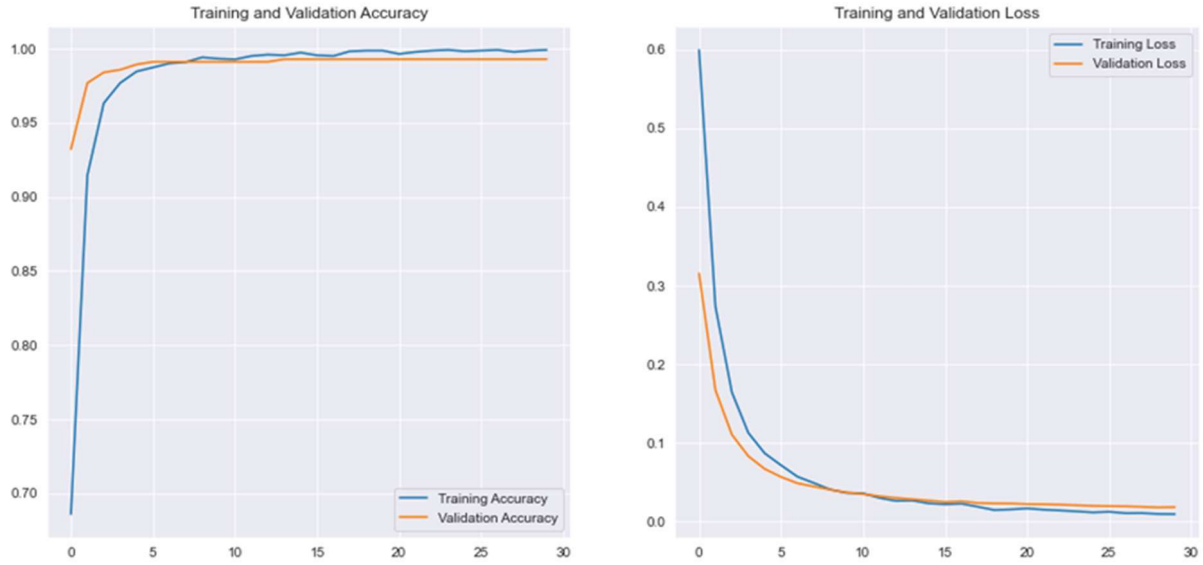

Fig 4.5.8 Training and Validation curves

	precision	recall	f1-score	support
Bird (Class 0)	1.00	0.99	0.99	280
Drone (Class 1)	0.99	1.00	0.99	280
accuracy			0.99	560
macro avg	0.99	0.99	0.99	560
weighted avg	0.99	0.99	0.99	560

Fig 4.5.9 Precision metric

Bird -> Class 0 Drone -> Class 1

```
[ ] img = image.load_img('E:/FYP/Classification/dataset-new/dataset/test/drone/OIP (32).jpg', target_size = (img_width, img_height))
    img = image.img_to_array(img)
    img = np.expand_dims(img, axis = 0)
    print(new_model.predict(img))
    print(new_model.predict_classes(img))

    [[0.00190556 0.9980945 ]]
    [1]

[ ] img = image.load_img('E:/FYP/Classification/dataset-new/dataset/test/bird/Bronzed_Cowbird_0039_24026.jpg', target_size = (img_width, img_height))
    img = image.img_to_array(img)
    img = np.expand_dims(img, axis = 0)
    print(new_model.predict(img))
    print(new_model.predict_classes(img))

    [[0.9760527  0.02394728]]
    [0]
```

Fig 4.5.10 Test results on new dataset

4.6 CNN-DRONE CLASSIFICATION

Lenet-5 Architecture

Training-accuracy: 93.55% | Validation-accuracy: 86.87%

```
[ ]  # Building the CNN
     # Initialising the CNN
     classifier = Sequential()

[ ]  # Convolution
     classifier.add(Conv2D(32, (3, 3), input_shape = (32, 32, 3), activation = 'relu'))

[ ]  # Pooling
     classifier.add(MaxPooling2D(pool_size = (2, 2)))

[ ]  # Adding a second convolutional layer
     classifier.add(Conv2D(32, (3, 3), activation = 'relu'))
     classifier.add(MaxPooling2D(pool_size = (2, 2)))

[ ]  # Flattening
     classifier.add(Flatten())

[ ]  # Full connection
     classifier.add(Dense(units = 128, activation = 'relu'))
     classifier.add(Dense(units = 1, activation = 'sigmoid'))
```

Fig 4.6.1 CNN-Architecture employed

```
Epoch 9/10
50/50 [==============================] - 17s 334ms/step - loss: 0.1777 - accuracy: 0.9279 - val_loss: 0.3359 - val_accuracy: 0.8250
Epoch 10/10
50/50 [==============================] - 16s 316ms/step - loss: 0.1588 - accuracy: 0.9355 - val_loss: 0.3849 - val_accuracy: 0.8687
```

Fig 4.6.2 Epoch summary

4.7 CNN-CLASSIFICATION

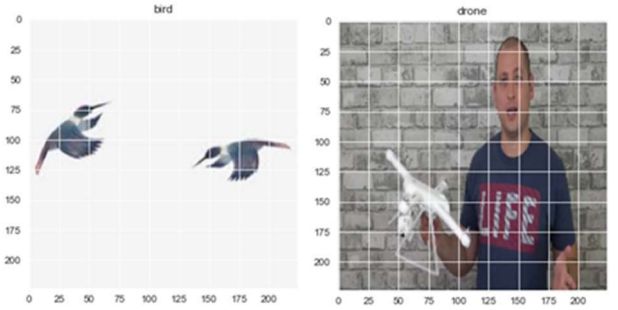

Fig 4.7.1 Old dataset

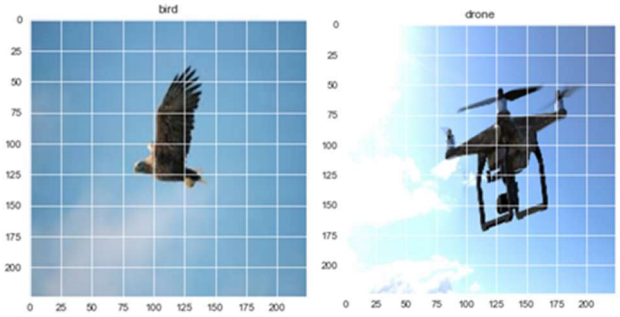

Fig 4.7.2 New dataset

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 224, 224, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 112, 112, 32)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 112, 112, 32)      9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 56, 56, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 56, 56, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 28, 28, 64)        0
_________________________________________________________________
dropout (Dropout)            (None, 28, 28, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 50176)             0
_________________________________________________________________
dense (Dense)                (None, 128)               6422656
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 258
=================================================================
Total params: 6,451,554
Trainable params: 6,451,554
Non-trainable params: 0
_________________________________________________________________
```

Fig 4.7.3 Architecture employed

```
Epoch 7/10
72/72 [==============================] - 149s 2s/step - loss: 0.0375 - accuracy: 0.9832 - val_loss: 0.4613 - val_accuracy: 0.8487
Epoch 8/10
72/72 [==============================] - 149s 2s/step - loss: 0.0282 - accuracy: 0.9889 - val_loss: 0.3834 - val_accuracy: 0.8605
Epoch 9/10
72/72 [==============================] - 149s 2s/step - loss: 0.0186 - accuracy: 0.9928 - val_loss: 0.4011 - val_accuracy: 0.8824
Epoch 10/10
72/72 [==============================] - 150s 2s/step - loss: 0.0420 - accuracy: 0.9858 - val_loss: 0.6840 - val_accuracy: 0.8655
```

Fig 4.7.4 Epoch summary on old dataset

```
Epoch 22/25
69/69 [==============================] - 102s 1s/step - loss: 0.0380 - accuracy: 0.9900 - val_loss: 0.4154 - val_accuracy: 0.8875
Epoch 23/25
69/69 [==============================] - 100s 1s/step - loss: 0.0286 - accuracy: 0.9950 - val_loss: 0.4099 - val_accuracy: 0.8839
Epoch 24/25
69/69 [==============================] - 99s 1s/step - loss: 0.0333 - accuracy: 0.9921 - val_loss: 0.4805 - val_accuracy: 0.8750
Epoch 25/25
69/69 [==============================] - 100s 1s/step - loss: 0.0350 - accuracy: 0.9902 - val_loss: 0.3873 - val_accuracy: 0.8768
```

Fig 4.7.5 Epoch summary on new dataset

	precision	recall	f1-score	support
Bird (Class 0)	0.58	0.75	0.65	100
Drone (Class 1)	0.95	0.89	0.92	495
accuracy			0.87	595
macro avg	0.76	0.82	0.78	595
weighted avg	0.88	0.87	0.87	595

	precision	recall	f1-score	support
Bird (Class 0)	0.87	0.88	0.88	280
Drone (Class 1)	0.88	0.87	0.88	280
accuracy			0.88	560
macro avg	0.88	0.88	0.88	560
weighted avg	0.88	0.88	0.88	560

Fig 4.7.6 Training and Validation curves (top); Metric (Bottom)

```
[ ]  img = image.load_img('E:/FYP/Classification/dataset/test/drone/0293.jpg', target_size = (img_width, img_height))
     img = image.img_to_array(img)
     img = np.expand_dims(img, axis = 0)
     print(new_model.predict(img))
     print(new_model.predict_classes(img))

     [[0. 1.]]
     [1]
```

```
[ ]  img = image.load_img('E:/FYP/Classification/dataset/test/bird/155.jpg', target_size = (img_width, img_height))
     img = image.img_to_array(img)
     img = np.expand_dims(img, axis = 0)
     print(new_model.predict(img))
     print(new_model.predict_classes(img))

     [[1. 0.]]
     [0]
```

Fig 4.7.7 Testing on old dataset

```
[ ]  img = image.load_img('E:/FYP/Classification/dataset-new/dataset/test/drone/OIP (32).jpg', target_size = (img_width, img_height))
     img = image.img_to_array(img)
     img = np.expand_dims(img, axis = 0)
     print(new_model.predict(img))
     print(new_model.predict_classes(img))

     [[0. 1.]]
     [1]
```

```
[ ]  img = image.load_img('E:/FYP/Classification/dataset-new/dataset/test/bird/Brandt_Cormorant_0018_23090.jpg', target_size = (img_width, img_height))
     img = image.img_to_array(img)
     img = np.expand_dims(img, axis = 0)
     print(new_model.predict(img))
     print(new_model.predict_classes(img))

     [[1. 0.]]
     [0]
```

Fig 4.7.8 Testing on new dataset

4.8 HARDWARE IMPLEMENTATION

- Jetson Nano interfaced with Rpi camera module v2
- Using SSD-mobilenet-v2 model
- Trained for 91-classes using MS-COCO dataset
- Uses Tensor-RT for real-time performance
- Function implemented in DetectNet network that return list of detections
- Saving the image with bounding boxes, label and confidence values

Fig 4.8.1 Jetson Nano

Fig 4.8.2 Results obtained

```
# load an image (into shared CPU/GPU memory)
img, width, height = jetson.utils.loadImageRGBA(opt.file_in)

# load the object detection network
net = jetson.inference.detectNet(opt.network, sys.argv, opt.threshold)

# detect objects in the image (with overlay)
detections = net.Detect(img, width, height, opt.overlay)

# print the detections
print("detected {:d} objects in image".format(len(detections)))

for detection in detections:
    print(detection)

# print out timing info
net.PrintProfilerTimes()

# save the output image with the bounding box overlays
if opt.file_out is not None:
    jetson.utils.saveImageRGBA(opt.file_out, img, width, height)
```

Open ▾ my-detection.py

```
import jetson.inference
import jetson.utils

net = jetson.inference.detectNet("ssd-mobilenet-v2", threshold=0.5)
camera = jetson.utils.gstCamera(1280, 720, "      0      ")
display = jetson.utils.glDisplay()

while display.IsOpen():
    img, width, height = camera.CaptureRGBA()
    detections = net.Detect(img, width, height)
    display.RenderOnce(img, width, height)
    display.SetTitle("Object Detection | Network {:.0f} FPS".format(net.GetNetworkFPS()))
```

Fig 4.8.3 Code Implemented

Fig 4.8.4 Snapshot of hardware implementation

CHAPTER 5

CONCLUSION & FUTURE WORK

5.1 CONCLUSION

It can be concluded that we have achieved the classification of bird and drone distinctly on PC with the accuracy as presented as well as the same for each input image on the Jetson Nano board. We have also achieved the real time detection on Jetson Nano with RPI camera.

We have employed various models with different architectures and worked on frameworks such as TensorFlow, Keras, PyTorch, OpenCV and made use of IDE's such as Google Colab & Anaconda. As there was an issue of overheating with the RPI board, we had to work upon the Jetson Nano.

We have used two different datasets for our project and compared the results so obtained. The old dataset contained images with objects that were far away and hence were tiny and posed a difficulty for the model to differentiate between the drone and bird. Hence, we have manually sourced images from Google, Pinterest, Kaggle etc. and prepared a new dataset that fetched better results.

5.2 FUTURE WORK

The project is believed to have a greater scope that includes

- Acquiring increased range for the prototype, which enhances its ability to track far away drones
- Practical deployment of the prototype in feasible locations to test for different conditions and factors (Different applications)
- Differentiating different types of drones (Such as weaponized etc.)
- Faster detection in order to be utilised by sensitive organizations
- Counter-drone technologies such as jammers, netgun, high powered laser beam etc.

REFERENCES

[1] W. Dai, T. Chang, and L. Guo, "Video object detection based on the spatial-temporal convolution feature memory model," 2020 IEEE International Conference on Power, Intelligent Computing and Systems (ICPICS), Shenyang, China, 2020, pp. 312-317,

DOI: 10.1109/ICPICS50287.2020.9202168.

[2] C. Aker and S. Kalkan, "Using deep networks for drone detection," 2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS), Lecce, 2017, pp. 1-6, DOI:10.1109/AVSS.2017.8078539.

[3] K. Kang, H. Li, J. Yan, et al., "T-cnn: Tubelets with convolutional neural networks for object detection from videos," IEEE Transactions on Circuits and Systems for Video Technology, vol. 28, no. 10, pp.

2896-2907, 2017.

[4] C. Feichtenhofer, A. Pinz, A. Zisserman., "Detect to track and track to detect," in Proceedings of the IEEE International Conference on Computer Vision, pp. 3038-3046, Venice, Italy, October 2017.

[5] H. Wei, K. Pooya, L. P. Tom, et al., "Seq-NMS for Video Object Detection," 2016.

[6] A. Dosovitskiy, P. Fischer, E. Ilg, et al., "Flownet: Learning optical flow with convolutional networks," in Proceedings of the IEEE international conference on computer vision, pp. 2758-2766, Santiago, Chile, December 2015.

[7] L. Mason, Z. L. Meng, W. Marie, et al., "Looking Fast and Slow:Memory-Guided Mobile Video Object Detection," 2019,.

[8] X. Z. Zhu, Y. J. Wang, J. F. Dai, et al., "Flow-guided feature aggregation for video object detection," in Proceedings of the IEEE International Conference on Computer Vision, pp. 108-417, Venice, Italy, October 2017.

[9] S. Wang, Y. Zhou, J. Yan, et al., "Fully motion-aware network for video object detection," in Proceedings of the European Conference on Computer Vision, pp. 542-557, Munich, Germany, September 2018.

[10] F. Gokc ¨ ¸e, G. Uc¨ ¸oluk, E. S¸ ahin, and S. Kalkan. Vision-based detection and distance estimation of micro unmanned aerial vehicles. Sensors, 15(9):23805–23846, 2015.

[11] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In ECCV, 2014.

[12] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y.Fu, and A. C. Berg. Ssd: Single shot multibox detector. In ECCV, 2016.

[13] J. Redmon and A. Farhadi. YOLO9000: better, faster, stronger. CoRR, abs/1612.08242, 2016.

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

[15] A. Rozantsev, V. Lepetit, and P. Fua. Detecting Flying Objects using a Single Moving Camera. PAMI, 39:879 – 892, 2017.