Indian Institute of Technology Kanpur

Sandeep Kumar Routray

SURGE 2019

Height Invariant Object Detection of Aerial Images Using Domain Adaptation

Under guidance of Dr. Vinay P Namboodiri

July 2019

Contents

List of Figures

Acknowledgement

I would like to thank all the people who contributed in some way or the other to the work described in the project. First and foremost I offer my sincerest gratitude to my mentor, Dr. Vinay P Namboodiri, who has supported me throughout this project with his patience and knowledge while allowing me the room to work in my way. He motivated me throughout the project by sharing his research experience and, insisted me to do experimental work on my own to learn through experience and figuring out solutions to issues that arrived while working. I would like to thank my co-mentor, Mr. Vinod Kumar Kurmi, PhD Scholar, and colleague, Mr.Vaibhav Jindal for fruitful guidance, intellectual discussions and help in experimental work. This report would not have been possible without their help and support while writing programs and studying research papers. Finally, I express my most profound gratitude to my family for everything they have given to me.

Abstract

Object detection is one of the most active areas of research in the computer vision community. Recent advances in deep learning have resulted in state-of-the-art object detection algorithms like YOLOv3, Faster RCNN, etc. Using object detection on aerial images is particularly useful for defense tasks like unmanned surveillance, disaster management tasks like automated food supply through drones, identification of people/animals in need of help, etc. But training models for object detection requires a large amount of labelled data. Hence, for tasks like aerial imaging, where getting a large labelled dataset is expensive, other algorithms for training have to be devised which can be used on a small labelled dataset along with a large unlabelled dataset. Current object detection models when trained on images captured from a fixed height, the model do not generalize well when tested on images captured from varying heights. We want to use unsupervised domain adaptation to bridge this domain gap among images taken from various heights. We use YOLOv3-tiny as our base model for its low online time of running.

We propose an architecture that augments our base model with an adversarial domain discriminator which would aid learning of features invariant to the height at which the images were taken. Our approach is unsupervised as, during training time, we only require labels of the images taken from a fixed height and the labels of images taken from other heights are not required. To our knowledge, this is the first work that tries to use domain adaptation on images captured at various heights to train domain adaptation algorithms.

Chapter 1

Introduction

1.1 Aim

The key aim of this project is to be able to devise a height invariant object detection system. It should be easily trainable with minimum supervision of data. To achieve this, we apply unsupervised domain adaptation techniques to an existing object detection module, so that the end model could be trained on labelled images captured from a particular height and unlabelled images captured from varying heights.

We take a state-of-the-art object detection module, YOLOv3 (You Only Look Once Version 3) as our base model. Specifically, we consider YOLOv3-tiny model, a smaller version of YOLOv3 model designed for constrained environments, where dataset is limited. First, we train our base model on images captured from a particular height and test on images taken from varying heights. We obtain a clear dip in mean Average Precision (mAP), an evaluation metric, using such approach demonstrating the need for adaptation to make features height invariant. Then, we devise an architecture for a domain discriminator capable of identifying the domain of the images, which in this case is the height from which the images were captured. The domain discriminator would be connected to appropriate feature output layers of the base YOLOv3 model through a gradient reversal layer. This arrangement would make the domain discriminator adversarial to the training of the base model, which in turn would cause the base model to learn features to confuse the domain discriminator. Thus, the features learned by the base model would be invariant to the height at which the images were taken. Our approach is unsupervised as, during training time, we only require labels of the images taken from a fixed height and the labels of images taken from other heights are not required.

All throughout the study, we make use of the Stanford Drone Dataset. We extract the frames from the videos to construct our main dataset. But that only gave images from one particular height. For images from other heights, we cropped the former images at various scale and resized them to original size.

1.2 Object Detection

Object detection is a computer vision task that deals with detecting instances of semantic objects of a certain class (such as human, buildings, or cars) in digital images and videos. It includes object localization and object classification as sub tasks which must be solved together to create an efficient object detection system. More precisely, given an image, the problem of object detection involves localizing object of interest by drawing a bounding box around it and correctly identifying the class of the object. It is widely used in face detection, self-driving cars, crowd monitoring. It is also used in tracking objects, for example tracking a ball during a football match or tracking a person in a video.

Every object has its own special features that help in classifying the class. Object detection exploits these special features. For example, when looking for circles, objects that are at a particular distance from a point (i.e. the center) are sought. Similarly, when looking for squares, objects that are perpendicular at corners and have equal side lengths are needed. A similar approach is used for face identification where eyes, nose, and lips can be found and features like skin color and distance between eyes can be found. Methods of object detection generally fall into either machine learning-based approaches or deep learning-based approaches. For Machine Learning approaches, it becomes necessary to first define features using one of the methods below, then using a technique such as support vector machine (SVM) to do the classification. On the other hand, deep learning techniques that are able to do end-to-end object detection without specifically defining features, and are typically based on convolutional neural networks (CNN). See Figure 1 for an illustration of a common CNN architecture.

- Machine Learning approaches:
  - Viola–Jones object detection framework based on Haar features
  - Scale-invariant feature transform (SIFT)
  - Histogram of oriented gradients (HOG) features[1]
- Deep Learning approaches:
  - Region Proposals (R-CNN, Fast R-CNN, Faster R-CNN)
  - Single Shot MultiBox Detector (SSD)
  - You Only Look Once (YOLO)

In this study, we will only consider YOLO object detection model, which would be described in detail in later sections. Also, we specifically focus on performing detections in aerial images captured by a drone. The dataset used would be described in detail in a later section.

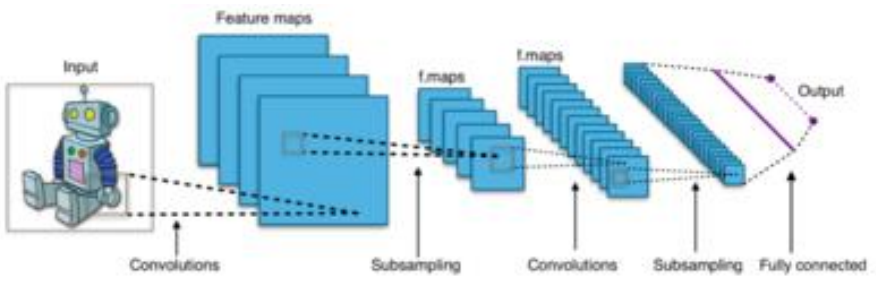


Figure 1 Typical CNN architecture, credits: https://www.jessicayung.com/explaining-tensorflow-code-for-a-convolutional-neural-network/

1.3 Domain Adaptation

Domain adaptation usually involves an algorithm trying to transfer two domains, usually called source and target, into a common domain. It can do so by either by translating one domain into the other, or to find a common embedding between the two domains (see Figure 2).

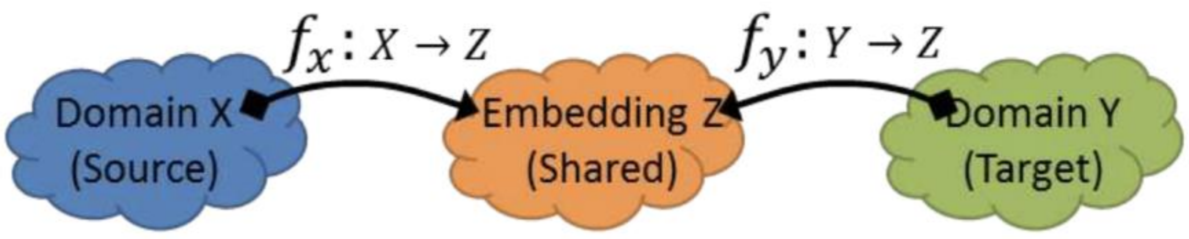


Figure 2 Domain adaptation by finding a common embedding, Image to Image Translation for Domain Adaptation, Murez et al., 2016

There are several contexts of domain adaptation differing in the information considered for the target task.

- The **unsupervised domain adaptation**: the learning sample contains a set of labelled source examples, a set of unlabelled source examples and a set of unlabelled target examples.
- The **semi-supervised domain adaptation**: in this situation, we also consider a "small" set of labelled target examples.
- The **supervised domain adaptation**: all the examples considered are supposed to be labelled.

We briefly describe the algorithmic principles based on which domain adaptation techniques are constructed.

- **Reweighting algorithms**: The objective is to reweight the source labelled sample such that it "looks like" the target sample (in term of the error measure considered)
- **Iterative algorithms**: A method for adapting consists in iteratively "auto-labelling" the target examples. The principle is simple:
    - a model h is learned from the labelled examples;
    - h automatically labels some target examples;
    - a new model is learned from the new labelled examples.

  Note that there exist other iterative approaches, but they usually need target labelled examples.
- **Search of a common representation space**: The goal is to find or construct a common representation space for the two domains. In the common space, the domains should be close to each other while ensuring good performances on the source labelling task. This can be achieved through the use of adversarial machine learning techniques where feature representations from samples in different domains are encouraged to be

indistinguishable. This is the approach followed in this paper and would be explained in details in the later sections.

- **Hierarchical Bayesian Model**: The goal is to construct a Bayesian hierarchical model $p(n)$, which is essentially a factorization model for counts n to derive domain-dependent latent representations allowing both domain-specific and globally shared latent factors.

In this study, we make use of unsupervised domain adaptation technique. In our case, the images captured from a fixed height forms the source domain. The target domain consists of images taken from all other heights. We augment the main object detection module with an adversarial domain discriminator to enable learning a common feature space invariant to height.

Chapter 2

Background

2.1 Intersection Over Union (IoU) Metric

The model output for predicted bounding box is extremely unlikely to be as an exact primary bounding box in reality. Therefore, to measure how accurate is the object identified in the Image/Frame we can make use of metric IoU. IoU is simply the ratio of area of intersection of the ground truth box and the predicted box, and the union of the area of these two boxes. This gives us an option to consider the object detected as complete or not. Generally, an IoU value greater than 0.5 for a prediction is considered good for object detection tasks.

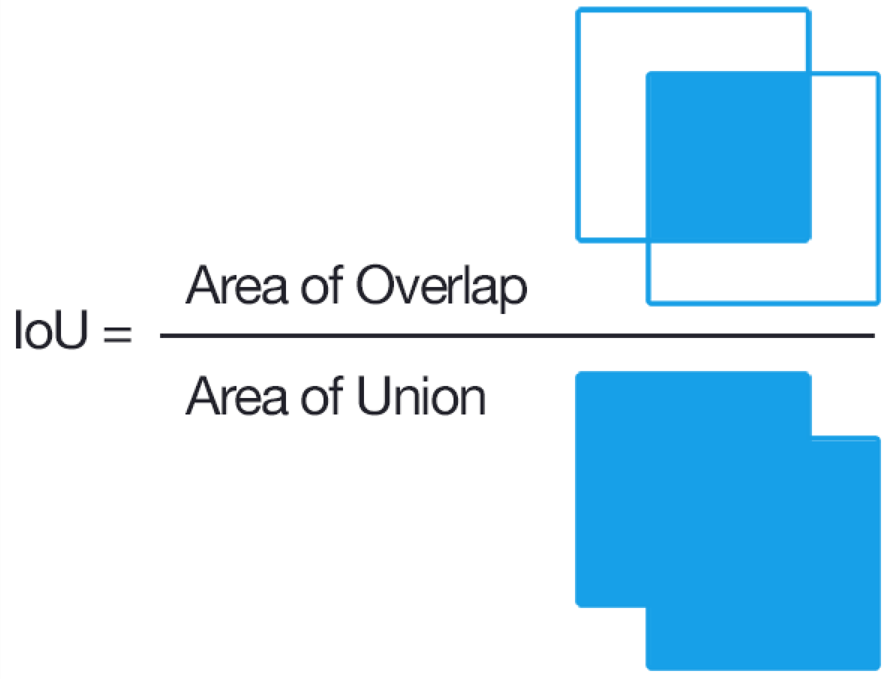


Figure 3 Pictorial depiction of IoU calculation, credit: https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/

2.2 Precision and Recall

Precision measures how accurate is your predictions. i.e. the percentage of your predictions are correct.

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

Recall measures how good you find all the positives.

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

F1 score is a popular metric defined as the harmonic mean of Precision and Recall.

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall}$$

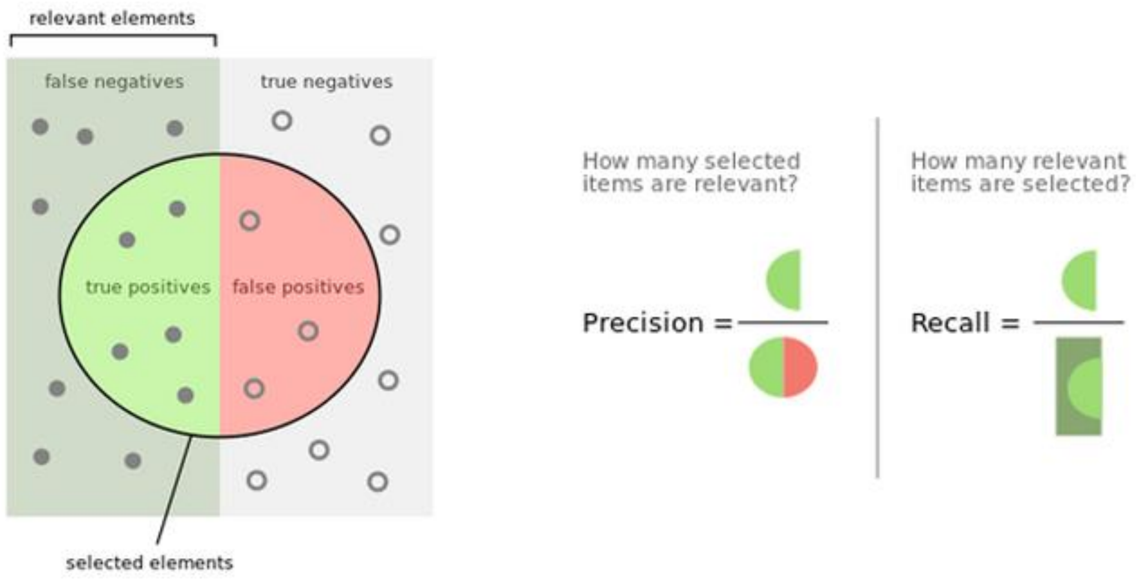


Figure 4 Illustrating Precision and Recall

2.3 Mean Average Precision (mAP)

mAP is a popular metric used to evaluate object detection algorithms. To calculate mAP, firstly AP (average precision) for each is class is calculated. Using the AP values for different classes, the final mAP score is calculated.

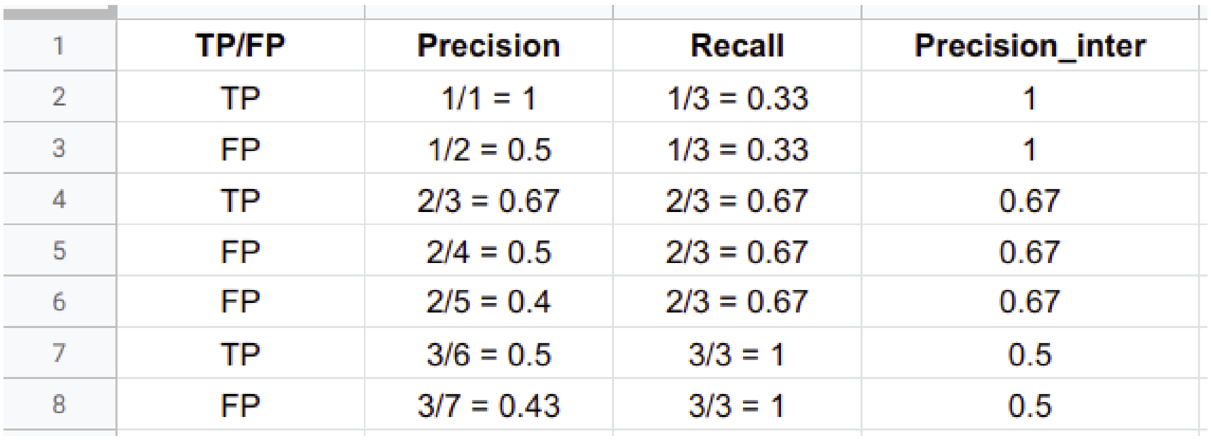
Calculating AP

To define AP, we first need to define True Positives and False Positives. For different datasets, we predefine an IoU threshold (say 0.5) in classifying whether a prediction is True Positive or False Positive.

AP is formally defined as the area under precision-recall curve as recall approaches 0. Each predicted bounding box would have its confidence level, usually given by its softmax layer, and would be used to rank the output. To improve the metric, AP is modified to calculate the area between the interpolated precision and recall curve. Their intention of interpolating the PR curve was to reduce the impact of “wiggles” caused by small variations in the ranking of detections. The interpolated precision, p_{interp}, is calculated at each recall level, r, by taking the maximum precision measured for that $r.$ The formula is given as such:

$$p_{interp} = max_{r':r'>r}\ p(r')$$

To get a better understanding of the process used for calculating AP, consider an example for the person class with 3 TP and 4 FP. We calculate the corresponding precision, recall, and interpolated precision given by the formulas defined above.



1	TP/FP	Precision	Recall	Precision_inter
2	TP	1/1 = 1	1/3 = 0.33	1
3	FP	1/2 = 0.5	1/3 = 0.33	1
4	TP	2/3 = 0.67	2/3 = 0.67	0.67
5	FP	2/4 = 0.5	2/3 = 0.67	0.67
6	FP	2/5 = 0.4	2/3 = 0.67	0.67
7	TP	3/6 = 0.5	3/3 = 1	0.5
8	FP	3/7 = 0.43	3/3 = 1	0.5

Figure 5 Calculation table for plotting PR curve with an example of 3 TP and 4 FP. Rows correspond to BB with person classification ordered by their respective softmax confidence.

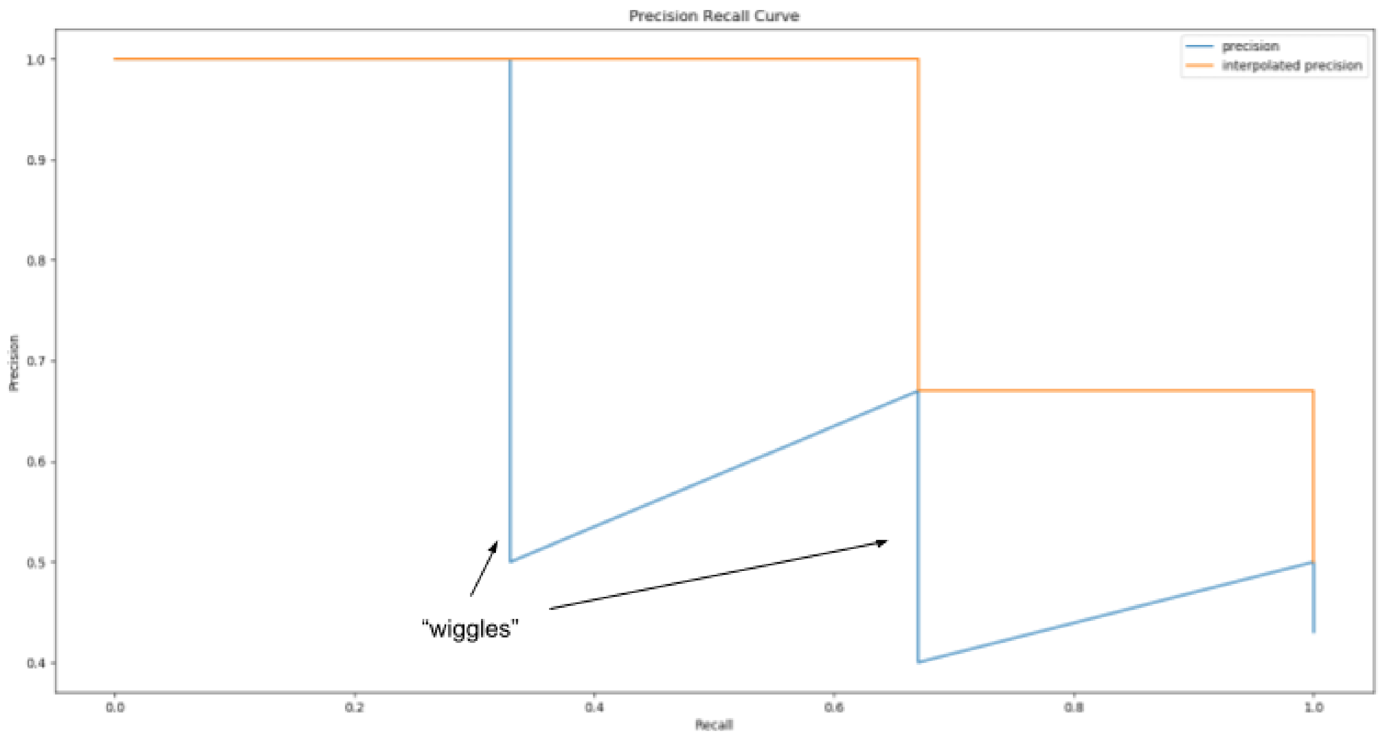


Figure 6 Precision-Recall Curve from the rankings given above, credits: https://towardsdatascience.com/breaking-down-mean-average-precision-map-ae462f623a52

The AP is then calculated by taking the area under the interpolated PR curve. This is done by segmenting the recalls evenly to 11 parts:

{0, 0.1, 0.2, …, 0.9, 1} and considering the corresponding precision values at those point. We get the following:

$$AP = \frac{1}{11} \sum_{r \in \{0,0.1,\ldots,1.0\}} p_{interp}(r)$$

$$= \frac{1}{11}(1+1+1+1+1+1+1+0.67+0.67+0.67+0.5)$$

$$\approx 0.865$$

Some other methods for calculating AP are also used which include calculating exact area of the orange curve in the figure above (later PASCAL VOC competitions) and using a 101-point interpolated AP (COCO mAP).

Calculating mAP

The mAP for object detection is the average of the AP calculated for all the classes. It is also important to note that for some papers, they use AP and mAP interchangeably.

2.4 YOLO Object Detection Algorithm

2.4.1 Introduction

You only look once (YOLO) is a popular state-of-the-art, real-time object detection system. The first version of the algorithm was proposed in Redmon et al., 2015. Since then, two more versions have been proposed by the author, with each new version better than the previous one. The latest version, YOLOv3, achieves real-time detection on videos with high mAP values, a metric used to compare different object detection algorithms. Other works on object detection repurpose classifiers to perform detection. Instead, YOLO frames object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. A single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation. Since the whole detection pipeline is a single network, it can be optimized end-to-end directly on detection performance.

2.4.2 Understanding YOLOv3

YOLO only uses convolutional layers, making it a fully convolutional network (FCN). It has 75 convolutional layers, with skip connections and upsampling layers. No form of pooling is used for downsampling the input, and instead a convolutional layer with stride 2 is used to downsample the feature maps. This helps in preventing loss of low-level features often attributed to pooling. Being an FCN, YOLO is invariant to the size of the input, but when working with batch sizes greater than 1, we need to give inputs of same size only.

The network downsamples the image by a factor called the **stride** of the network. For example, if the stride of the network is 32, then an input image of size 608 x 608 will yield an output of size 19 x 19. Each of these 19 x 19 boxes then predicts a fixed number of bunding boxes.

2.4.3 Network Output

Typically, the features learned by the convolutional layers are passed onto a classifier/regressor which makes the detection prediction (coordinates of the bounding boxes, the class label, etc). But since YOLO is a fully convolutional network, the prediction is done by using a convolutional layer which uses one-one convolutions. The output given by our network represents a feature map**.**

Depth-wise, we get $(B \times (5 + C))$ entries in the feature map. B represents the number of bounding boxes each cell can predict. According to the paper, each of these B bounding boxes may specialize in detecting a certain kind of object. Each of the bounding boxes have $5 + C$ attributes, which describe the center coordinates, the dimensions, the objectness score and C class confidences for each bounding box. YOLO v3 predicts 3 bounding boxes for every cell. It is expected that each cell will predict one or more boxes corresponding to its feature map.

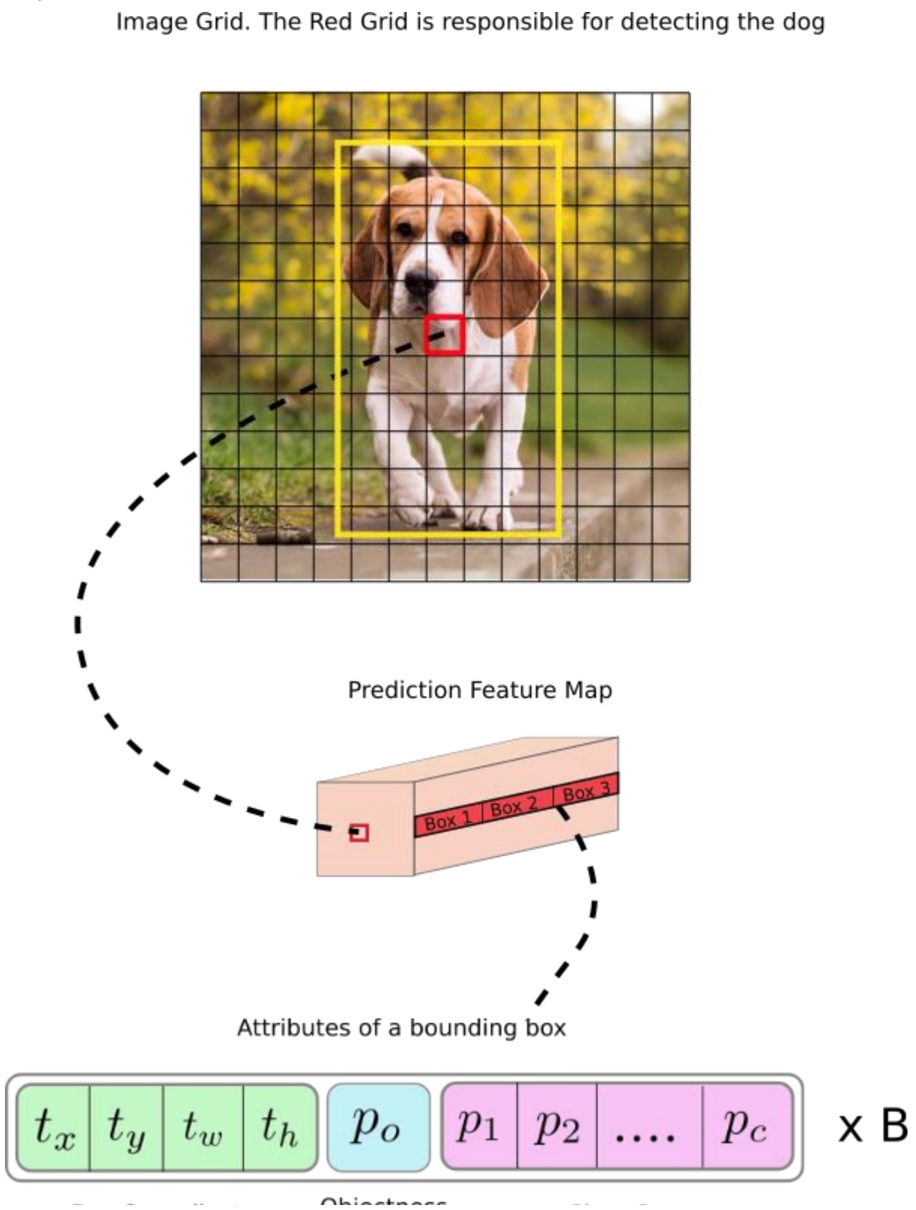


Figure 7 Depiction of outputs of the network, credits: https://blog.paperspace.com/how-to-implement-a-yolo-object-detector-in-pytorch/

2.4.4 Anchor Boxes

Since feature map corresponding to each cell in YOLOv3 can predict 3 bounding boxes, we need to know exactly which box will predict a particular object. It might make sense to predict the width and the height of the bounding box, but in practice, that leads to unstable gradients during training. Instead, most of the modern object detectors predict log-space transforms, or simply offsets to pre-defined default bounding boxes called anchors. Then, these transforms are applied to the anchor boxes to obtain the prediction. The bounding box responsible for detecting a particular object will be the one whose anchor has the highest IoU (Intersection over Union) with the ground truth box.

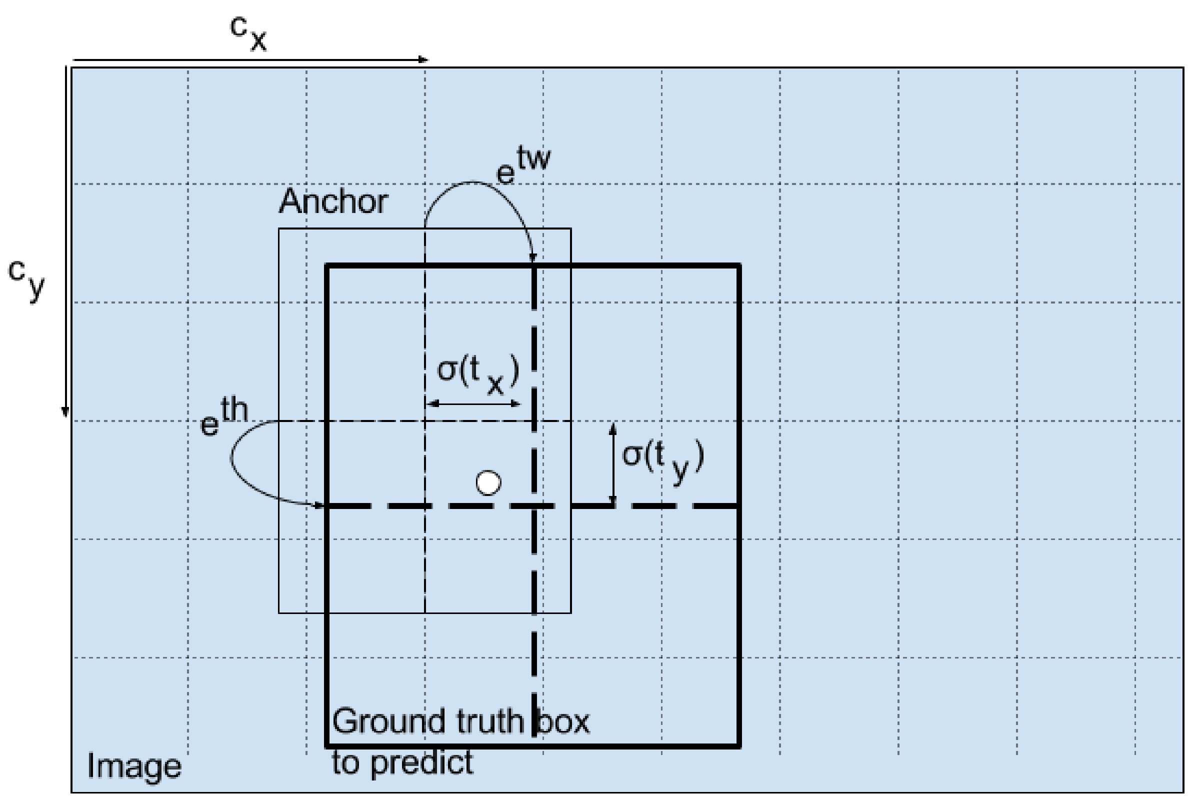


Figure 8 Illustration of an anchor box and the transformation used to predict the bounding box, credits: https://christopher5106.github.io/object/detectors/2017/08/10/bounding-box-object-detectors-understanding-yolo.html

The transforms used are:

$$b_x = \sigma(t_x) + c_x$$

$$b_y = \sigma(t_y) + c_y$$

$$b_w = p_w e^{t_w}$$

$$b_h = p_h e^{t_h}$$

Here b_x, b_y, b_w, b_h are the x, y center co-ordinates, width and height of our prediction. t_x, t_y, t_w, t_h is what the network outputs. c_x and c_y are the top-left co-ordinates of the grid. p_w and p_h are anchors dimensions for the box. The sigmoid term returns a number between 0 and 1 to locate the position of the centre relative to the top left corner of the cell.

2.4.5 Objectness Score and Class Confidences

Object score represents the probability that an object is contained inside a bounding box. It should be 1 for boxes that may contain an object and zero for others. Since this score is to be interpreted as probability, it is also passed through the sigmoid function.

Class confidences represent the probabilities of the detected object belonging to a particular class (Dog, cat, banana, car etc). This is also passed through the sigmoid function.

2.4.6 Different Scale for Prediction

YOLO v3 makes prediction across 3 different scales. The detection layer is used make detection at feature maps of three different sizes, having **strides 32, 16 and 8** respectively.

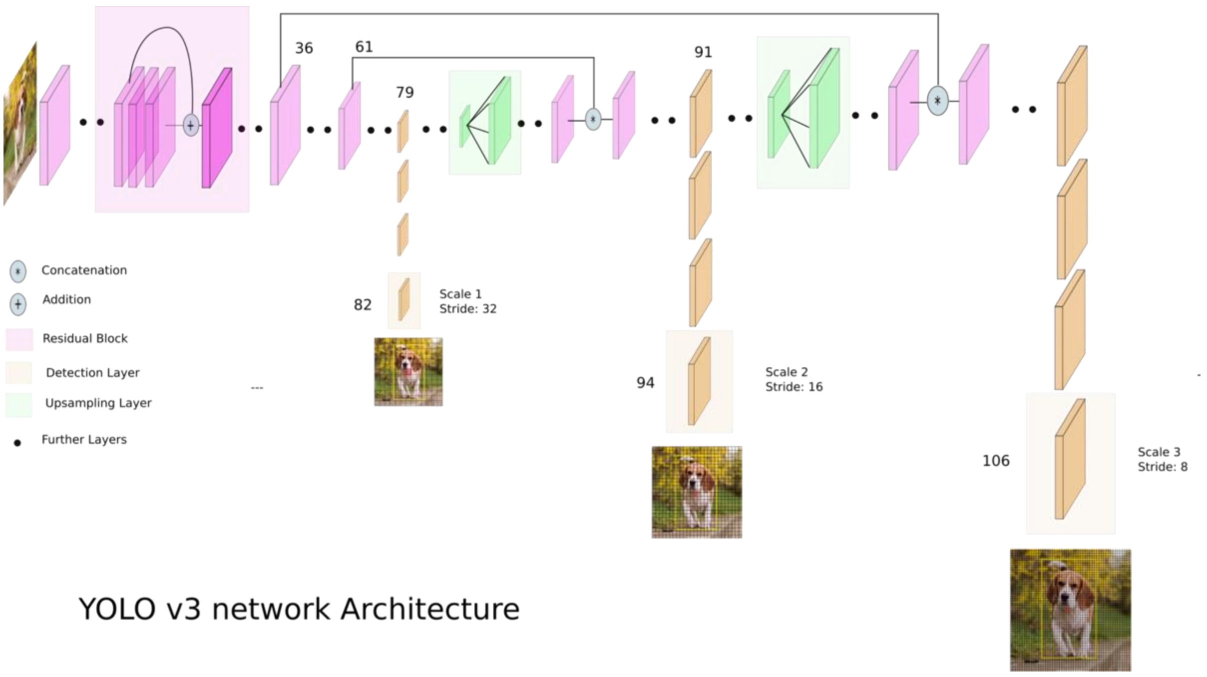


Figure 9 Multiscale prediction by YOLO network, credits: https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b

The network downsamples the input image until the first detection layer, where a detection is made using feature maps of a layer with stride 32. Further, layers are upsampled by a factor of 2 and concatenated with feature maps of a previous layers having identical feature map sizes. Another detection is now made at layer with stride 16. The same upsampling procedure is repeated, and a final detection is made at the layer of stride 8. At each scale, each cell predicts 3 bounding boxes using 3 anchors, making the total number of anchors used 9. The anchors are different for different scales with smaller anchors for smaller strides.

2.4.7 Output Processing

Since we have a large number of total bounding boxes that can be predicted, we need to somehow reduce the number of actual predictions. A two-step technique is used for this task.

1. Thresholding by Object Confidence – Boxes having confidence scores below the threshold are ignored.

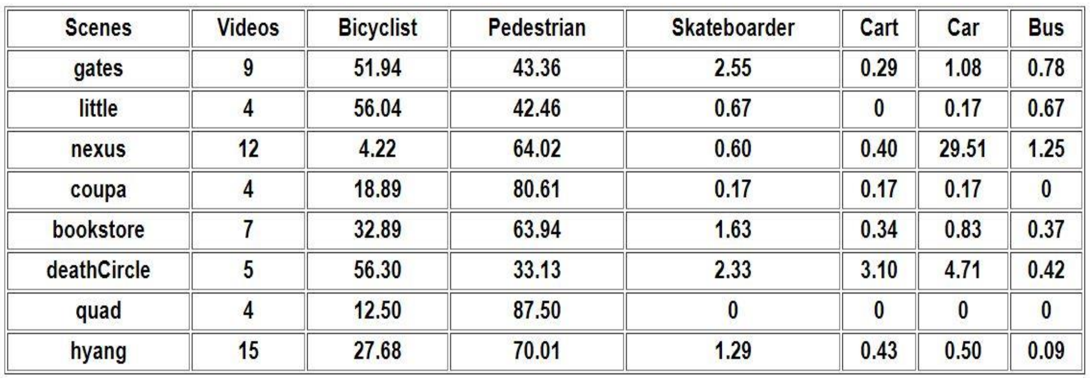
2. Non-maximum Suppression – If more than one boxes of a cell or boxes from adjacent cells predict for a single object, then NMS is used to eliminate these boxes. In such a case, the box with the highest score for a particular class is kept and other boxes which have an IoU greater than some threshold with this box are ignored. This process runs several times for a particular class until the NMS procedure is complete for that class.

This method effectively eliminates almost all of the wrong predictions.

2.5 Stanford Drone Dataset (SDD)

To advance the research in tasks such as aerial target tracking and trajectory forecasting for pedestrians, bicycles, cars etc., a large-scale dataset was released by Stanford's Computer Vision and Geometry Lab (Robicquet et al., 2016).

The dataset contains aerial videos captured around Stanford campus and has eight unique scenes. The videos have been captured from drones at an approximate height of 80 m. Rectangular bounding boxes and object classes for the bounding boxes have been provided for each frame in each video. The number of videos in each scene and the percentage of each agent in each scene is reported below.



Scenes	Videos	Bicyclist	Pedestrian	Skateboarder	Cart	Car	Bus
gates	9	51.94	43.36	2.55	0.29	1.08	0.78
little	4	56.04	42.46	0.67	0	0.17	0.67
nexus	12	4.22	64.02	0.60	0.40	29.51	1.25
coupa	4	18.89	80.61	0.17	0.17	0.17	0
bookstore	7	32.89	63.94	1.63	0.34	0.83	0.37
deathCircle	5	56.30	33.13	2.33	3.10	4.71	0.42
quad	4	12.50	87.50	0	0	0	0
hyang	15	27.68	70.01	1.29	0.43	0.50	0.09

Figure 10 Summary of Stanford Drone Dataset

Here we consider a few examples from SDD.

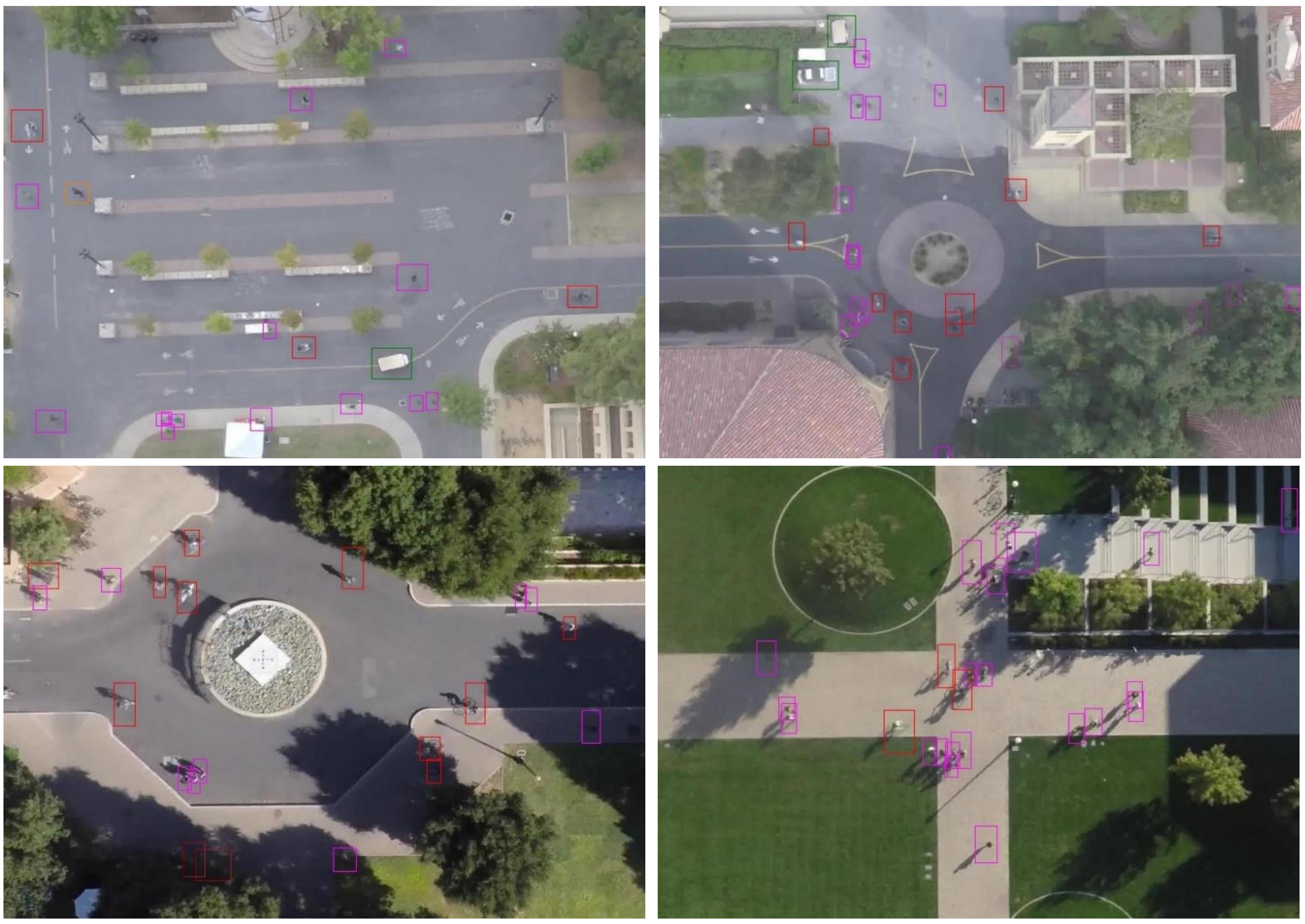
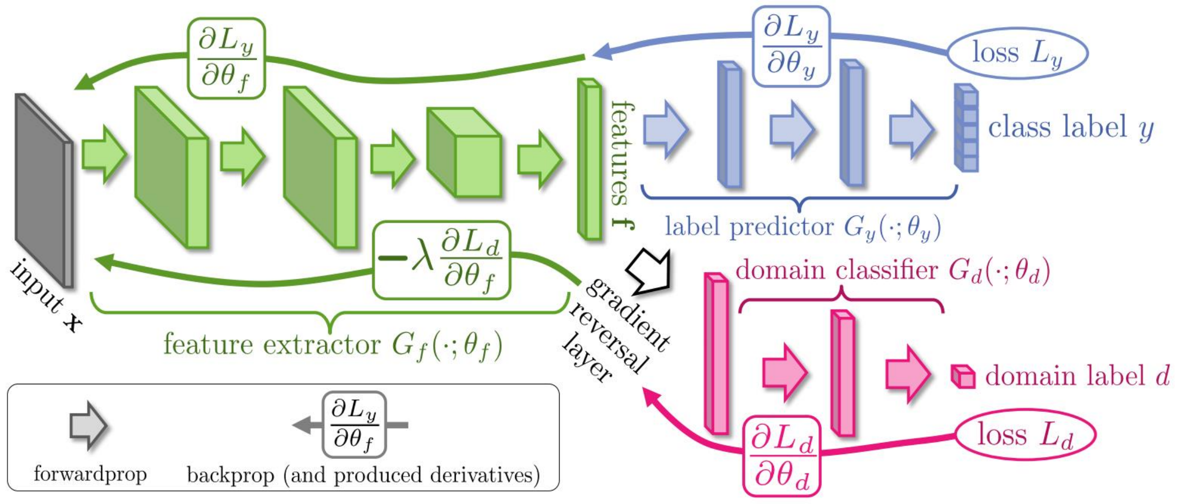


Figure 11 A few examples from Stanford Drone Dataset. Pedestrians are labelled in pink, bicyclists in red, skateboarders in orange, and cars in green.

2.6 Unsupervised Domain Adaptation Using Adversarial Methods

Using domain adaptation, we want to learn feature representations that are both discriminative and invariant to the changes of domain, i.e. have the same or very similar distributions in the source and target domains. This is achieved by attaching a domain classifier to the feature generator, in addition to the label classifier through gradient reversal layer (GRL). Due to the GRL, the feature generator becomes an adversary of the domain classifier. While the parameters of the classifiers are optimized in order to minimize their error on the training set, the parameters of the underlying feature generator are optimized in order to minimize the loss of the label classifier and to maximize the loss of the domain classifier. The latter encourages domain-invariant features to emerge in the course of the optimization. See the figure below for an illustrative explanation. We would now consider the details of the proposed model and formalize the approach.



$\frac{\partial L_y}{\partial \theta_f}$
$\frac{\partial L_y}{\partial \theta_y}$
loss L_y
features $\mathbf{f}$
class label y
label predictor $G_y(\cdot;\theta_y)$
$-\lambda \frac{\partial L_d}{\partial \theta_f}$
input x
feature extractor $G_f(\cdot;\theta_f)$
gradient reversal layer
domain classifier $G_d(\cdot;\theta_d)$
domain label d
$\frac{\partial L_d}{\partial \theta_d}$
loss L_d
forwardprop
backprop (and produced derivatives)

Figure 12 The proposed architecture includes a feature extractor (green) and a label predictor (blue), which together form a standard feed-forward architecture. Both feature extractor and label predictor are parts of YOLOv3-tiny network. Unsupervised domain adaptation is achieved by adding a domain classifier (red) connected to the feature extractor via a gradient reversal layer that multiplies the

gradient by a certain negative constant during the backpropagation-based training. Otherwise, the training proceeds in a standard way and minimizes the label prediction loss (for source examples) and the domain classification loss (for all samples). Gradient reversal ensures that the feature distributions over the two domains are made similar (as indistinguishable as possible for the domain classifier), thus resulting in the domain-invariant features. Unsupervised Domain Adaptation by Backpropagation, Ganin et al., 2015

2.6.1 Proposed Model

We assume that the model works with input samples $\mathbf{x} \in X$, where X is some input space and certain label y from the label space Y ($Y = \{1, 2, 3, \dots L\}$). We further assume that there exists two distributions $\mathcal{S}(x, y)$ and $\mathcal{T}(x, y)$ on $X \otimes Y$, which will be referred to as the source distribution and the target distribution (or the source domain and the target domain) and there exists a shift between the two domains (known as the domain shift).

We want to be able to predict y given the input $\mathbf{x}$ for the target distribution. At training time, we have a large set of training samples $\{\mathbf{x_1}, \mathbf{x_2}, \dots, \mathbf{x_n}\}$ from both source and target domains distributed according to the marginal distributions $\mathcal{S}(x)$ and $\mathcal{T}(x)$. Let d_i denote the binary variable for the ith training example indicating whether $\mathbf{x_i}$ come from the source distribution ($\mathbf{x_i} \sim \mathcal{T}(\mathbf{x})$ if $d_i = 0$) or from the target distribution ($\mathbf{x_i} \sim \mathcal{S}(\mathbf{x})$ if $d_i = 0$). For the examples from the source distribution ($d_i = 0$) the corresponding labels $y_i \in Y$ are known at training time. For the examples from the target domains, we do not know the labels at training time, and we want to predict such labels at test time.

We propose an architecture that for each input $\mathbf{x}$ predicts its label $y \in Y$ and its domain label $d \in \{0,1\}$. We decompose such mapping into three parts. We assume that the input $\mathbf{x}$ is first mapped by a mapping G_f (a *feature extractor*) to a D-dimensional feature vector $\mathbf{f} \in \mathbb{R}^D$ with parameters θ_f, i.e. $\mathbf{f} = G_f\ (\mathbf{x};\ \theta_f)$. Then, the feature vector $\mathbf{f}$ is mapped by a mapping G_y (*label predictor*) to the label y, and we denote the parameters of this mapping with θ_y. Finally, the same feature vector $\mathbf{f}$ is mapped to the domain label d by a mapping G_d (*domain classifier*) with parameters θ_d. (See Figure 3)

During the training stage, we aim to minimize the label prediction loss on the annotated part (i.e. the source part) of the training set, and the parameters of both the feature extractor and the label predictor are thus optimized in order to minimize the empirical loss for the source domain samples. This ensures the discriminativeness of the features $\mathbf{f}$ and the overall good prediction performance of the combination of the feature extractor and the label predictor on the source domain.

At the same time, we want to make the features $\mathbf{f}$ domain invariant. That is , we want to make the distributions $\mathcal{S}(\mathbf{f}) = \{G_f(\mathbf{x};\ \theta_f)\ |\ \mathbf{x}\sim\mathcal{S}(\mathbf{x})\}$ and $\mathcal{T}(\mathbf{f}) = \{G_f(\mathbf{x};\ \theta_f)\ |\ \mathbf{x}\sim\mathcal{T}(\mathbf{x})\}$ to be similar. We consider the loss of the domain classifier G_d as an estimate of the dissimilarity of the distributions $\mathcal{S}(\mathbf{f})$ and $\mathcal{T}(\mathbf{f})$, provided that the parameters θ_d of the domain classifier have been trained to discriminate between the two feature distributions in an optimal way. Thus, at training time, in order to obtain domain-invariant features, we seek the parameters θ_f of the feature mapping that maximize the loss of the domain classifier (by making the two feature distributions as similar as possible), while simultaneously seeking the parameters θ_d of the domain classifier that minimize the loss of the domain classifier. In addition, we seek to minimize the loss of the label predictor.

Formally, we consider the following functional:

$$\begin{aligned} E(\theta_f, \theta_y, \theta_d) \\ &= \sum_{\substack{i=1\ldots N\\ d_i=0}} L_y\big(G_y\big(G_f(\mathbf{x}_i;\ \theta_f);\theta_y\big), y_i\big) \\ &- \lambda \sum_{i=1\ldots N} L_d\big(G_d\big(G_f(\mathbf{x}_i;\ \theta_f);\ \theta_d\big), y_i\big) \\ &= \sum_{\substack{i=1\ldots N\\ d_i=0}} L_y^i(\theta_f, \theta_y) - \lambda \sum_{i=1\ldots N} L_d^i(\theta_f, \theta_d) \end{aligned}$$

Here, $L_y(.,.)$ is the loss for label prediction (e.g. cross entropy, mean squared error), $L_d(.,.)$ is the loss for the domain classification (e.g. logistic), while L_y^i and L_d^i denote the corresponding loss functions evaluated at the ith training example.

Our objective is to find the parameters $\hat{\theta}_f, \hat{\theta}_y, \hat{\theta}_d$ that deliver a saddle point of the above functional:

$$(\hat{\theta}_f, \hat{\theta}_y) = \arg\min_{\theta_f, \theta_y} E(\theta_f, \theta_y, \hat{\theta}_d)$$

$$\hat{\theta}_d = \arg\max_{\theta_d} E(\hat{\theta}_f, \hat{\theta}_y, \theta_d)$$

At the saddle point, the parameters θ_d of the domain classifier minimize the domain classification loss (since it enters into the functional with the minus sign) while the parameters θ_y of the label predictor minimize the label prediction loss. The feature mapping parameters θ_f minimize the label prediction loss (i.e. the features are discriminative), while maximizing the domain classification loss (i.e. the features are domain-invariant). The parameter λ controls the trade-off between the two objectives that shape the features during learning.

2.6.2 Optimization Procedure for the Proposed Model

A saddle point of the functional, as defined above, can be found as a stationary point of the following stochastic updates:

$$\theta_f \leftarrow \theta_f - \mu(\frac{\partial L_y^i}{\partial \theta_f} - \lambda \frac{\partial L_d^i}{\partial \theta_f^i})$$

$$\theta_y \leftarrow \theta_y - \mu \frac{\partial L_y^i}{\partial \theta_y}$$

$$\theta_d \leftarrow \theta_d - \mu \frac{\partial L_d^i}{\partial \theta_d}$$

where μ is the learning rate (which is usually varied over time).

The proposed update equations are very similar to the stochastic gradient descent (SGD) except for the $-\lambda$ factor in the first update equations. In order to implement the update equations as SGD, we introduce a special gradient reversal layer (GRL), which would be inserted between feature extractor and domain classifier. The gradient reversal layer has no parameters associated with it (apart from the meta-parameter λ, which is not updated by backpropagation). During the forward propagation, GRL acts as an identity transform. During the backpropagation though, GRL takes the gradient from the subsequent level, multiplies it by −λ and passes it to the preceding layer. Implementing such layer using existing object-oriented packages for deep learning is simple, as defining procedures for forwardprop (identity transform), backprop (multiplying by a constant), and parameter update (nothing) is trivial.

Mathematically, we can treat the gradient reversal layer as “pseudo-function” $R_\lambda(\mathbf{x})$ defined by two (incompatible)equations describing forward and backward propagation behaviour:

$$R_\lambda(\mathbf{x}) = \mathbf{x}$$

$$\frac{dR_\lambda(\mathbf{x})}{d\mathbf{x}} = -\lambda\mathbf{I}$$

where $\mathbf{I}$ is an identity matrix. We can then define the objective “pseudo-function” of $(\theta_f, \theta_y, \theta_d)$ that is being optimized by the stochastic gradient descent method.

$$\begin{aligned}\tilde{E}(\theta_f, \theta_y, \theta_d) \\ &= \sum_{\substack{i=1\dots N \\ d_i=0}} L_y\big(G_y\big(G_f(\mathbf{x}_i;\ \theta_f);\theta_y\big), y_i\big) \\ &+ \sum_{i=1\dots N} L_d\left(G_d\left(R_\lambda\left(G_f(\mathbf{x}_i;\theta_f)\right);\theta_d\right), y_i\right)\end{aligned}$$

This allows us to implement the required stochastic update equations for getting saddle point as doing SGD on the modified objective.

Chapter 3

Implementation Details

3.1 Dataset Preparation

Getting a suitable dataset for our study was a challenging task since, as per our knowledge, there is no existing dataset consisting of height-wise annotated images. So, we created a working dataset based on the Stanford Drone Dataset. We have covered the details about the dataset in the earlier chapter.

The dataset was particularly suited for our purposes since it consisted of videos taken from a fixed height. We extracted about 2400 annotated frames from these videos which we used as our source domain trainset (corresponding to the particular height at which the videos were captured). Similarly, we extracted another set of annotated 260 images to use as source domain testset.

Further, to create our target domain dataset, we simulated variations in height on another set of 2400 frames extracted from the videos. We cropped the images (to 90%, 80%, 70%) and resized them to original size in order to create effects of variation in height. An example of such simulation on a pair of images is given below.

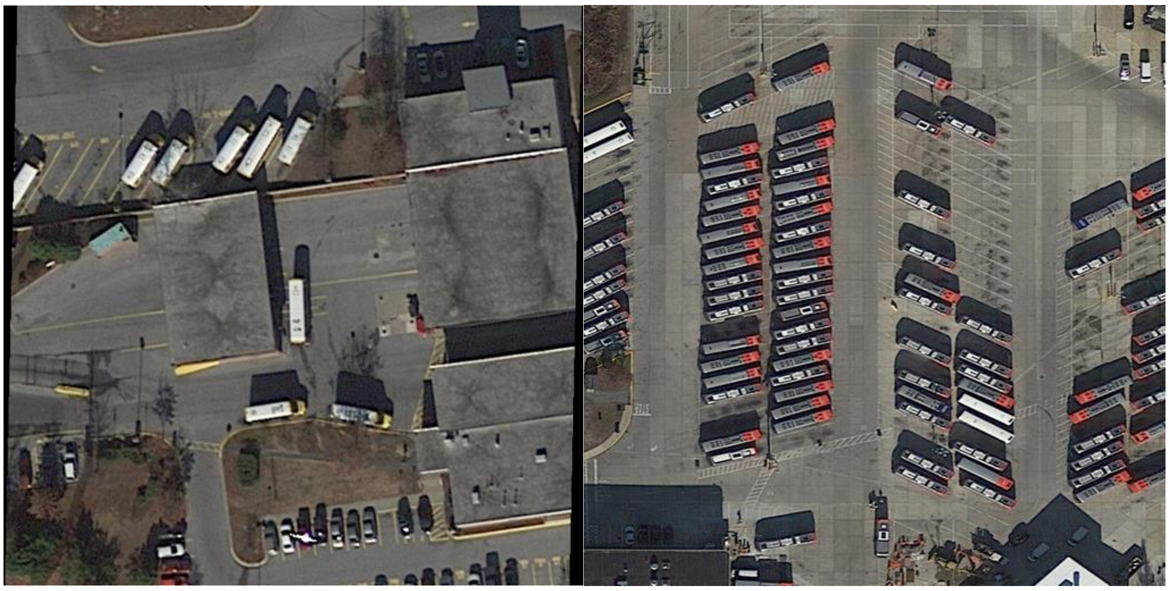


Figure 13 Before simulating height variation

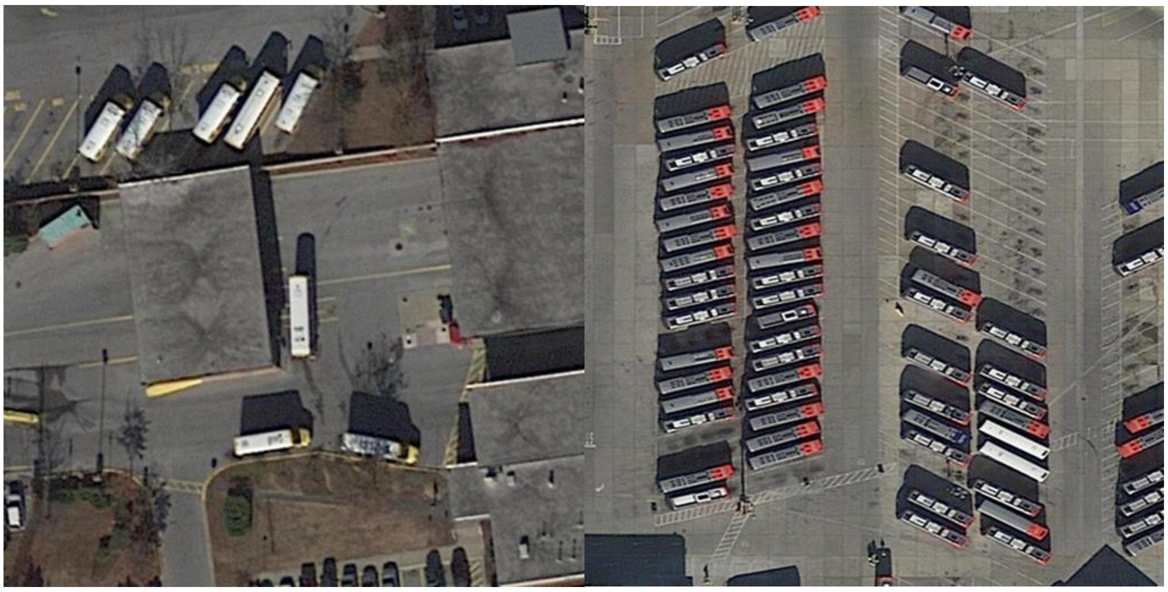


Figure 14 After simulating height variation

A point to note is that we did not create annotations for the target domain trainset images. We followed a similar procedure to create a target domain testset images consisting of 260 images. We annotated the testset images so that we could test the performance of our model.

3.2 Training Base Model and Establishing the Problem

Here, we describe the training process of the YOLOv3-tiny model. For each ground truth box, the model assigns a bounding box, whose anchor has the maximum overlap with the ground truth box. These predictions are preceded by confidence thresholding and non-max suppression (NMS). Confidence thresholding involves sorting all detections by the objectness score and removing the detections below a certain threshold. NMS, as described earlier chapter, helps in tackling the problem of multiple detections for the same object. Further, the YOLO model divides the input image into $S \times S$ grid. We use images of size 608×608 during training. YOLOv3-tiny makes use of $S = 19$ and $S = 38$ scales for the grid, thus, effectively making predictions on 32×32 and 19×19 patches of the image. If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object. Each grid cell predicts $B = 3$ bounding boxes and confidence scores for those boxes. Further, each grid cell also predicts $C = 6$ (since we have six classes) conditional class probabilities $\Pr(Class_i | Object)$. These probabilities are conditioned on the grid cells containing an object.

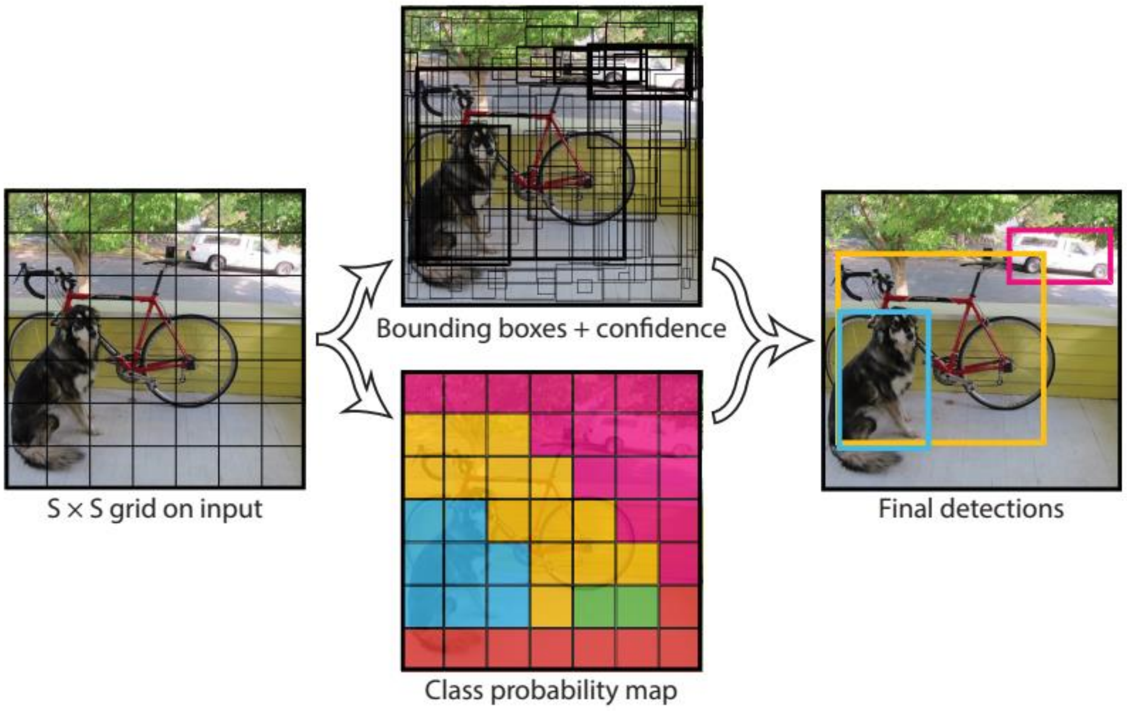


Figure 15 Pictorial depiction of the prediction process of YOLO model. You Only Look Once, Redmon et al., 2016

All these predictions are brought together to form the multi part loss function.

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_{ij}^{obj} [(x_i - \widehat{x}_i)^2 + (y_i - \widehat{y}_i)^2]$$

$$+ \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_{ij}^{obj} \left[\left(\sqrt{w_i} - \sqrt{\widehat{w}_i}\right)^2 + \left(\sqrt{h_i} - \sqrt{\widehat{h}_i}\right)^2 \right]$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_{ij}^{obj} BCE(C_i, \hat{C}_i) + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_{ij}^{noobj} BCE(C_i, \hat{C}_i)$$

$$+ \sum_{i=0}^{S^2} \mathbb{I}_{i}^{obj} CrossEntropy(over\ C\ classes)$$

The first two terms capture the error between the predicted bounding box coordinates and the ground truth boxes of matching classes added with appropriate weighting factor. The third term penalizes the objectness score prediction for bounding boxes responsible for predicting objects (the scores for these should ideally be one), the fourth term one for bounding boxes having no objects, (the scores should ideally be zero), and the last one penalizes the class prediction for the bounding box which predicts the objects. We have used cross entropy loss in the last three terms, the first two being binary cross entropy and the last being multi class cross entropy.

We have borrowed almost all the hyperparameters for the YOLOv3-tiny model from the original YOLOv3 parameters. We have used confidence threshold 0.1 and NMS IOU threshold 0.5. One of the important parameters of the model is the size of anchor box. It should be carefully chosen based on the size of all instances in the dataset. So, we took all the instances and applied K-Means algorithm to cluster the sizes around six cluster points. The results are shown in the figure below. We can observe the cluster points colored in black which were used as anchor box dimensions. Three of the larger boxes was used with $S = 19$ scale while the remaining three smaller boxes were used with $S = 38$ scale.

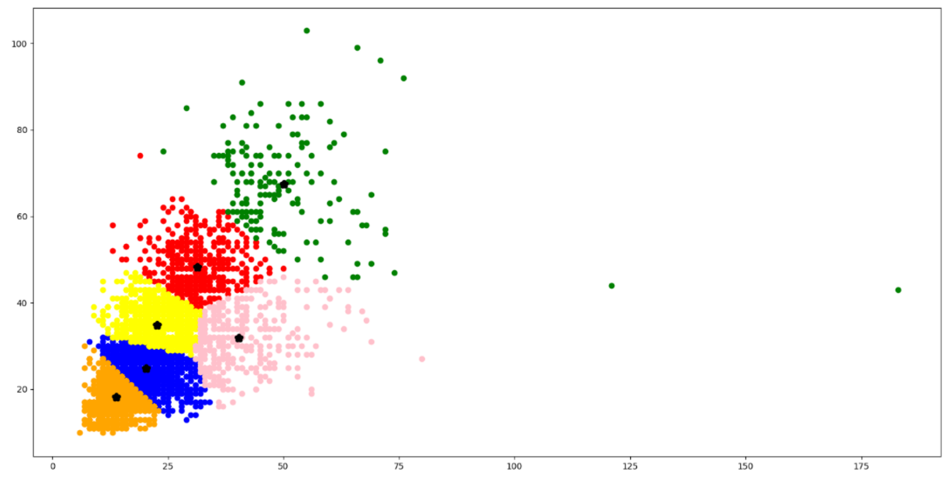


Figure 16 Results of K-Means clustering to find anchor box sizes. The black dots denote the final cluster means used in the model.

3.2.1 Results Obtained from Base Model

After training the base model (YOLOv3-tiny) on the source trainset consisting of images captured from a particular height, we tested the model on source domain testset and target domain testsets (consisting of images taken from 90%, 80%, 70% and 60% of the original height). The graph below illustrates the results obtained.

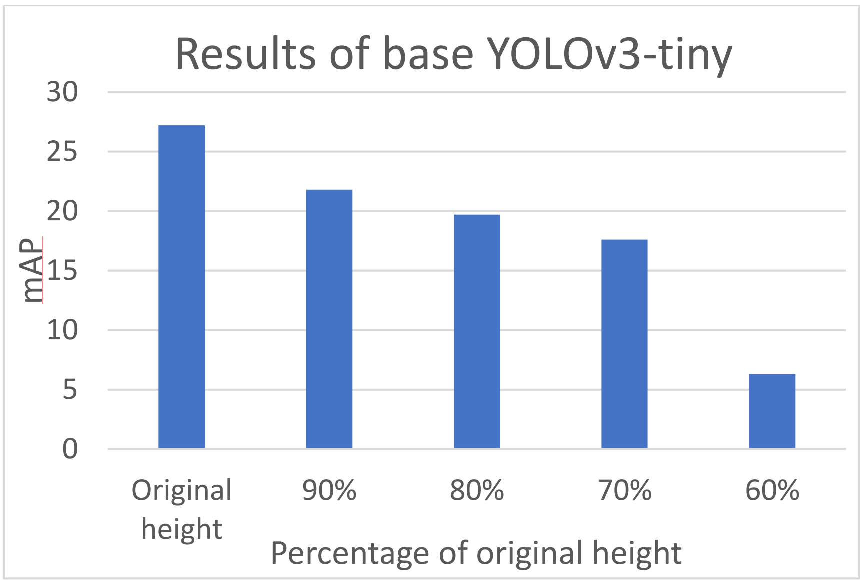


Figure 17 Results obtained from base model. This would be used as benchmark for comparison.

We obtained a clear dip in mAP on decreasing height. This suggests that model is basing its decisions on height dependent features. Since the model is trained on images captured from a particular height, it is not able to generalize well to images captured from other heights which forms the basis of our problem. Thus, we demonstrated the need of domain adaptation to extract height invariant features.

3.3 Performing Domain Adaptation

The first step involved creating a domain classifier which would act as an adversary to the feature generator of the base model. This also involved finding an ideal layer in the base model where the domain classifier would be attached through a gradient reversal layer. This would also theoretically decompose our base model into a feature part and classifier part. We experimented with attaching the domain classifier after the third, fourth and fifth convolutional block (each block consists of convolution with kernels, batch normalization, non-linear activation) of the base model. Finally, we decided to attach the domain classifier after the fifth convolution block since that seemed to perform the best during training time.

Since features output by the base model after the fifth convolution block is large in size (large depth space dimension), directly feeding them to a fully connected layer for classification was infeasible owing to very large number of parameters, which caused problems during training. So, we employed one-one convolution to reduce the depth space dimension of features and strided convolution to reduce the spatial dimension of the features. As an advantage, we also obtained the large receptive field associated with strided convolution, which would help in efficient capture of global features for discerning the domain of the image.

The first layer of domain classifier consisted of a one-one convolution layer to halve the features dimension in depth space. The following two layers consisted of strided convolutions, each layer reducing the spatial dimension of features by factor of two. In this way, we manged to achieve a reduction of $8\times$ in feature size, while ensuring we extract enough discriminative information to be able to correctly classify the domain of the image. Each application of convolution was followed by a batchnormalization operation to ensure efficient training. After achieving sufficient reduction, we flattened the features obtained and then attached a fully connected layer with 512 outputs. These outputs were then used in a fully connected manner to predict a single logit denoting the domain of the image. Sigmoid function was applied to the logit to enable a probabilistic interpretation of the output.

A simple binary cross entropy function was used for calculating the loss of the domain classifier. When added to the original YOLO loss, the overall loss of the entire model became:

$$Total\ loss = YOLO\ loss + \alpha * Domain\ loss$$

where, α is a hyperparameter denoting the relative importance of the base model and domain classifier.

Here, we describe some hyperparameters that were critical to the training of the entire network:

- **Learning rate:** We tried different learning rate of the base model and the domain classifier. It did not seem to affect trainability by much. So, when training both the base model and the network from random initializations, we set the initial learning rate of both the base model and the domain classifier as 0.001. We followed the same learning rate varying strategy as given in the YOLO paper. But, when using pretrained initializations, the learning rate of layers with pretrained initializations was 10 times less than that of those layers with random initializations.
- **$\boldsymbol{\alpha}$:** We found that the ratio of $Domain\ loss$ and $YOLO\ loss$, α, was critical to the trainability of the model. The model could be trained for values of α lying between 0.05 and 0.5. Higher value of α resulted in the model not learning, while lower values of α nullified the effect of the domain classifier.

- ***λ***: The parameter λ controls the relative importance between the domain classifier trying to correctly classify domain of the image and the feature generator trying to confuse the domain classifier. We found that, for values of λ lying between 3 to 15, the model performed well which was evident by the final loss of the domain discriminator after training. Further, the following strategy was used.

  $$\lambda_p = \sigma\left(\frac{2}{1 + \exp(-\gamma . p)} - 1 \right)$$

  where γ was set to 10 in all experiments, p denotes the progress of training and σ is the final value of the parameter. We also tried using a fixed value for λ, but its performance was worse than that obtained by following the above variation strategy .
- **Weight Initialization:** We experimented with various methods of weight initializations. We tried using the weights from training the base model for initializing the weights of feature extractor. This was to ensure that the network does not starts with an unusual weight initialization that leads to suboptimal selection of features. We obtained slightly better performance using this approach. Further, we tried fixing the weights first couple of layers of the feature generator as these layers are responsible for extracting very primitive features (like edges, corners, etc.) which would still be required after domain adaptation. This approach also led to small increase in the performance of the network.

3.3.1 Results Obtained After Adversarial Domain Adaptation

We trained around 30 different models with varying training conditions (see Hyperparameters in the last section) comprising of around 280 hours of GPU runtime. We noticed that towards the end of training, for every epoch, the mAP fluctuated with variations of 1-2% even though the loss functions showed very little variation indicating that it had settled. So, we found it difficult to report a fixed performance metric in terms of mAP. The best performance on the testset images captured from 70% of the original height was around 20%, which is 2.4% higher than the base model performance. So, we can say with certainty that our model with domain adaptation has achieved performance on par with the base model. But we cannot say with confidence that we have improved upon the base model using domain adaptation owing to the fluctuations in mAP. So, the problem remains to be solved satisfactorily and we present below some ideas that could be explored to tackle it. Further details have been mentioned in the future work section.

- Simple gradient reversal-based methods may not be sufficient to make the features height invariant due to the complexity of the YOLO model. We are working on an elaborate attention-based mechanism to achieve this task.
- Even though we have found the optimal ranges of the hyperparameters, we may not have found an optimal set all the hyperparameters, which can be used jointly to achieve superior performance.
- We used a very primitive technique based on similar triangles to simulate height variations. This method is prone to pixilation which can affect performance. A better technique to simulate height variation can alleviate this issue.

Chapter 4

Future Works

4.1 Use of the Latest, State-of-the-Art Domain Adaptation Techniques for Object Detection

To solve the problem of scarcity of good labelled data, recently many new state-of-the art domain adaptation methods have been proposed for various deep learning tasks. A lot of work is being done in both unsupervised and semi-supervised domain adaptation fields to apply these techniques to complex deep learning tasks and not only on simple classification tasks. Many new approaches for unsupervised domain adaptation proposed in the recent years outperform the conventional techniques. Deep discriminator networks involving multiple adversaries have been proposed to capture the multimode structure of data for fine-grained alignment of the source and target domains. Local attention-based models have been introduced to selectively transfer local image features that are more transferable across domains. Similarly, global attention helps in selectively aligning transferable images across the two domains. *Transferable Attention for Domain Adaptation* by Wang et al., 2018, makes use of both local and global attention for domain adaptation. These new models have shown great success in conventional domain adaptation tasks and may prove useful for aerial object detection also.

4.2 Generative Adversarial Networks (GANs) for Generating Height Based Image Translations

Generative Adversarial Networks by Goodfellow et al., 2014 have recently gained a lot of popularity and are being used for a variety of tasks. Conditional Adversarial Networks proposed for image to image translation problems (Image-to-Image Translation with Conditional Adversarial Networks, Isola et al., 2016) have shown good results for various translation tasks. Newer methods, for example, pix2pixHD (Wang et al., 2017) have shown much improved performance on such tasks. Such networks may be used to generate clever image transformations for height-invariant aerial object detection.

GANs have also shown impressive performance on image super-resolution tasks. SRGANs (Super Resolution GANs) and ESRGANs (Enhanced SRGANs) have been proposed providing generative adversarial methods for image super-resolution. Smart implementations of such networks along with Conditional GANs may help in generating highly realistic simulations of height change in an image.

4.3 Dataset Creation to Promote Studies on Aerial Images Captured at Different Heights

As mentioned earlier in the report, currently no open dataset exists for studying height-based transformations for object detection in aerial images. A large and well labelled dataset plays a crucial role in advancement of deep learning to newer problems in various domains. A large aerial image dataset with camera height information and annotations for object detection and semantic segmentation can be prepared to open up new avenues in aerial object detection and segmentation tasks. Such a dataset will not only help in academic research but would also play a pivotal role in many practical applications. For example, dropping relief supplies during calamities using drones to help the survivors can really benefit through this dataset. Automated food delivery using drones can also progress a lot using height invariant object detection, which will require a good dataset. Hence, an aerial image dataset with camera height information can be used for a lot of purposes and will help in progressing deep learning research.

References

Ganin, Yaroslav; Lempitsky, Victor. *Unsupervised Domain Adaptation by Backpropagation*. arXiv:1409.7495v2 [stat.ML] 27 Feb 2015

Redmon, Joseph; Divvala, Santosh; Girshick, Ross; Farhadi, Ali. *You Only Look Once: Unified, Real-Time Object Detection*. arXiv:1506.02640v5 [cs.CV] 9 May 2016

Redmon,Joseph; Farhadi,Ali. *YOLO9000:Better,Faster,Stronger*. arXiv:1612.08242v1 [cs.CV] 25 Dec 2016

Redmon,Joseph; Farhadi,Ali. *YOLOv3: An Incremental Improvement*. arXiv:1804.02767v1 [cs.CV] 8 Apr 2018

A. Robicquet, A. Sadeghian, A. Alahi, S. Savarese. *Learning Social Etiquette: Human Trajectory Prediction In Crowded Scenes* in European Conference on Computer Vision (ECCV), 2016

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio. *Generative Adversarial Networks*. arXiv:1406.2661v1 [stat.ML] 10 Jun 2014

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, Alexei A. Efros*. Image-to-Image Translation with Conditional Adversarial Networks*. arXiv:1611.07004v3 [cs.CV] 26 Nov 2018