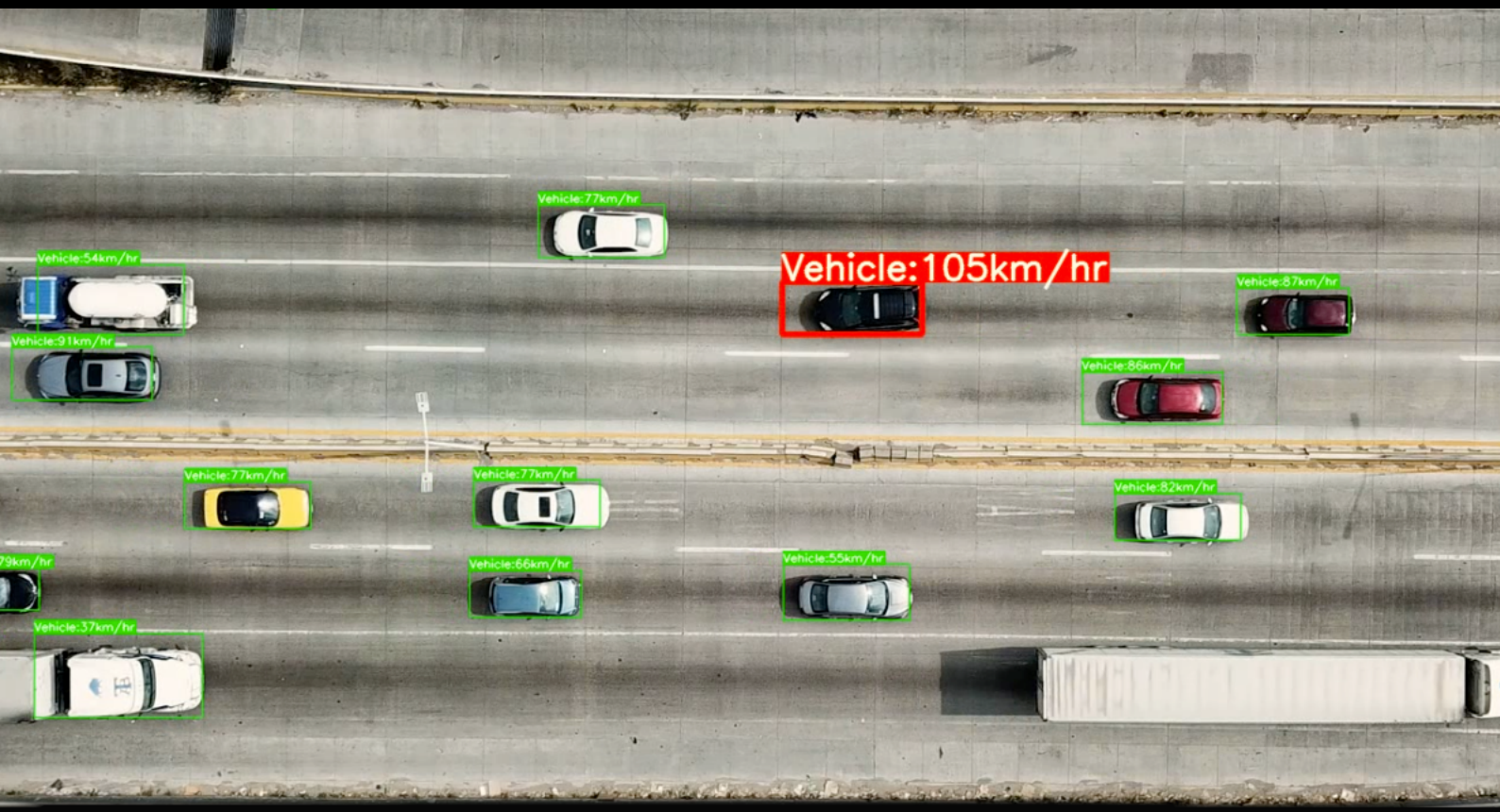
MultEYE

Real-time Vehicle Detection and Speed Estimation from Aerial Images using Multi-Task Learning

Navaneeth Balamuralidhar

MSc. Systems and Control
University of Twente



UNIVERSITY OF TWENTE.

MultEYE

Real-time Vehicle Detection and Speed Estimation from Aerial Images using Multi-Task Learning

by

Navaneeth Balamuralidhar

to obtain the degree of Master of Science in Systems and Control
at the University of Twente,
to be defended publicly on Wednesday November 4, 2020 at 2:00 PM.

Student number:	s2070707	
Project duration:	March 1, 2020 – September 30, 2020	
Thesis committee:	Prof. dr. ir. M.G. Vosselman,	Committee Chair, University of Twente
	Dr. ir. F. Nex,	Supervisor, University of Twente
	Dr. ir. D. Dresscher,	External Examiner, EEMCS Faculty, University of Twente
Subject Advisor:	S. M. Tilon,MSc	PhD Candidate, University of Twente

An electronic version of this thesis is available at https://essay.utwente.nl//.

UNIVERSITY
OF TWENTE.

Acknowledgement

The master's thesis before you, started out as an idea, to build an end-to-end system that can be used to monitor traffic situation using an Unmanned Aerial Vehicle and be used off-the-shelf. This idea when presented to me by my supervisor, Dr.Francesco Nex, little did I know that the path this research took me would elate, frustrate, confuse, excite and demoralize me through the course of seven months . At times, all in the course of a single day. Eventually, I began seeing the light at the end of the tunnel and an end result to my months worth of toil. However, all the hard work would have been for naught if there was no one to guide my work towards a focal point. I was fortunate as a student to have involved supervisors like Dr.Francesco Nex and Sofia Tilon, who made sure I was on track and were always available to address any queries that I had.For this reason, I would like to thank them for their excellent guidance and support during this process.

Working on a research through a global pandemic like COVID-19 would have put a tremendous amount of additional mental strain on me if it wasn't for my family and friends who supported me through these grey days. Last but not the least, my parents deserve a particular note of thanks: your wise counsel and kind words have, as always, served me well

Navaneeth Balamuralidhar
Delft, October 2020

Abstract

Though traffic monitoring systems, in the recent years, has seen automation incorporated into its infrastructure, the area under the scope of surveillance is still small. The per square kilometer investment required deters authorities from large-scale deployment plans. A UAV mounted surveillance solution can address this issue at a fraction of the cost. During the course of this research, an end-to-end system that can detect vehicles from aerial image sequences and estimate their speed in real-time was built. The system consists of three parts: Vehicle Detector, Vehicle Tracker and Speed Estimator. The vehicle detector uses the concept of multi-task learning to learn object detection and semantic segmentation simultaneously on an architecture custom designed for vehicle detection called **MultEYE**, which achieves 1.2% higher mAP score while being 91.4% faster than the state-of-the-art model on a custom dataset. An extremely fast algorithm called MOSSE, that runs multi-object tracking at around 300FPS, serves as the vehicle tracker for the system. Speeds of the tracked vehicles are estimated using a combination of optical flow for motion compensation and known estimates of vehicle sizes as reference for scale. Further, the complete system's performance is also optimized and benchmarked on an NVIDIA Jetson Xavier NX embedded computer to prove its deployability on mobile platforms capable of running on UAVs. The optimized system runs at an average frame-rate of upto 33.44 FPS on frame resolution 3072×1728 on the embedded platform.[1]

[1]The code for this project can be found at `https://gitlab.com/Navaneeth-krishnan/multeye`

Contents

List of Figures

List of Tables

1

Introduction

Highway traffic is a known complex phenomenon which depends on a multi-level interaction between vehicles and the interactions between the vehicles and the local road infrastructure. The control and and management of highway traffic is a complex task due to the constraints imposed by infrastructure and rising number of vehicles in the recent years. Enforcement of traffic control laws and situation monitoring has always been major choke-points in traffic management in the past 10 years due to these constraints.
Until a decade ago, traffic surveillance involved nothing other than the presence of patrol police personnel on the roads who responded to road violations or incidents. However, recently, a steady growth of automated solutions for traffic monitoring are seen, especially in the Netherlands. The Dutch government has implemented and integrated these automated systems into the road infrastructure network. In situ technology, like embed magnetometers and inductive detector loops, and video and image processing techniques have found widespread use across the country. Image and video processing based monitoring solutions have seen greater popularity in highways due to advantages like:

- Multi-lane monitoring
- Ease of modification of detection zones
- Availability of a rich array of data
- Wide area can be monitored if the cameras are used in a network

Such vision based solutions provide a plethora of data from which vehicle characteristics and its motion information can be extracted. This information can be used to detect overspeeding infractions, reckless driving or use of reserved lanes like the bus-lane. Further, it can also be used to identify traffic congestion and incidents after locating the same.
A typical vision based monitoring system consists of a CCD (Charged Coupled Device) camera, mounted on a high platform (typically a bridge), which captures video images online and sends the digitized version of the images to a computer. The algorithms on the computer process the image and perform detection and tracking of vehicles. The function of this system can be broadly classified into two types:

- *Traffic Monitoring*: This includes functions such as passive gathering of various traffic parameters, such as: traffic density, vehicle classification, average speed, etc. It also includes monitoring in order to detect vehicle crashes, also known as AID (Automatic Incident Detection). AID focuses on detecting traffic anomalies, such as: stopped or little traffic flow, traffic jams, vehicles outside of the road area(indicating crashes) etc.
- *Traffic Law Enforcement*: This consists of focusing on identifying traffic violations and uniquely identifying vehicles in case of speeding, reckless driving and unauthorised access of specialized infrastructure.

1.1. Problem Statement

Despite the advantages of vision based solutions and their role in helping the promotion of road safety and infrastructure development, large scale use of the system has not become widespread due to the high costs involved in the installation of the required number of camera units to effectively monitor a highway network. In 2013, each camera unit tendered to cost $\$125,000$ with installation and software charged separately [36]. The main reason for the cost stems from the fact that the field of vision of bridge-mounted camera span from $200m^2$ to a maximum of $1000m^2$. This problem can potentially be solved by a UAV mounted camera system.

Unmanned Aerial Vehicles (UAVs) were developed in the 1950s for the Central Intelligence Agency in the United States for the sole purpose of carrying out surveillance and reconnaissance missions in hostile territory. This was lauded as great decision morally as the vehicle was unmanned and no soldiers' lives were at stake.It also was a great financial investment as the UAVs cost only a fraction of the cost of a manned aircraft. Since entering the civilian domain, UAVs have been used extensively in various low altitude Remote Sensing based applications like crop-health monitoring [73, 93], forest cover estimation / tree crown extraction [34, 126] and Urban Planning [86].

In 2015, the Dutch ministry for infrastructure development successfully demonstrated the use of drones for traffic monitoring [92]. They used three remotely-piloted UAVs to monitor the traffic leading to and from the Concert at Sea festival in Zeeland. Together, the three drones monitored around 30 sq.km of area. Though there were no automated monitoring algorithms implemented during this flight, the demonstration proved that an aerial platform can cover a much larger area at a fraction of the price of ground based cameras. Further, the mobility of the UAV platform enables ease of deployment to areas of interest. The efficiency of UAV based traffic monitoring can be boosted significantly if the system is autonomous. Compared to traditional transportation sensors located on the ground or low angle cameras, UAVs exhibit many advantages, such as minimal cost, ease of deployment, mobility, greater scope of view, uniform scale, etc. In comparison with low angle cameras, UAV videos have lesser chances of occlusion which aids in tracking vehicle's position more accurately from the nadir or isometric view [29]. There have been several pieces of research in the field of autonomous UAVs for traffic monitoring [30, 58, 83], however, there have not been any significant works yet on automated vehicle detection and tracking system for these autonomous UAV systems. Most of the autonomous UAV solutions erroneously assume the state of the art vehicle detection algorithms can work off-the-shelf on these aerial platforms. The state-of-the-art solutions are generally designed to perform their best on curated benchmark datasets and their performance suffers when applied to real-world scenarios. These algorithms also perform their best when a lot of computational power is at hand which is not the case with mobile platforms. Therefore, there is a need to design a vehicle detection and tracking system that performs reliably in real-world scenarios while maintaining real-time processing speed on a mobile platform. This type of a system can also greatly benefit emergency service personnel, like fire brigade, police and ambulance, in assessing the situation while en route to the location of disturbance. The legal framework has not yet caught up with the increasing reliability of such technology but once this comes through, this system will be at hand to be deployed at large scale.

1.2. Research Objective and Questions

The main objective of this thesis is to develop and evaluate a vehicle detection and tracking system that can run real-time on an autonomous aerial surveillance platform. Adding situational awareness capabilities through multi-class semantic segmentation.

In order to achieve this, the following research objectives have to be addressed:

1) Detect vehicles from an aerial perspective image in real-time.

- What are the existing approaches to vehicle detection in aerial images?
- What methods can be used to improve the performance of a real-time approach that can be comparable to a computation heavy approach?

- Which approach strikes a good balance between performance and speed?

2) Enable real-time performance on a computationally constrained mobile platform.

- By how much does the speed performance suffer when a particular approach is transferred onto a mobile platform?
- Is there a way to optimize the approach such that the performance has little or no suffering?
- Does the approach work fast enough to run real-time?
- Can a tracking algorithm be run alongside this approach without adversely affecting the run time?

1.3. Outline

This thesis starts with a background on the deep learning leading up to state-of-the-art detection algorithms. This is followed by the methodology which involves data preparation, experiments with semantic segmentation and object detection separately and then together. The results for these experiments are presented right after each of the experiment methodology is presented. This is followed by the implementation of an optimized solution on a mobile computing device. Finally, the results are summarised and discussed, looking into any possible improvements that could be made in future research.
The main contributions of the research are :

1. A novel and tuned multi-task learning architecture to boost the performance of a computationally light vehicle detector that is robust to scale and view changes in aerial images
2. A novel vehicle speed estimation methodology for moving camera when extrinsic camera parameters are not available
3. An implementation methodology to enable fast execution on an embedded platform

2

Background and Related Work

This chapter introduces the topic of fully convolutional neural networks and networks devoted to semantic segmentation and object detection. It also introduces the concept of Multi-task learning and how the combination of object detection with semantic segmentation task results in improved performance of the object detector. An introduction to the basic artificial neural networks is given in section 2.1. Further, an artificial neural network capable of processing image data, known as Convolutional Neural Networks (CNNs), will be described in section 2.2. A variation of CNNs called Fully Convolutional Network (FCN) is discussed in section 2.3. The history and the latest developments in the field of semantic segmentation and object detection are described in sections 2.3.4 and 2.4 respectively. Final sections 2.5 and 2.6 introduce the concepts of multi-task learning and object tracking respectively.

2.1. Artificial Neural Networks

Artificial Neural Networks (or known simply as *neural network*s) is a computational model made as an information processing paradigm whose design was based on the biological central nervous system. A typical simple neural network consists of 3 parts: an input layer,one or more hidden layers and an output layer. The role of neural networks as function approximators was suggested by Cybenko [25]. He showed that a neural-network with one hidden layer could function as a universal function approximator as long as the function is continuous and the number of neurons in the network are finite.
The unambiguously named *Cybenko's Theorem*, states that if σ is a continuous discriminatory function, the finite sums of the form:

$$G(x) = \sum_{j=1}^{N} \alpha_j \sigma\left(y_j^T x + \theta_j\right) \tag{2.1}$$

are dense in $C(I_n)$. In other words, given any $f \in C\,(I_n)$ and $\epsilon > 0$, there is a sum $G(x)$, of the above form , for which:

$$|G(x) - f(x)| < \epsilon \quad \forall x \in I_n \tag{2.2}$$

Hence, provided a target function $f(x)$, that is to be approximated with a certain degree of accuracy $\epsilon > 0$, Cybenko's theorem states that there is always a network with output $G(s)$ which satisfies $|G(x) - f(x)| < \epsilon$, given that a sufficient number of (hidden) neurons are used .

2.1.1. Neuron Model

A mathematical neuron model is used to model a neural network. Earlier, works show that the development of neural networks was initially based on the imitation the biological neural system in a computational sphere. A *neuron* is the basic computational unit of the brain. It receives inputs on its dendrites and the output is delivered through its axons (Fig.2.1). The

axon terminals are called *synapses* which connect to the dendrites of other neurons. The synaptic strength is the factor that decides the degree of neuron interaction. These strengths are mathematically analogs to *weights* (w_i) in the artificial neural networks and these weights are learnable. The dendrites convey the input signal x_i to the cell-body, where these inputs are summed. The neuron then produces an output which is sent out through the axon once the sum crosses a certain threshold. The rate of the output generation is modeled by an activation function $f(x)$ that is non-linear in nature. An *activation function* decides the state of activation of a neuron by calculating a weighted sum and an additional bias. The input is transformed non-linearly which enables the network to learn tasks of high complexity.

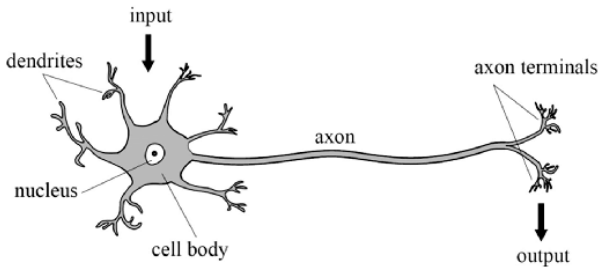


Figure 2.1: Example of a biological neuron

The decision making by an activated neuron can be modeled the following way:

$$a_i = f\left(\Sigma_i \left(w_i x_i\right) + b_i\right) \tag{2.3}$$

The activation of a neuron is represented by a_i while $f(x)$ is the non-linear activation function. w_i, x_i and b_i represent weight, input signal and bias respectively. Initially, the neurons were activated by a simple sigmoid or a hyperbolic tangential function but in the recent years, the Rectified Linear Unit (ReLU) [79] function has shown great promise as it was observed that ReLU was more computationally efficient that its counterparts[59][42]. This efficiency stems from the ability of ReLU to identically propagate all the positive inputs, which alleviates gradient vanishing and allows the supervised training of much deeper neural networks and additionally just output zero for negative inputs.

2.1.2. Network of Neurons

A *Neural Network* is a state of interconnected neurons arranged in layers. As seen in nature, the outputs of one layer of neurons become the inputs for the succeeding neuron layer. The inputs are generally passed forward without any signal-loops present in the network. Further, the neurons of a layer are not connected to each other. The *hidden layers* (Fig.2.2) are the layers of neurons arranged between the output layer and the input layer. Data can be perceived to be more abstracted the more layers the input passes through. The number of hidden layers constitutes the *depth* of the neural network.
The network is made to learn a general rule using a given set of examples. This process is called *Training*. It is done by updating the weights using a method called *backpropagation*. This method is further explained in section.2.2.4.

2.2. Convolutional Neural Network

A Convolutional Neural Network (CNN) [60] is a network of neurons that can process features from spatial data especially images. A CNN layer has neurons that are arranged in 3 dimensions. This arrangement enables it to not only process the 2D pixel arrangement in an image but also the channels of each pixel. CNNs have grown rapidly in the past 10 years and are now used to solve many of the image processing tasks such as image classification, segmentation and object detection. This section introduces the essential parts that influence the performance of a CNN such as Convolutional layer, Pooling layer, Backpropagation, Loss function, Learning Rate Scheduling and Hyperparameter Tuning. This section also describes a brief timeline outlining the development of CNNs through the ages.

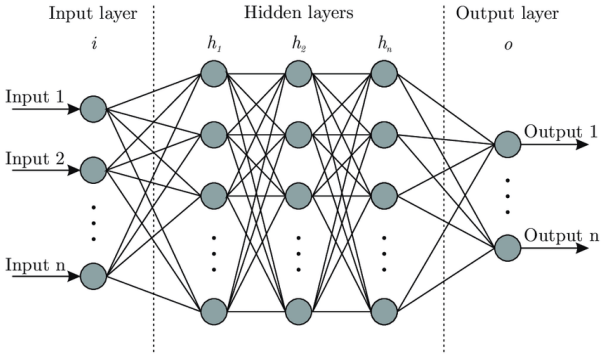


Figure 2.2: Neural network with hidden layers

2.2.1. Convolutional Layer

A standard digital image can be represented in a three-dimensional matrix of the form $h \times w \times 3$, where h represents the height of the image, w the width and the last dimension represent the number of channels with colour information which takes the value of 3 for the majority of images that have an RGB (Red-Green-Blue) or BGR (Blue-Green-Red) colour encoding. Multi- and Hyper-spectral images can expect the third dimensions to be of higher order. On the other hand, images can also have only 1 channel (Grayscale images). These representations are then fed into a convolutional layer. Convolutional layers are the most essential units of a CNN as they produce a mapping of features from input image or from other encoded features that are mappings from other convolutional layers .

Kernel is one of the center pieces of a convolutional layer. If kernel K has x rows, y columns and depth of d, the kernel that has a size of $(x \times y \times d)$ works on a receptive field of size $(x \times y)$ on the image. The kernel by design is smaller than the image. The kernel slides over the image (thereby convolving with it) and produces a feature map. A Convolution is the result of summation of element-wise multiplication of the spatial information with the kernel. The kernel *stride* is a parameter that is defined prior to training. Another parameter that is found in the definition of a convolutional layer is the stride,which is the number of pixels the kernel traverses during a convolution operation. Larger the stride, smaller is the resulting output. Equation.2.4 shows the relation between the size of the output O and size of the input image I after convolving with kernel K with stride s.

$$O_x = \frac{I_x - K_x}{s} + 1$$
$$O_y = \frac{I_y - K_y}{s} + 1 \tag{2.4}$$

After the convolution step, each pixel is then activated with an activation function (for example, ReLU non-linearity operation where every negative value is replaced with a 0). The output of this operation then proceeds either to another convolutional layer or a pooling layer.

2.2.2. Pooling Layer

Pooling Layers are used to decrease the size of the input by a certain fraction while attempting to retain the feature information of the input, essentially encoding them. The most commonly used pooling technique is the max pooling. The output of this layer is generated by choosing the maximum value within a selected kernel and replacing the pixel value with this chosen maximum value. Average pooling and L2-norm pooling are other pooling methods commonly used. The kernel and stride of this layer have the same length. Pooling layers decrease the number of trainable parameters thus reducing the chance of overfitting.

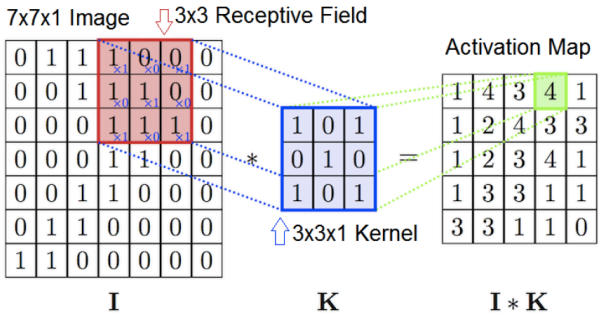


Figure 2.3: Single channel image I convolving with kernel K and stride 1.

2.2.3. Dropout Layer

Overfitting is the phenomenon in which a neural network loses its ability to generalize. By design, neural networks are overdefined and overfitting happens when there are too few training samples. Dropout layers prevent overfitting by randomly dropping nodes and its connections which adapts the network to the training samples while prevents the weights to be too much fitted to the training set. This will result in a significant reduction of the difference between the validation and training accuracy. Dropout layers are removed or deactivated during validation and testing.

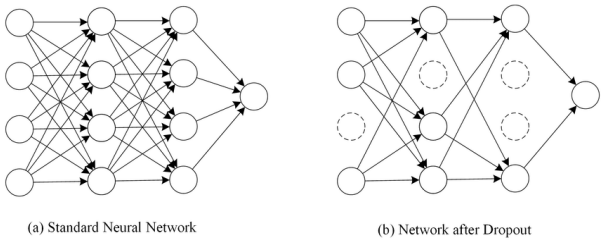


Figure 2.4: Neural network before and after a dropout layer [108]

2.2.4. Backpropagation

The training of a CNN involves adjusting the weights of the kernels. Backpropagation is an efficient method that calculates the gradients that are used by optimization algorithms [101]. The optimization problem is considered solved when the optimizer finds a unique combination of parameters that result in a minimum value of a loss function. The method involves calculation of gradient of the loss function. One of the pre-requisites of a loss function is that there should always exist a gradient in its domain or in other words, the loss function should always be continuous and differentiable through out its domain.
The weights of a neural network is initially assigned randomly. At, first, the network doesn't have a relation between the input image and output. The network is then trained by adaptation of the weights in a such a way that the difference between the predicted output and the expected output is minimized. There are two phases of computation of these weights, the forward pass and the backward pass. **Forward pass:** The image is quantized and fed to the input layer of the network which outputs an activated feature map which serves as the input for the second hidden layer which in turn computes its own feature map. This process is sequentially repeated for every connected layer in the network which eventually ends at the output layer.
Backward pass: The backpropagation updates the network weights. A single epoch of backpropagation has many parts and multiple epochs are required to be performed for a single image. Parts in an epoch are:

- **Loss function** A predefined loss function L minimizes the difference between the presented input-output pair. The weights are adjusted based on the calculated gradients from the loss function.

- **Backward pass** The total loss is reduced in this step by adjusting weights that had majority contribution to the calculated loss.
- **Weight update** Finally, all the weights are updated in the negative direction of the loss function gradient.Finally, all the weights are changed in the opposite direction of the gradient of the loss function.

As it can be seen, the main crux of the backpropagation problem is the computation of the loss function gradient with respect to the weights. This is done by minimizing the value of loss function using the computed partial derivative $\frac{\partial L}{\partial w}$. A common method for neural network optimization is the Stochastic Gradient Descent (SGD).

2.2.5. Loss Function

A function L that can quantify the difference between the input image and the annotated output after it has passed through the network is known as a loss function. A few of the most widely used loss functions are listed below. Assume x_i is the array of predicted outputs and $\hat{x}_i$ is the array of required outputs.

Quadratic Loss Function One of the simplest and most used loss functions is called the Mean Squared Error (MSE). It is defined as follows:

$$L = \frac{1}{N}\sum_{i=1}^{N}(x_i - \hat{x}_i)^2 \tag{2.5}$$

Cross-entropy Loss Function This loss function is used in the context of convolutional neural network applications.

$$L = \frac{1}{N}\sum_{i=1}^{N}(\hat{x}_i \ln(x_i) + (1-\hat{x}_i)\ln(1-x_i)) \tag{2.6}$$

Exponential Loss Function The exponential loss function requires an additional parameter τ.

$$L = \frac{1}{N}\tau \exp\frac{1}{\tau}\sum_{i=1}^{N}(x_i - \hat{x}_i)^2 \tag{2.7}$$

2.2.6. Learning Rate Scheduling

The learning rate of an optimization algorithm refers to the step-size it needs to take to get to a local or a global-minimum. Determination of the appropriate learning rate so as to achieve a global minimum is a complex problem. If the learning rate is too high, the model may never converge which leads to a sub-optimal performance while a learning rate set too low would lead to a slow convergence or the model may get stuck in a local minima. Learning scheduling is done to exploit the exploring nature of a high learning rate during the initial iterations in order to obtain a coarse estimation of the output and then decreased during further iterations to fine tune the model.

2.2.7. Hyperparameter Tuning

As explained in 2.2.4, the training of an artificial neural network is done using backpropagation. This process presents many choices a researcher should take, for example, batch size, optimizer, learning rate etc. These choices are what make up *hyperparameters* which act as a setting for the Neural Network. The optimal choice of values for these hyperparameters makes sure that the network gives its best performance. In order to make educated guesses on the values of these hyperparameters, the nature of each of these settings must be understood in the context given by the problem and the dataset. For example, the batch size is chosen based on the memory availability of the training machine. Larger the batch size, more memory required. The appropriate learning rate can be chosen by tracking the

loss per epoch (or a cycle of forward and backward pass). High learning rates will result in fast decrease of loss but may result in sub-optimal performance. The degree of over-fitting can be observed by comparing the training loss and the validation loss.

2.2.8. Applications of CNNs : A brief timeline

When CNNs were first introduced, they were designed to solve the *Image Classification* problem. Classification aims at labeling images and sorting them into pre-defined categories.
LeCun *et al.* were the pioneers in applying CNNs for complex tasks in 1998 [60]. **LeNet** was the first successful CNN built which was capable of reading handwritten digits and zip-codes. These were not widely popular until Krizhevsky et.al. [59] improved the CNN architecture to use Graphical Processing Units(GPU) to improve accuracy and faster training time. This provided a lauchpad for interest in Deep Learning for computer vision. **AlexNet** [59] is a deeper and larger version of LeNet. The network won the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC), by a large margin. It introduced the Rectified Linear Unit (ReLU) activation which significantly sped-up the optimizer (SGD), as expensive computational elements like tangentials and exponents were not required. Data augmentation and drop-out layers were used to further improve the network performance and reduce over-fitting.
In 2015, Google introduced inception module, as a part of the network called **GoogLeNet** [110] which won the ILSVRC2014 and made use of breakthrough approach in the design of CNN networks.
The Visual Geometry Group of the University of Oxford realised that a certain number of hidden layers were instrumental in efficiently encoding features, which resulted in **VGGNet** [107], which secured the second place in the ILSVRC 2014.

Microsoft, who introduced the **ResNet** architecture [46], won the ILSVRC2015. This network was considered a breakthrough by achieving an all time low rate of error of 3.6% which beat human performance which averaged around 7%. This network contains 152 layers.

Image classification later paved way for semantic and instance segmentation. Section.2.3 introduces and expands on Fully Convolutional Networks (FCNs) that were designed for the task of such segmentations.

2.3. Fully Convolutional Networks

Image segmentation is the classification of each pixel to the feature it belongs to in the image. The following sub-sections introduce Fully Convolutional Networks and their role in semantic segmentation.

2.3.1. Introduction

Fully Convolutional Network (FCN) was introduced for image segmentation in 2015 by Long *et. al.*[69]. FCNs are essentially modified CNNs. To make it suitable for pixel-wise segmentation, some layers are modified to enable generation of segmented maps as output. In other words, FCNs are created by replacing the fully-connected layers with convolutional layers. The main objective of an FCN is to extract contextual information; an object's identity and location. The architectures of FCNs inherently balances the coarse tuning of large-scale features and the fine tuning of small-scale local features.
Standard classifier networks can be used to learn semantic segmentation by the method of Transfer Learning. In this method, the location encoded in the trained classification network is exploited to generate segmented output maps by replacing the fully connected output layer by a convolutional layer.

2.3.2. Transposed Convolutional Layer

A Transposed Convolutional layer is used to obtain a dense map from a downsampled and coarse input [69]. They are also known as, *Deconvolutional Layer* but it is a misnomer as the layer also performs convolution.

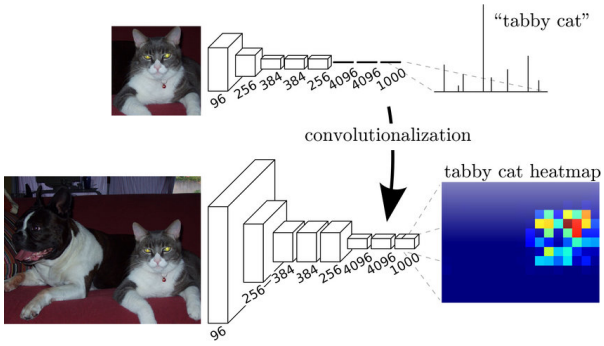


Figure 2.5: Depiction of how the class location information is encoded in a classification network [1]

This process differs from the process of UnPooling in a basic concept that the unpooling increases the size of the receptive field by a simple operation of nearest-neighbour or bilinear interpolation, while a transposed convolutional layer maps a single activation to a field of activations.

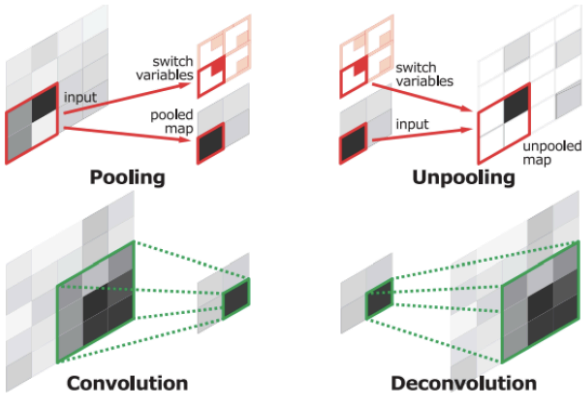


Figure 2.6: Difference between Unpooling and Deconvolution [85]

2.3.3. Types of FCNs

Since the introduction of FCNs, they have dominated the field of online image segmentations. The original FCN proposed by Long *et.al* [69] has a drawback of producing outputs of low-resolution due to the pooling and convolution operations in the layers. New network architectures have been proposed to tackle the output resolution problem. These methods have different approaches to collect both global and contextual features. The major four are :

- Encoder-Decoder
- Image Pyramid
- Spatial Pyramid Pooling
- Atrous convolutions

These methods have seen widespread use in the context of scene understanding, especially in the problem of semantic segmentation. The use of these methods in semantic segmentation is further discussed in section 2.3.4.

2.3.4. Semantic Segmentation

Image segmentation is an important aspect of many of the vision based systems. It involves classification of image pixels based on a high-level understanding of the type of information

which the pixel is a part of [111]. Segmentation plays an important role in many applications seen today [35], including medical imaging analysis (Tissue volume measurement, Tumour identification etc.), autonomous vehicles (ego-lane extraction, pedestrian detection, etc.), video surveillance and augmented reality. A plethora of semantic segmentation algorithms have been published in the past decade, from early pieces, such as thresholding [89], Clustering based on histogram [84], k-means clustering [28], local pixel topography based algorithms [80], to more modern and complex methods like Energy minimizing snakes [55], minimizing energy using computational graphs [14], advanced computational graphs in the form of random fields [1] and segmentation methods based on sparsity priors [109].
In the recent years, with widespread availability and accessibility of higher computational power, Deep Learning algorithms have enabled the development of segmentation algorithms that significantly outperforms its predecessors. These algorithms are popularly regarded as the front-runners of the paradigm shift that the field experienced.
FCNs were popularized in the field of semantic segmentation with the introduction of skip connections. Skip-connections are used to merge final layers of a network with a similarly sized initial layer in order to preserve features that may have been lost in the encoding process. This method of feature map merging results in a healthy mixture of fine and coarse details in the segmentation map. This resulted in performance bossts that propelled the FCNs to the top of standard benchmark challenges like PASCAL VOC, NYUDv2, and SIFT Flow. However, despite its popularity and effectiveness, traditional FCNs have a major limitation– it is not light enough to be implemented on an embedded platform, it could not perform real-time segmentation and it does not process the global context information efficiently. There were many efforts to over-come these limitations of standard FCNs.A few of these methods discussed in this section are, Encoder-Decoder, Image-Pyramid, Spatial Pyramid Pooling and Atrous Convolutions.

Encoder-Decoder

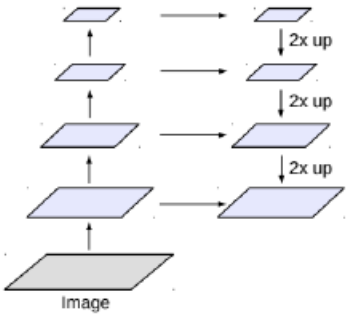


Figure 2.7: Generalized depiction of a typical Encoder-Decoder architecture [19]

The Encoder-Decoder type of network architecture (fig.2.7) consists of an encoder part and a decoder part. The encoder reduces the spatial dimensionality of the features and is known as the *latentspace*. The decoder part recovers the features encoded in the latent space and the spatial dimensions. A few examples of such an architecture are SegNet [4], U-Net [100], and RefineNet [63]. The SegNet architecture relies heavily on the VGG-16 model to reduce the dimensionality of the input image. The SegNet introduced a novel concept of using pooling-indices that are calculated in the max-pooling step of the encoder. These indices are then used to up-sample the feature maps in the decoder part non-linearly without having to use learnable parameters. Due to the absence of additional deconvolutional layers, SegNet is lighter than its contemporary models. SegNet was later upgraded to a Bayesian version of the same which could model the uncertainty inherent in the standard SegNet[56].

One main drawback of SegNet is that it cannot be used to get a high-resolution feature map as small artifacts are not efficiently encoded in the latent space. A solution for that problem was tackled by U-Net [100] and V-Net [78]. These architectures, though intended for medical image segmentations, it found widespread use in other domains too. Ronneberger *et al.* [100] proposed the use of U-Net to segment microscopy images. The architecture (Fig.2.8),inspired by the traditional FCNs and the encoder-decoder architecture, consists of two parts, an contracting encoder part used to encode contextual information, and a symmetric expanding decoder part that provides precise localization. The downsampling layers extracts features using 3×3 convolution much like FCNs. The upsampling layers use deconvolution layers which are concatenated with its corresponding encoder layer which helps it retain pattern information and avoid losing small features while encoding into the latent space.

The U-Net has been modified numerous times to cater to various types of images, for example, a U-Net architecture was proposed for 3-D images by Cicek *et al.* [23]. A nested architecture was proposed by Zhou *et al.* to produce a more robust architecture [134]. U-Net was also modified and applied to other problems such as road extraction and segmentation [131].

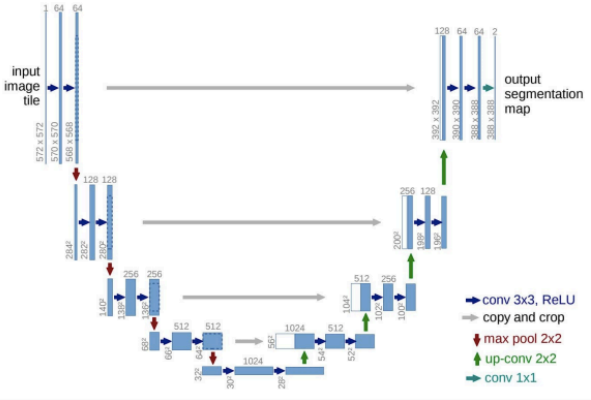


Figure 2.8: U-Net model architecture[100]

Image-Pyramid

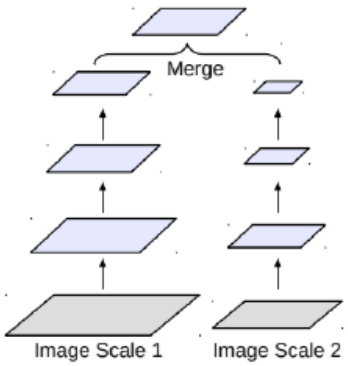


Figure 2.9: Generalized depiction of a typical Image-Pyramid architecture[19]

An Image Pyramid network architecture (Fig.2.9) works on multiple image resolution that accounts for multiple levels of feature encoding (from coarse at low resolution to dense at high resolution). An image is passed at multiple scales through the network and eventually merged at the end. The major disadvantage of such a structure is that it is not suitable for larger architecture as this requires high GPU memory to hold images at multiple resolutions

during training. One of the major model architectures employing this method is the Feature Pyramid Network (FPN) proposed by Lin *et al.* [64] which was aimed at use in object detection but was also applied to image segmentation.

Spatial Pyramid Pooling

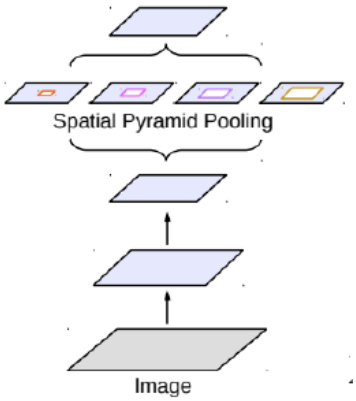


Figure 2.10: Generalized depiction of a typical Spatial Pyramid Pooling architecture[19]

Networks that use spatial pyramid pooling method, learn features at varying levels of details (Fig.2.10). Parse-Net[67] and PSPNet[132] are examples of networks that makes use of Spatial Pyramid Pooling.
The Pyramid Scene Parsing Network (PSPNet) [132] uses the Spatial Pyramid Pooling to better represent the global contextual information of a scene. Residual Network (ResNet) is used to extract the features and encode it into the latent space. The pyramid pooling identifies patterns in these feature maps that occur at varying levels of scaling. The outputs of these pyramid modules are upsampled and merged with the initial layers to capture varying levels of information resolution.

Atrous Convolutions

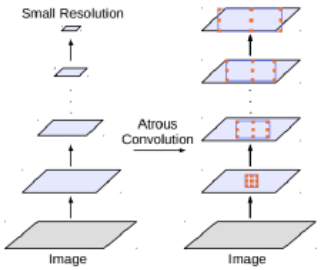


Figure 2.11: Generalized depiction of a typical architecture that uses Atrous Convolutions[19]

Some of the more recent network architectures use a new method that uses atrous convolutions [19] to recover spatial details rather than Deconvolutions like the older methods. This method introduces another parameter into the convolution layers called the 'dilation rate'. The atrous convolutions (or dilated convolution) (Fig.2.12) of a signal $x(i)$ is defined as $y_i = \sum_{k=1}^{K} x[i + rk]w[k]$, where r specifies the space between the kernel weights w, also termed as dilation rate. In other words, a kernel of size 3×3 with $r = 2$ will produce the same map size as a kernel with size 5×5 meanwhile retaining the original number of parameters of 9 which will result in an enlarged map with minimum computations. Atrous convolutions have seen popularity in many state-of-the-art papers in the field of real-time segmentation. A few of the notable publications include the DeepLab family of networks [18], Multi-scale

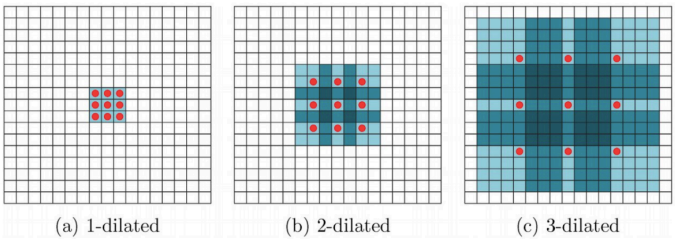


Figure 2.12: Visualization of different dilation rates of a 3×3 kernel

context aggregation [129], Hybrid dilated convolution [122], densely connected Atrous Spatial Pyramid Pooling (DenseASPP)[127],and the ENet [90].

Section.2.4 introduces another computer vision problem, Object Detection and section.2.5 describes how segmentation can be used to improve the accuracy of Object Detection models.

2.4. Object Detection

Object detection is a popular computer vision task that involves detecting instances of objects in an image that belong to a particular class. The task essentially locates and classifies class objects. As we have seen in the previous sections, advances in computer vision and deep learning have reached a saturation point with regards to the classification problem which caused a shift in focus to topics like adversarial image generation, neural style transfer, visual storytelling, and the topics of this thesis, object detection and tracking. This section tracks the significant milestones in the field of object detection up until the state of the art. This will pave way to the more specific problem of vehicle detection and tracking.

Most of the earliest object detectors were built around 20 years ago based on handcrafted features. This was necessary at that time as powerful computational resources were not widely available which resulted in algorithms that designed sophisticated feature representation algorithms. P.Viola and M.Jones achieved a real-time detection of human faces without using standard constraints like skin color segmentation which was a hundred times fastest that the state-of-the-art at that time [119] [120]. The detector was later dubbed ”Viola-Jones (VJ) Detector” by the research community honoring their contributions.

The algorithm uses a simple method of sliding windows for the detection process, which slides through all possible locations and scales. However, the computations behind this simple process was too computationally expensive for the time which resulted in most of the algorithms of the time to be extremely slow. VJ Detector sped up the process by incorporating techniques like integral image, feature selection and detection cascades.

Histogram of Oriented Gradients (HOG) feature descriptor was introduced by Dalal *et al.* in 2005 [27]. Feature invariance and non-linearity is balanced by the computation on a grid of cells that are uniformly spaced and dense in nature to improve the accuracy. Though HOG was initially designed for detection of pedestrians, it was shown to work well for other object classes too. To detect object of multiple sizes, the input image is scaled keeping the bounding box size constant. The HOG has been an important part of many pieces of research on object detection [32, 33, 74] and computer vision applications.

As the performance of handcrafted features saturated at around 2010, the rebirth of Convolutional Neural Networks and its ability to learn high level features robustly, garnered interest and gave birth to a whole family of CNN based object detectors.

CNN based Detectors

The earliest breakthrough achieved by a CNN-based object detector was proposed by R.Girshik *et al.* named Regions with CNN features (RCNN) for object detection [39, 40]. The **RCNN** (fig.2.13) detection pipeline starts with the generation of object proposals by selective search

[117]. The proposals are then scaled to a pre-defined size and fed into a pre-trained model to extract features. In the end, the presence of objects in the proposal and its eventual classification is handled by an SVM classifier.The RCNN yielded a significant performance boost on the PASCAL VOC 2007 dataset with the mean Average Precision (mAP) jumping to 58.5% from the then top performance of 33.7% achieved by DPM-v5 [41] .

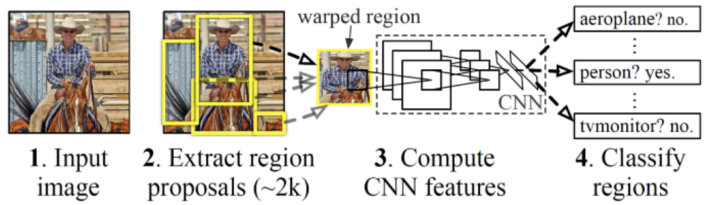


Figure 2.13: RCNN Framework [40]

Though RCNN made significant progress, it suffered from the problem of computing redundant features in the case of overlapping proposals result in extremely slow detection speeds (14s per frame on a GPU). SPPNet [45] was introduced later that year to overcome the problem.

The major advantage **SPPNet** had over other peer networks is that it was not constrained by the input image dimensions due to the presence of the Spatial Pyramid Pooling layer. The detection pipeline calculated the the entire feature map only a single time and the representations of fixed-size extracted from this feature map and used to train the model which made SPPNet about 20 times faster than RCNN without compromising on the detection accuracy (VOC07 mAP=59.2%).

In 2015, the authors of RCNN proposed an improvement model called **Fast RCNN** detector [38](Fig.2.14). The speed improvement was attributed to the ability to parallely train both a regressor and detector under the same hyperparameters of the network. Faster RCNN showed an 11.5 % improvement of mAP from the traditional RCNN on the VOC07 dataset while it clocked 200 times faster than its predecessor.
Though Fast RCNN exploits the characteristics of both RCNN and SPPNet, the detection

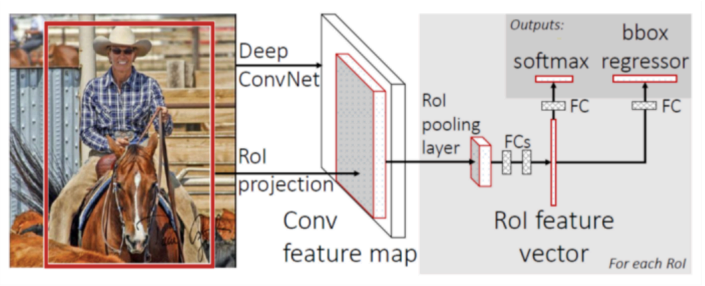


Figure 2.14: Fast-RCNN Framework [38]

speed was still limited by the proposal detections. This was solved by Faster RCNN by generation of object proposals by CNN.

The **Faster RCNN** [97](Fig.2.15) detector was proposed by Ren *et al.* shortly after the Fast RCNN. This was the first end-to-end , near-realtime deep-learning detector which brought the VOC07 mAP upto 73.2% running at 17 fps with ZFNet [130]. Near cost-free region proposals were generated using the Region Proposal Networks (RPN) and the unification the individual blocks like proposal detection, feature extraction, box regression etc. enabled the researchers to create such a fast end-to-end framework.

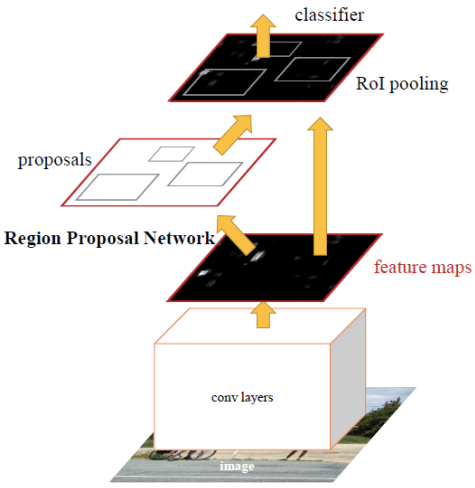


Figure 2.15: Faster-RCNN Framework [97]

Though the speed bottleneck was essentially broken through, there were quite a few computational redundancies in the detection stage. Although, many improvement models were subsequently proposed (RFCN [26] and Light head RCNN [62]), Feature Pyramid Networks (FPN) [64] proposed by T.Y.Lin *et al.* showed most promise among the detectors based off of Faster RCNN. This piece of research enabled building high-level semantics at all scales which earned it the position of state-of-the-art in MSCOCO dataset (COCO mAP@.5=59.1%). This is the reason why most of the latest detectors have made the FPN as a standard building block.

Despite the significant improvements of accuracy, widespread use of the methods in real-world applications were limited due to computational limitations of the commonly used mobile platforms. This brought the researchers to focus from *accuracy-oriented* solutions to a *speed oriented* one.

You Only Look Once, commonly known as **YOLO**[96], was proposed in 2015 by Redmon Joseph *et al.* This was regarded as the earliest deep-learning based one-stage detector. The family of YOLO (Fig.2.16) was very fast: The tiny-YOLO (a version designed for speed) ran at 155FPS achieving a mAP=52.7% on the VOC07 dataset. As the name suggests, the authors, completely rewrote the object detection paradigm from ”detection and verification” to the application of a single neural network to the whole image. This is done by dividing the whole image into grid cells and predicting bounding boxes and probabilities for each region simultaneously. Further versions of YOLO managed to improve detection accuracy while managing to keep the high detection speed.
Despite, the extremely fast detection speed, YOLO suffered a drawback. It suffered a significant drop in localization accuracy when compared to two-stage detectors especially when the objects in question were small. Later the Single Shot Detector addressed this problem.

Single Shot MultiBox Detector (**SSD**) [68] (Fig.2.17) was introduced in 2015 by Liu *et al.* This was the next generation of single-stage detectors to be introduced in the deep learning era. The main contribution of the paper was that SSD introduced multi-reference and multi-resolution detection techniques which significantly improved detection accuracy , especially for small objects. The SSD achieved a VOC07 mAP=76.8 % for a fast version that runs at 59fps.
In 2017, T.Y.Lin *et al.* claimed to have discovered the reason why the accuracy of one-stage detectors always trailed their two-stage counterparts. They had claimed that the extreme foreground -background class imbalance was the main reason for the gap. They attempted to bridge the gap using RetinaNet[65](Fig.2.18) which introduced a novel loss function designed

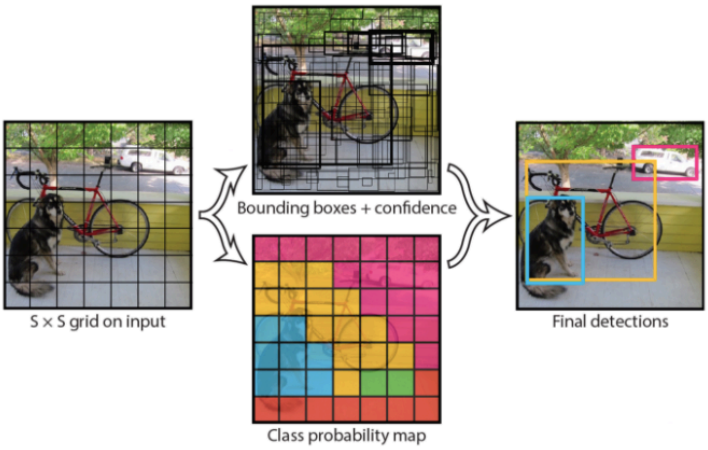


Figure 2.16: YOLO Framework [96]

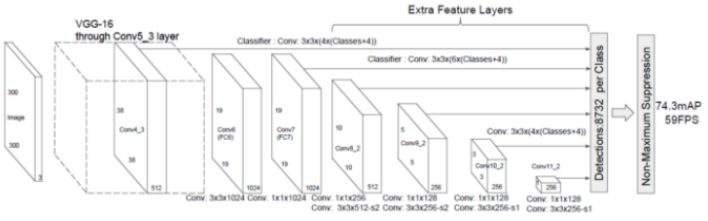


Figure 2.17: SSD Framework [68]

specifically for the task of object detection called 'Focal Loss'. This loss is calculated by reshaping the cross-entropy loss such that extra attention is paid to wrongly classified and difficult examples during training. This resulted in RetinaNet achieving similar performance levels to that of the two-stage detectors while maintaining high processing speeds (COCO mAP@.5=59.1 %).

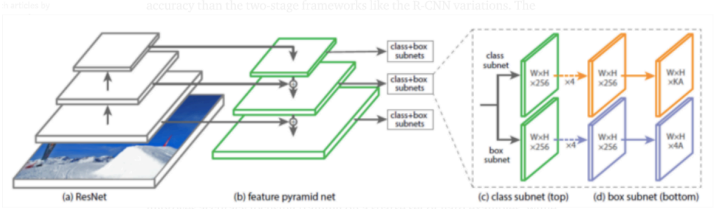


Figure 2.18: RetinaNet Framework [65]

In May 2020, the final version of YOLO was proposed by A.Bochkovskiy. Named **YOLOv4**[12], it is currently considered as the state-of-the-art real-time object detector. It uses the combinations of various features such as Weighted Residual Connections (WRC), Cross-Stage-Partial-connections (CSP), Cross mini-Batch Normalization, Self-adversarial Training, Mish Activation, Mosaic Data Augmentation, Drop-Block Regularization and CIoU loss. These are categorised by the authors into *Bag of Freebies* (Methods that improve the accuracy of detections which may influence the training time but inference time is not affected) and *Bag of Specials* (Methods that can be applied which slightly increases the inference time but significantly improves the detection accuracy). This results in performance much better and faster than its peer networks (COCO mAP@.5=65.2 %).
Recent research has shown that combining two related tasks together can boost up the accuracy of both the tasks simultaneously by the use of multi-task loss functions. This is

discussed further in section 2.5.

2.5. Multi-Task Learning

Multi-task learning is a method employed to improve learning efficiency and prediction accuracy by learning multiple objectives from a shared representation [17]. The use of multi-task learning has been prevalent in many applications like natural language processing and speech recognition. In the setting of visual scene understanding in computer vision, multi-task learning method has been used to improve object detection performance with the help of semantic segmentation.

Semantic segmentation has been shown to improve object detection due to 3 main reasons:

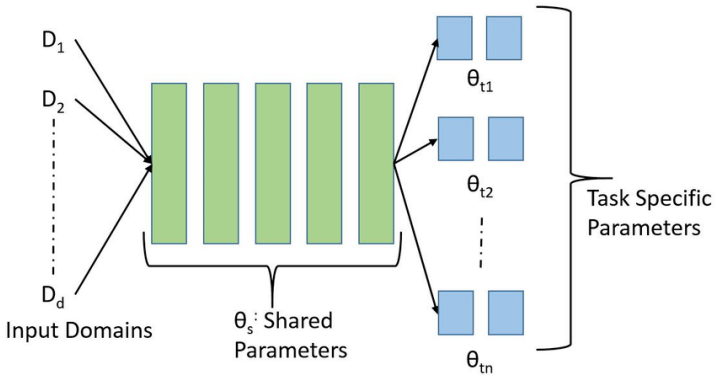


Figure 2.19: A general multitask learning framework for deep CNN architecture. The lower layers are shared among all the tasks and input domains.[94]

- *Improves Category Recognition*: Human visual cognition consists of edges and boundaries [7, 88]. In the setting of scene understanding, objects(eg. car, pedestrian, tree etc.) and background artifacts (sky, grass, water etc.) differ in the fact that the former has well defined boundaries within an image frame while the latter does not. As semantic segmentation clearly distinguishes these boundaries, it could help in category recognition.

- *Improves location Accuracy*: A clear and well established visual boundary is what defines an instance of an object in a ground truth. Some objects have special characteristic parts (for example long feathers of a peacock) that may result in incorrect or low location accuracy. As semantic segmentation problem encodes these boundaries very well, learning segmentation along with detection improves the localization accuracy.

- *Context Embedding*: Most of the objects in the scene understanding context have a standard pattern of surrounding backgrounds such as car is almost always found on road and not the sky. These arrangements constitute the context of an object which helps the object detection to improve object confidence accuracy.

There are two major training methodology employed when using segmentation to improve detection. One way is end-to-end learning with enriched features. This means that the segmentation is used as a fixed feature extractor which is integrated to the detection as additional features[15, 37, 106]. Though this method is easy to implement, a major drawback is the heavy computation cost of always having to calculate the segmentation even if only the detection is of interest during inference.

However, another training methodology introduces a segmentation head on top of the detection framework which trains the network with a multi-task loss function [15, 47]. Here, the input to the network produces two or more outputs each with its own loss function. During the backpropagation step, the optimizer tries to strike a balance between minimizing all the loss functions. This results in a sort of tug-of-war between the parameters in the backbone of the architecture which are shared between the tasks. This is beneficial when the tasks are

related as the relationship of the tasks can reduce the search space of the parameters in the backbone.

As long as there is no regularization connections between the segmentation and detection heads, the segmentation head can be decoupled during inference, the detection speed of the detector will not be affected as the computations required to calculate the segmentation map is no longer needed.

2.6. Multi-Object Tracking

Object tracking in a sequence of visual data is commonly known as Visual Object Tracking (VOT). Object tracking has been a focal point and have inspired many pieces of research in the past years due to the challenges faced by large variations in viewpoint, illuminations and occlusion. VOT tracking is broadly classified into two categories based on the number of objects tracked in the sequence: Single-Object Tracker (SOT) and Multi-Object Tracker (MOT). Kalmann and Particle filtering methods have been employed widely for single-object tracking tasks. These methods consider the object speed and position of motion which result in accurate object tracking [24] [87].
Bochinski *et al.*[11] proposed a simple Intersection over Union (IoU) based matching done by overlapping frames. This resulted in very fast tracking due to the use of positional information, however, the accuracy of this method suffered when used for complex objects or in difficult scenes. A similar position based tracking method gained popularity as an online tracker which means that the tracker learns the features of the object while performing the tracking task. This was called the Simple Online and Real-time Tracking (SORT) algorithm[9]. This predicts object location using Kalmann filtering using the location of the object in the previous frame. The Hungarian method is used to match the objects in the predicted locations and the IoU score of the detected and predicted bounding boxes are used as an affinity measure. The accuracy and precision of SORT outperforms the traditional IoU based methods but has a tendency to produce more false positives. DeepSORT [124] partially solved this issue by introducing re-identification to affinity between tracks and detection. Recurrent Neural Networks (RNNs) that use a combination of features, motion and affinity information have shown promising tracking performance [102]. Here deep learning methods are used for MOT for both object detection and affinity modeling by re-identification approach. Other deep learning based trackers relying on Correlation Fliters (CF) have shown greater performance accuracy when compared to peers using keypoint matching[54].

Multi-Domain Network (MDN)[81] makes use of the multi-task learning methodology to improve its tracking performances in different domains. While the shared layers, are trained in an offline fashion, the domain-specific layers are trained online. This results in a highly accurate tracking system but the methodology is rather slow. This method is recommended only if the accuracy of tracking is of much importance.

GOTURN [48] was an algorithm designed to improve the tracking speed while preserving the accuracy of tracking. The CNN layers of GOTURN are pre-trained on sequences images and video frames with bounding box annotations. These weights are frozen and used during inference without online training. This enables the algorithm to reach speeds of up to 100 FPS. The overview of the GOTURN algorithm is shown in Fig.2.20.

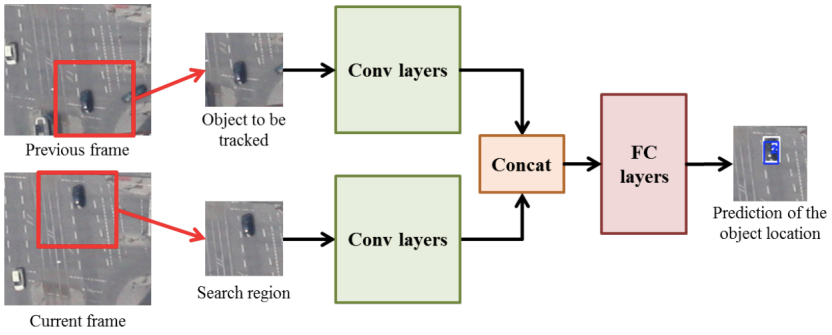


Figure 2.20: An overview of the GOTURN algorithm.[48]

2.7. Vehicle Speed Estimation

Visual speed estimation is done usually by tracking the objects through sequential frames. The displacement of the objects in pixels per second is converted to an inertial frame of reference using the camera's estimated pose. Speed estimation of tracked objects is often considered a sub-task when compared to object detection and tracking as it involves only the conversion of the speed of the bounding boxes across the frames to inertial frame. Due to this, it has been a neglected field in most traffic monitoring research pieces. The rise of vehicle speed estimation can be traced back to the beginning of the rise of computer vision applications for traffic monitoring. Most of the early methods involve using cameras mounted on road infrastructures that monitor and track vehicles[104][133]. The drawbacks in these methods is that they do not need to address the issue of a dynamic environment as the camera is fixed. This issue was addressed by Jing Li *et al.*[61] where they estimated the vehicle velocities from UAV video using motion compensations and priors. However, the method was tailor made for nadir view video frames and ran at a low frame rate on a GPU accelerated main-frame.

3

Vehicle Detection

The process of designing an end-to-end system pipeline that estimates the vehicle speed is divided into three major divisions: Vehicle Detection, Tracking and Speed Estimation. This chapter deals with the first step in the pipeline, that is vehicle detection. Object detection problems vary in difficulty depending on the object and scene complexity, which entails that the methods developed for object detection in ground context differs from that of aerial imagery.Therefore, there is a need to customize the object detector to extract the maximum performance out of object detectors benchmarked on standard ground datasets. One of the methods of customizing to boost performance is by training an object detector along with a segmentation head with shared backbone parameters. This is called *Multi-task Learning*. This chapter enumerates the process involved in building the custom object detector for vehicles using the concept of multi-task learning.

In order to build the architecture for the multi-task network, a modular approach is used. A broad template used to approach the architecture in a modular way is depicted in Fig.3.1. This schematic shows the propagation of image data from left to right. The image is fed into a *backbone module* which contains all the parameters that are shared between the object detection and segmentation tasks. The function of a backbone module is to extract all the features essential for both the tasks from the image and encode them into a lower dimensional representation called the *latent space*. This latent space representation is then sent to the task specific modules called *heads*, which contain parameters that are independent of the other task parameters. These task heads interpret the latent space information and present the required task outputs: detections and segmentation maps. Here it is aimed to design the best possible backbone and head modules.

The designing process starts with finding a segmentation task head that is optimized for speed and accuracy. Then the backbone and object detection task head are designed to provide maximum accuracy while managing real-time performance. The object detector and the segmentation head are individually trained to make this choice before being trained together using the multi-task methodology. The whole design is done keeping in mind that the final implementation of the model should run on embedded platforms like NVIDIA Jetson Xavier NX. The chapter begins with Section.3.1 that presents the choice of segmentation head for the multi-task network and its comparison to other commonly used segmentation decoders while section3.2 is where the choice of the object detector is presented for the detection task and compared with other state-of-the-art algorithms. Section.3.3 finally combines both of the choices and presents the improvement of performance. The chapter is concluded with further improvement of performance by performing hyperparameter tuning using the method described in section.3.4.

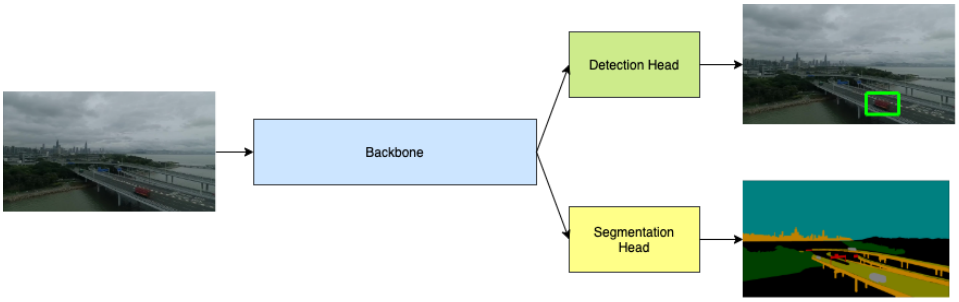


Figure 3.1: An overview of the architecture template for the Multi-task network

3.1. Semantic Segmentation Head

We start with the process of choosing the semantic segmentation head. The role of the semantic segmentor in this project is to provide contextual information to the object detector by shared representation in the latent space. In order to maximize the detection performance, there is a need for a semantic segmentor that efficiently classifies pixels into their respective categories as well as recognize small scale objects in the images. However, the effectiveness of a segmentation arm in a multitask network saturates after the segmentation reaches above a certain level of accuracy as shown in [113]. These saturation points vary with the type of tasks and the determination of this out of the scope of this research. An assumed saturation point of 0.9 average dice coefficient was chosen based on an observation made that differences in the segmentation map was barely noticeable to the naked eye for models with dice coefficient between 0.9 and 0.93. Therefore, the focus is to choose a segmentor that is small but achieves a dice coefficient score of at least 0.9 on the test dataset. This approximation is made under the assumption that higher accuracy levels focus on the fine details of the object in question which is not helpful to the extent that affects the detector as the detector only concerns itself with the location and size of the object and not the finer details like shape.

3.1.1. Model Architecture

The architecture for the segmentation is presented in table.3.2. This architecture is inspired by ENet proposed by Adam Paszke *et al.* [90] however, it is modified to get better results on the current problem statement. This is done mainly for the reason that the original ENet (Table.3.1) contains a very long backbone that consists of many atrous (or dilated) convolutional layers which results in a latent space that has a larger spatial dimension which is quickly and efficiently scaled up by a smaller decoder to generate a segmentation map. However, the object detector's latent space is much smaller than the one generated by the original ENet which requires some additional modification to the original architecture.

The encoder is not much in focus here as for the multi-task network, only the decoder is used as a branch from the object detector's backbone. For the sake of comparison, the MobileNetV3Small[50] is used as a common encoder here. The decoder network consists of bottleneck modules which have two branches: a branch that have convolutional layers and a main branch that merges with the first branch using element-wise addition.

Bottleneck:A bottleneck(Fig.3.2) in the context of deep learning, is a layer or a group of layers that reduces dimensionality of data. Bottleneck modules such as these help in forcing the backbone to retain only essential features as passing them along to the next layer which improves efficiency. In an ENet decoder bottleneck, the dimensionality is first reduced and then increased again using a deconvolutional layer. Each bottleneck module contains 3 convolutional layers. The first one is a 1×1 layer which reduces the spatial dimension of the input feature map. The second is the main convolutional layer which is a deconvolutional layer and finally, a 1×1 projection is used for expansion. A Batch Normalization[51] and PReLU[44] is placed between all convolutional layers in the bottleneck. The activations are zero-padded to match the number of feature maps. The regularizer used is the Spatial Dropout[114], with $p = 0.1$. The bias term in the 1×1 layers are not used in order to limit the calls to the kernel, resulting in faster computations. This is because cuDNN [20] uses separate kernels for

Name	Type	Output size
initial		$16 \times 512 \times 512$
bottleneck1.0	downsampling	$64 \times 128 \times 128$
$4\times$ bottleneck 1.x		$64 \times 128 \times 128$
bottleneck2.0	downsampling	$128 \times 64 \times 64$
bottleneck2.1		$128 \times 64 \times 64$
bottleneck2.2	dilated 2	$128 \times 64 \times 64$
bottleneck2.3	asymmetric 5	$128 \times 64 \times 64$
bottleneck2.4	dilated 4	$128 \times 64 \times 64$
bottleneck2.5		$128 \times 64 \times 64$
bottleneck2.6	dilated 8	$128 \times 64 \times 64$
bottleneck2.7	asymmetric 5	$128 \times 64 \times 64$
bottleneck2.8	dilated 16	$128 \times 64 \times 64$
Repeat stage 2, without bottleneck 2.0		
bottleneck4.0	upsampling	$64 \times 128 \times 128$
bottleneck4.1		$64 \times 128 \times 128$
bottleneck4.2		$64 \times 128 \times 128$
bottleneck5.0	upsampling	$16 \times 256 \times 256$
bottleneck5.1		$16 \times 256 \times 256$
fullconv		$C \times 512 \times 512$

Table 3.1: The original architecture of ENet proposed by Pazke *et al.* [90]. The number adjascent to the bottleneck module represents the stage which the bottleneck module belongs to.

convolution and bias addition. Also on the single main branch, Max Unpooling layers and spatial convolution layers are used in the decoder. A bare regular convolutional layer is used as the final output layer with C feature maps (12 in the case of Aeroscapes dataset).
The main modifications made on the ENet module are:

- Two additional upsampling bottleneck modules are introduced in the first stage of the decoder(bottlenecks 4.1 and 4.2) to compensate for the increase in the latent space dimensionality due to the lack of dilated convolutions in the backbone.
- 3 skip connections are introduced from the backbone to the outputs of bottlenecks 4.0, 4.2 and 5.0 respectively to enable learning of smaller scale features (as used in UNet[100])

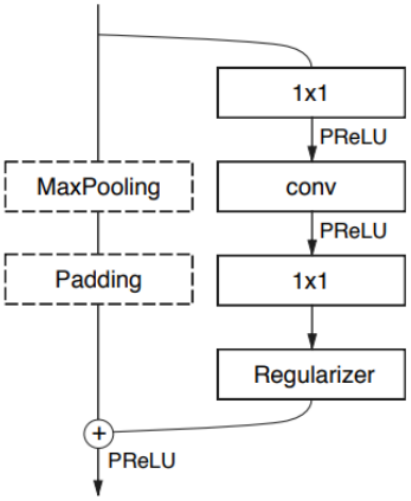


Figure 3.2: ENet bottleneck module

Name	Type	Output size
MobileNetV3Small	Backbone/Encoder	$16 \times 16 \times 576$
bottleneck4.0	upsampling	$32 \times 32 \times 256$
Concatenate	skip connection from backbone	$32 \times 32 \times 544$
bottleneck4.1	upsampling	$64 \times 64 \times 128$
bottleneck4.2	upsampling	$128 \times 128 \times 64$
Concatenate	skip connection from backbone	$128 \times 128 \times 136$
bottleneck4.3		$128 \times 128 \times 64$
bottleneck4.4		$128 \times 128 \times 64$
bottleneck5.0	upsampling	$256 \times 256 \times 16$
Concatenate	skip connection from backbone	$256 \times 256 \times 32$
bottleneck5.1		$256 \times 256 \times 16$
Deconvolution		$512 \times 512 \times C$

Table 3.2: Architecture of the Modified ENet for semantic segmentation. 2 additional bottleneck modules are introduced in stage 4 of the decoder along with 3 skip connections from the encoder (additional modules are highlighted)

3.1.2. Dataset preparation for Multi-class Semantic Segmentation

Supervised Learning method in Deep learning algorithms require training, validation and testing data. Most of the algorithms require a large number of training examples with good degree of context variety to effectively generalize the task at hand and prevent data overfitting.
Though multi-class segmentation is mainly used in this project as a performance enhancement technique for the object detector, a good segmentor can be instrumental in maximizing the potential of the object detector.
Semantic Segmentation has seen great improvements in the context of ground images but the scarcity of aerial datasets has highly limited the progress in developing methods for aerial images. A few well known aerial image semantic segmentation datasets are Inria Aerial Image Labeling Dataset [72], DLR-SkyScapes [2], Air-Ground-KITTI [76]and UAVID [71].However, in the context of aerial traffic monitoring, there are a few constraints that need to be taken into consideration.

- *Altitude*: As the system is designed to be implemented on a Hybrid-VTOL drone, the images should be captured between 20m and 150m (maximum allowed operating altitude by the Dutch Government[118]). This constraint eliminates most of the aerial images taken from an airplane and satellite images.
- *Number of Classes*: Higher the number of classes, greater the contextual awareness the object detector can learn. With this in mind, a minimum of 5 annotated classes are set as a requirement. This eliminates datasets with only road and/or buildings segmented.
- *Viewing Angle*: A charter on Drone Regulations by the Dutch Government states "*the flight is not performed over areas of contiguous buildings or structures, including industrial and port areas, or over crowds of people or over railway tracks or paved public roads accessible to motor vehicles*"[118]. This essentially renders all orthographic-only image datasets unusable in our context. There for only datasets that view the scene in a rough isometric as well as orthographic view would be used.

These constraints deplete an already scarce pool of candidate datasets. However, there are a few multi-class segmentation datasets that fall into this category which can be used to design and fine-tune a segmentor.
The **Aeroscapes**[82] dataset is an aerial semantic segmentation benchmark which is collected using a DJI Phantom3 fleet of drones between the altitudes of 5 and 50m. The dataset consists of 3269 720p non-sequential images and their respective 11 class annotations. The dataset is split in the ratio of 70:20:10 as training, validation and testing datasets. In other words, 2288 training samples, 654 validation samples and 327 test samples.

Once the design dataset is finalized, the data is prepared for training. The following steps are taken to prepare the data for training.

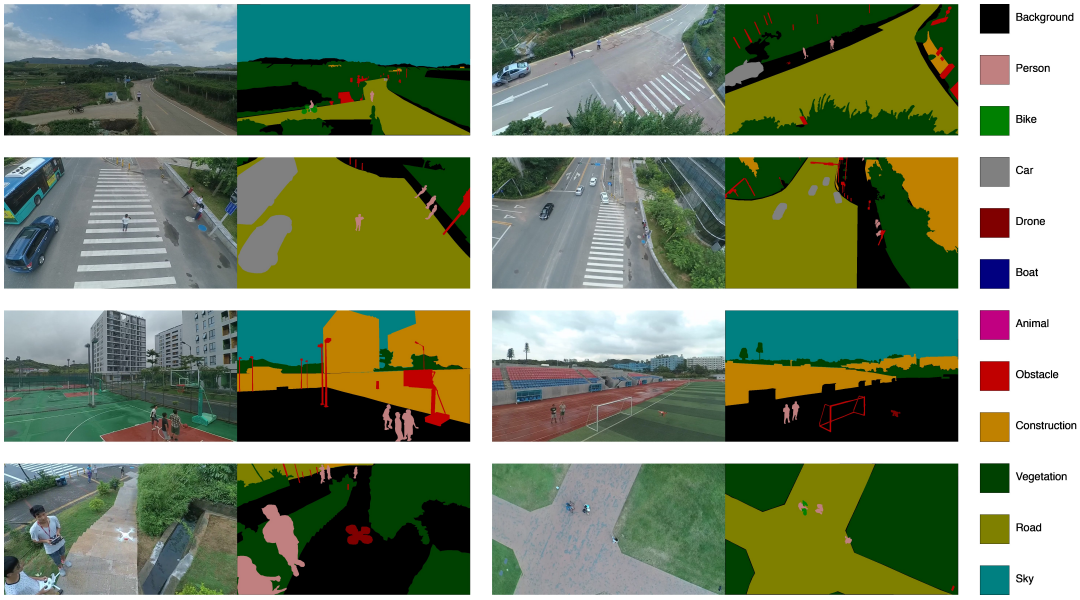


Figure 3.3: Sample images from the Aeroscapes dataset along with the annotation visualization [82]

- *Image Resize*: As a full-size (1280×720 pixels) requires a great amount of computational power during training, the image is resized to a respectable 512×512 and higher resolution images can be processed during inference by sliding a 512×512 window over the image. The resizing is done without affecting the aspect ratio by employing a method called **letterboxing**. This resizes the image to a desired dimension without loosing the aspect ratio (which holds a lot of object characteristics information) and filling the resulting empty pixels with a grey color. However, it should be noted that the resizing essentially increases the *Ground Sampling Distance*(GSD) (the distance on the ground the corresponding pixel on the image covers). This makes it difficult to identify the objects when the UAV altitude for inference is outside the range trained using the Aeroscapes dataset. This issue is resolved by using the previously mentioned sliding-window method to splice large images to smaller chunks which preserves the GSD. Thus, the model trained with 512×512 images can easily be used to infer larger image sizes.

- *Normalization*: Image Normalization is an important step in semantic segmentation as this process ensures that the pixel intensity and channel values have a similar data distribution throughout the image and the dataset. It is performed by finding the pixel-wise difference of the value and the global average of the pixels and then dividing it by the standard deviation of the original pixel values. It essentially converts an image of N channels: $I : \{\mathbb{X} \subseteq \mathbb{R}^n\} \rightarrow \{\text{Min}, \ldots, \text{Max}\}$ to a normalized image $I_N : \{\mathbb{X} \subseteq \mathbb{R}^n\} \rightarrow \{\text{NewMin}, \ldots, \text{NewMax}\}$ with pixel intensities ranging from (Min,Max) and (NewMin,NewMax) respectively, using eq.3.1.

$$I_N = (I - \text{Min})\frac{\text{newMax} - \text{newMin}}{\text{Max} - \text{Min}} + \text{newMin} \tag{3.1}$$

- *One-hot Encoding:* As annotations are mainly done by humans as of now, these annotations are usually done with different colors(which often fall in a finite set). These colors represent which class a pixel at that location in the original image belongs to. As humans, it is trivial to understand this type of classification. However, these annotations are not machine-readable. A neural network algorithm can understand 1s and 0s. Hence, we apply color mask to each of the colors corresponding to their respective class and extract a binary image for each of the classes. This will transform a 512×512×3 annotation into a 512×512×12 encoded annotation, where 12 is the number of classes.

It must be noted that Normalization is not applied to the annotations as the color value of the annotation changes that would affect the one-hot encoding process. However, all non spectral transformations on the original image must be reflected in the annotation too.

3.1.3. Training

The architecture was trained with the following model settings set:

- **Batch Size:** 5
- **Number of Epochs:** 250
- **Optimizer:** ADAM
- **Loss Function:** *Tversky Loss*[103]
- **Initial Learning Rate:** 1×10^{-3}
- **Learning Rate Decay factor:** 0.5 with a patience of 5 epochs

These setting values are chosen based on default settings used by Keras[22] platform and are not tuned to work on this task. The technical hardware and software specifications are listed below:

- **Training System Specification**: Intel Xeon Processor with 12 GB NVIDIA Titan Xp GPU
- **Software Packages**:Keras with Tensorflow 2 backend (Python3.7)
- **Evaluation Speed Bottleneck**: 1.8 GHz Intel Core i5

As it is has been shown to prevent overfitting due to dataset imbalance, the loss function used is the **Tversky Loss**[103] function. This function is based on the Tversky index[115] which is defined as:

$$S(P, G; \alpha, \beta) = \frac{|PG|}{|PG| + \alpha|P \backslash G| + \beta|G \backslash P|} \tag{3.2}$$

Where P and G are predicted and the ground truth segmentation maps respectively and α and β control the penalties of False Positives and False Negatives respectively. Based on this, the tversky loss function is formulated as:

$$T(\alpha, \beta) = \frac{\sum_{i=1}^{N} p_{0i} g_{0i}}{\sum_{i=1}^{N} p_{0i} g_{0i} + \alpha \sum_{i=1}^{N} p_{0i} g_{1i} + \beta \sum_{i=1}^{N} p_{1i} g_{0i}} \tag{3.3}$$

Where p_{1i} is the probability that the pixel does not belong to the particular class and p_{0i} is the probability that the pixel belongs to the particular class. The values of α and β were chosen to be 0.3 and 0.7 as per the original paper.

Further the evaluation is done on the test dataset using the **Dice Similarity Coefficient** measure which is defined as:

$$D(P, G) = \frac{2|PG|}{|P| + |G|} \tag{3.4}$$

3.1.4. Results and Discussion

The trained model is evaluated against other commonly used semantic segmentation models like SegNet[4], UNet[100] and the parent ENet[90]. As the decoder is the part of the model being evaluated, MobilenetV3Small[50] is used as a control backbone across all the tested models. This choice was made with the sole purpose of speeding up the training time and reducing GPU load. The models were evaluated on a CPU which served as the speed bottleneck which helped determine its suitability to be deployed on mobile platforms without GPU accelerated computing. SegNet[4] provided a very poor processing speed benchmark during the evaluation. This is attributed to the large memory requirement of the network .

Decoder	Average Tversky Loss	Average Accuracy	Average Dice Coefficient	FPS
Segnet[4]	23.15	0.8697	0.8968	-
U-Net [100]	23.115	0.9004	0.9208	2.3
ENet [90]	23.141	0.874	0.9019	4.1
Modified ENet	23.149	0.8855	0.9117	3.2

Table 3.3: Comparison of the Modified ENet with other commonly used segmentation models.The speed was benchmarked on an Intel i5 CPU

However, this network despite its slow segmentation, provides a good accuracy which helps set a benchmark to compare the other algorithms to. U-Net and ENet comparable in terms of performance by the fact that U-Net is able to capture small-scale features due to the skip connections from the encoder where as the ENet is able to scale up the image from the latent space with a small but efficient decoder which speeds up the inference time.

The skip connections introduced in the modified ENet decoder enables the model to recognize objects in small scale even with low resolutions like 512×512, which can be evident from the Figures 3.4 and 3.5. Table 3.3 shows that with a little sacrifice of frame rate, the modified ENet model gives an average Dice Coefficient of just above 90%. This drop in latency would not be of any consequence if the decoder is only used to boost object detection performance during training and later decoupled during inference. This decoder architecture is shallow when compared to the state-of-the-art, however, as discussed earlier, the performance of it would provide a reasonable guide for the object detector backbone to learn contextual information in an efficient way.

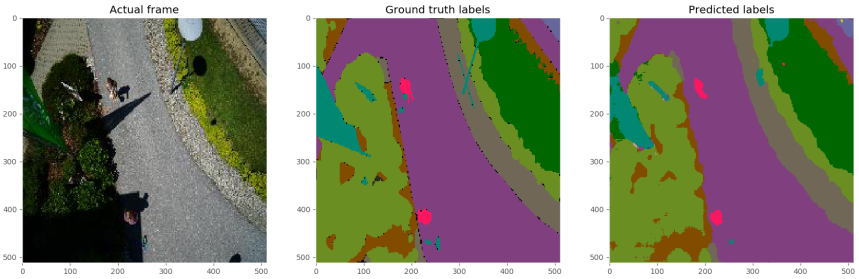


Figure 3.4: ENet decoder with MobilenetV3Small backbone

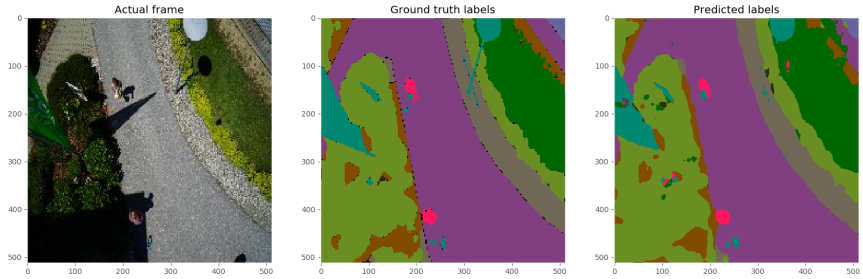


Figure 3.5: Modified ENet decoder introduced in this research with MobilenetV3Small backbone

3.2. Object Detection Head

Object Detection research in the past years has focused more on improving accuracy on carefully curated datasets at large costs of computational resources and time. However, recently, this trend has been shifting towards faster detection speeds while maintaining the accuracy levels. Recent one-stage detectors have shown promise in high-accuracy and low-latency detection with successful implementation on mobile platforms like NVIDIA Jetson TX2 and Xavier NX.

3.2.1. Model Architecture

YOLOv4 [12] is the obvious choice of detector as it is currently the state-of-the-art real-time object detector which has been shown to reach two-stage detector level accuracy with significantly lower computational cost. The Darknet implementation of YOLOv4 on the Jetson Xavier NX runs at around 8 FPS and at 2 FPS on the Jetson Nano on a 512x512 frame size. The research customizes a light version of YOLOv4 that has at least achieve 20FPS with a good and usable accuracy on the Jetson NX. YOLOv4 architecture is split into 3 parts: Backbone, Neck and the head.

Backbone

The YOLOv4 uses a **CSPDarknet53**[121] backbone which was shown to be the backbone that most effectively captures features for object detection [12]. This backbone consists of two modules: Space-to-Depth Transformation and the CSP Bottleneck module.

- **Space-to-Depth Transformation:** This module is inspired by input procedure described in TResNet[99] (dedicated to improving GPU inference throughput). The paper argues that as most neural networks aim to quickly reduce the input dimensions at the initial layers, this can be efficiently achieved by using a **Space-to-Depth Transformation** layer which arranges blocks of spatial data into depth.This results in the transformation of data from the form $[b, h, w, c] \rightarrow [b, h/2, w/2, 4c]$. Where b is the batch size, h is the height of the images, w is width and c is the number of channels of the image. This is followed by a simple 1×1 `Conv` layer to match channel depth of the network. This type of dimensionality transformation resulted in 4.2% increase in computational speed on ResNet50 backbone with 0.1% increase in the Top-1 accuracy.[99]

- **Cross-Stage Partial Bottleneck:** The Cross-Stage Partial(CSP) Bottleneck is designed based on the CSPNet [121] which was proposed as a network that respects the variability of the gradients by integrating feature maps from the beginning and the end of a network stage, which reduces computations by 20% without any detrimental effect on the accuracy. A CSP block integrated with a residual block of Darknet53 architecture is visualized in Fig.3.6

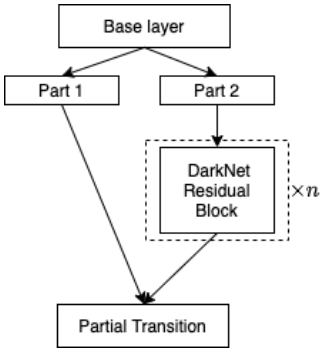


Figure 3.6: Integration of CSP module with a native Darknet53 Residual block

Name	Times Repeated	Filter size
Input	1	-
Space-to-Depth	1	64
Conv	1	128
BottleneckCSP	1	128
Conv	1	256
BottleneckCSP	3	256
Conv	1	512
BottleneckCSP	3	512
Conv	1	1024

Table 3.4: The architecture of the lite version of the CSPDarknet53 backbone

The custom light CSPDarknet53 obtained by reducing the number of convolutional residual blocks of the Darknet53 and replacing them by BottleneckCSP modules. Though the number of convolutional layers are almost the same, the CSP blocks have higher inference speed as compared to the native convolutional-residual blocks of Darknet53. The standard convolutional block (convolutional layer+batch normalization) and the BottleneckCSP modules are placed in an alternating way to optimize speed and accuracy of the standard and bottleneck layers as per BlockType selection experiments by Ridnick *et al.*[99]

Detection Head: YOLO Neck and Head

The complete detection head consists of 2 parts:The YOLOv4 Neck and the YOLOv3 Head. The neck consists of the following modules: Path-Aggregation Network or PANet[66] and Spatial Pyramid Pooling Module or SPP[45]. The function of the neck is to combine various scales of features that are generated in the different levels of the backbone. This is done by attaching an SPP module to the backbone in order to increase the receptive field and choose the most important features from the backbone. These features are then fed to the PANet that does the feature aggregation(Fig.3.7). The YOLO head generates detections from the multi-

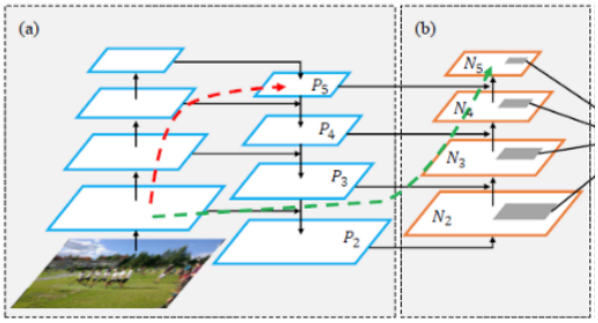


Figure 3.7: The original PANet that is used as a feature aggregator in the architecture.[66]

level aggregated features generated by the PANet. The head used is the same head used by YOLOv3[96]. The PANet generates 3 levels of features of size 13x13, 26x26, and 52x52. The head divides these features into a grid of 9 cells each. Then, there are anchor boxes assigned to each grid cell in each level. There are 9 different anchor boxes defined, each with its own aspect ratio with each feature level being assigned 3 anchor boxes. The anchor boxes that are closest to the ground truth are retained and rest are discarded during the inference by the process called *non-maximum suppression.*

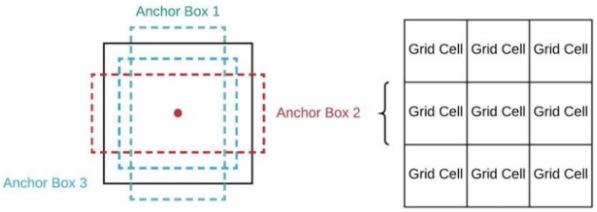


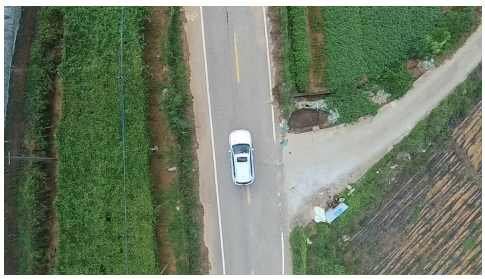
Figure 3.8: Examples of Anchors and how they are initialized on to grid cells(black)

3.2.2. Dataset preparation for Object Detection

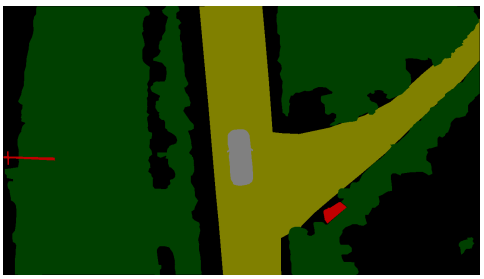
As the task requirements dictate to perform multi-task training for both multi-class semantic segmentation and object detection of class vehicles, a dataset with both task annotations are required. However, keeping view of the constraints discussed in 3.1.2, there are no available datasets with multi-class semantic segmentation as well as vehicle detection annotations on the same training image. Thus the *Aeroscapes* dataset was annotated for vehicle class using the existing multi-class segmentation ground-truth visualization.

The steps involved in the automatic label generation methodology is enumerated below:

1. A training image is retrieved from the dataset and its corresponding label visualization image is taken in for processing.



(a) The training image sample from Aeroscapes [82]



(b) Multi-class segmentation ground truth visualization

2. The colour ID (R,G and B values) of the vehicle class is extracted from the class dictionary (`Grey[128,128,128]`) and a binary color mask is generated(Fig.3.10) that isolated vehicle class pixels. The binary image is then eroded with a kernel of size 10 pixels by 10 pixels so that nearby car classes do not fuse and merge into a single blob.

3. The contour is extracted from the resulting mask using the computational edge detection method proposed by J.Canny [16]. The threshold range for this is set between 1 and 2. A padding of thickness 1 pixel is recommended in order to obtain a complete closed contour when the object to be annotated is at the edge of the image. Multiple contours are extracted in case of multiple object instances.

4. Once the contour is extracted, a bounding box can be fit over this contour by finding the `xmin,ymin,xmax and ymax` of each contours(1 here) in the image.

Figure 3.10: Generated color mask

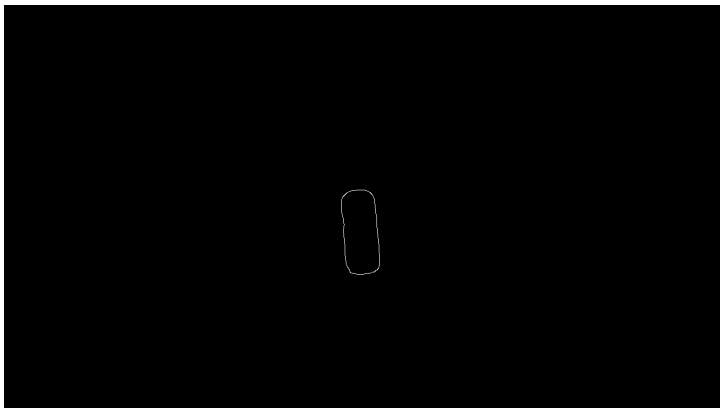


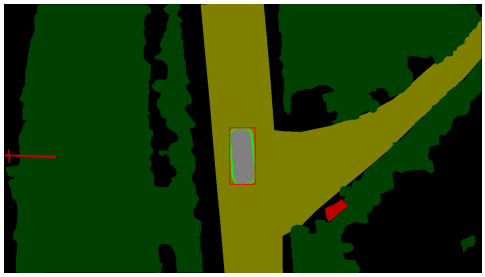
Figure 3.11: Canny-Edge Contour

5. Once the bounding boxes are extracted, the box information can be written into the annotation file in either PASCAL VOC format or YOLO format.

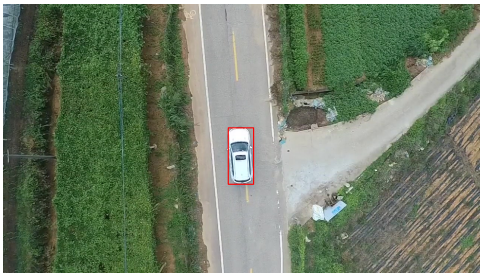
There are a few images in the dataset where the vehicle instances are clustered together which shows up as a big blob in the segmentation visualization. These images were manually annotated in the same format as the automatic method.
Once all of the data is annotated and verified, dataset augmentation techniques are used to balance the dataset. These methods have been heavily inspired by the *Bag-of-Freebies* introduced in [12], which essentially refers to a set of techniques which improves the model's generalizing ability. The geometric transformations performed on the images are replicated on the annotations but the spectral transformations are only performed on the images. The augmentation is performed for each pass through the dataset which results in a different dataset being trained during each forward pass.

The following image augmentation techniques are used to prepare the data for training:

- *Letterbox Resize:* As explained in section.3.1.2, letterboxing resizes the image while maintaining the aspect ratio of the image which makes sure that the image objects are not distorted by the resizing. This is applied to all the images uniformly.
- *Random Resize and Crop:* Training images are randomly chosen and are reshaped to a random aspect ratio. The image is then cropped to the target image size (512×512).
- *Probabilistic Horizontal Flip:* The training images are chosen and flipped about the y-axis based on a pre-defined probability.
- *Random Brightness:* The intensity of each pixel changed uniformly across the image by a random level.



(a) The estimated box fit on the label visualization



(b) The estimated box translated onto the original image

- *Random Color Level Adjustment:* The chroma of an image is enhanced by a random value.
- *Random Contrast:* A random contrast value is randomly applied to a randomly chosen image
- *Random Sharpness:* A random sharpness value is randomly applied to a randomly chosen image
- *Random Grayscale:* A randomly chosen images is converted to a 3-channel grayscale image.
- *Probabilistic HSV Distort:* The hue,saturation and value of all the pixels are changed by a set of probabilities.
- *Random Mosaic Augment:* The mosaic augmentation technique arranges 4 different images, at different scales, into a single one that encourages the model to learn small scale features and also helps it to generalize the variations present in the images. This method was introduced in the YOLOv4 paper [12].

Figure 3.13: Mosaic data augmentation introduced in YOLOv4 [12]

Other image augmentation methods introduced in [12] like Vertical Flip, Shear and MixUp essentially generates images that have negligible chance of being observed in reality and hence were not implemented. Vertical flip augmentation does not make sense to include unless the UAV is acrobatic in nature and ends up flying up-side-down. Shear augmentation modifies the aspect ratio of the vehicle which does not reflect the reality of how vehicles look in the wild. Finally, MixUp augmentation mixes up two images by changing the transparency of one and overlaying it on the other. This was introduced to improve multi-class detections of the original model benchmarked on PascalVOC and MSCOCO datasets. This does not make sense in this research context as there is only one class.
Apart from data augmentation, a set of anchor boxes have to be determined for a particular

dataset to be able to effectively detect objects. Anchor boxes are defined by their width and height parameters, and these boxes generate region proposals for the object through out the image in the last layers of the detector. The *k-means clustering* methodology is generally used to determine the anchor boxes.
The following algorithm is used to determine the anchors:

- *Step 1 :*The algorithm generates a TXT file containing the position of the anchor box, each of which contains $(x_j, y_j, w_j, h_j), j \in \{1, 2, \dots, N\}$ that is, the coordinates of the annotated boxes with respect to the original image, where (x_j, y_j) is the centre of the box and (w_j, h_j) is the width and height of the box respectively and N is the number of all the label boxes.

- *Step 2 :* First give k cluster center points,$(W_i, H_i), i \in \{1, 2, \dots, k\}$, where W_i and H_i are the width and height of the anchor boxes. Since the anchor boxes are not fixed, there is no (x, y) coordinates, only width and height.

- *Step 3 :*Calculate the distance d=1-IOU (labeling box, cluster center) of each label box and each cluster center point. When calculating, the center point of each label box coincides with the cluster center, so that the IOU value can be calculated.

$$d = 1 - IOU\left[(x_j, y_j, w_j, h_j), (x_j, y_j, W_i, H_i)\right], j \in \{1, 2, \dots, N\}, i \in \{1, 2, \dots, k\} \tag{3.5}$$

  Assign the anchor box to the nearest cluster center of Distance.

- *Step 4 :*After all the label boxes are allocated, the cluster center point is recalculated for each cluster using the formula below.

$$W_i' = \frac{1}{N_i}\sum w_i, H_i' = \frac{1}{N_i}\sum h_i \tag{3.6}$$

  Where N_i is the number of label boxes of the i-th cluster, which is the average of the width and height of all the label boxes in the cluster.

- *Step 5 :*Repeat steps 3 and 4 until the cluster center has a small amount of change.

Once the anchor sizes are determined, the data is ready for training. Both the data preparation method, for the segmentation and detection, is combined in order to prepare the data for the multi-task training. The dataset is split just as it was done for segmentation in the ratio of 70:20:10 as training, validation and testing datasets. In other words, 2288 training samples, 654 validation samples and 327 test samples. However, the test data is expanded further using the SODA Test Dataset.

SODA Test Dataset

Defining a test dataset as a subset of the aeroscapes dataset makes sense from a qualitative evaluation standpoint. However eventually, the designed system is planned to be implemented on the DeltaQuad drone(Fig.3.14) which uses a Sensefly SODA camera to capture images. The SODA dataset contains 52 images taken at 30,60 and 120m altitudes at 5472×3648 resolution using the DeltaQuad UAV. The images in the dataset were manually annotated for cars using the LabelImg tool[116].

3.2.3. Training

The primary focus after deciding the model architecture is to choose the loss functions involved in training and the hyperparameters. The loss used in the training of the object detector comprises of a combination of three different loss functions used to achieve their respective objectives. They are namely: Confidence loss, Class loss and Location loss. The choice of loss functions for each of the losses are discussed as follows:

- **Confidence Loss**: The **Focal loss function**[65] is used to obtain the confidence loss as it has been proven to be effective in handling the class imbalance problem.This is

Figure 3.14: SenseFly SODA camera(left) and the DeltaQuad on which it is mounted(right)

formulated as a weighted cross-entropy function. The standard cross-entropy function can be written as follows

$$CE(p,y) = \begin{cases} -\log(p), y = 1 \\ -\log(1-p), \text{ otherwise} \end{cases} \tag{3.7}$$

Where p is the predicted output and y is the ground=truth label. A simplified modification of this can be written as

$$p_t = \begin{cases} p, y = 1 \\ 1-p, \text{ otherwise} \end{cases} \tag{3.8}$$

$$CE(p,y) = CE\,(p_t) = -\log\,(p_t) \tag{3.9}$$

An additional α term is added to eq.3.9 to account for the class imbalance. It is a hyperparameter which can be used with the CE loss for cross-validation. α_t is the weight term that assumes the value of α for the positive value of class and $1-\alpha$ for the negative value of class.

$$\alpha_t = \begin{cases} \alpha, y = 1 \\ 1-\alpha, \text{ otherwise} \end{cases} \tag{3.10}$$
$$CE\,(p_t) = -\alpha_t * \log\,(p_t)$$

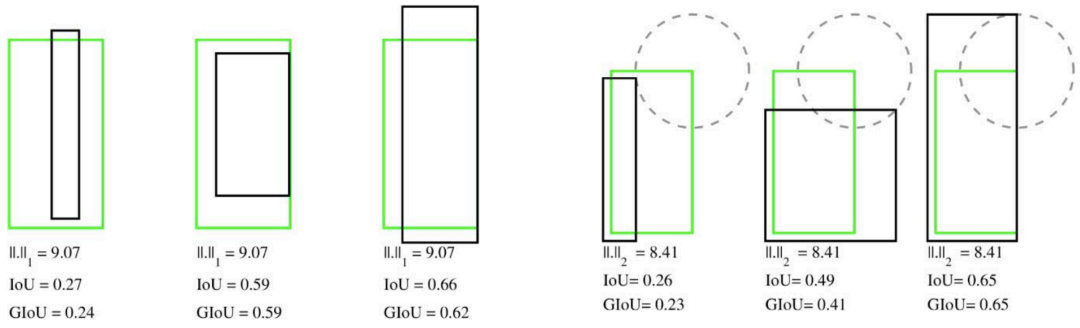
However, eq.3.10 only takes into account the class positivity. A final modification is done to this function to take hard and easy samples into account during training.

$$FL\,(p_t) = -\alpha_t\,(1-p_t)^{\gamma}\log\,(p_t) \tag{3.11}$$

Where γ is the relaxation parameter.

- **Location Loss:** The most commonly used location loss functions are the ℓ_1 norm and the ℓ_2 norm. However, these loss functions have been shown to be unreliable due to the reliance on distance measure more than the degree of overlap as evidenced by fig.**??**. It is generally common practice to train with norm function losses and Intersection over Union (IOU) is used to evaluate the overlap quality of each of the predicted bounding boxes. However, IOU cannot be used as a loss function due to two reasons. The IOU does not discriminate between predictions that are closer to the ground truth when there is no overlap. Secondly, the backpropagation algorithm requires that the loss function be differentiable throughout its domain. This is not possible in the case of IOU where no gradient can be obtained when there is no overlap and there exists a sharp corner at the point of first overlap.
To counter this drawback, Rezatofighi *et al.* proposed a new loss function called the **Generalised Intersection over Union (GIoU)**[98], which is formulated as follows.

$$GIoU = \frac{|A \cap B|}{|A \cup B|} - \frac{|C \backslash (A \cup B)|}{|C|} = IoU - \frac{|C \backslash (A \cup B)|}{|C|} \tag{3.12}$$



(a) Example of situations where the $l1$ loss is the same with different IOU and GIOU score

(b) Example of situations where the $l2$ loss is the same with different IOU and GIOU score

Figure 3.15: Evidence of the reason many neural-network based object detectors fail when using $l1$ or $l2$ loss functions [98]

Where A and B are the prediction and ground truth bounding boxes. C is the smallest convex hull that encloses both A and B. Fig.3.16 shows that the GIoU loss function

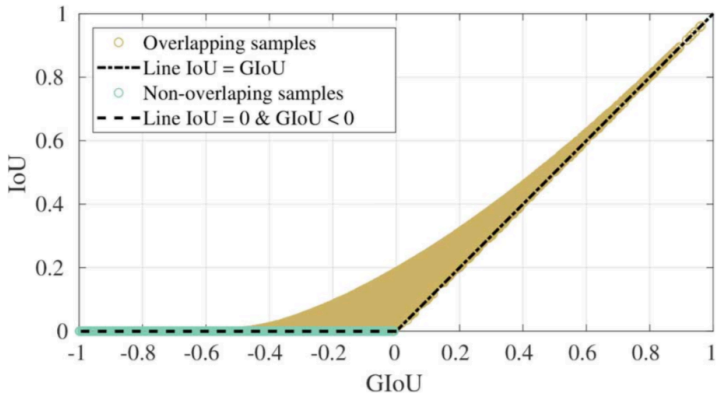


Figure 3.16: GIoU loss vs IoU with varying overlap [98]

is differentiable everywhere even when there is no overlap. It can be seen that GIoU falls in the range [-1, 1] and the negative values occur when the area enclosing both bounding boxes,C, is greater than IoU. As the IoU component increases, the value of GIoU converges to IoU.

- **Class Loss:** As there are only 1 class of object in the research scenario, the class loss is calculated by a simple binary cross-entropy function.

These losses are added with equal weighting to obtain a final loss value which the optimizer tries to minimize.

$$TotalDetectionLoss = (\texttt{giou} \times GIoU) + (\texttt{obj} \times FL) + (\texttt{cls\_pw} \times Class\_Loss) \tag{3.13}$$

Once the losses are decided, the model training settings are declared before training.The architecture was trained with the model settings used for the original YOLOv4 model:

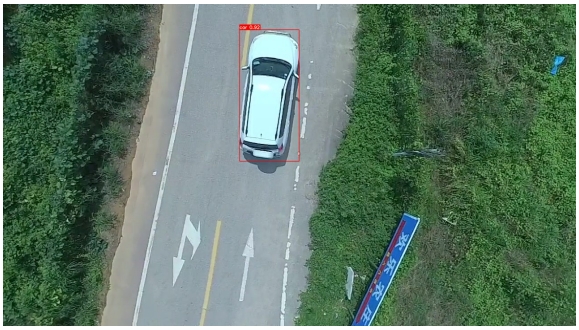
- **Batch Size:** 4 (Minimum for the Mosaic Data Augmentation)
- **Number of Epochs:** 250
- **Optimizer:** ADAM
- **Initial Learning Rate:** 1×10^{-2}
- **Learning Rate Decay factor:** 0.5 with a patience of 5 epochs

The technical hardware and software specifications are listed below:

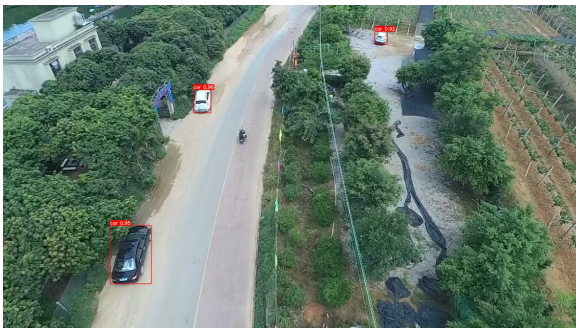
- **Training System Specification**: Intel Xeon Processor with 12 GB NVIDIA Titan Xp GPU
- **Software Packages**:Keras with Tensorflow 2 backend (Python3.7)

The custom lite-version of the CSPDarkNet53 was first trained on the ILSVCR ImageNet[59] dataset with a classifier head. This dataset with annotations for 1000 classes, consists of around 14 million images of varying sizes and the average resolution was found to be 469×387. The training was performed on the cloud system, by Google called Colaboratory, for 200 epochs. This is done so that a basic representation of common visual objects are encoded into the latent space which would serve as a guide to training the full network.
Once the weights are finalized for the backbone, the classifier head is replaced by the YOLOv4 neck and head. For the first 20 epochs of training, the weights of the backbone were frozen and only the head is trained. After 20 epochs, the backbone is unfrozen and the whole network is trained to fine-tune the accuracies. This *training strategy* is used to stabilize the losses and to prevent division by 0 errors that frequent when training on non-standardized datasets like Aeroscapes.

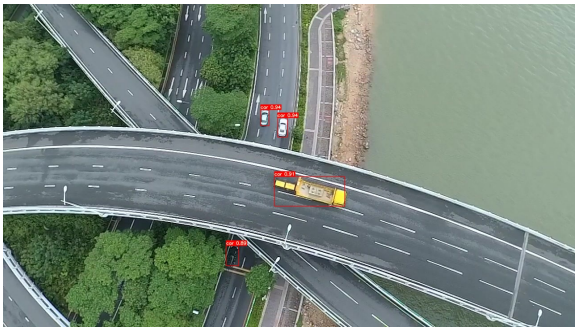
3.2.4. Results and Inference



(a) Example of a large-scale detection



(b) Example of multiple scales in a single image



(c) Examples of different vehicle types

Figure 3.17: Example detections from the test dataset

Model	Backbone	mAP@0.5	No. of Parameters	FPS
YOLOv4[12]	CSPDarkNet53 (Lite)	0.8073	27.4M	43.88
	CSPDarkNet53[12]	0.8172	37.3M	31.19
	EfficientNet(B1)[112]	0.824	58.6M	22.72
	MobileNetV3[50]	0.746	29.13M	41.5
	MobileNetV3Small[50]	0.694	11.6M	57.13
TinyYOLOv4 (Custom)	EfficientNet (B1)[112]	0.7082	27.42M	37.2
	MobileNetV3 Small[50]	0.6733	5.41M	110.11

Table 3.5: Results of YOLOv4 and Custom Tiny-YOLOv4 with different backbones and evaluated on 10% set of Aeroscapes images resized at 512×512 resolution.The evaluation was bechmarked on a single NVIDIA Titan Xp GPU with CUDA 10.1. Output models are in the Keras model format.

The trained network is evaluated on the images from the SODA dataset which were not used for training. Evaluation is done by calculating Mean Average Precision (mAP). mAP for general object detection is formulated as follows:

$$mAP = \frac{1}{|\text{ classes }|} \sum_{c \in \text{classes}} \frac{TP(c)}{TP(c) + FP(c)} \tag{3.14}$$

Where TP stands for total number of true positive in the evaluation batch and FP stands for the total number of false positives in the evaluation batch. As the current research scenario has only one class, this equation boils down to:

$$mAP = \frac{TP}{TP + FP} \tag{3.15}$$

This evaluation metric works well in classification problems however in a detection problem, mAP is related to the IoU threshold. A detection is classified as true positive if the IoU score of the predicted bounding box and the ground truth are above the declared IoU threshold.Here, the IoU threshold was set at 0.5. The detection speed evaluation was performed on an NVIDIA Titan Xp GPU on the same evaluation dataset. The speed of detection as well as the time taken for Non-Maximum Suppression (NMS) is accounted during the frame speed calculations. The YOLOv4 with the modified backbone is compared with other backbones which are commonly used and also with a custom built tiny version of YOLOv4 following the model ablation techniques used to design tiny-YOLOv3. The result for the evaluations have been tabulated in Table.3.5.

It can be observed that the modified CSPDarknet53 backbone has reduced the size of the model by 31.1% and increased the inference speed by 38% when compared to the original YOLOv4 architecture at a low cost of mAP. The EfficientNet backbone gives the maximum mAP for the Aeroscapes dataset however, the model size is above 250 megabytes which makes it hard to deploy on a mobile platform. The custom designed Tiny YOLOv4 provides small models with high inference speed which make it attractive for deployment on low-computational platforms however, the modified CSPDarknet53 model is shown to perform slightly faster than the best performing tiny-YOLOv4 model with much higher mAP.
In conclusion, the modified CSPDarknet53-YOLOv4 achieves 2 times the computational speed than the model with the highest accuracy (YOLOv4 with EfficientNet backbone) with a small sacrifice of mAP score which makes it an optimal choice for the object detection arm of the multi-task network.

3.3. Multi-task Learning

Now that we have the choice of the best model architectures for both object detector and the segmentation head, we combine the both architectures to enable multi-task learning.
The contextual information provided by the semantic segmentation is very useful in locating instances of objects in an image frame. *Multi-task Learning* is a training strategy that exploits the features extracted for one task to perform a related task or tasks. This is done by the

concept of parameter sharing where two or more tasks share the same feature extractor and negotiates the trainable parameters within the shared network. There are two main types of parameter sharing architectures: Hard-parameter sharing and Soft-Parameter sharing. In Hard-parameter sharing, a set of parameters mainly in the hidden-layers are shared between two or more distinct but related tasks. However, in soft parameter sharing, the layers of all the tasks are kept separate while regularizing interconnections between the hidden layers of the tasks encourage the parameters of the layers to obtain similar values . The hard parameter sharing methodology is used to build the architecture in this research as it is known to greatly reduces the risk of overfitting. Baxter *et al.* [6] showed the risk of overfitting reduces by the order of the number of tasks sharing a set of parameters. This is intuitive as higher number of tasks forces the model to find a parameter value that produces satisfactory representation that makes sense to all the other tasks simultaneously. This results in a lower chance of the model overfitting to any one task. The following sections discusses the methodology of combining two related tasks discussed in previous sections: Object Detection and Semantic Segmentation.

3.3.1. MultEYE Model Architecture

The model for implementing a multi-task learning system would be a conglomeration of the semantic segmentation and the vehicle detection. The backbone used for this network is the same as the one used for the vehicle detection. This choice is made because vehicle detection is the focus of this research and a backbone enabling a good vehicle detection is of greater importance than the one designed for semantic segmentation.Therefore, the vehicle detection head and the semantic segmentation decoder are used with the **modified CSPDarkNet53** or CSPDarkNet53(Lite) backbone.The same backbone used in sec.3.2.1.
A major addition to the architecture that is not seen in the respective individual task architecture is that the loss is calculated as a layer.In other words, the losses are calculated within the architecture. This entails that the losses of each tasks are combined and the architecture's output is this combined loss. This implemented because it was observed during experiments that the Keras[22] platform tries to minimize the output when using a custom layer using the *lambda* function which is used to define the YOLOv4 post-processing head. This was also required for the segmentation decoder when training it with the multi-task strategy. The complete architecture can be visualized in Fig.3.18.
It can be observed that the training architecture accepts the training image as the input. The detection ground truth and the corresponding segmentation ground truth are given to the network as auxiliary inputs only for the purpose of calculating the corresponding losses. These losses (segmentation and detection) are added to produce output of total loss, which is minimized during training. However, the ground truth inputs and loss layers are stripped from the architecture during inference. This multi-task model is named **Mult**i-task **E**ntwined **Y**OLO and **E**Net(**MultEYE**) for reference in the rest of the report.

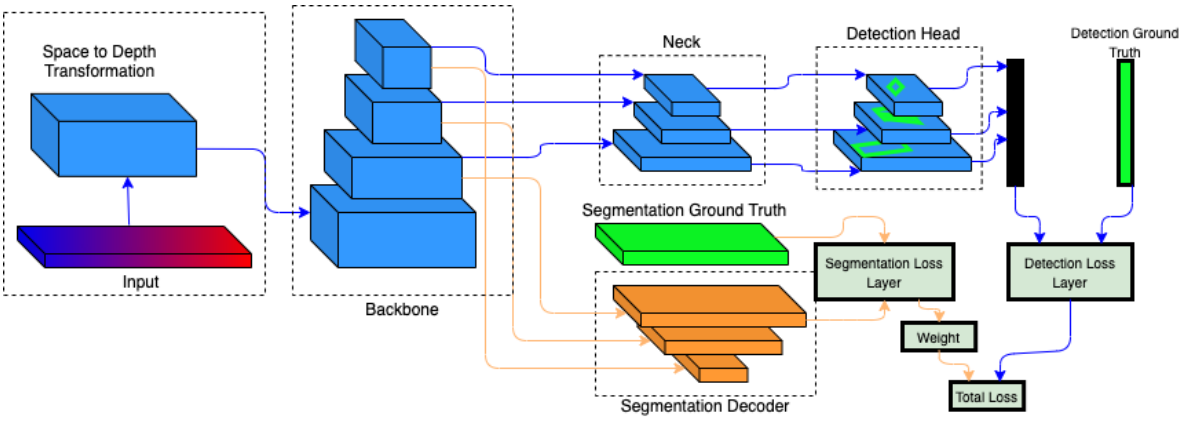


Figure 3.18: Vizualization of the MultEYE

Model	Backbone	mAP@0.5%	No. of Parameters	FPS
MultEYE*	CSPDarkNet53(Lite)	0.834	12.4M	43.5
MultEYE	CSPDarkNet53 (Lite)	0.834	27.5M	–
YOLOv4[12]	CSPDarkNet53 (Lite)	0.8073	12.4M	43.88
	CSPDarkNet53[12]	0.8172	37.3M	31.19
	EfficientNet(B1)[112]	0.824	58.6M	22.72
	MobileNetV3[50]	0.746	19.13M	41.5
	MobileNetV3Small[50]	0.694	11.6M	57.13
TinyYOLOv4 (Custom)	EfficientNet (B1)[112]	0.7082	27.42M	37.2
	MobileNetV3 Small[50]	0.6733	5.41M	110.11
YOLOv3[95]	Xception[21]	0.7625	42.53M	20.8
SSD[68] (300x300)	VGG 16[107]	0.7739	24.0M	39
	MobileNetV3[50]	0.7156	9.6M	59.3
Faster RCNN[97]	VGG 16 [107]	0.7910	71.93M	6.62

Table 3.6: Comparison of the MultEYE network with other state-of-the-art models evaluated on a combination of 10% set of Aeroscapes and SODA Dataset images resized at 512×512 (Except the SSD network that was trained with 300×300 resolution).The evaluation was benchmarked on a single NVIDIA Titan Xp GPU with CUDA 10.1.
*:Segmentation Decoder is detached from the model

3.3.2. Training

The training strategy used for the multi-task training is similar to the one employed for the vehicle detection. The backbone is initally loaded with pre-trained ImageNet[59] weights. Once the pre-trained weights are loaded, the backbone is frozen and only the detection head (and neck) and the segmentation decoder are trained for 20 epochs.For comparison purposes, the hyperparameters used for MultEYE training are kept the same as the one used for the vehicle detection which is as listed:

- **Batch Size:** 4
- **Number of Epochs:** 250
- **Losses:** Tversky Loss[103] for Segmentation, GIoU[98] for Vehicle Location and Focal loss[65] for Vehicle confidence.
- **Segmentation Loss Weight:**1
- **Optimizer:** ADAM
- **Initial Learning Rate:** 1×10^{-2}
- **Learning Rate Decay factor:** 0.5 with a patience of 5 epochs

The technical hardware and software specifications are listed below:

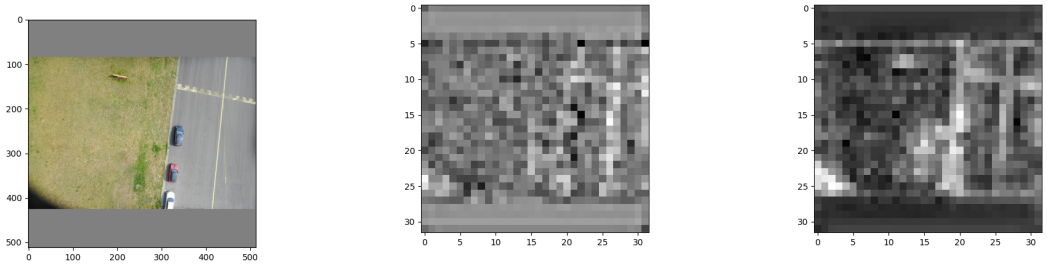
- **Training System Specification**: Intel Xeon Processor with 12 GB NVIDIA Titan Xp GPU
- **Software Packages**:Keras with Tensorflow 2 backend (Python3.7)

A new hyperparameter is introduced called the *Segmentation Loss Weight* which essentially adds a weighing factor onto the level of influence that the segmentation task has on the vehicle detection.This value is set as 1 for the testing phase which gives equal weighting to both detection and semantic segmentation.

3.3.3. Results and Discussion

The MultEYE network was evaluated against the best performing models of different state of the art models that claimed real-time (>30 FPS)performance. The network was evaluated in two different modes: the inference version of the multi-task trained network that generates

the detections as well as the segmented maps of the input frame and the stripped down version that only generates the detections. The performance throughput of the model with the segmentation head attached could not be benchmarked due to a technical issue with saving the Keras model with multi-task heads. However, as expected intuitively, stripping the segmentation head doesn't affect the accuracy of the detection head. The main improvements



(a) The original 512×512 letterbox resized image fed to the networks

(b) Feature activation map from the 3^{rd} Conv module of the backbone of the single task network

(c) Feature activation map from the 3^{rd} Conv module of the backbone of the multi task network

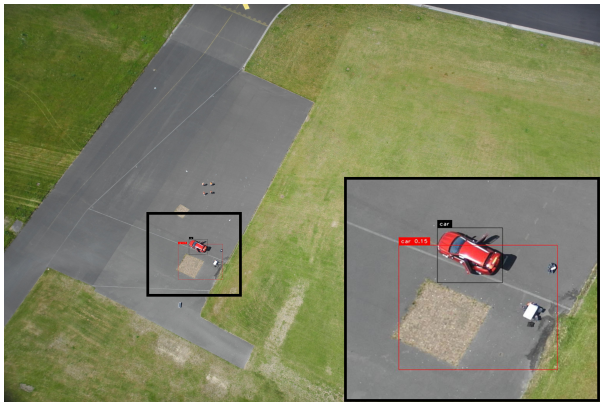
Figure 3.19: The comparison of the difference in features learned in a standard learning methodology and the features learned with an auxiliary segmentation task

that were observed have been described below:

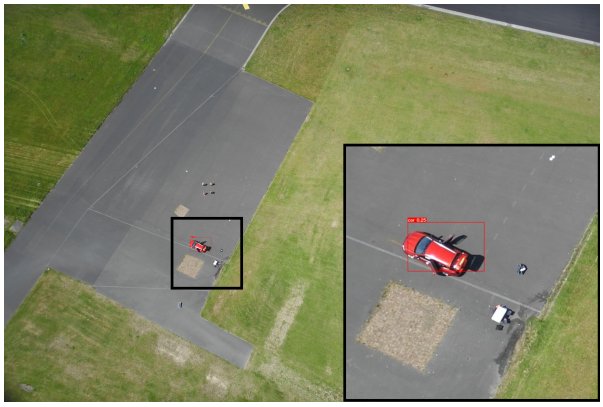
Higher Test mAP:
As evidenced in table.3.6, having an auxiliary task improves the performance of the main task.

The effect of having an auxiliary task to learn the features can be visualized in Figure.3.19. This figure depicts the activation in the first channel of one of the final convolutional layer output of the backbone. It can be observed that the single task backbone activation(Fig.3.19b) is much more noisier than the same learned with a multi-task methodology(Fig.3.19c). This could be explained by the implicit data augmentation capability of multi-task learning which essentially states that all tasks generally tend to be noisy and have their own noise signature which the network learns to ignore and generalizes well. Different tasks have different noise pattern as the errors propagate differently each of these tasks. Therefore, learning the vehicle detection task along with the semantic segmentation enables the network to better learn the features by averaging out the noises.

Better Generalization:
It was also observed that MultEYE network exhibited a better generalizing ability on the SODA image set. This was more apparent in some of the images that were taken at 120m altitude which was the maximum flown by the DeltaQuad. Such images are much harder to get good detections from due to the scale of the vehicle and the low resolution (512×512). An image example of the generalizability can be seen in Fig.3.20b. The YOLOv4 detection has a low IoU score of 0.132 while the detection using MultEYE has a high IoU score of 0.844 while the object confidences of both remain similarly low.
This difference in detection accuracies could be attributed to the limited training dataset causing the YOLOv4 to overfit to the aeroscapes image pattern while the MultEYE avoids this due to the introduction of inductive bias which in turn helps in regularization. Such regularization provided by having the segmentation task is most likely the reason why the model does not overfit and may reduce its Rademacher complexity (the ability to fit to random noise).



(a) YOLOv4 detection on a SODA image taken at 120m altitude.The region of interest is zoomed in and displayed at the bottom right corner.



(b) MultEYE detection on a SODA image taken at 120m altitude.The region of interest is zoomed in and displayed at the bottom right corner.

Figure 3.20: Evidence of high generalising ability of the MulEYE network

3.4. Hyperparameter Optimization

Most of the applications of deep learning models tend to use the model settings or *hyper-parameters* that are used for benchmarking the model architecture. Setting these hyper-parameters that are tuned for another task would not enable the model to perform to its maximum capacity. These hyperparameters need to be tuned to extract the model's full potential. However, the YOLOv4 has 15 tunable parameters which makes it impossible to tune manually. In the recent years, hyperparameters for deep learning models are tuned using meta-heuristic algorithms like Genetic Algorithms [75], Particle Swarm Optimization[57] and Bayesian Optimization algorithms[125] etc.
The algorithm chosen to optimize the MultEYE model is the Invasive Weed Optimization (IWO)[77] as it has been shown to find isolated global minima better that peer meta-heuristic algorithms for at least 40 parameters[5]. The IWO algorithm is based on the colonizing behaviour of weeds found in the ecological sphere. Before the algorithm is explained, some key terms are defined below.

- **Seed:** The individual in the colony that represents a value of a variable to be optimized.
- **Fitness:** The value representing how good the solutions is.
- **Plant:** Seed after the fitness is evaluated.
- **Colony:** The entire set of seeds.
- **Population Size:** Number of plants in the colony

- **Max. Number of Plants:** The maximum number plants that can produce seeds.

The variables that are optimized during this process are:

- Initial learning rate (`lr0`) : The initial value of learning rate which is later decayed as the validation loss reaches a plateau.
- Adam β_1 (`momentum`) : The exponential decay rate for the first moment estimates of the adam optimizer
- Adam weight decay (`weight_decay`) : Decay rate of the learning rate from the initial learning rate.
- GIoU Loss gain (`giou`) : Percentage contribution of the GIoU loss to the total loss
- Class loss gain (Segmentation Loss weight) (`cls`) : Percentage contribution of the classes in the tversky loss for the segmentation task
- Weighting for the positive class (`cls_pw`) : Percentage contribution of the binary cross-entropy loss to the object detection task.
- Focal Loss gain (`obj`): Percentage contribution of the Focal loss to the total loss
- Object Binary Cross-entropy Loss positive weight (`obj_pw`) : Percentage contribution of the binary cross-entropy for object confidence.
- Anchor-multiple threshold (`anchor_t`): The threshold-multiple filter out generated anchors based on the IoU score for the three scales of detection proposals.
- Hue, Saturation and Value augmentation (`hsv_h`,`hsv_s`,`hsv_v`): The values for hue, saturation and value for image augmentation used in train.
- Translation augmentation (`translate`): Probability of a random translation augmentation applied to a training image
- Image scale augmentation (`scale`): Probability of random image scale augmentation applied to a training image
- Mirror augmentation probability (`fliplr`): Probability that a training image is mirrored before training

These hyperparameters are arranged so that they are augmentable by the optimization algorithm. The steps involved in the optimization algorithm are described as follows:

1. The 15 parameters to be optimized are selected and a 15-dimensional search space is defined in which the solution is searched.
2. The parameters or seeds are initialized by randomly distributing them in the previously defined search space. Each seed is an initial solution and thus the population is initialized.
3. The fitness of each seed is calculated by running the training for 200 epoch using these initial hyperparameters and obtaining the score as a weighted sum of their test mAPs (both at 0.5 and 0.75 IoU), class confidence, object confidence and the Tversky loss. Each seed then becomes a flowering plant that creates its own seeds.
4. The number of seeds produced by a flowering plant is decided by the fitness value. Therefore the plants generated from seeds with better solution to the optimization problem creates more seeds around itself in the search space.

5. The newly generated seeds are dispersed around the search space with their mean as the location of the parent plant and with varying variance. The standard deviation of dispersion during a particular iteration is calculated as follows.

$$\sigma_{iter} = \frac{(iter_{\max} - iter)^n}{(iter_{\max})^n}\left(\sigma_{initial} - \sigma_{final}\right) + \sigma_{final} \tag{3.16}$$

   Where, $iter$ and $iter_{max}$ are the current iteration index and the maximum number of iterations, $\sigma_{initial}$ and σ_{final} are defined initial and final standard deviation respectively and n is the non-linear modulation index. This makes sure that for the initial iterations, the majority of the seeds are dispersed widely to search for a global optima while the dispersion variance reduces as iterations progresses in order to fine-tune the solutions gradually.

6. The newly planted seeds are evaluated and thus generated plants are ranked together with their parent plants. Lowest ranking plants are eliminated to maintain the maximum population size.

7. The surviving plants then produce seeds of their own based on their ranking in the population. This process from step 3 is repeated till a desired fitness value is obtained or the maximum iteration is reached.

These steps can also be visualized in Fig.3.21.

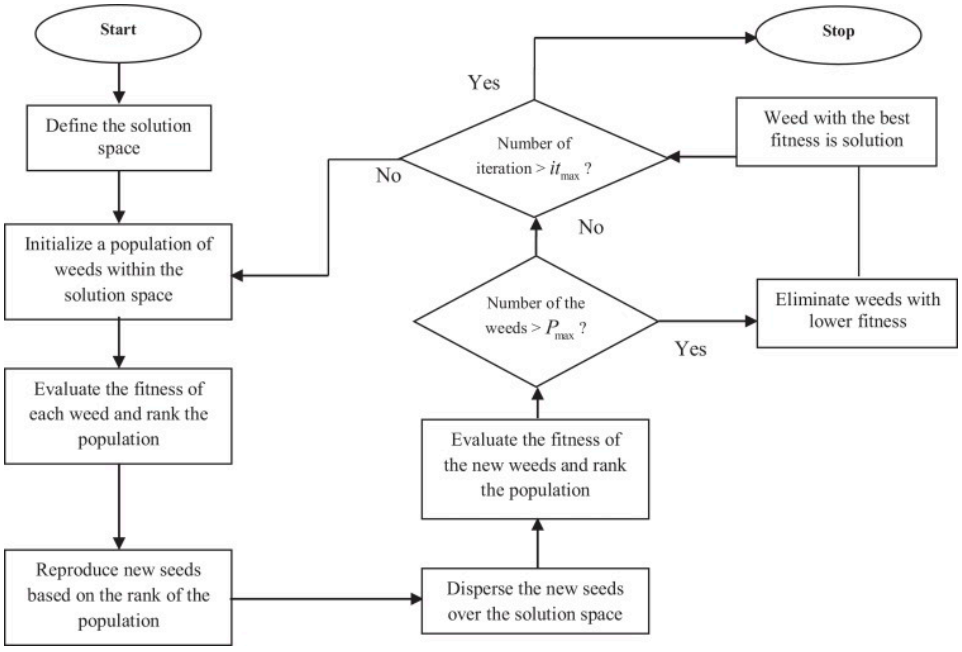


Figure 3.21: Visual Schematic of the Invasive Weed Algorithm (IWO)

3.4.1. Results

The MultEYE is optimized by running a 200 epoch training for 300 iterations using the hyperparameters as the arguments. The results of the optimization can be visualized in Fig3.22. The figure plots the values tested in the scope of the optimization procedure against the fitness score obtained during that particular iteration. The colours represent the density of the weeds that survived the population limit. Highest density is indicated by the colour red while sparsely distributed individuals are marked in blue. It can be noted that the chosen value of the hyperparameter (indicated with a + sign) lies close to the most dense cluster. This shows that the algorithm is able to localize the local minimum with good precision. Further, the presence of multiple red clusters, indicates the presence of multiple local minima and

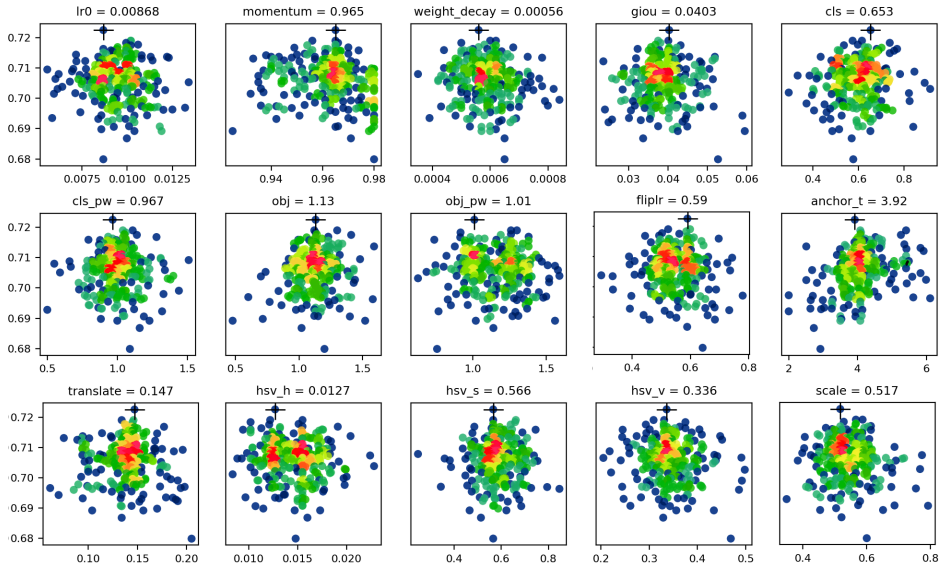


Figure 3.22: Results of IWO optimization of MultEYE hyperparameters. The best seed of the hyperparameters from each iteration is plotted in the x-axis versus the fitness score in represented in the y-axis

Hyperparameter	Value	95% Confidence Interval
lr0	0.00868	[0.00793,0.00869]
momentum	0.965	[0.959,0.970]
weight_decay	0.00056	[0.00053,0.00057]
giou	0.0403	[0.0371,0.0418]
cls	0.653	[0.058,0.073]
cls_pw	0.967	[0.959,1.142]
obj	1.13	[1.083,1.292]
obj_pw	1.01	[1.0, 1.088]
flip_lr	0.59	[0.55,0.622]
anchor_t	3.92	[3.918,3.924]
translate	0.147	[0.13,0.156]
hsv_h	0.0127	[0.0125,0.0136]
hsv_s	0.566	[0.45,0.6]
hsv_v	0.336	[0.31,0.354]
scale	0.517	[0.47, 0.589]
Optimized Model mAP: 0.862		

Table 3.7: Optimized Hyperparameter Values and their respective 95% Confidence Intervals

the algorithm has managed to find the global minimum in the course of the optimization process.
The optimized hyperparameter values and their 95% confidence interval are tabulated in Table3.7. A *95% Confidence Interval* is the range of parameter values where the probability of the location of the solution is 95%. It was observed that training with the optimized hyperparameters **improved the test mAP from 0.834 to 0.862**.

3.5. Summary

This chapter describes the steps involved in building a detector model optimized for accuracy and performance throughput. The model uses a stripped down version of the CSPDarkNet53 network as the backbone. Then the network branches out to two heads: The detector head and the segmentor head. The detector head is the same used for YOLOv4 and no further modifications are made. The segmentor head is designed based on the ENet model with additional modules and skip connections from the CSPDarkNet53 backbone. The complete model is named MultEYE and is trained on the Aeroscapes dataset with a set of tuned hyperparameters and tested on a combination of both Aeroscapes and SODA datasets. Once

the MultEYE model is trained, the segmentation head is detached from the model in order to boost its performance throughput to prepare its implementation on the Jetson Xavier NX platform.

4

Vehicle Tracking and Speed Estimation

Vehicle tracking is the next step after a successful detection of vehicles in the aerial images acquired by the UAV. The vehicle detections provided by the object detector serves as an initialization for the tracking algorithm. There are two main reasons why tracking of vehicles is important in the context of traffic surveillance: Firstly, the computationally heavy neural network running a detection for each frame of the video input costs time and energy, the later of which could be made better use of extending the flight time of the UAV. This computational cost can be reduced by using the neural network model to only initialize the detections in the video while a computationally light algorithm tracks these detections across frames. Secondly, neural network detectors consider each frame to independent to the previous frames and therefore does not track any detections across frames. In other words, standard neural network detections do not preserve identity across image sequences.
This chapter discusses the choice of tracker used for the project and presents a comparison between the choice and the other state-of-the-art trackers and also some classic trackers. Further, two methods of extracting the inertial speed estimates from the tracked objects are compared.

4.1. Minimum Output Sum of Squared Error based Tracking

The tracking algorithm used for the current problem of tracking vehicles is a form of Optimized Correlation Output Filter called the Minimum Output Sum of Squared Error (MOSSE)[13]. The tracker uses the first two frames to initialize itself. The area enclosed by the bounding box of a detection is cropped from the first frame of the sequence which is used as the template for initialization. This template is first transformed using a natural logarithmic transformation to reduce lighting effects and for contrast enhancement. This is done using eq.4.1.

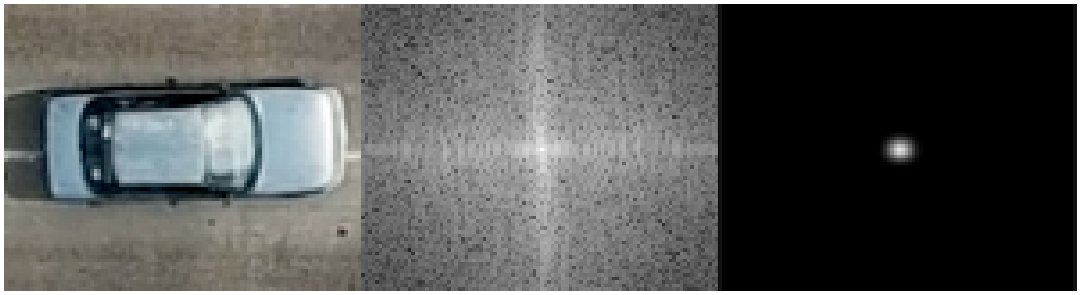
$$y = \ln(x + 1) \tag{4.1}$$

Where, x and y are the input and output pixel values. Once the logarithmic transformation is done, the template is normalized to reduce the effect of variant illumination. Further, the template is expressed in its frequency domain by finding its Discrete Fourier Transform (DFT) using eq.4.2(Fig.4.1b).

$$F(u,v) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-2\pi i\left(\frac{xy}{M}+\frac{yv}{N}\right)} \quad u = 0,\dots,M-1; \quad v = 0,\dots,N-1 \tag{4.2}$$

Once the DFT is generated, a synthetic target has to be generated to be used in the initialization of the tracker and for updating the filter during tracking. This synthetic target contains a Gaussian peak centered at the object in the template and is generated using the eq.4.3(Fig.4.1c).

$$G_i = \sum e^{-\frac{(x-x_j)^2+(y-y_j)^2}{\sigma^2}} \tag{4.3}$$



(a) Original Image (b) Image in frequency domain (c) Synthetic Gaussian Peak

Figure 4.1: Initializing the MOSSE filter requires the Fourier transformation of the image and a synthetic gaussian peak that represents the position of the vehicle that is being tracked

Where, G_i represents the synthetically generated target of i^{th} frame, x and y are the pixel location coordinates and x_j and y_j specify the coordinates of the centre of the object to be initialized on. The σ term represents the radius of the gaussian peak.
With the Fourier transform and the synthetic target extracted, the MOSSE filter is initialized using the following formula.

$$H^* = \frac{\sum_i G_i \odot F_i^*}{\sum_i F_i \odot F_i^* + \epsilon} \tag{4.4}$$

Where H^* is the complex conjugate of the filter, F_i and G_i are the fourier transform and the synthetically generated target of the template in the i^{th} frame respectively, F_i^* is the complex conjugate of F_i. The symbol $\odot$ denotes an element-wise multiplication. The value of σ in eq.4.3 is taken to be 2 as per the original paper and ϵ is the regularizing variable introduced to prevent division by 0 and has a value of 0.001.
Once the filter has been initialized, the tracking filters can be estimated using the following formula.

$$\begin{gathered} N_i = \eta\left(G_i \odot F_i^*\right) + (1-\eta)N_{i-1} \\ D_i = \eta\left(F_i \odot F_i^* + \epsilon\right) + (1-\eta)D_{i-1} \\ H_i^* = \frac{N_i}{D_i} \end{gathered} \tag{4.5}$$

The term η represents the learning rate which ranges from 0 to 1. Once H_i^* (the MOSSE filter of i_{th} frame) has been determined, the position of the new object is determined by element-wise multiplication of the MOSSE filter with the cropped and transformed tracking window(eq.4.6).

$$G = H^* \odot F \tag{4.6}$$

This describes the original MOSSE tracker that uses the first 2 frames for initialization .

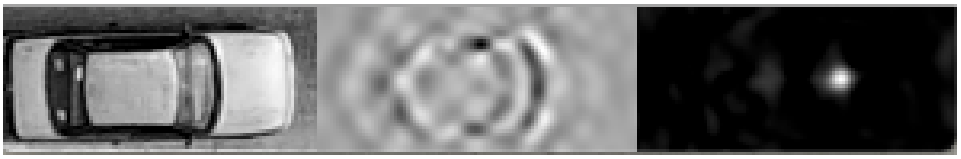


Figure 4.2: The MOSSE filter (middle) of the tracked car (left) and its predicted position (right)

However, tracking performance can be boosted if there are higher number of training samples or initialization frames. This can be achieved by increasing the number of initialization frames by performing geometric affine transformations on the image in addition to the normalization and logarithmic transformation. Scale ,translation and rotation transformation are performed using the following equations.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -t_x \\ 0 & 1 & -t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 0 \end{bmatrix} \tag{4.7}$$

	MOTA	MOTP	Avg.Framerate(FPS)
BOOSTING[43]	35.92	44.32	<1
Multiple Instance Learning[3]	60.4	64.0	<1
Kernalized Correlation Filters[49]	80.2	87.8	8.3
Tracking Learning and Detection[53]	78.34	82.3	<1
MEDIANFLOW[52]	94.73	63.14	6.6
GOTURN*[48]	67.5	75.2	20.0
CSRT[70]	95.0	94.3	1.3
MOSSE[13]	90.91	92.11	227.5
POI*[128]	96.83	97.71	11.2
DeepSORT*[124]	94.85	93.29	40.4

Table 4.1: Comparison of commonly used trackers with established state-of-the-art deep learning based trackers on a custom dataset.The * denotes that the tracker algorithm is deep-learning based and the speed was evaluated on a GPU (NVIDIA Titan Xp).

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -t_x \\ 0 & 1 & -t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 0 \end{bmatrix} \tag{4.8}$$

Where, u and v are the coordinates of the new pixel and x and y that of the old pixel. The symbols t_x and t_y represents the distance of the centre of the image from the x and y axis respectively. The rotation angle and the scale is represented by θ and s respectively.

4.2. Experiment and Results

The MOSSE algorithm's multi-object tracking capability is evaluated on a custom dataset with 100 image sequences which is a derived subset of the KIT AIS Dataset [123]. The image sequences are captured 0.33 seconds apart with an average of 14 targets to track throughout the 100 images. There is no movement of the camera frame and none of the targets in the image are occluded from the point of their appearance in the sequence and their eventual exit from the frame of observation. The vehicle initializations are manually annotated so that the errors caused by an object detector does not influence the performance evaluation of the tracker.

The evaluation metrics used for the trackers are: Multi-object Tracking Accuracy (MOTA), Multi-object Tracking Precision (MOTP)[8] and the Average Framerate. MOTA is calculated using the following formula:

$$MOTA = 1 - \frac{\sum_t (m_t + fp_t + mme_t)}{\sum_t g_t} \tag{4.9}$$

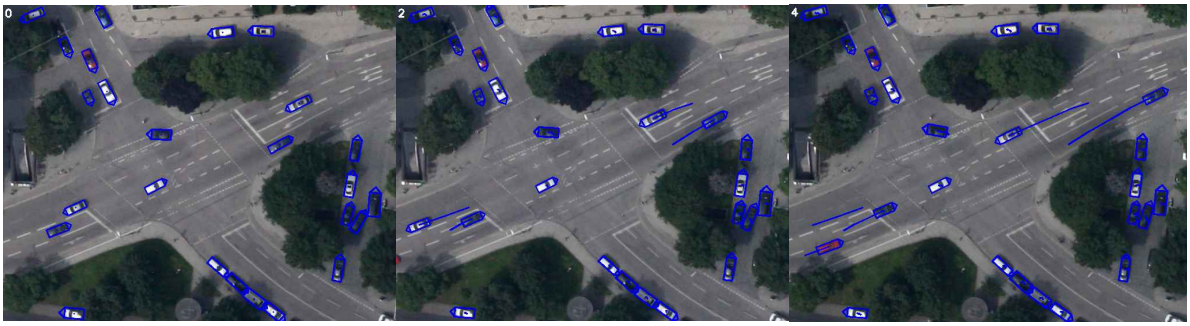
Where m_t, fp_t and mme_t are the number of initialization misses, false positives and mismatches respectively of time t, while g_t is the number of ground truth instances at time t.
The MOTP is calculated the following way:

$$MOTP = \frac{\sum_{i,t} d_{i,t}}{\sum_t c_t} \tag{4.10}$$

Which is essentially the ratio of the cumulative sum of position errors to the total number of detections.

Table.4.1 shows the comparison of MOSSE with other commonly used trackers.It can be seen that old algorithms like BOOSTING[43], MIL[3] and TLD[53] perform poorly on the custom dataset with very low framerate. This is most likely due to difficulty of these algorithms to maintain tracker identity when implemented as a Multi-Object Tracker as these were designed to be single object tracker. Only GOTURN[48], DeepSORT[124] and MOSSE[13] have tracking speed that is faster than the speed of object detection. The GOTURN and DeepSORT are deep learning based algorithms that credit their accuracies and speed to GPU

based computing and do not maintain its performance when limited to CPU. It can be seen that MOSSE's execution speed is far higher than its counter-parts despite running on CPU while performing fairly well in the tracking aspect.
Further, as there is large numbers of tracking target that enter and leave the frame in a short amount of time during a typical surveillance mission, it is essential that the trackers are initialized frequently to purge the trackers of objects that have already left the frame and introduce trackers of objects that have newly entered the frame. This update frequency was estimated to be once in every 10 frames by brute-force trial and error method. With this tracker update rate, it was found that GOTURN, DeepSORT and MOSSE performed almost identically with respect to the tracking accuracy. Therefore, the lightweight MOSSE was the obvious choice of tracker that was tasked with tracking vehicles in this research.



(a) Frame-0 (b) Frame-2 (c) Frame-4

Figure 4.3: Tracking of manually initialized vehicles through 5 frames of the test dataset

4.3. Speed Estimation

Tracking is important in the context of vehicle speed estimation as the tracker preserves the identity of the object through two or more frames.The relative displacement of the object in two consecutive frames can be estimated easily. However, converting that displacement to the world coordinate system or the inertial coordinate system requires the knowledge of two more parameters. The two major parameters are the Ground Sampling Distance(GSD) and the UAV's velocity vector. GSD is the distance in inertial coordinates that corresponds to the length of 1 pixel in the image coordinates. There are two methods of acquiring these parameters, a) Estimation of the parameters using context priors and optical flow and b) Parameter calculation using real-time flight data.

4.3.1. Parameter Estimation using known context Priors and Optical Flow (Non-Parametric Method)

This type of vehicle speed estimation is chosen in cases where the platform running the detection and tracking has no way to acquire parameters such as flight altitude, global position and camera angle. These parameters are necessary for a fairly accurate estimate of the speed, hence they are estimated using visual approximation and using expected parameter values of known objects which are called *context priors*.
The first of the parameter that is estimated is the flight velocity vector in pixels per second.This is estimated by first assuming that majority of the pixels in the frames belong to inertially static objects like road markings,trees,buildings etc. The Farneback optical flow[31] is computed for two consecutive frames. When the view is nadir(Fig.4.4), the statistical mode of the flow magnitudes in the frame would correspond to the displacement of the frame with respect to the static objects in the images or flight displacement in pixels. This can be visualized by plotting the probability density function of the magnitudes using the following formulation.

$$f(x) = \frac{\exp\left(-(x-\bar{x})^2/(2\times\sigma^2)\right)}{\sqrt{2\pi}\times\sigma} \tag{4.11}$$

Where x is set of flow magnitudes in the whole frame, $\bar{x}$ represents it's mean and σ represents the standard deviation of the distribution.

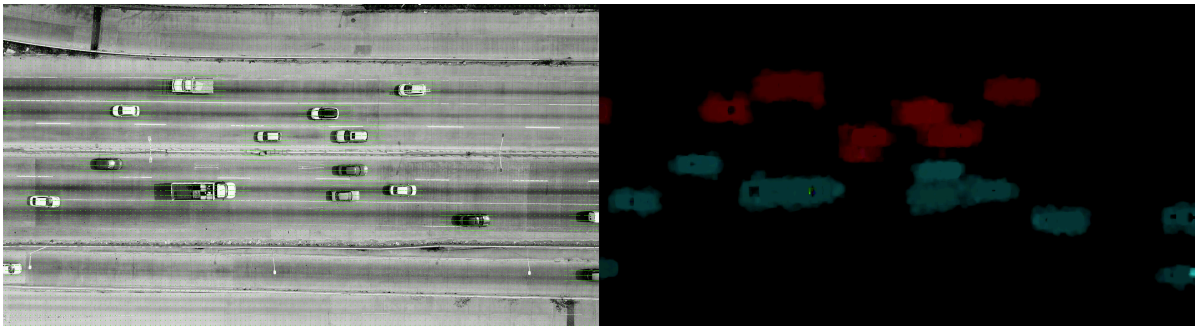


Figure 4.4: Visualization of optical flow when the camera frame is static with respect to the inertial frame. It can be observed that the majority of the flow magnitude is 0 which corresponds to the flight velocity.

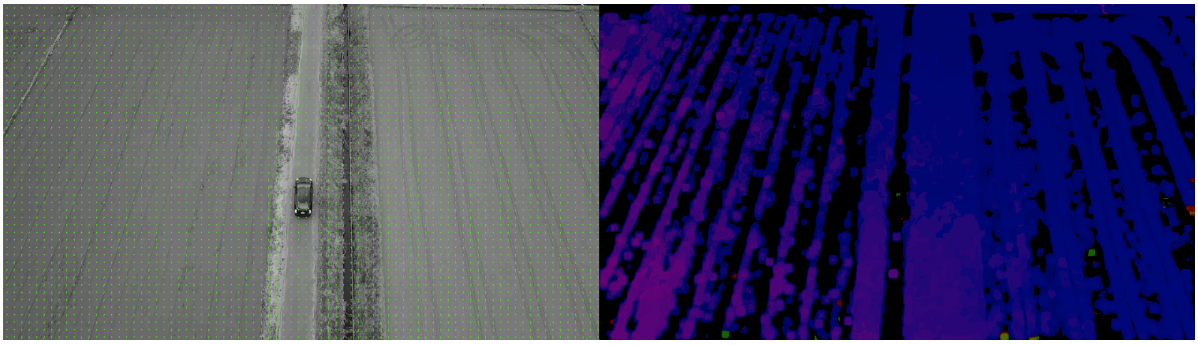


Figure 4.5: Visualization of optical flow when the camera frame is moving with respect to the inertial frame. It can be observed that the flow magnitudes vary linearly in the vertical direction but the flow immediately surrounding the car has similar values.

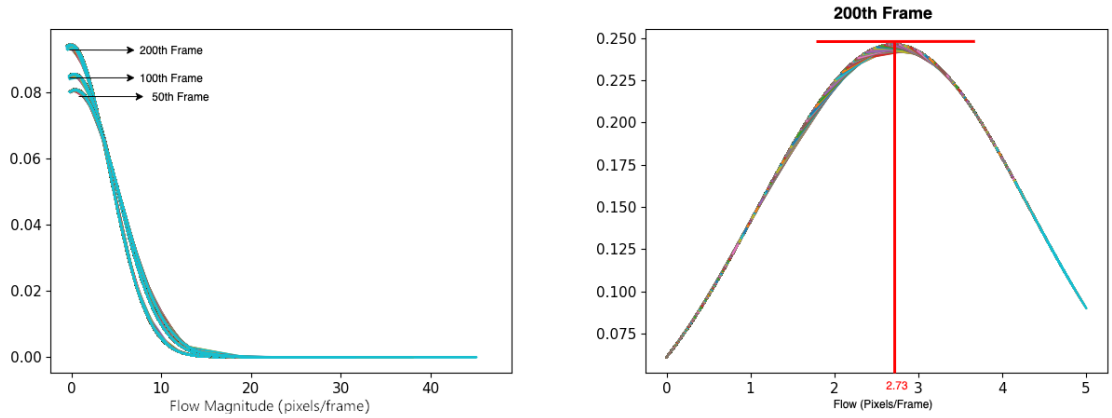
It can be observed in Fig.4.6a that the most of the flow, in the image sequence captured by a UAV in hover mode, is concentrated around the value 0. This is also valid across different frames in the sequence although the value of density changes, the mode remains at 0. Similarly for a sequence captured by a UAV in forward flight, it can be seen in Fig.4.6b that the peak of the probability density function is shifted by the value of the approximate flight displacement.

The error of the approximation will be higher when the view is in non-nadir due to variable Ground Sampling Distance (GSD) across the frame (Fig.4.5). This error can be mitigated to an extent by finding the statistical mode of the optical flow just around the object of interest. This method makes sure that the GSD variation has minimal impact on the velocity calculation. However, this would result in separate fight displacement estimates for each tracked object. Once the flight displacement is estimated, the speed of the car in pixels/s can be estimated the following way.

$$V_{pixel} = \frac{|\vec{D}_{UAV} - \vec{D}_{box}|}{f} \tag{4.12}$$

Where V_{pixel} is the speed of the vehicle in pixels/s, $\vec{D}_{UAV}$ and $\vec{D}_{box}$ are the estimated displacement of the UAV and the displacement of the tracked object across the corresponding frames. The time between each frame is denoted as f.

The GSD is a parameter that is essential to be able to convert from the image coordinate system to the inertial coordinate system. The GSD can be estimated by calculating the ratio of known measures to the same measures in the image coordinate system. This issue is generally solved in aerial photogrammetry by the use of *Ground Control Points*(GCPs) which



(a) Flight velocity = 0

(b) Flight Velocity ≠0

Figure 4.6: Probability Density Function of flow magnitude of the frame with hover and forward flight conditions

are strategically placed markers of known dimensions. However, in majority of the cases, GCPs are not widely available which leads us to use car width as method of estimating scale. Car width is chosen mainly due to the fact that the it does not vary across cars as much as the car length does. According to [10], the width of common European cars ranges from 1.68 m for small hatchbacks to 1.8 m for SUVs with average width of 1.71 m. Once this is known, the GSD is calculated the following way.

$$GSD = \frac{1.71}{w} \tag{4.13}$$

Where the numerator is the average width of European cars in meters and w is width of the bounding box enclosing the car in pixels. The speed of the vehicle in meters per second is then calculated using the following formula.

$$V = V_{pixel} \times GSD \tag{4.14}$$

4.3.2. Parameter estimation using Real-time Flight Data (Parametric Method)

The companion computer that runs the detection and tracking on the UAV is normally connected to its autopilot generally. In this case, the UAVs altitude and angle of the camera can be streamed from the autopilot. Further, the intrinsic parameters of the camera is required for the calculation of the GSD which entails that the camera should be calibrated pre-flight. Figure.4.7 helps in the visualization of the parameters required for the calculation of the GSD and subsequently estimate the speed. With these parameters available, we can calculate GSD in meters/pixel using the following formula.

$$GSD = \frac{H \times S_w}{F \times im_w} \tag{4.15}$$

Where, H is the flight altitude in meters, S_w is width of the camera sensor in millimeters, F is the distance between the camera lens and the sensor in millimeters, im_w is the width of the frame captured. This formula is valid only for nadir views and the GSD can be approximated to be constant throughout the frame.
For oblique angle of camera, the nadir GSD can be used to calculate the oblique GSD at the angle of the camera by multiplying the nadir GSD with the *GSD Rate*.

$$GSDRate = \frac{H \times (tan(\theta + \phi) - tan(\theta))}{GSD \times im_H} \tag{4.16}$$

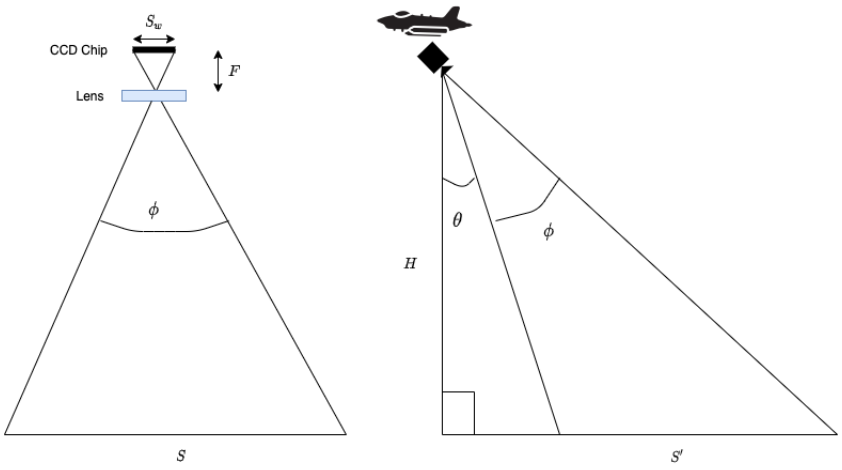


Figure 4.7: Visualization of the difference between nadir and off-nadir angle of view.

Where, θ is the camera angle, im_H is the frame height in pixels and ϕ is the total field of view which is calculated using the following formula.

$$\phi = tan^{-1}\frac{GSD \times im_h}{H} \tag{4.17}$$

The GSD Rate is essentially a function of camera angle and altitude during flight as all the other parameters can be determined from the camera's intrinsic parameters during calibration and kept constant. Therefore, the speed of the detected vehicle can be formulated in the following way.

$$V = \frac{|\vec{D}_{UAV} - (\vec{D}_{box} \times GSD(H) \times GSDRate(\theta))|}{f} \tag{4.18}$$

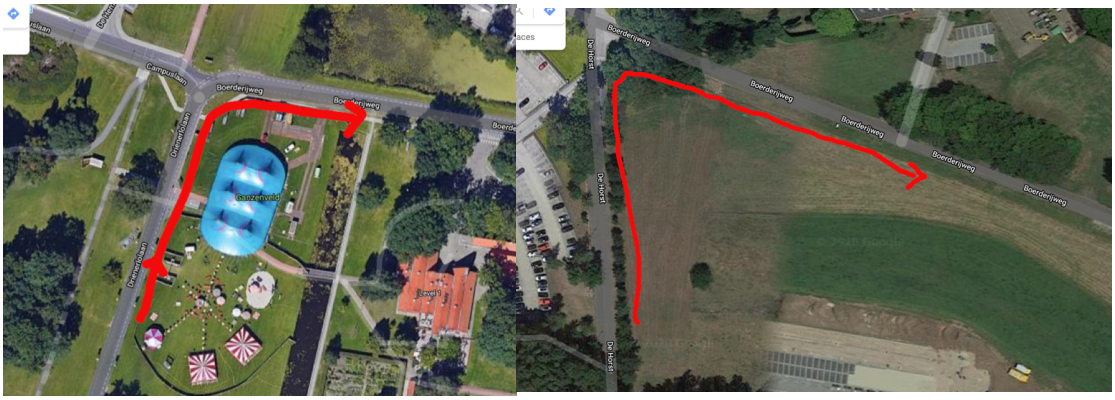
Where, $\vec{D}_{UAV}$ can be estimated using the UAV positioning difference obtained from the onboard GPS.

4.3.3. Method Comparison

The effectiveness of the methods was compared using a custom dataset comprised of data gathered at 2 locations (Fig.4.8) in the University of Twente campus using a DJI Phantom 3 UAV. [1] The detection and tracking of bicycles were used instead of cars for the ease of ground truth data gathering and organization of experiments. Collecting the speed ground truth for bicycles proved easier than that of cars. Further, organizing and coordination of a data collection experiment for 10 cars in the university campus presented complexities like having to increase the area covered by the flight plan and the requirement to cordon-off roads used for these experiments made it not feasible to use cars. Bicycles provided an easy alternative to cars to prove the speed methodology without any changes to the model architecture and hence throughput speed.

The bicycle detector was trained using the same steps described in chapter.3, except dataset was prepared with Aeroscapes where the automatic annotations were done on bicycle pixels instead of cars. This resulted in MultEYE model for bicycles. For the data gathering experiments, 10 participants were asked to ride their bicycles on a pre-defined stretch of road at a constant speed of their choice while using an app-based speed tracker to help themselves maintain this chosen speed. While the participants performed the circuit, the UAV was flown parallel to the predefined route at different altitudes. The essential flight data like GPS location, altitude and camera angle was logged for each frame of the captured video. Six videos

[1] Data collection experiments were conducted on 12^{th} June, 2020 following the social distancing guidelines set by the Rijksinstituut voor Volksgezondheid en Milieu (RIVM) and the University of Twente



(a) Drienerlolaan (b) Boederijweg

Figure 4.8: Locations of the data gathering experiments and the planned flight path at the University of Twente campus

were identified and each of the videos were sampled at 5 frames per second to generate 6 corresponding image sequences. These sequences were further divided into subsequences that contained 8 images each. Ideally, 2 tracked image frames are enough to make an estimation of the vehicle speed, however having 8 tracked frames can be used to average out the errors induced by outliers. Longer subsequences do not make sense from an application point of view where vehicles tend to have variable speeds through the sequence. These sequences were used to estimate the speed of the bicycles using the parametric as well as the non-parametric method.

Figure.4.10 shows the comparison of speeds for an example 8-image sub-sequence from Sequence-1. Here, estimates the speed of a static object is depicted when the UAV is at hover condition ($|\bar{V}_{UAV}| \approx 0km/h$). It is interesting to observe that the parametric method estimates have minimal variance but have a negligible but constant offset from the ground truth which can be attributed to errors in the measurement of the parameters. The non-parametric measurements are much more erratic in comparison but apart from 2 outlier data-points, the other values tend to be comparable with the parametric estimates. This sequence, where both the camera and the target are stationary, helps set the baseline errors in both the estimation methods.

Figure.4.11 shows the estimated speed profile of both the methods where both the target and the camera (UAV) are in motion(8^{th} sub-sequence of Sequence1). In this sample subsequence, the target is moving at 15 km/h (recorded by a mobile app-based speedometer) and the UAV is moving at 27 km/h (calculated using IMU and GPS based measurements) in the opposite direction. The parametric method predicts an average speed of 16.25 km/h while the non-parametric method predicts it to be 16.75 km/h. Intuitively, it is expected that the error in the non-parametric estimation to be higher than that of its counterpart. However, the difference between the errors in the sequence under observation is less than 0.5 km/h.

Across the all of the 6 sequences(Table.4.2), the mean base-line compensated average error for the non-parametric method was calculated to be 1.56 km/h and for the parametric method, it was calculated to be 1.13 km/h. This shows that overall, the accuracy of both the methods are comparable and the non-parametric method is a good alternative when the computation speed is not of the essence.

	Number of Subsequences	Avg. Error Parametric	Avg. Error Non-Parametric
Sequence1	8	0.85	1.32
Sequence2	9	1.03	1.41
Sequence3	15	1.43	1.96
Sequence4	14	0.71	1.3
Sequence5	9	1.25	1.52
Sequence6	15	1.52	1.86
Mean Avg. Error		**1.13**	**1.56**

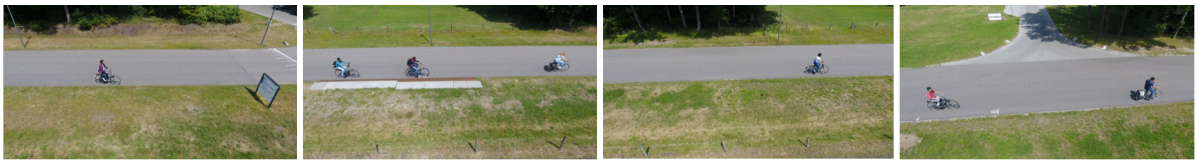
Table 4.2: Average errors in the speed estimation on the collected image sequences

(a) Sequence1:Drienerlolaan, Altitude: 5m

(b) Sequence2:Drienerlolaan, Altitude: 10m

(c) Sequence3:Drienerlolaan, Altitude: 30m

(d) Sequence4: Boerderijweg, Altitude: 5m



(e) Sequence5: Boerderijweg, Altitude: 10m

(f) Sequence6: Boerderijweg, Altitude: 30m

Figure 4.9: Sample frames from 6 image sequences captured at Drienerlolaan and Boerderijweg with different flight altitudes

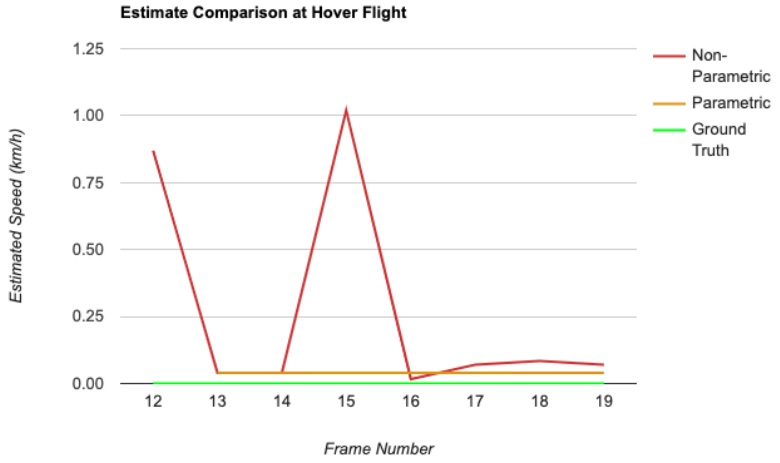


Figure 4.10: Comparison of speeds of static targets estimated using parametric and non-parametric methods while the UAV is in hover mode

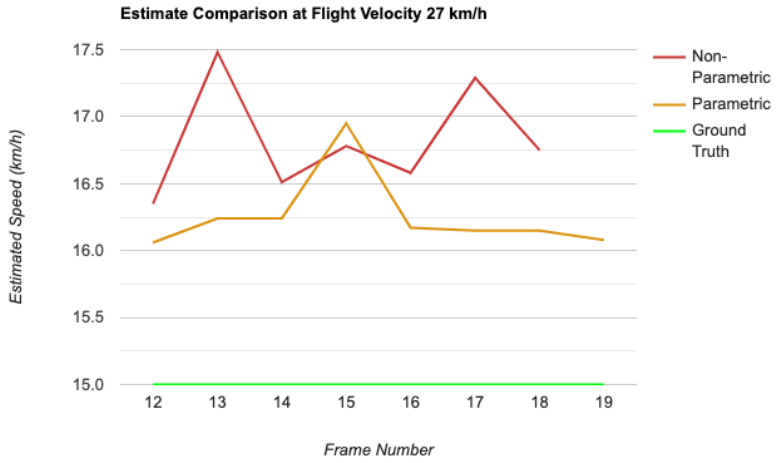
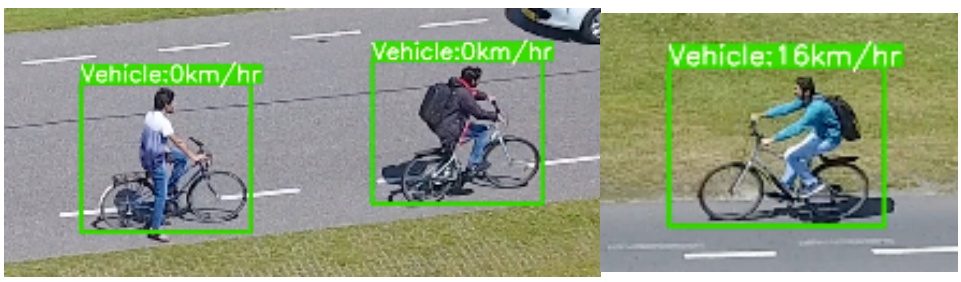


Figure 4.11: Comparison of speeds of moving targets estimated using parametric and non-parametric methods while the UAV is flying at 27 km/hr



(a) Flight velocity ≈0 (Static targets)

(b) Flight Velocity ≠0

Figure 4.12: Examples of detections when the flight velocity is 0 and non-zero

5

Inference on Embedded Platform

Now that all the sections of the pipeline are ready, the pipeline should be implemented on an embedded platform. Running the pipeline on a high-performance GPU device may be an attractive option as the model can be run unbridled without limitation and at its maximum potential. However, the sheer size and the energy consumption of such devices make it non-viable to be mounted on mobile platforms like UAVs. The whole pipeline starting from detection inference, vehicle tracking and speed estimation should be able to run at a fairly high frame-rate on commonly used embedded computers. While the vehicle detector requires GPU to run at its maximum potential, the tracking and speed estimation algorithms require only CPU to run at maximum performance. In the current research, the embedded device used to benchmark the performance of MultEYE is the Jetson Xavier NX development kit released by NVIDIA. The specifications of the system is described below:

- **Size :** 103 mm ×90.5 mm ×34 mm
- **GPU :** NVIDIA Volta architecture with 384 NVIDIA CUDA cores and 48 Tensor cores
- **CPU :** 6-core NVIDIA Carmel ARM v8.2 64-bit CPU 6 MB L2 + 4 MB L3
- **Memory :** 8 GB 128-bit LPDDR4x 51.2GB/s
- **Storage :** 128GB MicroSD

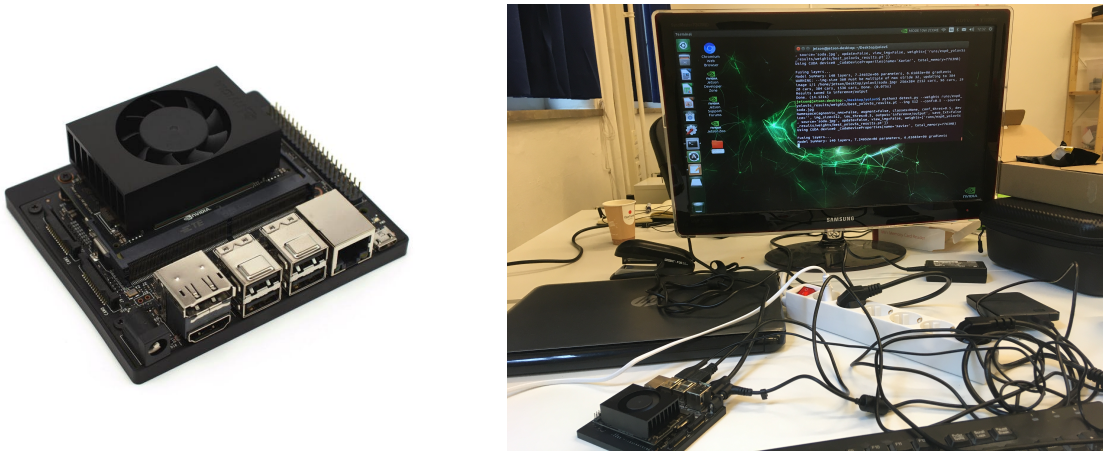


Figure 5.1: NVIDIA Jetson Xavier NX (left) and its user interface with all attached peripherals

In this chapter, Section.5.1 explains the steps taken to improve inference throughput and the subsequent improvement is compared. Section.5.2 discusses and compares the inference

speeds of the MultEYE model on the Jetson Xavier NX board with different image sizes and different power modes. The chapter ends with section.5.3 discussing the result of combining all the elements of the pipeline i.e vehicle tracking and speed estimation (parametric and non-parametric) along with the MultEYE detection.

5.1. Graph Optimization

There are a lot of modules and settings that are present in the model graph used for training which are not required for inference. Graph Optimization is the method of stripping the graph of these additional modules and settings in order to improve its inference speed and reduce model size [105]. Before going into the steps involved in the graph optimization, the issue regarding the Keras [22] model saving function should be discussed.
As of September 2020, the Keras framework has a known issue when saving trained models with multiple outputs which results in incorrect outputs. Further, Keras platform is meant for fast-prototyping and does not have libraries optimized for inference. Due to these reasons, the Keras `.h5` model was converted to a PyTorch[91] model.
Once the PyTorch version of the MultEYE is generated, the model can be optimized for inference using the following steps:

1. The model should be set to `eval` mode. This makes sure that the modules used only for training, such as dropout and batch normalization layers, can be removed.
2. The model in `eval` mode is run though a `detach` function that cuts variables from the computation graph.
3. After running through the `detach` method, the optimizer is stripped from the model graph.
4. Finally, the float precision of the model is halved from 32 to 16.

The steps can be visualized in Fig.5.2. The optimized model was found to contain 7.8 million

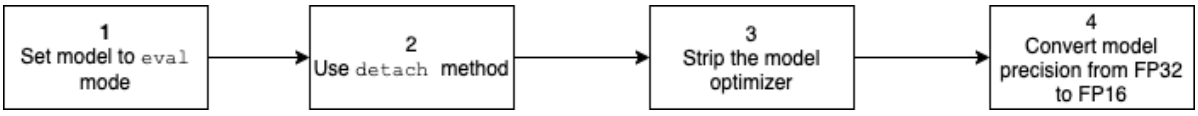


Figure 5.2: Steps involved in improving the computation throughput of the model graph

parameters which is a **71.5**% decrease in model size with only 0.03% decrease in test mAP score. However, the optimization resulted in a boost of framerate by 25.3% on the Titan Xp GPU. Once this optimized model is obtained, the inference speed on the Jetson Xavier NX can be benchmarked.

5.2. MultEYE Model Inference on Jetson Xavier NX

The Jetson Xavier NX board has 6 CPU cores and a GPU with 384 CUDA cores. Only 2 of the CPU cores are used in this research to enable parallel processing of other applications on the other 4 cores. However, all of the GPU's resources are used to run the pipeline as a neural network uses all of the available GPU to perform at its best. The board runs in 2 power modes : the 10 Watt mode and the 15 Watt mode. Figure.5.3 show the frame rate achieved for different input image resolutions run at 10 and 15 Watt modes respectively.

It can be observed that the model achieves an almost real-time performance of 29.41 FPS for input resolution 512×320 running with 15 W power mode. However, as the input resolution is increased, the difference of runtime between the two power modes decrease. Therefore, if there is a requirement to process image sizes of 2048×1152 and above, power consumption can be reduced by choosing to run at 10 W without loosing much on the performance speed. Now that the model inference speed on the board have been tested, the throughput performance of the whole pipeline of vehicle detection and tracking with speed estimation is required to be benchmarked on the platform.

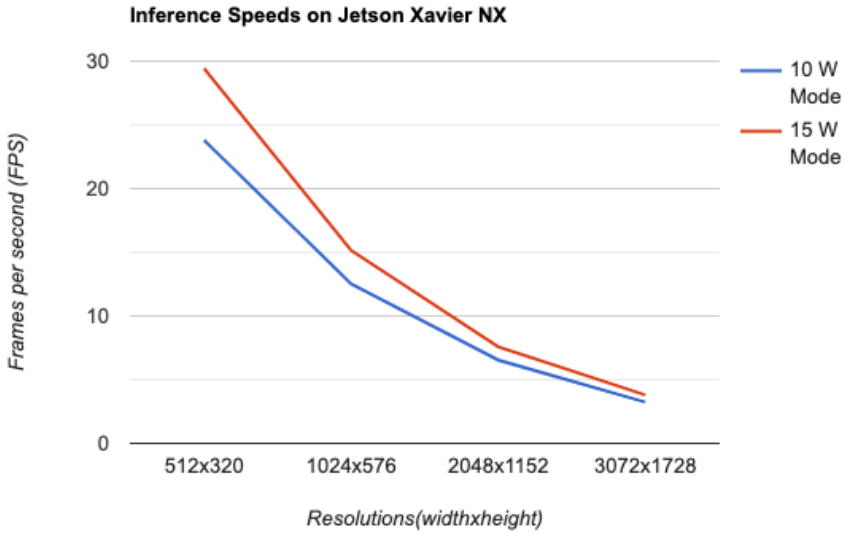


Figure 5.3: MultEYE inference speeds for different input resolutions for 10W and 15W power modes

5.3. Pipeline Inference

Algorithms run on CPU rarely show any difference in performance speed across devices. However, it is very insightful when the performance of CPU based algorithms for vehicle tracking and speed estimation is combined with the detection on the embedded platform.Figure.5.4 shows the contribution of each algorithm of the pipeline to the runtime for each of the resolutions , tested in Fig.5.3, run at 15 W. The speed estimation used here is the non-parametric estimation using optical flow and context priors. It can be seen that the non-parametric

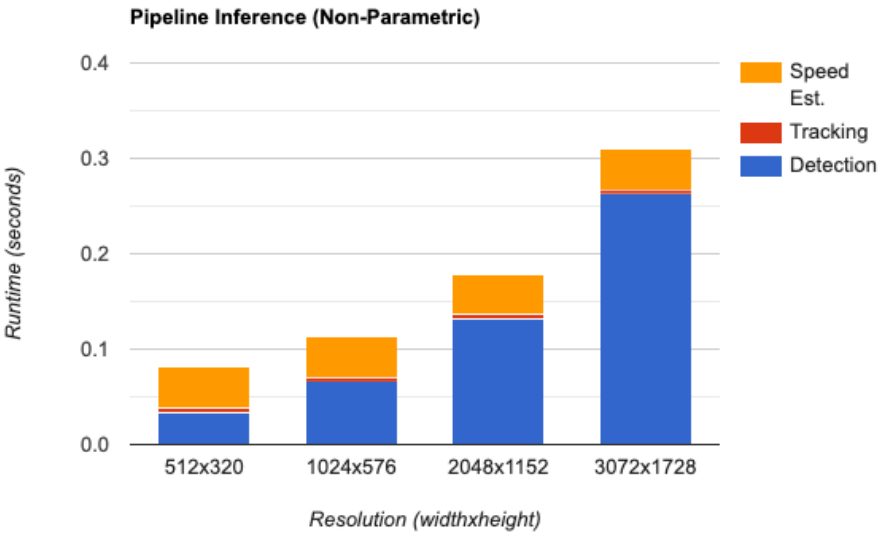


Figure 5.4: Contribution of algorithms in the pipeline running at 15 W power mode (Non-parametric speed estimation)

speed estimation contributes a significant percentage to the overall runtime of the pipeline. Especially for the resolution 512×320, the speed estimation accounts for 52.65% of the total runtime which means that the speed estimation takes longer to process than the MultEYE

Resolution	% contribution of Detection	% contribution of Tracking	% contribution of Speed Estimation	Total Runtime (seconds)
512x320	41.9	5.42	52.65	0.0811
1024x576	58.35	3.89	37.7	0.1131
2048x1152	73.7	2.45	23.8	0.1791
3072x1728	84.8	1.41	13.76	0.3101

Table 5.1: Percentage contribution of each algorithm in the pipeline towards the total runtime for 4 different resolutions(Non-Parametric method)

Resolution	% contribution of Detection	% contribution of Tracking	Total Runtime (seconds)
512x320	88.54	11.45	0.0384
1024x576	93.74	6.25	0.0704
2048x1152	96.8	3.2	0.1364
3072x1728	98.36	1.64	0.2674

Table 5.2: Percentage contribution of each algorithm in the pipeline towards the total runtime for 4 different resolutions(Parametric method)

detection. The contributions of each algorithm in the non-parametric pipeline is tabulated in Table.5.1. The main speed bottleneck for the pipeline in the speed estimation arises in the calculation of dense optical flow in the frame.

This issue can be averted by using the non-parametric method of speed estimation given that a steady stream of time synchronized fight data is available to the Jetson board. The images and the flight data should be time synchronous so that the state of the UAV can be assigned to the image captured at that particular instant which helps in accurate measurement of the parameter.

Fig.5.5 shows the total runtime for each of the resolutions chosen with individual con-

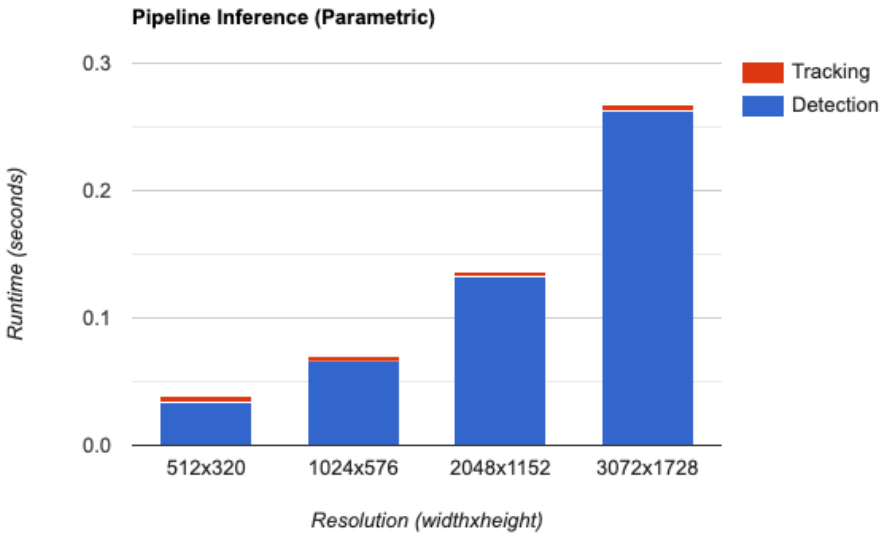


Figure 5.5: Contribution of algorithms in the pipeline running at 15 W power mode (Parametric speed estimation). The contribution of the parametric speed estimation cannot be plotted due to it being in the order of 10^{-5} seconds.

tributions of detection and tracking highlighted. The contribution of the parametric speed estimation is not plotted on the graph due to the scale being of the order 10^{-5} seconds. The reason for such a high-speed calculation is mainly attributed to the presence of only algebraic

and trigonometric calculations which are highly optimized in the python language libraries. Meanwhile, the contributions of the other two algorithms have been tabulated in Table.5.2. In a nutshell, the pipeline is able to run at 12.33, 8.84, 5.58, 3.22 frames per second while using non-parametric method of speed estimation and 26.04, 14.2, 7.33, 3.74 frames per second while using parametric method of speed estimation for resolutions 512×320, 1024×576, 2048×1152 and 3072×1728 respectively.
Observing this trend, it can be said that using the parametric method over the non-parametric one provides a significant boost in processing speed in lower resolutions. Therefore, non-parametric method should be avoided when the speed of detection is of utmost importance and flight data is readily available for streaming.

5.4. Streaming Optimization

The frame rates presented in the previous section are achieved under the assumption that one image goes through the pipeline during every iteration. This is not the case when streaming camera images in real-time (Fig.5.6). A stack of images are fed into the pipeline from the stream buffer. Only the first image of the stack is processed by the MultEYE detector which initializes the MOSSE tracker for each of the vehicles while the rest of the images in the buffer are directly fed sequentially to the tracker algorithm which then estimates the speed. This results in a higher average frame-rate as the detector algorithm is not invoked for each image being streamed.
This can be demonstrated by an example streaming case. A buffer of 10 images of resolution 3072×1728 are fed into the pipeline that runs the non-parametric version of the speed estimation. The detector takes 0.263 seconds to process the first frame while the rest of the 9 frames are processed by the tracking and speed estimation algorithms at a rate of

$$0.0044 + 0.0427 = 0.0471[seconds/frame]$$

In total, it takes

$$0.263 + (0.0471 \times 9) = 0.6869[seconds]$$

to process the entire buffer, which puts the average runtime per image to be 0.06869 seconds per image or 14.55 FPS.
Average frame rate for other resolutions for the same sample buffer size of 10 images (parametric and non-parametric methods) is tabulated in Table.5.3. It can be seen that the average frame rate sees a significant boost when the pipeline is processed in this way. The full resolution stream achieves real-time processing speeds while speed estimation is performed parametrically and the non-parametric pipeline also achieves a good processing rate of 14.55 FPS.

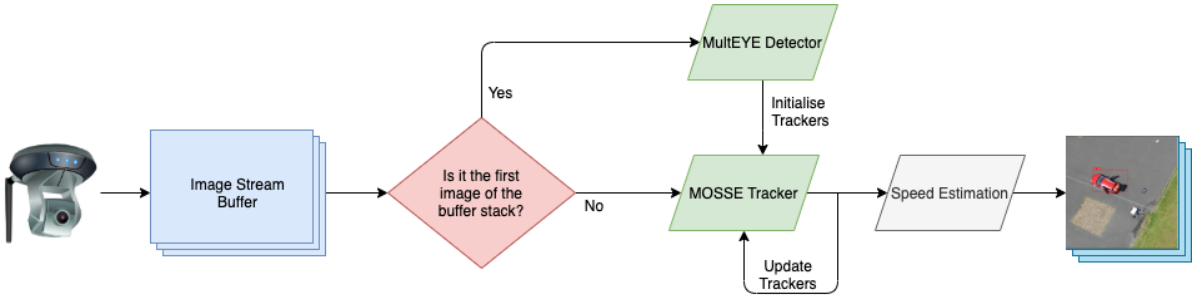


Figure 5.6: The flow of information through the pipeline when streaming from a camera in real-time

Resolution	Average FPS (Non-Parametric)	Average FPS (Parametric)
512x320	21.83	142.85
1024x576	20.41	98.03
2048x1152	17.98	59.52
3072x1728	14.55	33.44

Table 5.3: Average frame rates for the pipeline for a sample stream buffer size of 10 images for 4 different resolutions.

5.5. Summary

This chapter describes the pruning of the MultEYE model and generating a model optimized for inference on the Jetson Xavier NX platform. The inference throughput of the model is studied at different power modes of the NX platform for resolutions 512×320, 1024×576, 2048×1152 and 3072×1728. It was found that running the model at 15 Watt power mode for the image resolution 512×320 provided the maximum framerate of 29.41 FPS.
The contribution of the detection, tracking and speed estimation algorithms to the throughput of the pipeline was studied for both parametric method as well as non-parametric method of speed estimation. Further, the pipeline was optimized for streaming applications by using a stack of images which are processed sequentially by the pipeline. The first image of the stack runs through the detector which serves as the initialization for the tracker which then processes the rest of the images in the stack. This method of processing boosts the average framerate of the stack. A 10-image stack provided an average FPS of 33.44 when run through a parametric pipeline of image resolution 3072×1728.

6

Conclusion and Future Work

The goal of this research was to design a system that could detect, track and estimate the velocity of vehicles in a sequence of aerial images. The research also aimed to optimize the execution of the system so that it could process the images in real-time on an embedded computer mounted on a UAV.

The use of commonly used object detectors for the detection of vehicles came with several limitations. Accuracy of the detectors were directly proportional to the size of the network which entailed fast processing networks like one-shot detectors lacked the accuracy required to build a reliable system. Further, the lack of region-proposal networks in these one-shot detectors resulted in difficulty in identifying vehicles when the UAV is flying at high altitudes. In order to overcome these limitations, a multi-task learning methodology was employed by adding a segmentation head to the object detector backbone. Learning the contextual features along with the detections helped the backbone of the network to better encode the features, especially the lower scale features. The backbone and the segmentation head were customized for vehicle detection. The number of bottlenecks in the backbone was reduced as it was able to generate a good latent-space representation with fewer parameters, while additional bottlenecks and skip connections were introduced in the segmentation head to be able to resolve smaller-scale features in the images. The detection speed was preserved by detaching the segmentation head after training. This resulted in a detector model that performed 4.8% better than the state-of-the-art for vehicle detection in aerial images, in terms of accuracy while preserving the processing speed.

Once the detector was finalized, the detections are used to initialize MOSSE trackers to track vehicles through the image sequences. The MOSSE algorithm was proved to be extremely fast and fairly accurate in the context of vehicle tracking in aerial images as there is minimum chances of occlusions.

Speed of the tracked vehicles were estimated using two methods. The first method is used when the geographic location and pose of the camera is not available. This method estimates the ground sampling distance(GSD) based on an approximation of the detected vehicles size and the camera velocity using optical flow. The second method is used when the location and pose of the camera are available and the GSD and camera velocity can be directly calculated. As intuitively expected, the direct calculation method is significantly faster than the former.

Finally, the detector is stripped of its non-essential layers and the float-precision is halved to generate an optimized inference model. The detector-tracker-speed estimator pipeline is then designed such that it accepts a batch of images that are streamed from the camera and processes it sequentially to improve the average throughput. This results in the whole pipeline running at around 33 FPS while processing a full-resolution 10-image sequence, on a Jetson Xavier NX platform.

In summary, the goal of designing a vehicle detection, tracking and speed estimation system that can run in real-time on an embedded computer was successfully realised.

Future Work

The system in its current state is able to track vehicles real-time and estimate their speeds while running on a mobile platform. Several possible improvements and extensions can be considered while developing the system further.

Improved training dataset

Aeroscapes[82] was the only existing publicly available dataset that conformed to the requirements to train a multi-task vehicle detector during the period of research. However, the dataset is not very balanced and has only 625(≈20%) images that have vehicles in them. Further, the images are taken at altitudes ranging from 5 to 50m, while the DeltaQuad experiments are conducted at altitudes of up to 120m. Thus the detector accuracy can be significantly increased if a custom dataset can be created with images taken from the DeltaQuad.

Multi-task Inference

Due to issues with the Keras framework, infering the complete multi-task model could not be realized. Future work could consider re-constructing the model in the Pytorch[91] framework. Having a segmentation map alongside the detection would help in providing a contextual awareness to the UAV autonomy system especially in cases where emergency landing could be made safer by navigating the UAV to 'safe' spots like fields or buildings rather that busy roads.

Parallel-processing

In the case of streaming, the frames are processed sequentially in the pipeline. Each part of the pipeline have different execution times. The batch processing time can be further optimized by splitting the detection and tracking between two cores of the CPU. This will result in fewer kernel calls per core and a more optimized processing.

Tracking as a part of the Multi-task training

Deep-learning based trackers like GOTURN and DeepSORT have been shown to work best when trained along with the detector. This could be extrapolated to training the detector with a tracking head that can possibly perform tracking and detection faster and with more accuracy than when implemented sequentially.

Anomaly Identification

A true autonomous system is able to distinguish between an expected event and an anomaly in the context of aerial surveillance. Further information like vehicle orientations, traffic density and track variances can be used to identify anomalous behavior and send the report to the ground station for review.
Such a system will employ self-supervised learning that can be trained online and the detection system gets better, the longer it is used.

Bibliography

[1] Anurag Arnab, Shuai Zheng, Sadeep Jayasumana, Bernardino Romera-Paredes, Måns Larsson, Alexander Kirillov, Bogdan Savchynskyy, Carsten Rother, Fredrik Kahl, and Philip HS Torr. Conditional random fields meet deep neural networks for semantic segmentation: Combining probabilistic graphical models with deep learning for structured prediction. *IEEE Signal Processing Magazine*, 35(1):37–52, 2018.

[2] Seyed Majid Azimi, Corentin Henry, Lars Sommer, Arne Schumann, and Eleonora Vig. Skyscapes fine-grained semantic understanding of aerial scenes. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 7393–7403, 2019.

[3] Boris Babenko, Ming-Hsuan Yang, and Serge Belongie. Visual tracking with online multiple instance learning. In *2009 IEEE Conference on computer vision and Pattern Recognition*, pages 983–990. IEEE, 2009.

[4] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017.

[5] Navaneeth Balamuralidhar, Pranjal Biswas, Saumya Kumaar Saksena, Gautham Anand, and Omkar. State-space identification of unmanned helicopter dynamics using invasive weed optimization algorithm on flight data. *Journal of American Helicopter Society*, 2018.

[6] Jonathan Baxter. A bayesian/information theoretic model of learning to learn via multiple task sampling. *Machine learning*, 28(1):7–39, 1997.

[7] Anthony J Bell and Terrence J Sejnowski. The "independent components" of natural scenes are edge filters. *Vision research*, 37(23):3327–3338, 1997.

[8] Keni Bernardin, Alexander Elbs, and Rainer Stiefelhagen. Multiple object tracking performance metrics and evaluation in a smart room environment. In *Sixth IEEE International Workshop on Visual Surveillance, in conjunction with ECCV*, volume 90, page 91. Citeseer, 2006.

[9] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. Simple online and realtime tracking. In *2016 IEEE International Conference on Image Processing (ICIP)*, pages 3464–3468. IEEE, 2016.

[10] Blog. *Dimensions of European Cars*, 2019 (accessed September 10, 2020). URL `https://www.automobiledimension.com/`.

[11] Erik Bochinski, Volker Eiselein, and Thomas Sikora. High-speed tracking-by-detection without using image information. In *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pages 1–6. IEEE, 2017.

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

[13] David S Bolme, J Ross Beveridge, Bruce A Draper, and Yui Man Lui. Visual object tracking using adaptive correlation filters. In *2010 IEEE computer society conference on computer vision and pattern recognition*, pages 2544–2550. IEEE, 2010.

[14] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on pattern analysis and machine intelligence*, 23(11): 1222–1239, 2001.

[15] Samarth Brahmbhatt, Henrik I Christensen, and James Hays. Stuffnet: Using 'stuff'to improve object detection. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 934–943. IEEE, 2017.

[16] John Canny. A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, (6):679–698, 1986.

[17] Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

[18] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2017.

[19] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017.

[20] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014.

[21] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.

[22] Francois Chollet et al. Keras, 2015. URL `https://github.com/fchollet/keras`.

[23] Özgün Çiçek, Ahmed Abdulkadir, Soeren S Lienkamp, Thomas Brox, and Olaf Ronneberger. 3d u-net: learning dense volumetric segmentation from sparse annotation. In *International conference on medical image computing and computer-assisted intervention*, pages 424–432. Springer, 2016.

[24] Erik V Cuevas, Daniel Zaldivar, and Raul Rojas. Kalman filter for vision tracking. 2005.

[25] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[26] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-fcn: Object detection via region-based fully convolutional networks. In *Advances in neural information processing systems*, pages 379–387, 2016.

[27] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 886–893. IEEE, 2005.

[28] Nameirakpam Dhanachandra, Khumanthem Manglem, and Yambem Jina Chanu. Image segmentation using k-means clustering algorithm and subtractive clustering algorithm. *Procedia Computer Science*, 54:764–771, 2015.

[29] Rongyi Du, Zhongren Peng, and Qingchang Lu. Comparison of sms calculation methods based on ngsim data for uav detection. In *2012 4th International Conference on Intelligent Human-Machine Systems and Cybernetics*, volume 2, pages 112–115. IEEE, 2012.

[30] M. Elloumi, R. Dhaou, B. Escrig, H. Idoudi, and L. A. Saidane. Monitoring road traffic with a uav-based system. In *2018 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6, 2018.

[31] Gunnar Farnebäck. Two-frame motion estimation based on polynomial expansion. In Josef Bigun and Tomas Gustavsson, editors, *Image Analysis*, pages 363–370, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. ISBN 978-3-540-45103-7.

[32] Pedro Felzenszwalb, David McAllester, and Deva Ramanan. A discriminatively trained, multiscale, deformable part model. In *2008 IEEE conference on computer vision and pattern recognition*, pages 1–8. IEEE, 2008.

[33] Pedro F Felzenszwalb, Ross B Girshick, and David McAllester. Cascade object detection with deformable part models. In *2010 IEEE Computer society conference on computer vision and pattern recognition*, pages 2241–2248. IEEE, 2010.

[34] Xiaoxue Feng and Peijun Li. A tree species mapping method from uav images over urban area using similarity in tree-crown object histograms. *Remote Sensing*, 11(17): 1982, 2019.

[35] David A Forsyth and Jean Ponce. *Computer vision: a modern approach*. Prentice Hall Professional Technical Reference, 2002.

[36] Howard Frank. *Expanded traffic-cam system in Monroe County will cost PennDOT 4.3M*, 2013 (accessed July 27, 2020). URL `http://www.poconorecord.com/apps/pbcs.dll/articlAID=/20130401/NEWS/1010402/-1/NEWS`.

[37] Spyros Gidaris and Nikos Komodakis. Object detection via a multi-region and semantic segmentation-aware cnn model. In *Proceedings of the IEEE international conference on computer vision*, pages 1134–1142, 2015.

[38] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[39] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[40] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Region-based convolutional networks for accurate object detection and segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 38(1):142–158, 2015.

[41] Ross B Girshick, Pedro F Felzenszwalb, and David McAllester. Discriminatively trained deformable part models, release 5. 2012.

[42] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323, 2011.

[43] Helmut Grabner, Michael Grabner, and Horst Bischof. Real-time tracking via on-line boosting. In *Bmvc*, volume 1, page 6, 2006.

[44] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[45] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(9):1904–1916, 2015.

[46] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[47] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[48] David Held, Sebastian Thrun, and Silvio Savarese. Learning to track at 100 fps with deep regression networks. In *European Conference on Computer Vision*, pages 749–765. Springer, 2016.

[49] João F Henriques, Rui Caseiro, Pedro Martins, and Jorge Batista. High-speed tracking with kernelized correlation filters. *IEEE transactions on pattern analysis and machine intelligence*, 37(3):583–596, 2014.

[50] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1314–1324, 2019.

[51] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[52] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. Forward-backward error: Automatic detection of tracking failures. In *2010 20th International Conference on Pattern Recognition*, pages 2756–2759. IEEE, 2010.

[53] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. Tracking-learning-detection. *IEEE transactions on pattern analysis and machine intelligence*, 34(7):1409–1422, 2011.

[54] Ugur Kart, Alan Lukezic, Matej Kristan, Joni-Kristian Kamarainen, and Jiri Matas. Object tracking by reconstruction with view-specific discriminative correlation filters. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1339–1348, 2019.

[55] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *International journal of computer vision*, 1(4):321–331, 1988.

[56] Alex Kendall, Vijay Badrinarayanan, and Roberto Cipolla. Bayesian segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding. *arXiv preprint arXiv:1511.02680*, 2015.

[57] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, pages 1942–1948 vol.4, 1995.

[58] Muhammad Arsalan Khan, Wim Ectors, Tom Bellemans, Davy Janssens, and Geert Wets. Uav-based traffic analysis: A universal guiding framework based on literature survey. *Transportation research procedia*, 22:541–550, 2017.

[59] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[60] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[61] Jing Li, Shuo Chen, Fangbing Zhang, Erkang Li, Tao Yang, and Zhaoyang Lu. An adaptive framework for multi-vehicle ground speed estimation in airborne videos. *Remote Sensing*, 11(10):1241, 2019.

[62] Zeming Li, Chao Peng, Gang Yu, Xiangyu Zhang, Yangdong Deng, and Jian Sun. Light-head r-cnn: In defense of two-stage object detector. *arXiv preprint arXiv:1711.07264*, 2017.

[63] Guosheng Lin, Anton Milan, Chunhua Shen, and Ian Reid. Refinenet: Multi-path refinement networks for high-resolution semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1925–1934, 2017.

[64] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.

[65] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[66] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8759–8768, 2018.

[67] Wei Liu, Andrew Rabinovich, and Alexander C Berg. Parsenet: Looking wider to see better. *arXiv preprint arXiv:1506.04579*, 2015.

[68] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[69] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[70] Alan Lukezic, Tomas Vojir, Luka ˇCehovin Zajc, Jiri Matas, and Matej Kristan. Discriminative correlation filter with channel and spatial reliability. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6309–6318, 2017.

[71] Ye Lyu, George Vosselman, Gui-Song Xia, Alper Yilmaz, and Michael Ying Yang. Uavid: A semantic segmentation dataset for uav imagery. *ISPRS Journal of Photogrammetry and Remote Sensing*, 165:108 – 119, 2020. ISSN 0924-2716. doi: https://doi.org/10.1016/j.isprsjprs.2020.05.009. URL `http://www.sciencedirect.com/science/article/pii/S0924271620301295`.

[72] Emmanuel Maggiori, Yuliya Tarabalka, Guillaume Charpiat, and Pierre Alliez. Can semantic labeling methods generalize to any city? the inria aerial image labeling benchmark. In *IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*. IEEE, 2017.

[73] Maitiniyazi Maimaitijiang, Vasit Sagan, Paheding Sidike, Ahmad M Daloye, Hasanjan Erkbol, and Felix B Fritschi. Crop monitoring using satellite/uav data fusion and machine learning. *Remote Sensing*, 12(9):1357, 2020.

[74] Tomasz Malisiewicz, Abhinav Gupta, and Alexei A Efros. Ensemble of exemplar-svms for object detection and beyond. In *2011 International conference on computer vision*, pages 89–96. IEEE, 2011.

[75] K. F. Man, K. S. Tang, and S. Kwong. Genetic algorithms: concepts and applications [in engineering design]. *IEEE Transactions on Industrial Electronics*, 43(5):519–534, 1996.

[76] Gellért Máttyus, Shenlong Wang, Sanja Fidler, and Raquel Urtasun. Hd maps: Fine-grained road segmentation by parsing ground and aerial images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3611–3619, 2016.

[77] Ali Reza Mehrabian and Caro Lucas. A novel numerical optimization algorithm inspired from weed colonization. *Ecological informatics*, 1(4):355–366, 2006.

[78] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. V-net: Fully convolutional neural networks for volumetric medical image segmentation. In *2016 fourth international conference on 3D vision (3DV)*, pages 565–571. IEEE, 2016.

[79] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

[80] Laurent Najman and Michel Schmitt. Watershed of a continuous function. *Signal Processing*, 38(1):99–112, 1994.

[81] Hyeonseob Nam and Bohyung Han. Learning multi-domain convolutional neural networks for visual tracking. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[82] Ishan Nigam, Chen Huang, and Deva Ramanan. Ensemble knowledge transfer for semantic segmentation. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1499–1508. IEEE, 2018.

[83] Haoran Niu, Nuria Gonzalez-Prelcic, and Robert W Heath. A uav-based traffic monitoring system-invited paper. In *2018 IEEE 87th Vehicular Technology Conference (VTC Spring)*, pages 1–5. IEEE, 2018.

[84] Richard Nock and Frank Nielsen. Statistical region merging. *IEEE Transactions on pattern analysis and machine intelligence*, 26(11):1452–1458, 2004.

[85] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. In *Proceedings of the IEEE international conference on computer vision*, pages 1520–1528, 2015.

[86] Norzailawati Mohd Noor, Alias Abdullah, and Mazlan Hashim. Remote sensing UAV/drones and its applications for urban areas: a review. *IOP Conference Series: Earth and Environmental Science*, 169:012003, jul 2018. doi: 10.1088/1755-1315/169/1/012003. URL https://doi.org/10.1088%2F1755-1315%2F169%2F1%2F012003.

[87] Kenji Okuma, Ali Taleghani, Nando De Freitas, James J Little, and David G Lowe. A boosted particle filter: Multitarget detection and tracking. In *European conference on computer vision*, pages 28–39. Springer, 2004.

[88] Bruno A Olshausen and David J Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609, 1996.

[89] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics*, 9(1):62–66, 1979.

[90] Adam Paszke, Abhishek Chaurasia, Sangpil Kim, and Eugenio Culurciello. Enet: A deep neural network architecture for real-time semantic segmentation. *arXiv preprint arXiv:1606.02147*, 2016.

[91] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[92] Press. *Dutch Government successfully uses Aerialtronics drones to control traffic*, 2015 (accessed July 27, 2020). URL https://www.suasnews.com/2015/07/dutch-government-successfully-uses-aerialtronics-drones-to-control-traffic/.

[93] Paulina Lyubenova Raeva, Jaroslav Šedina, and Adam Dlesk. Monitoring of crop fields using multispectral and thermal imagery from uav. *European Journal of Remote Sensing*, 52(sup1):192–201, 2019.

[94] Rajeev Ranjan, Swami Sankar, Carlos Castillo, and Rama Chellappa. An all-in-one convolutional neural network for face analysis. 11 2016.

[95] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[96] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[97] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[98] Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, and Silvio Savarese. Generalized intersection over union. June 2019.

[99] Tal Ridnik, Hussam Lawen, Asaf Noy, and Itamar Friedman. Tresnet: High performance gpu-dedicated architecture. *arXiv preprint arXiv:2003.13630*, 2020.

[100] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[101] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[102] Amir Sadeghian, Alexandre Alahi, and Silvio Savarese. Tracking the untrackable: Learning to track multiple cues with long-term dependencies. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 300–311, 2017.

[103] Seyed Sadegh Mohseni Salehi, Deniz Erdogmus, and Ali Gholipour. Tversky loss function for image segmentation using 3d fully convolutional deep networks. In *International Workshop on Machine Learning in Medical Imaging*, pages 379–387. Springer, 2017.

[104] Todd N Schoepflin and Daniel J Dailey. Dynamic camera calibration of roadside traffic management cameras for vehicle speed estimation. *IEEE Transactions on Intelligent Transportation Systems*, 4(2):90–98, 2003.

[105] Sean O Settle, Manasa Bollavaram, Paolo D'Alberto, Elliott Delaye, Oscar Fernandez, Nicholas Fraser, Aaron Ng, Ashish Sirasao, and Michael Wu. Quantizing convolutional neural networks for low-power high-throughput inference engines. *arXiv preprint arXiv:1805.07941*, 2018.

[106] Abhinav Shrivastava and Abhinav Gupta. Contextual priming and feedback for faster r-cnn. In *European conference on computer vision*, pages 330–348. Springer, 2016.

[107] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[108] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[109] J-L Starck, Michael Elad, and David L Donoho. Image decomposition via the combination of sparse representations and a variational approach. *IEEE transactions on image processing*, 14(10):1570–1582, 2005.

[110] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[111] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[112] Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019.

[113] M. Teichmann, M. Weber, M. Zöllner, R. Cipolla, and R. Urtasun. Multinet: Real-time joint semantic reasoning for autonomous driving. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1013–1020, 2018.

[114] Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun, and Christoph Bregler. Efficient object localization using convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 648–656, 2015.

[115] Amos Tversky. Features of similarity. *Psychological review*, 84(4):327, 1977.

[116] Darrenl Tzutalin et al. Labelimg, 2018. URL `https://github.com/tzutalin/labelImg`.

[117] Koen EA Van de Sande, Jasper RR Uijlings, Theo Gevers, and Arnold WM Smeulders. Segmentation as selective search for object recognition. In *2011 International Conference on Computer Vision*, pages 1879–1886. IEEE, 2011.

[118] MH Schultz van Haegen. *Model Flying Scheme*, 2019 (accessed July 22, 2020). URL `https://wetten.overheid.nl/BWBR0019147/2019-04-01`.

[119] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001*, volume 1, pages I–I. IEEE, 2001.

[120] Paul Viola and Michael J Jones. Robust real-time face detection. *International journal of computer vision*, 57(2):137–154, 2004.

[121] Chien-Yao Wang, Hong-Yuan Mark Liao, Yueh-Hua Wu, Ping-Yang Chen, Jun-Wei Hsieh, and I-Hau Yeh. 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*, pages 390–391, 2020.

[122] Panqu Wang, Pengfei Chen, Ye Yuan, Ding Liu, Zehua Huang, Xiaodi Hou, and Garrison Cottrell. Understanding convolution for semantic segmentation. In *2018 IEEE winter conference on applications of computer vision (WACV)*, pages 1451–1460. IEEE, 2018.

[123] Werner (IPF) Weisbrich. Startseite. URL `http://www.ipf.kit.edu/downloads_data_set_AIS_vehicle_tracking.php`.

[124] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. Simple online and realtime tracking with a deep association metric. In *2017 IEEE international conference on image processing (ICIP)*, pages 3645–3649. IEEE, 2017.

[125] Jia Wu, Xiu-Yun Chen, Hao Zhang, Li-Dong Xiong, Hang Lei, and Si-Hao Deng. Hyperparameter optimization for machine learning models based on bayesian optimization. *Journal of Electronic Science and Technology*, 17(1):26–40, 2019.

[126] Xiangqian Wu, Xin Shen, Lin Cao, Guibin Wang, and Fuliang Cao. Assessment of individual tree detection and canopy cover estimation using unmanned aerial vehicle based light detection and ranging (uav-lidar) data in planted forests. *Remote Sensing*, 11(8):908, 2019.

[127] Maoke Yang, Kun Yu, Chi Zhang, Zhiwei Li, and Kuiyuan Yang. Denseaspp for semantic segmentation in street scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3684–3692, 2018.

[128] Fengwei Yu, Wenbo Li, Quanquan Li, Yu Liu, Xiaohua Shi, and Junjie Yan. Poi: Multiple object tracking with high performance detection and appearance feature. In *European Conference on Computer Vision*, pages 36–42. Springer, 2016.

[129] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.

[130] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[131] Zhengxin Zhang, Qingjie Liu, and Yunhong Wang. Road extraction by deep residual u-net. *IEEE Geoscience and Remote Sensing Letters*, 15(5):749–753, 2018.

[132] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2881–2890, 2017.

[133] He Zhiwei, Liu Yuanyuan, and Ye Xueyi. Models of vehicle speeds measurement with a single camera. In *2007 International Conference on Computational Intelligence and Security Workshops (CISW 2007)*, pages 283–286. IEEE, 2007.

[134] Zongwei Zhou, Md Mahfuzur Rahman Siddiquee, Nima Tajbakhsh, and Jianming Liang. Unet++: A nested u-net architecture for medical image segmentation. In *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support*, pages 3–11. Springer, 2018.