San Jose State University
SJSU ScholarWorks

Master's Projects | Master's Theses and Graduate Research

Spring 2018

FINE-GRAINED OBJECT DETECTION

Rahul Dalal
San Jose State University

Follow this and additional works at: https://scholarworks.sjsu.edu/etd_projects

Part of the Computer Sciences Commons

Recommended Citation
Dalal, Rahul, "FINE-GRAINED OBJECT DETECTION" (2018). *Master's Projects*. 609.
DOI: https://doi.org/10.31979/etd.f9gs-dd3b
https://scholarworks.sjsu.edu/etd_projects/609

This Master's Project is brought to you for free and open access by the Master's Theses and Graduate Research at SJSU ScholarWorks. It has been accepted for inclusion in Master's Projects by an authorized administrator of SJSU ScholarWorks. For more information, please contact scholarworks@sjsu.edu.

FINE-GRAINED OBJECT DETECTION

A Writing Project

Presented to

The Faculty of Department of Computer Science

San José State University

In Partial Fulfillment

Of the Requirements for the Degree

Master of Computer Science

By

Rahul Dalal

Spring 2018

©2018

Rahul Dalal

ALL RIGHTS RESERVED

SAN JOSÉ STATE UNIVERSITY

The Undersigned Thesis Committee Approves the Thesis Titled

FINE-GRAINED OBJECT DETECTION

By

Rahul Dalal

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

Dr. Teng Moh, Department of Computer Science 5/9/2018

Dr. Suneuy Kim, Department of Computer Science 5/9/2018

Prof. James Casaletto, Department of Computer Science 5/9/2018

Abstract

Object detection plays a vital role in many real-world computer vision applications such as self-driving cars, human-less stores and general purpose robotic systems. Convolutional Neural Network(CNN) based Deep Learning has evolved to become the backbone of most computer vision algorithms, including object detection. Most of the research has focused on detecting objects that differ significantly e.g. a car, a person, and a bird. Achieving fine-grained object detection to detect different types within one class of objects from general object detection can be the next step. Fine-grained object detection is crucial to tasks like automated retail checkout. This research has developed deep learning models to detect 200 types of birds of similar size and shape. The models were trained and tested on CUB-200-2011 dataset. To the best of our knowledge, by attaining a mean Average Precision (mAP) of 71.5% we achieved an improvement of 5 percentage points over the previous best mAP of 66.2%.

Keywords: Object Detection, Computer Vision, Deep Learning, Convolutional Neural Networks, Region based Convolutional Neural Network, Inception, You Only Look Once, Single Shot Detection.

Acknowledgement

I am grateful to my project advisor Dr. Teng Moh for his guidance and insights that helped steer the project in the right direction. I would also like to thank my committee members Professor James Casaletto and Dr. Suneuy Kim for dedicating their precious time to my project.

I would like to thank the San Jose State University and Computer Science department for giving me this opportunity and the necessary resources to pursue this project.

Lastly, I would like to thank the deep learning research community that has shown great openness in sharing their experiences and learnings with fellow researchers and deep learning enthusiasts.

Table of Contents

Abstract **4**

Acknowledgement **5**

1 Introduction **8**

2 Related Works **12**

2.1 Machine Learning Approaches **12**

2.1.1 Viola-Jones Object Detection [14, 15] 12

2.2 Deep Learning Approaches **13**

2.2.1 Convolution Neural Network 14

2.2.2 Object Detection 24

3 Dataset **36**

4 Evaluation Metric **37**

5 Proposed Approach **40**

5.1 Baseline approach **40**

5.2 Project approach **42**

5.2.1 Implementation pipeline 43

5.2.2 Models trained 50

6 Experiments and Results **55**

7 Conclusion **60**

References **61**

Table of Figures..... 65

Table of Tables..... 67

1 Introduction

Advancements in computer vision have made self-driving cars and automated retail a possibility. One of the key computer vision tasks in all these applications is object detection. Object detection has evolved from image classification. Image classification is a supervised learning problem that takes images and corresponding class labels as input data for training. Once the model is trained it can classify a test image with a class label. For e.g. once trained, an image classification model can label a given image as a car, a pedestrian, a bicycle or background (none). In this case, we have four classes i.e. car, pedestrian, bicycle and background. Object localization builds on image classification. Given an image, it classifies it into a class and locates a bounding box around the object in the image. Object localization requires the image to contain only one object of any of the classes under consideration. Object detection is the supervised learning problem of classifying multiple objects and localizing them in an image. Thus, object detection consists of two main tasks:

- Identify the class label for each object in the image
- Create bounding boxes for all the objects

Machine learning techniques like Support Vector Machine (SVM) [1] classifiers have solved image classification problems with high accuracy. LeCun et al. [2] in their monumental research in 1998, developed a deep Convolutional Neural Network(CNN) to identify human hand written images with an accuracy of more than 99%. Since then CNN has been the de facto architecture for image classification. It was a natural progression to develop CNN [3] based models for object detection. However, object detection encounters significant additional challenges than image classification and object localization such as:

- Variable number of objects: As image classification and object localization contains only one object, the input label and the output is of the same size for all

the images. As the number of objects in object detection is variable, the size of input labels and output can change from image to image depending on the number of objects in the image. This represents a crucial challenge as almost all previous machine learning and deep learning algorithms took an input of fixed size and generated an output of fixed sized.

- Combining classification and localization: Object detection performs classification and localization simultaneously for multiple objects. Object detection deep learning models must perform both these tasks accurately while performing them simultaneously.

Given the success of CNN based deep learning models in image classification and the non-linear nature of the task, all latest successful object detection models employ CNN based deep learning. As of now, two approaches are at the forefront of object detection research:

- Region Based Convolutional Neural Network (R-CNN) [4, 5]: Faster R-CNN [6] is the latest and most accurate model in this series. R-CNN based models use a sliding window approach and CNN based classification for object detection.
- Regression based models – While these models also use a CNN based architecture, they deviate from the traditional viewpoint of seeing object detection as a classification problem and characterize it a regression problem. YOLO [7] was the first algorithm to propose this approach. Since then YOLO:9000 [8] and Single Shot MultiBox Detection (SSD) [9] have emerged as the most accurate models in this category. While these models are generally less accurate then their R-CNN counterparts, they are significantly faster.

In fine-grained object detection, we split a class of objects into further classes and apply object detection to them. General purpose object detection tasks use object detection datasets like COCO [10] and PASCAL VOC [11] where the object classes are significantly different from each other e.g. a person, a car, a bird etc. Characteristics such as shape, size, color of a person, car and a bike are very different from each other. Fine grained object detection splits a class in general object detection into different categories and uses them as classes for object detection. For our research, we use CUB-200-2011 [12] dataset, which consists of 200 classes of birds. It can be viewed as the birds' class in general object detection being split into 200 classes. Fine-grained object detection encounters additional challenges to general purpose object detection.

Firstly, the model needs to learn fine-grained features. In general purpose object detection, the model can learn to detect a bird (as opposed to a person or a car) by learning its features like a beak, a tail, feathers, its body shape etc. In our case, all the 200 classes will have these features. Hence to detect these objects the model must learn more fine-grained features. For e.g. two of the classes in the dataset are black footed albatross and laysan albatross [12]. The model should first learn to classify an albatross from other type of birds and then learn that the black footed albatross is dark gray/ black in color while the laysan albatross has a significant white colored portion. Secondly, as the number of classes is 200 and they are similar, increasing the probability of false positives significantly.

Training a model from start can require huge resources and even then, takes a lot of training time. To reduce the training time, we used transfer learning. We retrained COCO object detection models on the CUB dataset using transfer learning. We used data augmentation to distinguish between similar classes. The detailed approach is explained in Section 3. We achieved a mean

average precision(mAP) of 71.5 % on the test dataset. To the best of our knowledge, it represents an improvement of 5 % points from the previous best of 66.24% [13].

The paper is structured with section 2 surveying related works, sections 3 and 4 covering the dataset and evaluation metric respectively, section 5 explaining the proposed approach, section 6 describing the experiments and how their results compare with the baseline. Finally, we outline our conclusions in section 7.

2 Related Works

Fine-grained object detection builds upon object detection. In this section, we survey the landscape of object detection.

2.1 Machine Learning Approaches

2.1.1 Viola-Jones Object Detection [14, 15]

Viola-Jones [14] in 2001 was the first machine learning algorithm devised to detect objects accurately in real-time. The framework carried out its experiments by detecting faces in images. Viola-Jones object detection consists of four main stages:

- Haar feature selection
- Creating integral image
- Adaboost training
- Cascading classifiers

The major highlights of this approach were its use of Haar features to detect facial features and Adaboost training to reduce training time. Haar features are rectangular kernels that can be used to detect areas in the image that are alike. The Haar feature shown below can be used to detect the eye region.

Figure 1. Haar feature for eye region [14]

The Haar feature shown in Fig. 1, slides across the entire image, shifting by 1 pixel in each operation and gives an output value. The output value will be higher for regions that resemble the filter and lower for other regions. This operation is comparable to convolution that is covered in

detail in Section 2.2.1.1. The eye region is darker than the region below it and hence it can be detected by the Haar feature in Fig. 1. Viola-Jones divided the image into a 24x24 sub windows and ran multiple Haar features over it. Considering all the different positions, scales and types of Haar features, a 24x24 window yielded 160,000 features, which was extremely high to perform real time object detection. Viola-Jones used the Adaboost algorithm to consider only relevant features. Adaboost trains a strong classifier as a weighted combination of these features. Only those features that perform better than random guessing are considered. Thus, Adaboost combines several weak classifiers linearly to form a strong classifier. The Viola-Jones was the first object detection model that gave a high detection rate of 76.1% (with a threshold of 10 false positives) in real time and at a speed of 67 ms [14]. One of the major limitations in adapting the model for general purpose object detection is to identify Haar features to be used. The complexity of identification of Haar features increases as the number of classes to be detected increases. Another challenge is the need for domain knowledge to select Haar features. Also, Viola-Jones considered detection accuracy as the metric. With the evolution of object detection, mean Average Precision (mAP) became the metric for measuring accuracy of a model. The mAP metric is explained in detail in section 4.

2.2 Deep Learning Approaches

CNNs are an integral part of all deep learning approaches for object detection. CNNs gained prominence when LeCun et al. [2] used a deep CNN, LeNet-5, to classify hand written digits with an accuracy of more than 99%. In this section, we look at the important components and concepts of a CNN [3, 16].

2.2.1 Convolution Neural Network

2.2.1.1 Convolution [3,16]

The convolution operation takes an image and a filter (also referred to as kernel) as an input. Consider an image and a filter as shown in Fig. 2:

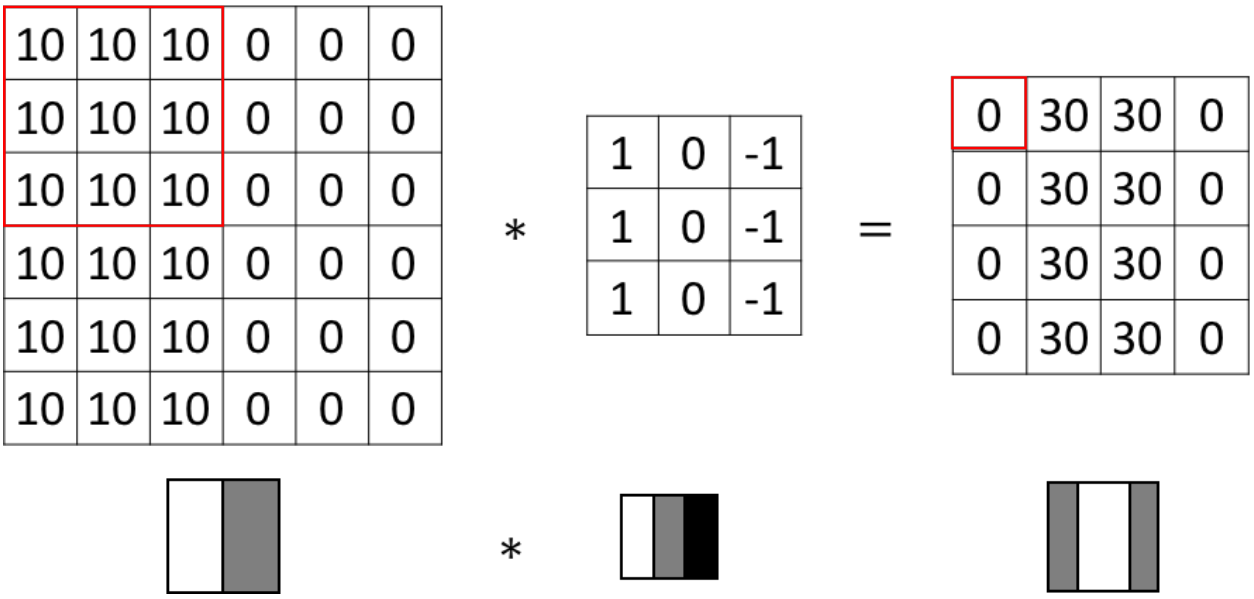

Figure 2. Convolution operation in CNN

A filter of 3x3 size performs convolution on the input image. The red boxes highlighted in the figure denote input and output of one convolution operation. The filter then strides over the image one pixel towards the right at a time until it reaches the end while staying within the boundary of the image and performs a convolution every time. When it reaches the end of the row it takes a stride down. The convolution operation reduces the size of the input. In the example in Fig. 2, the output size reduces to 4x4 from 6x6 because of convolution. To maintain the input size, we can pad the input image with zeros. When we pad to maintain the size of the input image during convolution it is known as same padding. When no padding is employed it is called valid padding. The convolution filter in Fig. 2. detects a vertical edge. In the above example, we have considered a gray scale image as the input. For a RGB image the input image will have 3 channels (depth).

For e.g. The RGB equivalent of the image considered in Fig. 2 would be of size 6x6x3. It is an accepted norm that the filter has the same number of channels as the input image. As a result, the filter would also be of size 3x3x3 size having 27 weights. Since the input and the filter also have a third dimension they are also referred as volumes. A convolution layer can contain many such filters and it stacks the output of each filter along the channels of the output. So, the number of channels in the output will be equal to the number of filters used in a convolution layer. Thus, the output of convolution also has a third dimension (channels) and hence is referred to as volume activation.

2.2.1.2 Pooling [3, 16]

The output of convolution is fed to a pooling layer. Max pooling is the most common pooling used.

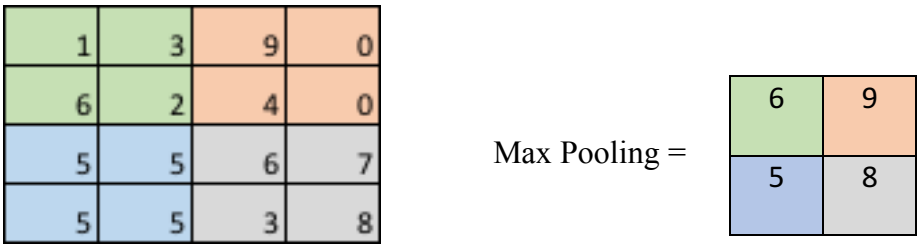

Figure 3. Pooling operation in CNN

Consider a 2x2 max pool applied to the output of a convolution layer as shown in Fig. 3. Each pooling operation selects the maximum value from a 2x2 area from the input and produces the output. The maximum value in the 2x2 area starting at pixel (0,0) is 6. Hence pixel (0,0) of the output will be six. Generally, the pooling operation takes a stride equal to its size. We follow the same norm in this example. So, the pooling operation moves to the 2x2 region starting at pixel (0,2) and gives a value of 9 for output pixel (0,1).

Pooling helps to reduce the height and width of the input layer while still maintaining relevant features learnt by the network. Suppose we are learning to identify faces and perform convolution using a vertical edge filter. The convolution operation should detect a nose in all the input images

and thereby give a high value in the output for the pixels in the middle region (where the nose is situated). The exact pixels where the convolution output gives high values will differ slightly for different images. Our network need not learn the exact pixel locations where the vertical edge is detected but rather learn that there should be a vertical edge in the center region. Max pooling helps us achieve that. This behavior of pooling also has a regularization effect on the model. Further pooling by its nature reduces the height and width of the input, thus playing an important role in developing a convergent CNN architecture.

The advantages achieved from CNN architecture are:

1. In a CNN, the weights of a specific filter remain same across the image. This sharing of weights reduces the number of weights dramatically as compared to a fully connected layer.
2. Sharing and localization of weights allows the same feature to be detected in different areas of the image. If a filter learns to detect a vertical edge, it can be used to detect vertical edges in all areas of the image, which is accomplished by the convolution operation.

2.2.1.3 VGG-16 [17]

A CNN architecture uses three types of layers:

- Convolutional layer
- Pooling layer
- Fully connected layer (traditional neural net layer)

A CNN stacks multiple layers of these types and operates on an input volume of activations to generate an output volume of activations. The final layer can be a softmax [3] or sigmoid layer depending on the task on hand. While a variety of CNN architectures have emerged, some of the classical CNN architectures LeNet-5[2], AlexNet [18] and VGG-16 [17] have shaped general themes on CNN architecture. We will consider VGG-16 for understanding these themes:

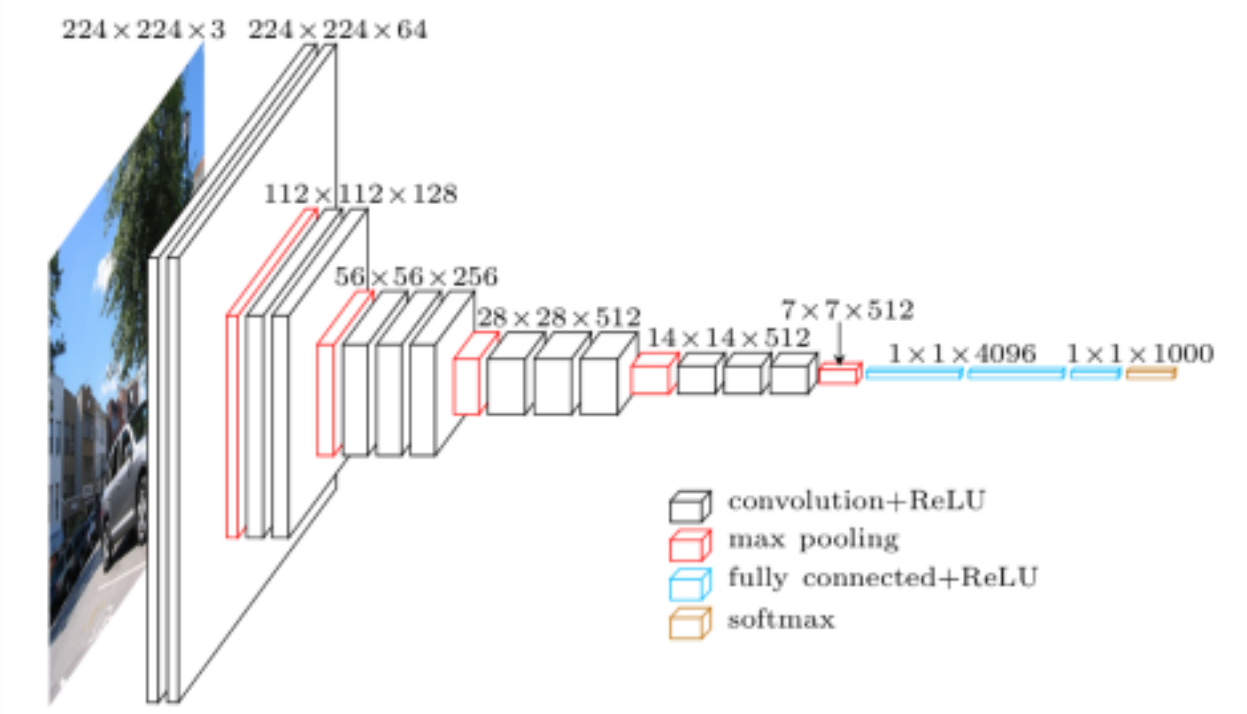

Figure 4. VGG-16 architecture [17]

2.2.1.3.1 Description of VGG-16 [17]

The input image is a RGB image of size 224x224x3. Before VGG-16, many architectures used filters and max-pool units of different sizes across different layers. However, VGG-16 used uniform sized filters and max-pool units across all layers and achieved excellent results. Subsequently, most of the architectures followed uniform filter and max-pool sizes across layers unless otherwise required. In VGG-16 every convolution filter is of size 3x3. Each 3x3 filter will have the same number of channels as the input layer e.g. first convolution layer operating on input image will have 3x3x3 filters. By convention we only refer to the first two dimensions of the filter as the third dimension can be deduced from the input. Henceforth we will refer to a filter with two dimensions. Every max-pool unit is of size 2x2. Max-pooling units do not look at the activations across channels (depth). Hence, they preserve the number of channels (depth) of the input.

- Filter details:

- Size =3x3
- Stride=1
- Padding = same

- Max-pool details:
  - Size=2x2
  - Stride=2

Convolution (Layer 1) and Max-Pool (Layer-2):

We explain the principle behind the initial convolutional and max-pool layers. The remaining layers can be interpreted on similar lines from Fig. 4. CNN layer 1 contains 64 filters of 3x3 size. As all of them use same padding, the width and height of the output is still 224x224. As there are 64 filters, the output of the convolutional layer is 224x224x64. CNN layer 2 doubles the number of channels of the output with same padding, giving output activation volume of 224x224x128.This is then operated by a max-pool layer. Max pooling with a 2x2 unit produces an output of 112x112x128.

2.2.1.3.2 Themes of CNN architecture

- One or more convolutional layers are followed by a max-pool layer. This pattern is repeated and at the end there are one or more fully connected layers followed by a softmax layer or a sigmoid layer or a linear layer depending on the task at hand.
- Convolution operation with same padding maintains the width and height from the input. The number of filters in convolutional layers increases from left to right, thereby increasing the depth of output volume activations.
- Max-pooling reduces the width and height from the input activation while maintaining its depth.

- Each convolution filter detects a specific feature across the entire image.
- Max-pooling aggregates this information and reduces the number of neurons making the network convergent. Generally, as we move from left to right in a neural network we want the number of neurons to gradually decrease.
- This architecture combines elementary features into more sophisticated features as we move from left to right.
- At the end of the network, the output of convolutional and max-pool layers is flattened by one or more fully connected layers and fed to a softmax layer or a sigmoid layer or a linear layer depending on the task at hand.

Subsequently VGG-19 architecture was developed which consisted of 19 layers as the name suggests.

2.2.1.4 Inception Network [19]

One of the key decisions in designing a CNN is to choose the size of the filter in each layer. 3x3 and 5x5 are common filter sizes used. Inception network by Szegedy et al. in 2014 proposed that instead of selecting the filter size, multiple filter sizes could be computed and the network be allowed to select the most suitable size during training.

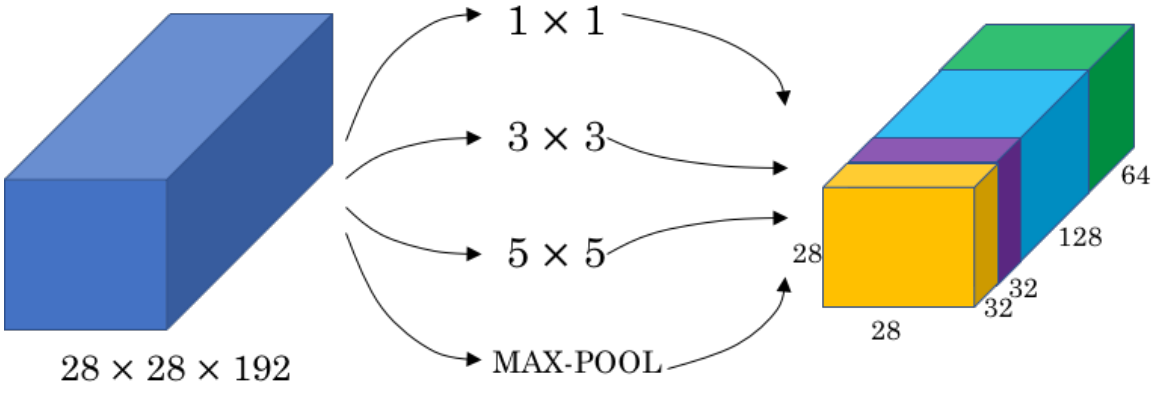

Figure 5. Inception layer without channel reduction [16, 19]

As shown in Fig. 5, the input convolutional layer contains 1x1, 3x3, 5x5 convolution filters and a max-pool filter. All the convolutions and max pool use same padding, and the results are concatenated channel wise to get the output. One of the issues faced by such an architecture was a huge increase in computations. Lin et al. in their network in network architecture [20] in 2013 had used a 1x1 convolution to reduce the number of channels. The modified inception network shown in Fig. 6 uses 1x1 filters to reduce channels in the intermediate layer and achieves significantly lesser computations.

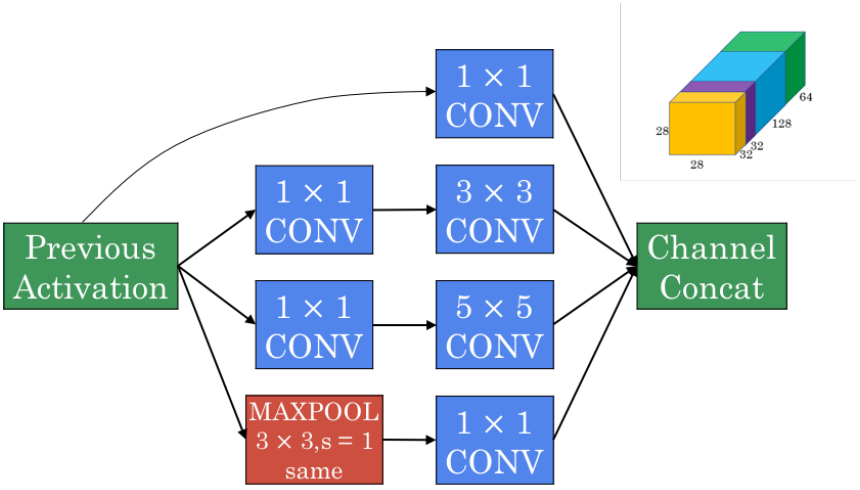

Figure 6. Inception layer with channel reduction [16, 19]

The Inception network comprises of many such Inception layers, followed by fully connected layers and a softmax layer at the end for classification.

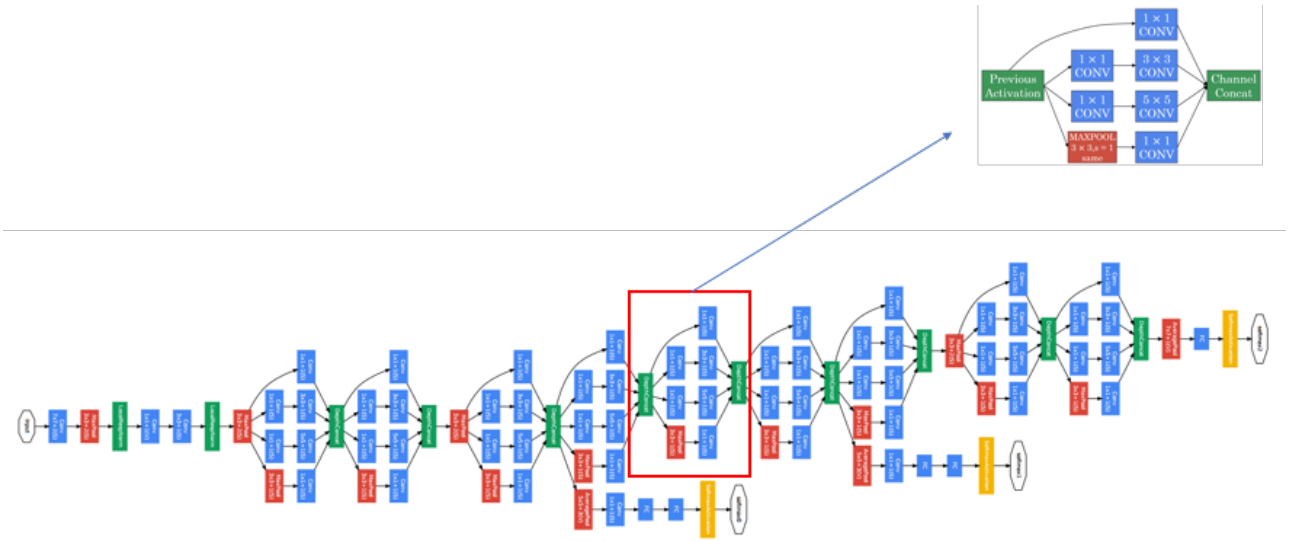

Figure 7. Inception CNN architecture [19]

The Inception network uses softmax layers at intermediate stages to ensure that intermediate layers are also performing well at classifying the object.

2.2.1.5 Residual Network (ResNet) [21]

Ideally, a deeper neural network should perform more accurately, assuming training time and training resources are not constraints and there is sufficient data. However, it was observed that training deeper neural networks was difficult due to vanishing and exploding gradients. Gradient clipping can mitigate exploding gradients but vanishing gradients remained a major hindrance in training deep neural networks. He et al., in 2015 [21] developed the ResNet architecture to train very deep neural networks. The key element of this architecture was a residual block.

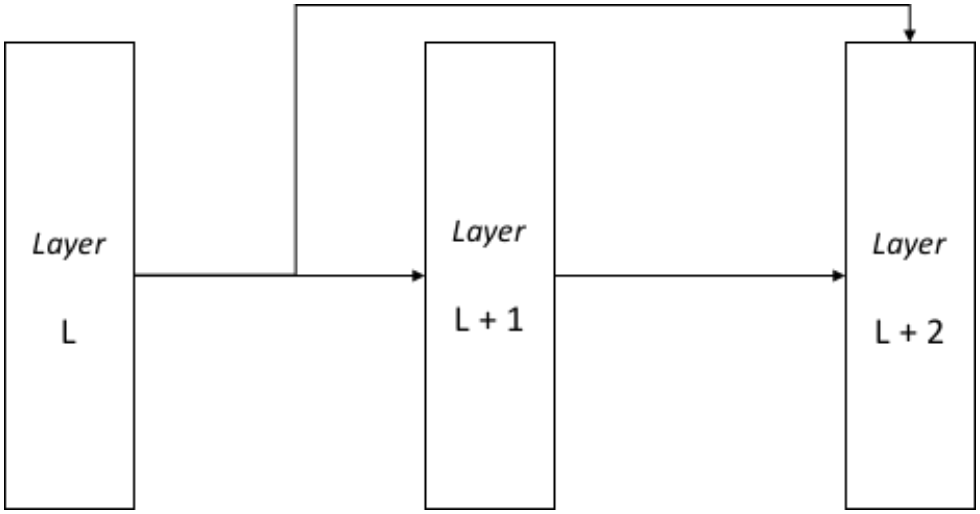

Figure 8. ResNet block

As shown in Fig. 8, in a residual block the $L+2^{th}$ layer receives the activation of the L^{th} layer in the network through a skip connection. The activation of L^{th} layer is added to the logits calculated from activation of $L+1^{th}$ layer. Activation function of the $L+2^{th}$ layer is then applied to this sum to obtain activation of the $L+2^{th}$ layer. If the weights in the $L+1^{th}$ layer vanish (approach towards zero), $L+2^{th}$ layer will still receive the L^{th} layer's activation and continue to train. The argument presented is for weights but can be extended to gradients. Thus, ResNet architecture allowed training of very deep neural networks up to 101 layers.

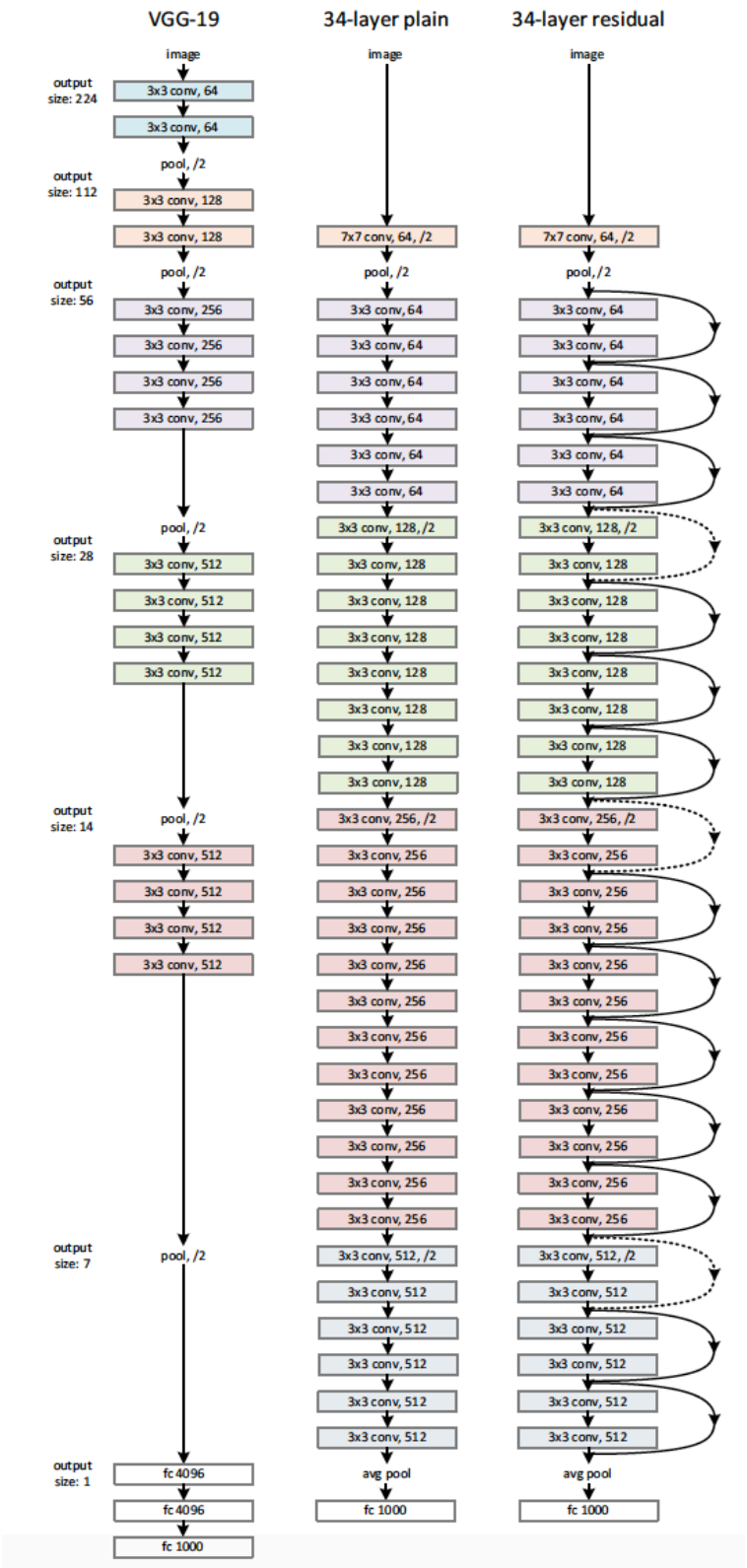

Figure 9. Comparison of 34 layered normal CNN vs 34 layered ResNet architecture [21]

2.2.2 Object Detection

2.2.2.1 Sliding Window Detection

This model first trains a CNN to classify the objects to be detected.

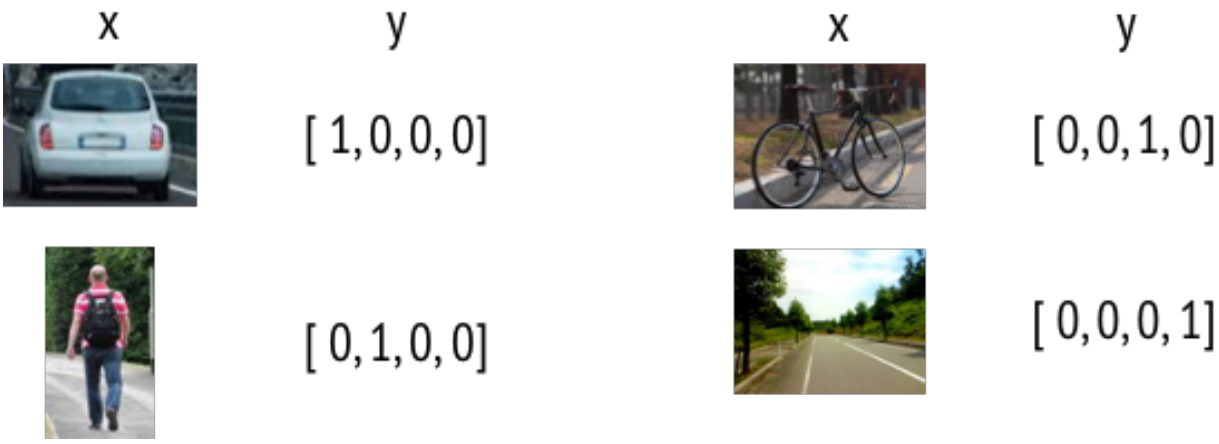

Figure 10. Data labelling for CNN classification

As shown in Fig. 10 a CNN can be trained to classify images of a car, a pedestrian, a bike or background. While training the CNN, the images are cropped to fit the object exactly. This ensures accurate localization during detection.

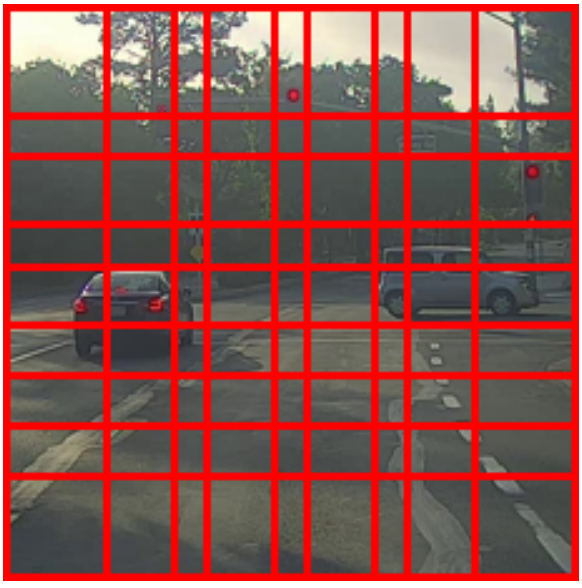

Figure 11. Sliding window object detection using CNN

As shown in Fig. 11, windows of different sizes, aspect ratios and strides are slid across the image and fed as input to the CNN individually. The CNN classifies the object in that window. The window's bounding box is fine-tuned to obtain the bounding box for the object classified. By having a low stride and trying windows of various sizes, reasonable accuracy can be achieved. CNN's are accurate as compared to linear machine learning classifiers but they are computationally expensive. Hence the sliding window framework is extremely expensive to achieve acceptable real time object detection.

2.2.2.2 OverFeat [22]

OverFeat designed by Sermanet et. al. [22] in 2013 was the first Deep Learning model that performed object detection using CNN efficiently. To reduce the computations in sliding window object detection, Overfeat modified the final layers of CNN into a convolutional layer. Consider a simple CNN that takes 14x14x3 image as input and classifies the 4 object classes as shown in Fig. 12.

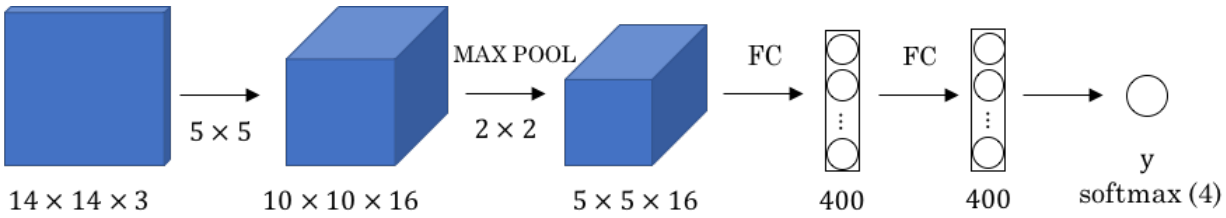

Figure 12. CNN classifying 4 object classes [22, 16]

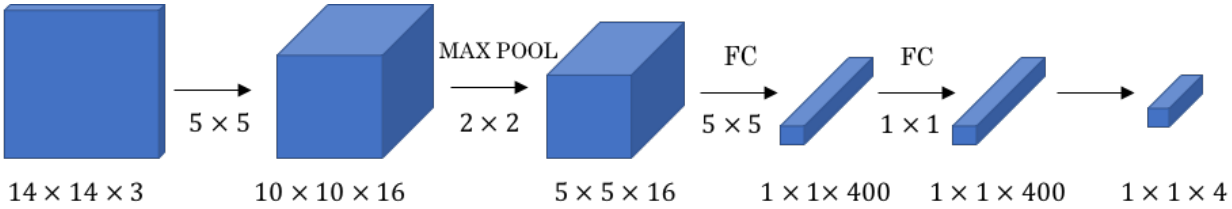

Figure 13. OverFeat transformation of CNN classifying 4 objects [22, 16]

In Fig. 13, the CNN is transformed by OverFeat. The fully connected layers of 400 neurons are transformed into convolution layers of 1x1x400. Both the original and transformed layers have the same number of neuros but OverFeat transformation reduces computations in sliding window object detection significantly.

Suppose an input image of size 16x16x3 is fed to a sliding window object detection model having a CNN accepting an input of size 14x14x3 as shown in Fig. 12. Four 14x14x3 windows starting at locations (0,0), (0,2), (2,0) and (2,2) are cropped from the image. Each of the cropped window is fed as input to the CNN whose output is 4x1 tensor (softmax) indicating class of the object in that window. This requires 4 passes through the CNN.

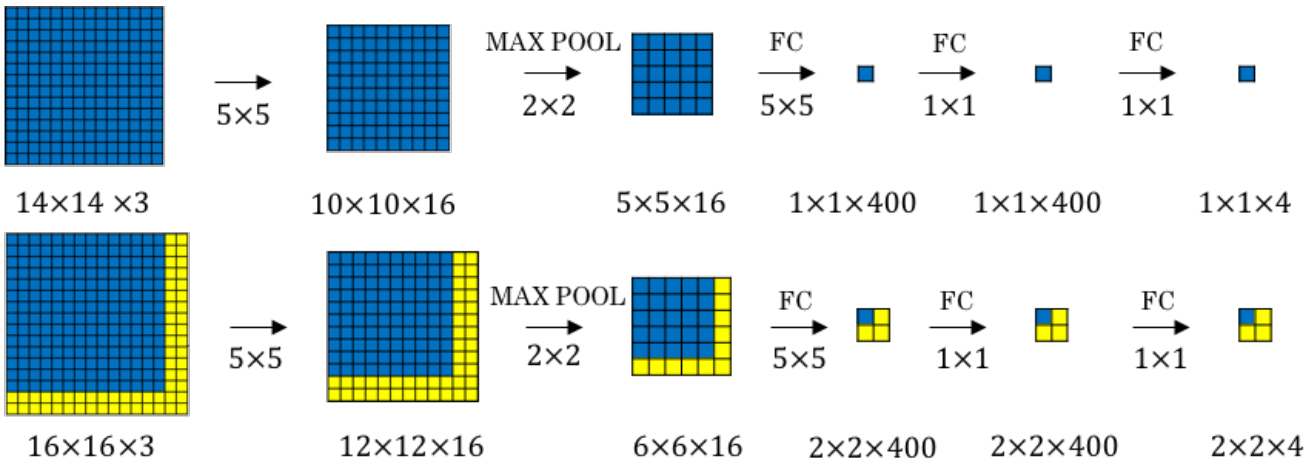

Figure 14. Sliding window object detection in one pass of CNN using OverFeat [22]

The same computations can be performed by OverFeat in one pass as shown in Fig. 14. The OverFeat model feeds the 16x16x3 image as input to the CNN. With the transformation of fully connected layers to convolutional layers the output is 2x2x4 instead of 4x1. The 1x1x4 tensor located at (0,0) highlighted by blue color in Fig. 14 is the output of the 14x14x3 input window starting at (0,0) (also highlighted by blue color in input). Similarly, the outputs for the 14x14x3 windows starting at (0,2), (2,0) and (2,2) can be found in the output at (0,1), (1,0) and (1,1)

respectively. Thus, OverFeat performs classification for all the windows in one pass through the CNN. The improved efficiency made it possible to detect objects by sliding windows object detection utilizing CNNs.

2.2.2.3 R-CNN

One of the issues with sliding window object detection, even with convolutional implementation, was that it processed a lot of windows that did not contain objects. Girshick et al. [4] came up with the Region based Convolutional Neural Network (R-CNN) model. R-CNN's salient feature was identifying regions with a high probability of containing an object before feeding them to a CNN. This approach achieved great object detection results on PASCAL VOC 2012 dataset and put deep learning on the map of object detection.

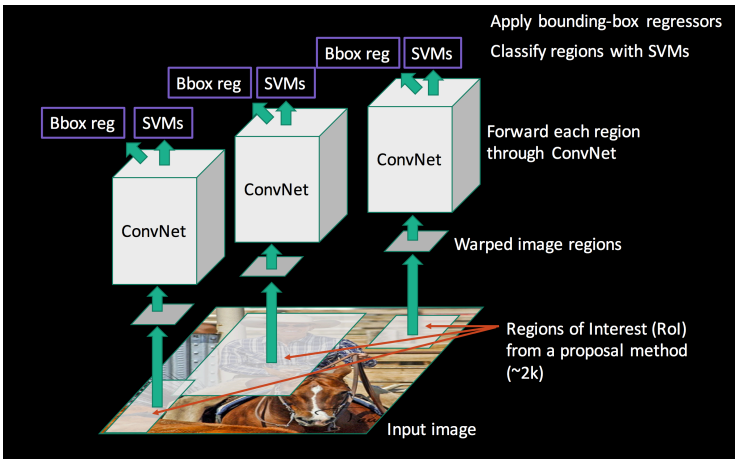

Figure 15. R-CNN model [4]

R-CNN consists of the following main activities:

2.2.2.3.1 Region Proposal

A region proposal algorithm scans the image to identify regions that have a high probability of containing an image. The most popular algorithm (also used in R-CNN) is selective search

algorithm [23]. R-CNN uses selective search to generate approximately 2000 proposals [4] called Regions of Interest (ROI) that have a high probability of containing an object.

2.2.2.3.2 CNN extraction

Each of the RoI is then fed to a CNN to extract features.

2.2.2.3.3 Support Vector Machines (SVM) classifier

A SVM classifier then operates on these CNN features to classify the object in the RoI.

2.2.2.3.4 Bounding box regressors

The model then uses bounding box regressors to refine the RoI's bounding box.

2.2.2.3.5 Limitations of R-CNN

For each image, ~ 2000 RoI proposals were produced which were then fed to a CNN individually. Hence, R-CNN was very slow in detection and its test time per image was 50 seconds. [4].

2.2.2.4 Fast R-CNN

Girshick et al. [24] continued to improve their model and Fast R-CNN was published in 2015. The main bottleneck in R-CNN was, it fed each RoI to a CNN separately. Fast R-CNN uses a principle like OverFeat (Section 2.2.2.2) to pass all RoI's in one pass to the CNN.

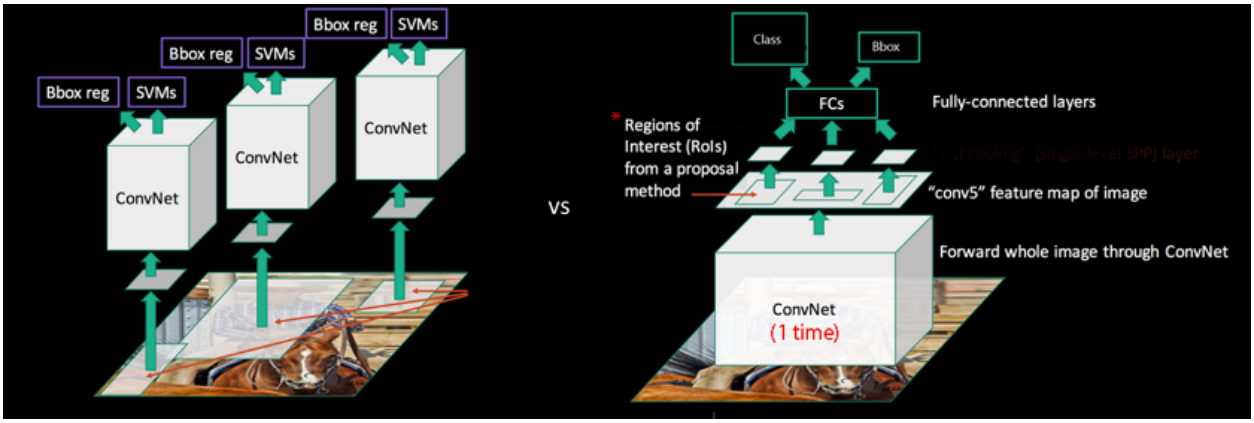

Figure 16. Fast R-CNN model [24]

As shown in Fig. 16, Fast R-CNN feeds the entire image to the CNN once, and the generates the RoI's from convolutional feature maps using selective search algorithm. This improved the test time per image by 25 times to 2 seconds [24].

2.2.2.5 Faster R-CNN

Ren et al. (also co-authored by Girshick) [6] then proposed the Faster R-CNN model in 2017. Both R-CNN and Faster R-CNN used selective search as its region proposal method to generate RoI's. Faster R-CNN eliminated the need for region proposal method. Instead it trained a neural network called Region Proposal Network (RPN) [6] to generate RoI's. These proposals were then fed as an input to the RoI pooling layer. Rest of the architecture remained like Fast R-CNN.

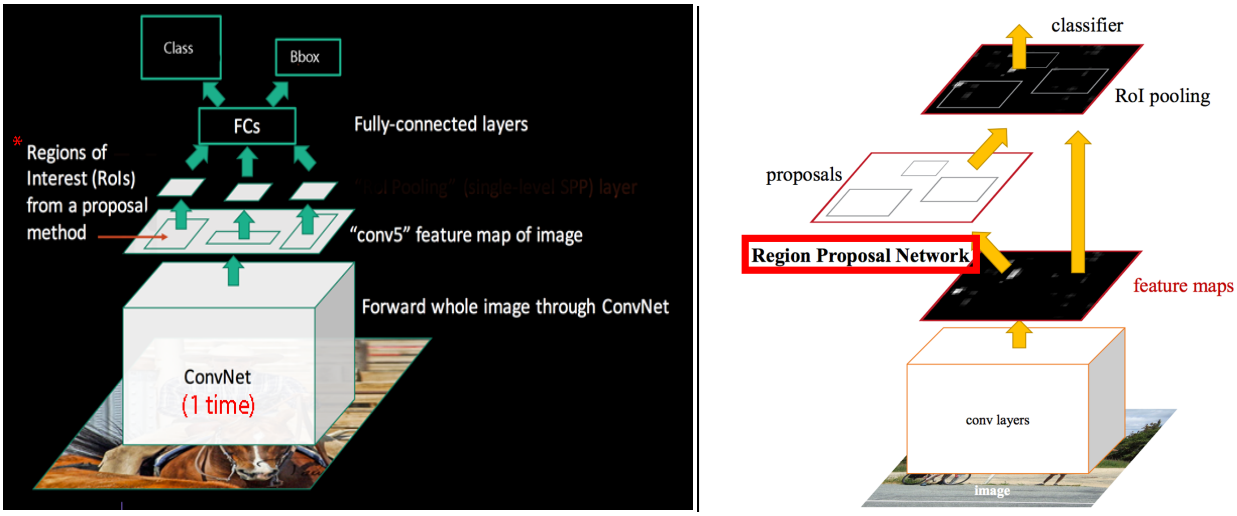

Figure 17. Faster R-CNN model [6]

With this Faster R-CNN achieved a 250 times improvement over R-CNN and 10 times improvement over Fast R-CNN and the test time per image was reduced to 0.2 seconds. [6]

2.2.2.6 You Only Look Once (YOLO)

You only look once (YOLO) [7] brought a paradigm shift in object detection in 2016. It was the first model to frame object detection as a regression problem instead of a classification problem. We consider four object classes a car, a pedestrian, a bicycle and background as before. We first

explain localization using the principles of YOLO and then extrapolate it to object detection. The localization problem is, given an image of any of the four classes our CNN should be able to provide a class label and a bounding box for the object. Since the problem at hand is localization, the image will contain only one object situated near the center.

The training data is labeled as below:

- x – RGB pixel values
- y – 8x1 tensor = (p_c, b_x, b_y, b_w, b_h, c_1, c_2, c_3)
    - p_c – Probability that there is an object. It is 1, if there is an object in the image and 0 for background
    - b_x – 'x' coordinate of the center of the bounding box containing the object
    - b_y – 'y' coordinate of the center of the bounding box containing the object
    - b_w – Width of the bounding box
    - b_h – Height of the bounding box
    - c_i – Probability of class i. It will be labelled 1 if the image contains an object of class i, where i = 1,2,3 in our case (since there are 4 classes including background)
    - b_x, b_y, b_w, b_h are normalized between 0 and 1. The left most corner of the image is considered (0,0) and the right most (1,1)

Figure 18. Object localization data labelling for car object class [16]

An image with a car as in Fig. 18, will have a ‘y’ label of (1, 0.5, 0.6, 0.5, 0.5, 1, 0, 0) assuming c_1 corresponds to car.

Figure 19. Object localization data labelling for background class (None) [16]

On the other hand, a background image will be labeled as (0, None, None, None, None, None, None, None, None). As there is no object in this image we are only concerned with the first value.

The CNN outputs an 8x1 tensor. The network can be trained using a mean squared error loss function and a gradient descent algorithm. Some models use maximum likelihood loss for the classification portion and mean squared error for localization (bounding box).

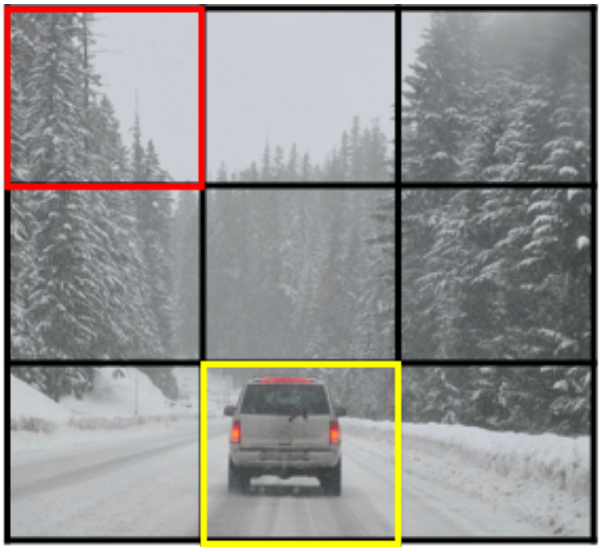

Figure 20. YOLO data labelling of an input image

To perform object detection YOLO first divides the image into a SxS grid. In the original publication, Redmon et al. [7] used a 7x7 grid. Many implementations of YOLO have used a 19x19

grid to achieve more accurate results while still maintaining its speed. For illustration, we will consider a 3x3 grid as shown in Fig. 20. The grid that contains the center of an object, is responsible for detecting that object. For e.g. grid 8 (highlighted yellow in Fig. 20) is responsible for detecting the car. For each grid an input of (x, y) is generated as shown earlier. For e.g. red grid (background) and yellow grid (car) in Fig. 20 are labelled as below:

- Background grid (red)
    - x – RGB pixel values of the grid
    - y – (0, None, None, None, None, None, None, None, None)

  Note: 'y' has 8x1 dimensions as we have 4 classes including background
- Car grid (yellow)
    - x – RGB pixel values of the grid
    - y – $(1, b_x, b_y, b_w, b_h, 1, 0, 0)$
        - b_x, b_y, b_w, b_h – Define the bounding box coordinates with respect to grid 8 as defined earlier in this section
        - Assuming car class is the first class

YOLO first generates (x, y) input labelled data for each of the nine grids for an image and feeds it to a CNN. However instead of feeding them separately to the CNN, it does so by convolutional implementation of sliding window object detection as in OverFeat (Section 2.2.2.2). Thus, in one pass of an image to the CNN, YOLO generates output for all the nine grids. The CNN architecture of YOLO is as below:

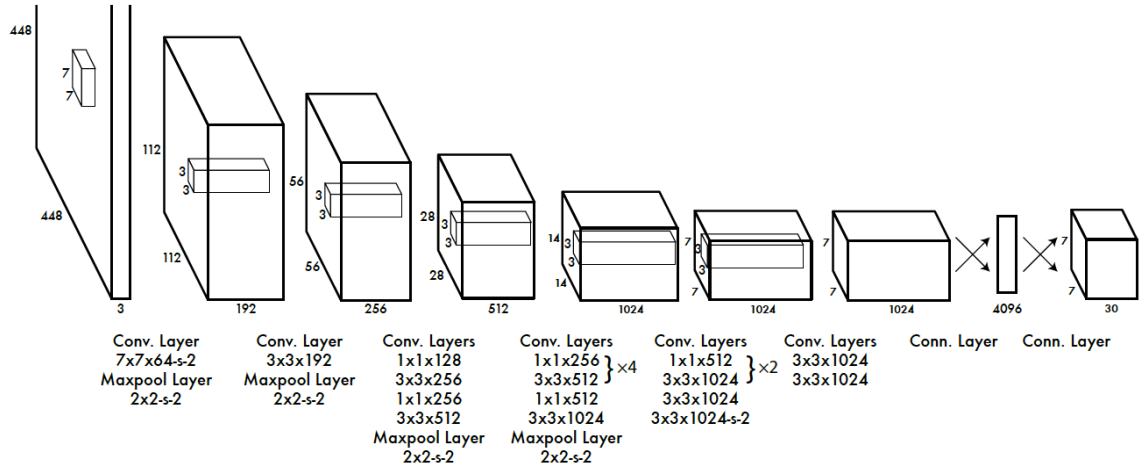

Figure 21. YOLO CNN architecture [7]

It is on the lines of a typical CNN architecture we discussed in Section 2.2.1.3. YOLO used mean squared loss function where classification and localization components are weighted differently. It used mini batch momentum gradient descent algorithm for training with a weight decay.

One of the issues faced by YOLO model was that it could give rise to multiple detections of the same object, thereby increasing the false positive rate (The definition and calculation of false positives is explained in detail in Section 4). For e.g. in our example in Fig. 20, grids 7,9,5 could also detect the car assuming its center lies in them. For illustration purposes the image was only divided into 3x3 grid and hence the grid size is large. For practical applications, the image is divided into higher number of grids to achieve more localization accuracy. However, this makes it possible that adjacent grids can detect the same object. YOLO uses non-max suppression to identify the best bounding box for each object and remove duplicate detections of the same object. Due to its simple pipeline, YOLO can be trained efficiently to achieve high speed. YOLO achieved a mean Average Precision (mAP) of 63.4% as compared to Faster R-CNN VGG-16 model's mAP of 73.2% on the PASCAL VOC datasets 2007 and 2012 combined. However, YOLO's detection

time was much faster and it could process 45 frames per second (fps) as compared to Faster R-CNN VGG-16 model's detection time of 7 (fps).

2.2.2.6.1 Limitations of YOLO

One grid can detect only one object. Hence if two small objects have their centers in the same grid, YOLO will miss one of them. To mitigate this limitation a 19x19 grid is used to reduce the probability of two objects having their centers in the same grid. Still it remained a limitation of YOLO version 1.

2.2.2.7 YOLO 9000 (version 2) [8]

YOLO version 2 was named as YOLO 9000 [8] and published in 2017. It improved on several features of YOLO version 1. The most significant improvement was the use of anchor boxes to improve the limitation mentioned above. It used anchor boxes to detect multiple objects having their centers in the same grid. YOLO9000 achieved a mAP of 73.4% on PASCAL VOC 2007+2012 dataset at 67 fps at 544x544 image resolution [8]. This is slightly less than Faster R-CNN's mAP of 78.8% on PASCAL VOC 2007 and 75.9% on VOC 2012[6]. However, YOLO 9000 is much faster than Faster R-CNN during detection.

2.2.2.8 Single Shot Multibox Detector (SSD)

SSD was inspired by YOLO and bears a lot of resemblance with it. Convolutional layers reduce spatial dimension and resolution. Hence, YOLO could detect only relatively large objects. SSD used output of each of the convolutional layers for object detection to overcome this limitation. This improved the mAP to 74.3% [9] over PASCAL VOC 2007 and 2012 datasets while maintaining high speed of detection.

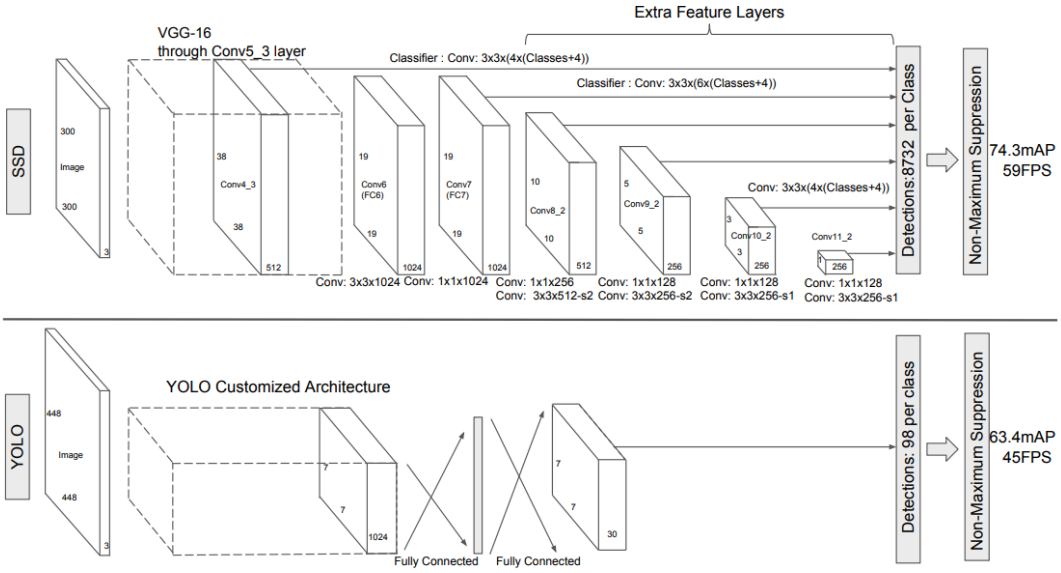

Figure 22. SSD CNN architecture [9]

3 Dataset

For this research, we used the Caltech-UCSD Birds 200-2011(CUB-200-2011) [12] dataset. CUB-200-2011 is an image dataset of 200 categories of bird species:

- No. of object classes: 200
- No. of images: 11,788
    - Train images: 5,994
    - Test images: 5,794
- Images are annotated with bounding boxes

The bird species closely resemble each other. The dataset was created to research on fine-grained image classification and object detection.

We carved out a validation set from the training set as below:

- Train images: 5,394
- Validation images: 600
- Test images: 5,794

The validation set is used to tune hyper parameters.

4 Evaluation Metric

For this section, we use the following acronyms:

- TP - True Positive
- TN - True Negative
- FP - False Positive
- FN - False Negative

Some of the challenges in evaluating object detection are:

1. Accuracy = (TP+TN) /(TP+FP+TN+FN) may be difficult to define as an image may contain objects of multiple classes. In sliding window object detection algorithms, many windows may only contain background(negatives) making accuracy a biased measure.
2. Along with evaluating classification, localization needs to be evaluated.

Before defining an appropriate measure, we consider some commonly used terms in evaluation.

- Precision = TP/(TP+FP): It is the ratio of true positives to the model predicting a positive outcome.
- Recall = TP/(TP+FN): Recall indicates ratio of actual positive occurrences correctly predicted by the model.
- Intersection over Union (IoU): Localization is the second key component of object detection. Along with predicting the class of the object in the image, we are also concerned with how accurately the network predicts the bounding box around the object.

  IoU = (Area of overlap between predicted box and ground truth box) / (Area of union of predicted box and ground truth box)

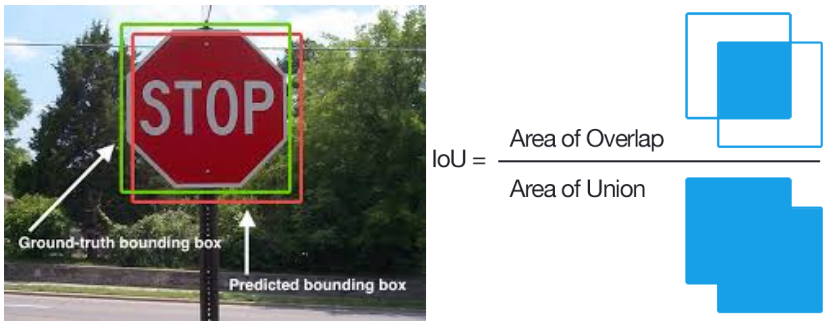

Figure 23. Intersection over Union (IoU) [25]

We now try to gain an understanding of mean average precision (mAP). Firstly, we need to define what is meant by a positive detection for object detection. When the model predicts an object, if its confidence score is above a threshold and bounding box has an IoU greater than the IoU threshold, it is considered a positive detection. The default value for confidence score threshold and IoU threshold is 0.5. If the predicted class matches the ground truth class it is considered a true positive else it is considered as a false positive. An object detection model could detect multiple boxes for one object. Only the first detection (detection with the highest confidence score) is considered as true positive while the remaining are considered as false positives. For e.g. if the model predicts five positive detections for an object in the image, only the first detection is considered as a true positive while remaining ones are considered as false positives.

Precision and recall values vary depending on the threshold of confidence score (p_c in Section 2.2.2.6) and IoU. For e.g. if we set the confidence score threshold at 0.5 we may get a certain number of positive detections, while if we increase the confidence threshold to 0.9 the number of positive detections will decrease. Changing the IoU threshold for detection also affects the precision and recall values. As we relax the confidence score and IoU threshold, the number of true positives will increase at the expense of including many more false positives. This reduces the precision. At the same time, the number of false negatives will reduce and hence recall increases. Similarly increasing the confidence and IoU threshold will increase the precision while reducing

recall. This is the precision recall tradeoff for a given IoU threshold. We fix the IoU at its default value 0.5 and calculate the precision recall value for every class by varying the confidence threshold. For e.g. the precision recall curve for car class in our continued example at IoU of 0.5 may be as below:

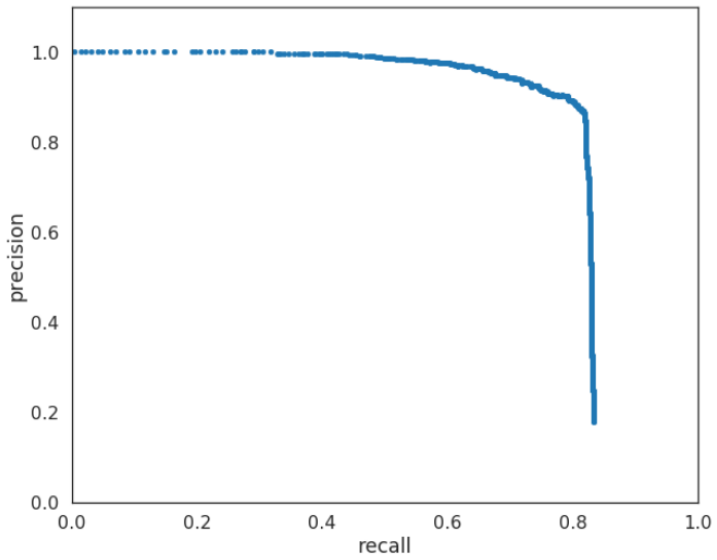

Figure 24. Sample precision vs recall curve for a specific IoU threshold [26]

Based on the precision recall curve, we calculate the Average Precision(AP) as the mean precision at equally spaced recall values. In this research, we have considered PASCAL mAP@0.5 metric that calculates AP across eleven equally spaced recall values given by $Recall_i$ = [0, 0.1, 0.2, …, 1.0].

Average Precision (AP) = $(1/11) * \sum_i Precision(Recall_i)$

The mAP for a class is calculated by taking the mean of the AP for that class over all the test images. The mAP for the entire dataset is calculated by taking the mean over all the classes. As the mAP was calculated at an IoU of 0.5 it is denoted as mAP@0.5.

5 Proposed Approach

Fine-grained object detection aims at training general purpose object detection models for fine-grained datasets. All the object detection models surveyed in the Related Works (section 2) are trained on COCO [10] or PASCAL VOC [11] datasets. These datasets have common objects from real life for e.g. a car, a pedestrian, a bike that are relatively easier to differentiate from each other.

5.1 Baseline approach

Turner et al. [13] in 2016 trained Fast R-CNN network on the CUB-200 dataset for fine-grained object detection. Turner et al. used keypoint density region proposal algorithm instead of selective search algorithm to generate RoI's. Remaining architecture of Fast R-CNN network was the same. This algorithm tries to identify areas within the image with a high density of keypoints. The success of this algorithm depends on the premise that areas which are dense in keypoints are most likely to contain objects. Let us first comprehend what keypoints and density of keypoints mean in this context. To identify keypoints, the algorithm generates Scale-Invariant Feature Transform (SIFT) features [13]. SIFT features identify points with a large change in gradient and these are referred to as keypoints. Consider an image with mean 'μ' and standard deviation 'σ' keypoints in an area of size 's'. Given a region 'r' of the same size 's' in the image, having 'x' keypoints, the density for region 'r' is:

Density of keypoints for 'r' = $(x-\mu)/\sigma$

The keypoint density region proposal algorithm [13] is defined as below:

KDRP (image, regionsNeeded) → outputRegions

1. keypointCoordinates ← SIFT-like feature generation(image)
2. keypointMean ← mean(numberOfKeypoints([256 uniform regions of keypointCoordinates])

3. keypointStdDev ← stdDev(numberOfKeypoints([256 uniform regions of keypointCoordinates])
4. lengthOutput ← 0
5. RoI ← [ ]
6. while (lengthOutput < regionsNeeded)
    a. r ← generateRandomRegion(image)
    b. densityPercentile ← percentile(zScore(numberOfKeypoints(r), keypointMean, keypointStdValue))
    c. if binomialTrialSuccess(densityPercentile)
    d. then RoI ← RoI + r
7. return outputRegions

The algorithm slides a square window with a uniform stride over the image to calculate the mean and standard deviation of keypoints (steps 1-3). Then in step 6, it repeatedly and stochastically generates a region 'r' (not necessarily a square) and examines if its keypoint density is within a certain threshold percentile of the mean and standard deviation. Regions within the threshold density are candidates for becoming a RoI. A candidate is then binomially sampled to determine whether it should be a RoI. The number of regions to be generated in RoI can be specified. The remaining pipeline of Fast R-CNN was maintained as it is in this approach. The algorithm used momentum mini-batch gradient descent with learning rate decay. Key parameters of the training are as below:

- Base learning rate of .01, decreasing by a factor of 10 every 500,000 iterations
- Momentum term 0.9, batch size

The model trained for 5,000,000 iterations and achieved a mAP of 66.24% on the CUB-200-2011 dataset with a detection time of 1 second on a test image.

5.2 Project approach

In this research, we aimed to improve the mAP of fine-grained object detection on CUB-200-2011 dataset. We trained models for fine-grained object detection on CUB-200-2011 using both the state-of-art paradigms in object detection i.e. region proposal (R-CNN family) and regression based (SSD).

For region proposal methods, Faster R-CNN models were trained as they ~10 times faster in detection than the Fast R-CNN models [6]. For regression based models, we trained SSD models. This necessitated building an object detection pipeline that could enable training of different types of models without any code changes. To accomplish this, we used TensorFlow's object detection framework. TensorFlow [27] is an open source machine learning framework based on dataflow graphs. The graph nodes represent mathematical operations and graph edges represent multidimensional data arrays known as tensors. TensorFlow's Object Detection framework [28] is built on top of TensorFlow.

5.2.1 Implementation pipeline

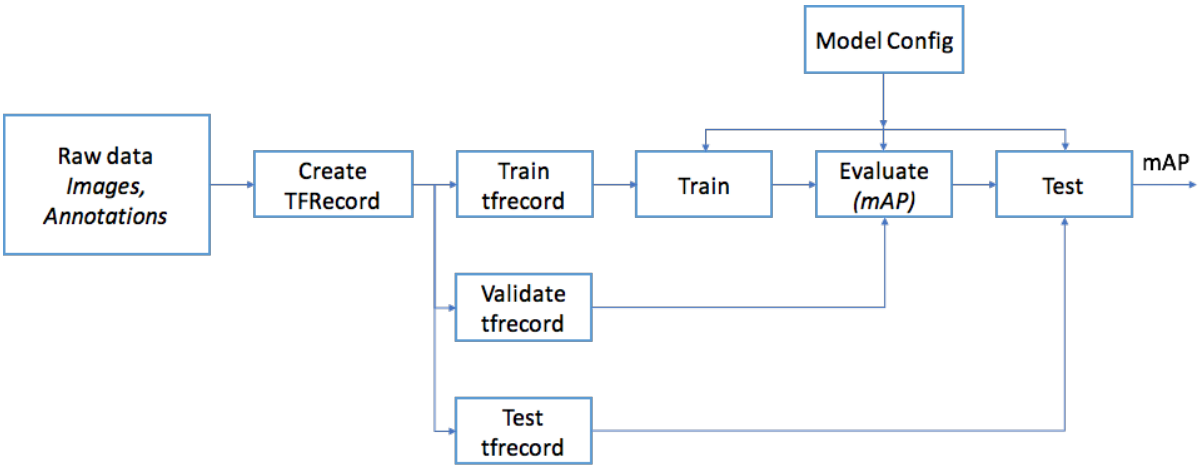

Figure 25. Implementation pipeline for fine-grained object detection using TensorFlow object detection

TensorFlow requires the data to be in the 'tfrecord' format. The tfrecord format enables splitting, creating batches, shuffling data and providing a uniform format across network architectures and systems. CUB-200-2011 dataset provided the image labels and their bounding boxes in text files. These were converted into a tfrecord.

The original dataset comprised of only train and test data. The train data was split into train and validation set as mentioned in Section 3. Then the train, validation and test splits were converted into three separate tfrecord files.

One of the key design principles was to develop a model independent implementation pipeline. This would enable us to train all the models using the same pipeline. Hence the train and test scripts were independent of the model. The model configuration containing the network architecture, training parameters like batch size, initialization, training data path, test data path, and other parameters were specified using a config file. The config file for a Faster R-CNN model is as below:

```
model {
  faster_rcnn {
    num_classes: 200
    image_resizer {
      fixed_shape_resizer {
        height: 500
        width: 500
      }
    }
    feature_extractor {
      type: 'faster_rcnn_inception_v2'
      first_stage_features_stride: 16
    }
    first_stage_anchor_generator {
      grid_anchor_generator {
        scales: [0.25, 0.5, 1.0, 2.0]
        aspect_ratios: [0.5, 1.0, 2.0]
        height_stride: 16
        width_stride: 16
      }
    }
    first_stage_box_predictor_conv_hyperparams {
      op: CONV
      regularizer {
        l2_regularizer {
          weight: 0.01
        }
      }
      initializer {
```

```
    truncated_normal_initializer {
      stddev: 0.01
    }
  }
}
first_stage_nms_score_threshold: 0.0
first_stage_nms_iou_threshold: 0.7
first_stage_max_proposals: 300
first_stage_localization_loss_weight: 2.0
first_stage_objectness_loss_weight: 1.0
initial_crop_size: 14
maxpool_kernel_size: 2
maxpool_stride: 2
second_stage_box_predictor {
  mask_rcnn_box_predictor {
    use_dropout: false
    dropout_keep_probability: 1.0
    fc_hyperparams {
      op: FC
      regularizer {
        l2_regularizer {
          weight: 0.0
        }
      }
      initializer {
        variance_scaling_initializer {
          factor: 1.0
          uniform: true
          mode: FAN_AVG
```

```
                }
              }
            }
          }
        }
        second_stage_post_processing {
          batch_non_max_suppression {
            score_threshold: 0.0
            iou_threshold: 0.6
            max_detections_per_class: 100
            max_total_detections: 300
          }
          score_converter: SOFTMAX
        }
        second_stage_localization_loss_weight: 2.0
        second_stage_classification_loss_weight: 1.0
      }
    }
    train_config: {
      batch_size: 16
      optimizer {
            adam_optimizer: {
                learning_rate {
                exponential_decay_learning_rate:
{initial_learning_rate:0.00001}
                }
            }
      }
      gradient_clipping_by_norm: 10.0
```

```
        fine_tune_checkpoint:
"faster_rcnn_inception_v2_coco_2018_01_28/model.ckpt"
        from_detection_checkpoint: true
        batch_queue_capacity:200
        data_augmentation_options {
          random_horizontal_flip {
          }
          random_crop_pad_image{
          }
          rgb_to_gray {
          }
          random_black_patches {
          }
        }
      }

      train_input_reader: {
        tf_record_input_reader {
          input_path: path to train tfrecord
        }
        label_map_path: path to label
      }
      eval_config: {
        num_examples: 600
      }
      eval_input_reader: {
        tf_record_input_reader {
          input_path: path to validation tf record
        }
```

```
    label_map_path: path to labels
    shuffle: false
    num_readers: 1
  }
```

Multiple models were then trained. To monitor the convergence of models and tune hyperparameters we used the validation set.

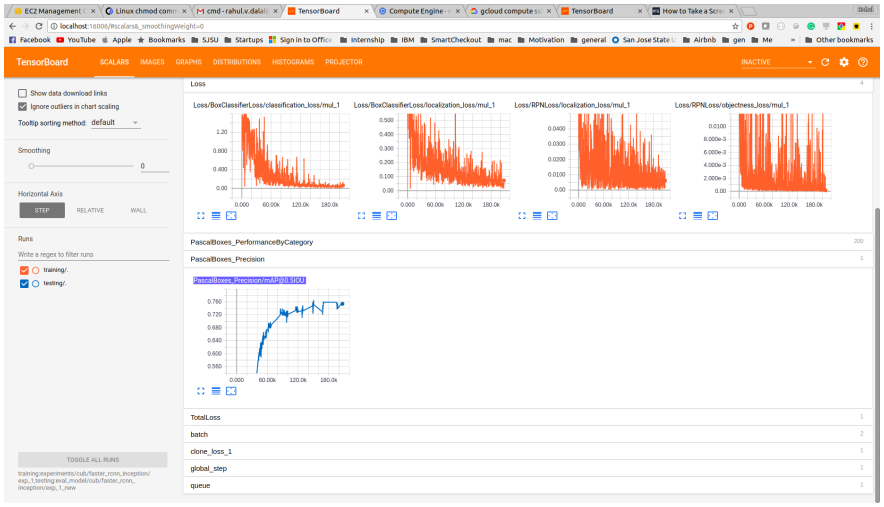

Figure 26. Monitoring validation performance on TensorBoard while training

While a model was being trained on the train tfrecord data, its mAP was evaluated at regular intervals on the validation tfrecord as shown in Fig. 27. Also, the model weights were stored at regular intervals as checkpoints. This mechanism allowed us to determine if the model was overfitting and select the best trained model from the available checkpoints. We tested our three best trained models on the entire test dataset at the end. The technologies used in developing the pipeline are:

- Language: Python 3.6
- TensorFlow v1.5
- TensorFlow Object Detection prerequisites [28]
    - Protobuf 2.6

    - Python-tk
    - Pillow 1.0
    - lxml
    - tf Slim
    - Jupyter notebook
    - Matplotlib
    - TensorFlow
    - Cython
    - cocoapi
- TensorFlow Object Detection
- Anaconda virtual environment
- GitHub

Salient features of the implementation pipeline:

- It could train different types of models using the same codebase. Both SSD and Faster R-CNN were trained using the same codebase.
- Transfer learning could be easily employed.
- Time to initiate training of a new model was reduced significantly as opposed to developing each model individually.
- Multiple models could be trained easily, constrained only by system resources.
- Models could be monitored for underfitting and overfitting. This enabled efficient utilization of computation resources as deep learning is extremely resource (computation) and time intensive.

5.2.2 Models trained

Training deep neural networks on a dataset as large as CUB-200-2011can take several days. Transfer learning was used to train existing COCO based object detection models on CUB-200-2011. We began with training SSD Mobilenet [29] models as they have the lowest depth and took least training time. Based on the results, we identified the techniques that were crucial to training general purpose object detection models for fine-grained object detection. We used our learnings to train deeper SSD models - SSD Inception [29], to obtain more accurate results. SSD Inception models use the SSD architecture and replace a normal convolutional layer with an Inception convolutional layer. The details of Inception layer are explained later in Section 2.2.1.4. This helped us corroborate and adapt our findings from SSD Mobilenet. We then proceeded to train Faster R-CNN Inception models within a restricted scope dictated by our findings from training SSD models. Finally, we experimented by training substantially deep Faster ResNet 101 models [29], which have 101 layers and use ResNet architecture [21]. Some of the key techniques used by this research are highlighted in the following sections.

5.2.2.1 Transfer Learning

Instead of training a model from scratch, the training time can be reduced by initializing the weights of our model with that of an already trained model solving a problem similar to the one under consideration. In this way, the new model can incorporate the learnings of the already trained model. Transfer learning is used to significantly reduce the training time in many cases. This research used models trained on the COCO dataset [29], that contains many more object classes than a bird. However, with transfer learning our models can utilize learnings from COCO models like identifying vertical edges, differentiating objects from background etc. that will be common in general purpose as well as fine-grained object detection.

5.2.2.2 Data Augmentation

Data augmentation creates more diverse data. With data augmentation, we can learn to identify the object of interest in varying sizes, lighting conditions, at different areas of the image etc. Also, as the object classes are very similar in nature, data augmentation forces the model to learn features beyond what is obvious in the normal dataset, which can be useful in differentiating similar object classes. During each training iteration, every data augmentation technique specified in the config file is either applied with a probability of 0.5 or the image is used as it is without any augmentation. The following data augmentation techniques [30] were used in the research:

1. Random horizontal flip – It prevents the model from assuming that a feature is present in only a certain area of the image. Instead it helps the model focus on the relation between features. Most of the bird images are top down and hence the beak is present in the upper half of the image. As a result, the model may fail to detect the object in the below image.

Figure 27. Black footed albatross from CUB-200-2011 with beak in lower right half [12]

2. Random crop image with padding – Random cropping makes the model better in differentiating the object from the background. Padding is used as the CNN expects a fixed size input. Random cropping also helps the model learn different parts of an object better.

3. Random RGB to Gray – Many of the object classes (birds) have one dominant color but also have additional colors.

Figure 28. Object class with primary color as red and secondary color as pastel green

In Fig. 25 cardinal class has red color as the dominant color but occasionally it may also have the pastel green color as shown. Random RGB to Gray transformation influences the CNN to learn finer features of the bird and not just depend on the color. This helps in detecting even the secondary colored images of the bird.

4. Random black patches – Random black patches introduce limited noise during the training process that makes the model more stable and robust. For e.g. If the beak of the bird was covered with a black patch the model will learn to detect the object despite the beak not being visible. During testing if the beak is not clear in one of the images, the model can still detect the object.
5. Random distort color – Random color distortions can provide better results in cases where one object class can have more than one color. As most of the classes in our dataset had more than one color, random distort color was used as a data augmentation technique.
6. Random adjust saturation – It modifies the intensity of colors in the image and helps the model learn the object across different color intensities.

7. Random adjust brightness- Random adjust brightness can improve the model to detect objects in different lighting conditions.
8. Random adjust contrast – Random adjust contrast was used in a similar way to random adjust brightness.

We used a systematic approach in applying the data augmentation techniques. Random horizontal flipping and random cropping with padding are unbiased towards the remaining six techniques. Hence, they were combined with the other data augmentation techniques. We researched on three separate themes for data augmentation:

- Transform color to gray: In this theme, we used random RGB to gray and random black patches. Color and brightness variations were not used in these cases. The idea was to see the effect of color to gray transformation on the performance of the model.
- Color variations: In this theme, we considered the model's performance by varying colors in the image. For this we used random distort color and random adjust saturation.
- Brightness variations: In this theme, we varied the lighting conditions of the image by using random adjust brightness and random adjust contrast.

5.2.2.3 Adam Optimization [31]

The models discussed in related works and the baseline approach used gradient descent with momentum. Recent research has made breakthroughs in optimization algorithms for deep learning. Adam optimization combines Momentum and RMSProp [32] optimization algorithms, and is known to yield faster training convergence than Momentum in most cases [31]. Hence, we utilize Adam optimization algorithm for training our models.

5.2.2.4 Dropout Regularization [33]

Another suggested best practice for deep learning in recent years is Dropout Regularization [33]. Dropout as the name suggests ignores ('drops out') randomly selected neurons during an iteration's forward pass while training. Their contribution of activation to downstream neurons is nullified during that iteration. For each iteration, the set of neurons that are ignored are randomly chosen as per a probability. This research used a keep probability of 0.8 (thus dropout probability was 0.2) for all the models trained during research.

5.2.2.5 Network architectures

This research used Inception and ResNet architectures for the CNN in object detection models. Inception architecture has yielded better results than a conventional CNN in many applications. ResNet architecture allows to train deeper networks. This enables us to train a more accurate and deeper CNN.

6 Experiments and Results

We trained the following COCO models [29] on the CUB-200-2011 dataset. For easier representation of results their names have been shortened as below:

- ssd_mobilenet_v1_coco: ssd_mobile
- ssd_inception_v2_coco: ssd_inception
- faster_rcnn_inception_v2_coco: faster_rcnn_inc
- faster_rcnn_resnet101_coco: faster_rcnn_res
- faster_rcnn_resnet101_coco_low_proposal: faster_rcnn_res_low_prop

No.	Model	Batch Size	Optimizer	Hor. Flip	Crop Img.	RGB to Gray	Black Patches	Dist. Color	Adj. Sat.	Adj. Bright.	Adj. Contr.	Valid. mAP
1	ssd_mobile	32	RMSProp	0	0	0	0	0	0	0	0	**52.2**
2	ssd_mobile	64	RMSProp	0	0	0	0	0	0	0	0	**55.8**
3	ssd_mobile	64	Adam	0	0	0	0	0	0	0	0	**57.2**

Table 1. Validation results for SSD Mobilenet models traiend on CUB-200-2011 dataset without data augmentation

Table 1. indicates the validation results for SSD Mobilenet models trained on CUB-200-2011. The data augmentation transformation outlined in Section 5.2.2.2 were used and names have been shortened for display purposes. A ‘0’ in a data augmentation option indicates that the data augmentation technique was not used, while a ‘1’ indicates the technique was used. *Valid. mAP* refers to the mAP achieved on validation dataset.

As shown in Table 1, SSD Mobilenet models were trained without any data augmentation. Adam optimizer and batch size of 64 gave the best results. These hyperparameters were maintained in subsequent experiments for SSD Mobilenet.

No.	Model	Batch Size	Optimizer	Hor. Flip	Crop Img.	RGB to Gray	Black Patches	Dist. Color	Adj. Sat.	Adj. Bright.	Adj. Contr.	Valid. mAP
4	ssd_mobile	64	Adam	1	1	0	0	0	0	0	0	**60.8**
5	ssd_mobile	64	Adam	1	1	1	1	0	0	0	0	**64.8**
6	ssd_mobile	64	Adam	1	1	0	0	1	1	0	0	**53.8**
7	ssd_mobile	64	Adam	1	1	0	0	0	0	1	1	**60.7**

Table 2. Validation results for SSD Mobilenet models trained on CUB-200-2011 dataset with data augmentation

Random horizontal flipping and random cropping image, increased the mAP to 60.8% on the validation dataset. As already highlighted, random horizontal flipping helped the model to detect the object in different positions. Random cropping image improves the model's capabilities in differentiating the object from background. We then probed along the three themes of data augmentation as explained earlier:

- RGB to gray
- Color variations
- Brightness variations

Random horizontal flipping and random cropping image are unbiased towards all the three themes and hence were continued during training these three models. Most of the birds had one dominant color and one or two secondary colors. Random RGB to gray augmentation enhanced the model's capability to detect the bird in its secondary colors. Random RGB to gray along with random black patches improved the validation mAP by 4 percentage points. Data augmentations using color and saturation distortions performed poorly. One possible reason for poor performance of color variation technique could be that this data augmentation technique could introduce some color

variations during training which the model may not encounter during testing at all (as each bird has at most three colors). Color distortions in fact reduced the model's performance as compared to without any data augmentation. Contrast and brightness adjustments yielded neutral results and needed further probing in subsequent models.

No.	Model	Batch Size	Optimizer	Hor. Flip	Crop Img.	RGB to Gray	Black Patches	Dist. Color	Adj. Sat.	Adj. Bright.	Adj. Contr.	Valid. mAP
8	ssd_inception	64	Adam	1	1	0	0	0	0	0	0	**69.4**
9	ssd_inception	64	Adam	1	1	1	1	0	0	0	0	**70.7**
10	ssd_inception	64	Adam	1	1	0	0	1	1	0	0	**57.9**
11	ssd_inception	64	Adam	1	1	0	0	0	0	1	1	**69.6**

Table 3. Validation results for SSD Inception models trained on CUB-200-2011 dataset with data augmentation

We then trained, SSD inception models on the same data augmentation techniques and found Random RGB to gray with random black patches along with random horizontal flipping and random cropping of image performed the best. Also, the effects from color and saturation distortions were poor again. Hence while training the much deeper faster R-CNN models we did not use color and saturation distortions.

Mod. #	Model	Batch Size	Optimizer	Hor. Flip	Crop Img.	RGB to Gray	Black Patches	Adj. Bright.	Adj. Contr.	Valid. mAP
12	faster_rcnn_inc	16	Adam	0	0	0	0	0	0	**67.7**
13	faster_rcnn_inc	16	Adam	1	1	0	0	0	0	**75.2**
14	faster_rcnn_inc	16	Adam	1	1	1	1	0	0	**75.6**
15	faster_rcnn_inc	16	Adam	1	1	0	0	1	1	**70.4**

Table 4. Validation results for Faster R-CNN Inception models trained on CUB-200-2011 dataset with data augmentation

Faster R-CNN model generally provides more accurate predictions by trading off speed with accuracy as compared to SSD. Faster R-CNN model combined with Inception CNN architecture gave a significant improvement in mAP. Subsequently we could use data augmentation to improve the model's performance. Random RGB to gray combined with random black patches and random horizontal flipping with random cropping of image outperformed other data augmentation techniques giving a mAP of 75.6% on the validation set after training for 200,000 steps. All the inception models were trained for ~200,000 steps. The models started overfitting around 220,000 steps and gave best results in the range of 190,000 – 210,000. Constrained by computation and GPU resources, Faster R-CNN Inception models could only be tested for a maximum batch size of 16 at which we observed better training convergence as compared to batch size of 8 and 1.

Mod. #	Model	Batch Size	Optimizer	Hor. Flip	Crop Img.	RGB to Gray	Black Patches	Adj. Bright.	Adj. Contr.	Valid. mAP
16	faster_rcnn_res	1	Adam	1	1	0	0	0	0	**44.8**
17	faster_rcnn_res	1	Adam	1	1	1	1	0	0	**41.2**
18	faster_rcnn_res_low_prop	1	Adam	1	1	0	0	0	0	**47.1**
19	faster_rcnn_res_low_prop	1	Adam	1	1	1	1	0	0	**50.1**

Table 5. Validation results for Faster R-CNN ResNet models trained on CUB-200-2011 dataset with data augmentation

ResNet's are significantly deeper than a normal Inception network. In this research, we trained 101 layer ResNet architecture for Faster R-CNN models. Ideally, they should have performed better than other networks. However, the Faster R-CNN models did not converge even after training for ~4,000,000 steps at a batch size of 1. Mobilenet and inception architectures (for both SSD and Faster R-CNN) had performed poorly with a batch size of 1. One of the plausible reasons could be the constraint of batch size 1. The poor performance of the ResNet models could also be

due to inadequate data for training such deeper network or our inability to tune hyperparameters for such a deep network.

The top 3 models were tested on the entire test dataset.

Mod. #	Model	Batch Size	Optimizer	Hor. Flip	Crop Img.	RGB to Gray	Black Patches	Valid. mAP	Test mAP
1	faster_rcnn_inc	16	Adam	1	1	1	1	**75.6**	**71.5**
2	faster_rcnn_inc	16	Adam	1	1	0	0	**75.2**	**70.8**
3	ssd_inception	64	Adam	1	1	1	1	**70.7**	**65.9**

Table 6. Test results for top 3 models (validation mAP) trained on CUB-200-2011 dataset

Faster R-CNN Inception model with random horizontal flipping, random cropping image, random RGB to gray and random black patches performed the best and gave a mAP of 71.5% on the test dataset.

Object detection times for TensorFlow implementation of Faster R-CNN and SSD [29] are:

Model	Detection time (ms)
ssd_mobilenet_v1_coco	30
ssd_inception_v2_coco	42
faster_rcnn_inception_v2_coco	58
faster_rcnn_resnet101_coco	106
faster_rcnn_resnet101_lowproposals_coco	82

Table 7. Detection time for object detection models

Faster R-CNN Inception takes ~42 ms for detection.

Model	mAP
KDRP Fast R-CNN (baseline)	66.24
faster_rcnn_inception_v2_coco with data augmentation (our model)	71.5

Table 8. Research achievement (mAP) vs baseline

To the best of our knowledge, the research achieved an improvement of more than 5 percentage points in fine-grained object detection on CUB 200-2011 dataset attaining a mAP of 71.5%.

7 Conclusion

The paper proposed deep learning models for fine-grained object detection. We trained Faster R-CNN object detection model with Inception architecture on the CUB-200-2011 dataset to achieve a mAP of 71.5%. To the best of our knowledge this represents the best mAP on CUB-200-2011 dataset. In addition to accomplishing the best results on CUB-200-2011, we explained our approach and experiments that could provide insights into fine-grained object detection. To reduce training time, we used transfer learning to retrain COCO object detection models for fine-grained object detection on CUB-200-2011 dataset. One of the key discernments from our experiments was that data augmentation could improve accuracy of fine-grained object detection. We also implemented an object detection pipeline using TensorFlow's object detection framework that could enable training of different models without changes in source code. We experimented with 8 data augmentation techniques. Future research could experiment on effectiveness of other data augmentation techniques for fine-grained object detection. Training deeper networks like Faster R-CNN ResNet 101 COCO with higher batch size could be explored to achieve better results.

References

[1] Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3), pp.273-297.

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

[3] I. Goodfellow, Y. Bengio and A. Courville, Deep Learning, MIT Press, 2016.

[4] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In CVPR, 2014.

[5] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Region based convolutional networks for accurate object detection and segmentation. TPAMI, 2015.

[6] S. Ren, K. He, R. Girshick and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, pp. 1137-1149, 2017.

[7] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. arXiv preprint arXiv:1506.02640, 2015.

[8] J. Redmon and A. Farhadi. Yolo9000: Better, faster, stronger. arXiv preprint arXiv:1612.08242, 2016.

[9] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.- Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In European Conference on Computer Vision, pages 21–37. Springer, 2016.

[10] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollar, and C. Lawrence Zitnick. Microsoft ´ COCO: Common objects in context. In ECCV, 1 May 2014.

[11] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. International journal of computer vision, 88(2):303–338, 2010.

[12] Wah C., Branson S., Welinder P., Perona P., Belongie S. “The Caltech-UCSD Birds-200-2011 Dataset.” Computation & Neural Systems Technical Report, CNS-TR-2011-001.

[13] J. T. Turner, K. M. Gupta, and D. W. Aha. Keypoint density-based region proposal for fine-grained object detection and classification using regions with convolutional neural network features. CoRR, abs/1603.00502, 2015.

[14] P. Viola and M. Jones. Robust real-time object detection. 2nd Intl. Workshop on Statistical and Computational Theories of Vision, 2001.

[15] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In Proc. CVPR, pages 511–518, 2001.

[16] A. Ng, “deeplearning.ai,” deeplearning.ai. [Online]. Available: https://www.deeplearning.ai/. [Accessed: 16-Apr-2018].

[17] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In ICLR, 2015.

[18] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In NIPS, 2012.

[19] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. CoRR, abs/1409.4842, 2014.

[20] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. CoRR , abs/1312.4400, 2013.

[21] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” arXiv:1512.03385, 2015.

[22] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, “Overfeat: Integrated recognition, localization and detection using convolutional networks,” in International Conference on Learning Representations (ICLR), 2014

[23] J. R. Uijlings, K. E. van de Sande, T. Gevers, and A. W. Smeulders, “Selective search for object recognition,” International Journal of Computer Vision (IJCV), 2013.

[24] R. Girshick, “Fast R-CNN,” in IEEE International Conference on Computer Vision (ICCV), 2015.

[25] A. Roserbock, “Intersection over Union (IoU) for object detection,” PyImageSearch, 27-Sep-2016. [Online]. Available: https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/. [Accessed: 16-Apr-2018].

[26] T. C. Arlen, “Understanding the mAP Evaluation Metric for Object Detection,” Medium, 01-Mar-2018. [Online]. Available: https://medium.com/@timothycarlen/understanding-the-map-evaluation-metric-for-object-detection-a07fe6962cf3. [Accessed: 16-Apr-2018].

[27] “TensorFlow,” TensorFlow. [Online]. Available: https://www.tensorflow.org/. [Accessed: 16-Apr-2018].

[28] TensorFlow, “TensorFlow object detection,” *GitHub*. [Online]. Available: https://github.com/tensorflow/models/tree/master/research/object_detection. [Accessed: 16-Apr-2018].

[29] TensorFlow, “TensorFlow object detection model zoo,” GitHub. [Online]. Available: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md. [Accessed: 16-Apr-2018].

[30] TensorFlow, “TensorFlow object detection data augmentation techniques,” GitHub. [Online]. Available: https://github.com/tensorflow/models/blob/master/research/object_detection/protos/preprocessor.proto. [Accessed: 16-Apr-2018].

[31] Kingma, Diederik and Ba, Jimmy. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

[32] Tieleman, T. and Hinton, G. Lecture 6.5 - RMSProp, COURSERA: Neural Networks for Machine Learning. Technical report, 2012.

[33] Srivastava, Nitish, Hinton, Geoffrey, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan. Dropout: A simple way to prevent neural networks from overfitting. J. Mach. Learn. Res., 15(1):1929–1958, January 2014.

Table of Figures

Figure 1. Haar feature for eye region [14] 12

Figure 2. Convolution operation in CNN 14

Figure 3. Pooling operation in CNN 15

Figure 4. VGG-16 architecture [17] 17

Figure 5. Inception layer without channel reduction [16, 19] 19

Figure 6. Inception layer with channel reduction [16, 19] 20

Figure 7. Inception CNN architecture [19] 21

Figure 8. ResNet block 22

Figure 9. Comparison of 34 layered normal CNN vs 34 layered ResNet architecture [21] 23

Figure 10. Data labelling for CNN classification 24

Figure 11. Sliding window object detection using CNN 24

Figure 12. CNN classifying 4 object classes [22, 16] 25

Figure 13. OverFeat transformation of CNN classifying 4 objects [22, 16] 25

Figure 14. Sliding window object detection in 1 pass of CNN using OverFeat [22] 26

Figure 15. R-CNN model [4] 27

Figure 16. Fast R-CNN model [24] 28

Figure 17. Faster R-CNN model [6] 29

Figure 18. Object localization data labelling for car object class [16] 30

Figure 19. Object localization data labelling for background class (None) [16] 31

Figure 20. YOLO data labelling of an input image 31

Figure 21. YOLO CNN architecture [7] 33

Figure 22. SSD CNN architecture [9] 35

Figure 23. Intersection over Union (IoU) [25]..... 38

Figure 24. Sample precision vs recall curve for a specific IoU threshold [26] 39

Figure 26. Implementation pipeline for fine-grained object detection using TensorFlow object detection..... 43

Figure 27. Monitoring validation performance on TensorBoard while training 48

Figure 28. Black footed albatross from CUB-200-2011 with beak in lower right half [12] 51

Figure 29. Object class with primary color as red and secondary color as pastel green 52

Table of Tables

Table 1. Validation results for SSD Mobilenet models traiend on CUB-200-2011 dataset without data augmentation 55

Table 2. Validation results for SSD Mobilenet models trained on CUB-200-2011 dataset with data augmentation 56

Table 3. Validation results for SSD Inception models trained on CUB-200-2011 dataset with data augmentation 57

Table 4. Validation results for Faster R-CNN Inception models trained on CUB-200-2011 dataset with data augmentation 57

Table 5. Validation results for Faster R-CNN ResNet models trained on CUB-200-2011 dataset with data augmentation 58

Table 6. Test results for top 3 models (validation mAP) trained on CUB-200-2011 dataset 59

Table 7. Detection time for object detection models 59

Table 8. Research achievement (mAP) vs baseline 59