Master's Thesis

Authors:
Christoffer Bøgelund Rasmussen

Supervisors:
Kamal Nasrollahi

Aalborg University
VGIS
10th semester
Group 17gr1041

TBA

AALBORG UNIVERSITY

Title:
Master's Thesis
Subject:
Vision, Graphics and Interactive Systems
Project period:
1/2-2017 to TBA
Project group:
17gr1041
Participants:
Christoffer Bøgelund Rasmussen

TBA

Supervisor:
Kamal Nasrollahi
Printed copies:
TBA
Number of pages:
TBA
Appendix media:
AAU digital exam zip file
Finished:
TBA

Preface

Reading Guide

Tables, code listings and figures are numbered sequentially within each chapter. Citations are written as [x] where x denotes the reference number used in the bibliography. Code classes and functions are written as `class` and `function()`, respectively. Additional files have been uploaded to the AAU Digital Exam.

Christoffer Bøgelund Rasmussen

Contents

1 Introduction

Object detection is a fundamental area of computer vision that has had a great amount of research over the past decades. The general goal of object detection is to find a specific object in an image. The specific object is typically from within a pre-defined list of categories that are of interest for a given use case. Object detection generally consists of two larger tasks; localisation and classification. It is assumed that the objects of interest are not already located in the image and as objects can vary in number of pixels depending on factors such as distance and scale, objects must be both localised in an image and classified accurately. Localisation is typically done by with a bounding-box indicating where a given object is in the image. However, other methods such as objects' centres and closed boundaries can also be used. Not only is object detection an important task in localising and classifying, it is also a necessary earlier step in larger computer vision pipelines. For example, object detection is needed within the tasks such as activity and event recognition, scene understanding, and robotic picking.

Object detection is a challenging problem due to both some large scale issues and minute differences. Firstly, there is the challenge of differentiating objects between classes. Depending on the problem at hand the sheer number of potential categories present can be into the thousands or tens of thousand. On top of this separate object categories can be both very different in appearance, for example an apple and an aeroplane, but separate categories can also be similar in appearance, such as dogs and wolves.

Current state-of-the-art within object detection is also within the realm of deep learning with Convolutional Neural Networks (CNNs). This is exemplified with almost all entries in benchmark challenges such as Pattern Analysis, Statistical Modelling and Computational Learning Visual Object Classes (PASCAL VOC) [1], ImageNet [2], and Microsoft Common Objects in Context (MS COCO) [3] consisting of CNN based approaches. However, improvements are still needed before object detection can be used in real-world scenarios that require a high level of precision, accuracy, and performance.

1.1 Initial Problem Statement

An initial problem statement can be formed as follows:

How is object detection performed with CNNs?

Based upon this, Chapter *2 Problem Analysis* will outline these challenges. On top of this, related work into object detection, both current state-of-the-art and also classic methods, will be researched.

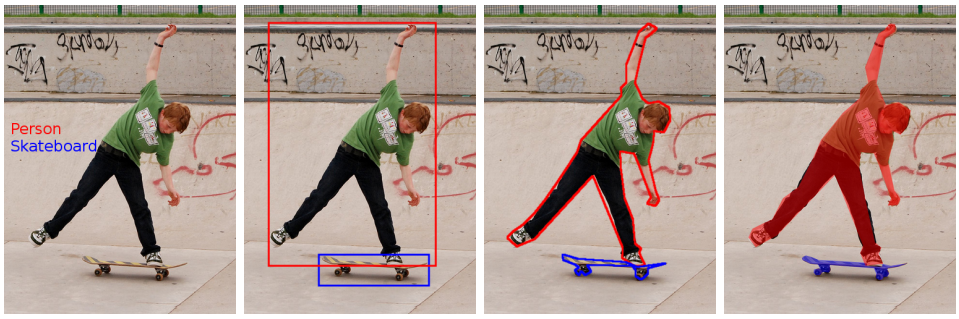
2 Problem Analysis

This chapter will outline object detection and it's key challenges. This includes aspects within robustness, computational-complexity and scalability. Once completed the key works within object detection will be analysed, both current state-of-the-art and notable older methods.

2.1 Object Detection

As mentioned in Section *1.1 Initial Problem Statement*, object detection consists of two larger tasks; classification and localisation. Depending on the problem at hand, object detection can be split into two categories. If only a single class is of interest, such as detecting a specific traffic sign, the object detection task is denoted as class-specific detection. Whereas, the more general case when multiple classes are of interest in an image is denoted as multi-class detection [4]. Key challenges such as PASCAL VOC, ImageNet, and MS COCO are of the latter task. This thesis will be within the multi-class detection domain and take these challenges into account when analysing related works in Section **?? ??** and determining the algorithm to be implemented and evaluated in Section **?? ??**. An analysis of these key challenges is done in Section **?? ??**. The goal of a detector is to output a list of labels from a predefined list of categories indicating which objects are present and where they are located in an image. Object detection has a number of related fields which share to common goal of categories relevant objects. This can be seen in Figure 2.1. In all four instances the goal is to categorise the two objects person and skateboard, however, the difference lies in the level of localisation precision. In Figure 2.1a, object categorisation aims to only classify the objects in the image without providing any indication as to where the objects are located. Object class detection in Figure 2.1b, localises the classified objects with the use of bounding-boxes, where ideally the bounding-boxes are placed as tightly around the given object as possible. Figure-ground segmentation in Figure 2.1c, indicates localisation with a lasso outline around the objects. Finally, in Figure 2.1d, semantic-segmentation localises objects at a pixel-level classifying each pixel that is related to the given object.

correct section refs to above



(a) **(b)** **(c)** **(d)**

Figure 2.1: Example of vision tasks related to object detection. All tasks have the common goal of categorising predefined objects. Methods are: object categorisation (a), object class detection (b), figure-ground segmentation (c), semantic Segmentation (d). Image and class labels taken from MS COCO [3].

A more recent example of segmentation is that of instance segmentation. Instance

segmentation varies to semantic segmentation in that individual instances of objects are classified as such. If multiple instances of the same object is present, such as an image of a crowd with many people, in semantic segmentation all people will be given the same label as one large group. However, in instance segmentation the people are still given the same label but individual instances of a person is also found. This area of research within segmentation is relatively new, however, is beginning to become more popular in comparison to semantic segmentation. For example, the MS COCO segmentation challenge which has been held in 2015 and 2016 only accepts instance segmentation entries.

2.2 Main Challenges

The challenges of object detection can be split into two groups as per [4]:

- Robustness-related.
- Computational-complexity and scalability-related.

The following sections will outline the above.

2.2.1 Robustness-related Challenges

Robustness-related refers to the challenges in appearance within the both of intra-class and inter-class. Intra-class is the differences in appearance of objects which are of the same class. For example as seen in Figure 2.2, all of the images belong to the superclass chair from the ImageNet training set [2], however, vary greatly in their overall appearance.

(a) **(b)** **(c)** **(d)**

Figure 2.2: Examples of intra-class appearance variation. All images have the label chair in the ImageNet training set [2].

An object detection system must be able to learn

explanation that typically obj detection is supervised learning

the appearance variations that can occur intra-class. These variations can be categorised into two types as per [5]:

- Object variations.
- Image variations.

Object variations consist of appearance differences between instances of colour, texture, shape, and size. Image variations are differences not related to the object instances themselves but rather consist of conditions such as lighting, viewpoint, scale, occlusion, and clutter. Based upon these conditions the task of both classifying a given object as a given class but also differentiating the potentially largely varying objects into the same class challenging.

unsure if to add explanation of structured and unstructured classes

Robustness-related challenges can also occur with inter-class appearance differences. This refers to the differences between objects that are regarded as different categories. Challenges arise in scenarios where an object detector must decide if an instance is between classes that are very similar. For example using images and their respective classes from ImageNet [2], in Figure 2.3 and Figure 2.4 the differences between the two examples are very similar, however, their class labels are different. In Figure 2.3a and Figure 2.3b the class labels are mini-bus and delivery truck respectively. In Figure 2.4a and Figure 2.4b the labels are white wolf and German shepherd.

(a) **(b)**

Figure 2.3: Examples of inter-class appearance variation. Both images are from the ImageNet training set [2] and have the labels mini-bus (a) and delivery truck (b).

(a) **(b)**

Figure 2.4: Examples of inter-class appearance variation. Both images are from the ImageNet training set [2] and have the labels White wolf (a) and German shepherd (b).

It should be noted that this is a task-specific if inter-class appearance similarities is a problem or not. It can be argued that both the examples in Figure 2.3 and Figure 2.4 can be grouped into a larger superclass label if the given task does not require training of a model to such granularity. In both examples the classes stated are of the lowest class available in the overall hierarchy. ImageNet has labels available for each image along a larger array of classes and sub-classes. Figure 2.5 visualises the granularity possible where both images belong to the superclass animal.

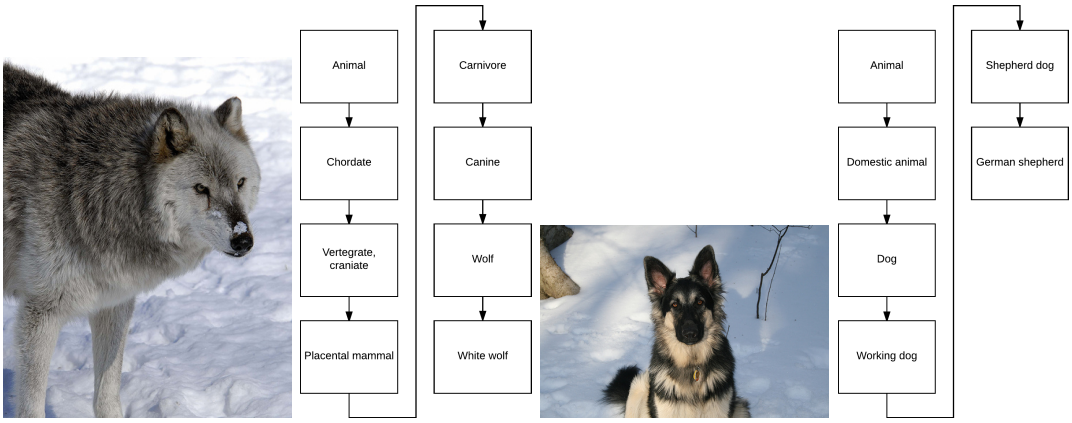


Figure 2.5: Visualisation of the hierarchy of potential classes for two examples in the ImageNet training set [2].

2.2.2 Computational-complexity and Scalability-related Challenges

The second challenge as per [4] is related to the potential scale of object detection. When deciding on which type of model to use it must be complex enough to be able to capture the previously mentioned challenges both in inter- and intra-class. On top of this there is an extremely large number of potential classes in object detection. If the aim is to train

a model to classify between an extreme number of classes then naturally a large number of images are also needed for each category. The large number of images need also to be representative enough in training to capture the necessary visual features to generalise on non-training images. In 2016 the ImageNet object detection challenge there is a total of 200 object categories, with 456,567 images comprising the training set. Therefore, a complex enough model is needed in order to learn and generalise on such a dataset but this of course places high requirements on the amount of training needed.

Issues can also arise over time when designing an object detection system. Over time the visual appearance of an object can change, which is very difficult to take into account when training a model. For example, the visual appearance of televisions have changed greatly in the past century. If a system were only to be trained on images from an earlier time period it may not be able to generalise on new instances. Therefore, it is important that a model is able to be updated as the appearance of objects change. On top of this, new categories of objects can come to fruition which may needed to be added to a model.

2.3 Implementation Outline

As per [4] the steps in the general pipeline for an object detection system is as follows:

1. Find all possible object regions in the image.
2. Determine if the regions correspond to any of the predefined categories.
3. Evaluate all responses from step 2 to determine final detections.

2.4 Related Work

One of the first methods to show that CNNs could significantly improve object detection was that of R-CNN [6]. The method obtains the name R-CNN based upon a CNN is used on regions of the image. Many earlier object detection approaches were used in a sliding window fashion testing all areas of an image. This can lead to a huge amount of potential testing windows especially if the object detection is done at a multitude of different scales. The method was heavily inspired by the AlexNet model that started the deep learning renaissance in 2012 by winning classification challenge in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC). The authors of R-CNN aimed to show that the advances in classification with a model such as AlexNet could also be done in object detection. In R-CNN the model is used as a feature extractor from which a class-specific linear Support Vector Machine (SVM) can be trained on top of. The AlexNet-based feature extractor is firstly pre-trained on a large dataset designed for classification, in this case the training set from ILSVRC 2012. This pre-trained model is then adapted to the new domain of object detection by fine-tuning the model accordingly. In this instance the authors fine-tuned warped training instances from the PASCAL VOC dataset. The AlexNet model was also altered to classify the 20 classes present in PASCAL VOC rather than the 1000 classes in ILSVRC. The pipeline of the R-CNN is split into 3 modules:

1. Region proposals.

2. Feature extraction.

3. Class-specific linear SVMs.

In this first module, region proposal, there is a large number of choices of methods to produce a suitable number of windows in comparison to a sliding window approach. R-CNN is agnostic to the region proposal method chosen and in the original work SelectiveSearch [7] is used. Module two, as explained earlier, is the use of a CNN as a feature extractor. This is in the form of a 4096-dimensional feature vector from the domain-specific PASCAL VOC trained AlexNet model. These feature vectors are used in the third module, class-specific linear SVMs. In the case of PASCAL VOC a total of 21 SVMs are trained, one for each of the 20 classes in the challenge and one for a background class. The training of the SVMs is done by forward propagating a large number of both positive and negative region proposals found with SelectiveSearch and storing each 4096-dimensional feature vector to disk. After this the appropriate labels are applied to each vector and a linear SVM is optimised for the 21 classes. At test time, for a given image SelectiveSearch is used to produce around 2000 proposals. Each of the proposals are propagated through the network to extract their respective feature vectors. Each feature vector is then tested against every SVM to produce a score for each class. Finally greedy Non-maximum Suppression (NMS) is applied to remove overlapping detections. The approach outlined in R-CNN produced a significant improvement in object detection with an improvement of roughly 13%, compared to previous state-of-the-art methods, to an overall 53.7% Mean Average Precision (mAP) on the PASCAL VOC 2010 test set. Similar results were also found on the PASCAL VOC 2011/12 set with mAP of 53.5%. Despite the significant improvements with a CNN-based method on region proposals there are still issues with the R-CNN. Firstly, the testing time per image is very slow at roughly 47 seconds on an Nvidia K40 GPU. Also extracting features for each proposal in order to train the SVMs takes a large amount of disk space and may not be feasible on all hardware configurations. Finally, as the R-CNN is made up of 3 modules the training is done in a multi-stage manner rather than end-to-end. Therefore, the loss calculating when optimising the SVMs are not used to update the CNN parameters.

The R-CNN method was improved the following year with Fast R-CNN [8] and aimed to improve speed and accuracy. One of the significant changes is that the detection training done is now end-end rather than in the multi-stage pipeline in R-CNN. Due to this the large requirements of disk space due to feature caching is no longer required. The Fast-RCNN method takes both an image and a set of pre-computed object proposals. A CNN forward propagates the entire image, rather than individual proposals in R-CNN, through several convolutional and max-pooling layers to produce a feature map. Features are extracted for each proposal in their corresponding location in the computed feature map with a Region of Interest (RoI) pooling layer. The RoI feature is calculated by splitting the $h \times w$ proposal into $H \times W$ sub-windows of size $h/H \times w/W$. Where h and w is the height and width respectively of a proposal. H and W are hyper-parameters specifying the fixed spatial extent of the extracted feature. Each sub-window has max-pooling applied and with the resulting value being placed in the corresponding output cell. Once the RoI pooling layer has been applied to a pre-computed object proposal the forward pass continues through two fully-connected layers followed by two sibling output layers. The sibling outputs are a softmax classification layer that produces probabilities for the object classes and another

Table 2.1: A comparison of R-CNN and Fast R-CNN PASCAL VOC mAP results on the test set from 2007, 2010, and 2012.

	2007	2010	2012
R-CNN	66.0	62.9	62.4
Fast R-CNN	**66.9**	**66.1**	**65.7**

Table 2.2: Speedup between R-CNN and Fast R-CNN in regards to both training and testing. Both methods are train a VGG16 network for object detection.

	R-CNN	Fast R-CNN
Train time (hours)	84	**9.5**
Train speed-up	1x	**8.8**×
Test time (seconds/image)	47.0	**0.32**
Test speed-up	1x	**146**×

layer for bounding-box regression. These two layers replace the respective external modules in R-CNN and make it possible to train the entire detection network in a single-stage. As in R-CNN, pre-training a CNN on a large classification dataset and fine-tuning towards detection and a specific object classes is done in a similar fashion. In R-CNN, the only deep network used was AlexNet [9], however, in Fast R-CNN the authors experiment with networks of different size. It was found that the deeper network VGG-16 [10] for computing the convolutional feature map gave a considerable improvement in mAP. For a fair comparison of results against R-CNN, its CNN was the same pre-trained VGG-16 network. The mAP results on PASCAL VOC 2007, 2010, and 2012 compared against R-CNN can be seen in Table 2.1.

However, as the name Fast R-CNN implies the main improvement is the speed in respect to both training and testing. By computing a convolutional feature map for an entire image rather than per object proposal the number of passes in the network is lowered significantly. Also by makign the detection training end-to-end with the two sibling layers lowers the training time needed. An overview of time spent training and testing both Fast R-CNN and R-CNN with a VGG16 CNN can be seen in Table 2.2.

While Fast R-CNN provided improvements in both accuracy and speed, the increase in speed is only in relation to the actual object detection and assumes that the region proposals are pre-computed. Therefore, there is still a significant bottleneck per image as a region proposal method can typically take a couple of seconds.

The region proposal bottleneck was addressed in the third iteration of the R-CNN network with Faster R-CNN [11]. In this method it was shown how to compute region proposals with a deep CNN with a part of the network called Region Proposal Networks (RPN). A RPN shares the convolutional layers and feature map used for computing features with RoI pooling in Fast R-CNN. As this deep network is already being computed on the entire image for the classification pipeline the added time for proposals using the RPN is negligible (10ms) in comparison to a method such as SelectiveSearch. Apart from the change in how region proposals are computed there is no change in comparison to Fast R-CNN. An RPN takes the convolutional feature map as input and returns a number of object proposals. Each proposals is fed into two sibling layers, similar to that in Fast R-CNN, one layer scoring how

likely to be an object or background and another performing bounding-box regression. The proposals are found through a method denoted as anchors. At each sliding-window location k proposals are with anchors that are user-defined reference boxes for how an object proposal may be formed. The anchors can be built based upon scale and aspect ratio altering the size. In the original work three different scales and three aspect ratios are built, yielding a total of $k = 9$ anchors at each sliding window position. These anchors are then placed on the feature map and the sibling layers calculate the likelihood of an object and regress the anchor as necessary. Once the proposals have been found with the RPN these are placed on the same convolutional feature map as earlier and the rest of the pipeline is identical to Fast R-CNN, classifying and regressing bounding-boxes with another set of sibling layers. As the only change is the addition of computing proposals in the network with RPN, the results are similar in respect to mAP. Only a slight improvement in made on PASCAL VOC 2007 and 2012, from 66.9% to 69.9% and 65.7% to 67.0% respectively. However, the main contribution to the work is the speed-up of the entire object detection pipeline as the object proposal time is now minimal. On average processing an image on PASCAL VOC 2007 with an Nvidia K40 with Fast R-CNN including proposals took 2 seconds per image. While in Faster R-CNN with the same hardware takes 0.2 seconds per image. A speed-up of 10× from Fast R-CNN to Faster R-CNN and 250× from the original R-CNN. The Faster R-CNN methods has also proved to be the foundation for the winning entry in multiple detection challenges including MS COCO. The results for this challenge with a VGG-16 model for Fast R-CNN were 35.9% mAP@0.5 Intersection-Over-Union (IoU) and 19.7% mAP@[0.5, 0.95]. Faster R-CNN improved this to 42.7% mAP@0.5 and 21.9% mAP@[0.5, 0.95].

Much of the recent work within object detection has been based upon the Faster R-CNN framework. This is exemplified by looking at the MS COCO detection leaderboard, with 15 of the 21 approaches being Faster R-CNN related in some way. Firstly, the winner of the MS COCO 2015 and ILSVRC 2015 detection challenge was with the use of deep residual networks [12]. As is well known with CNNs, deeper networks are able to capture richer higher-level features and the authors showed that this is also beneficial in the object detection domain. In [12] an ensemble of three deep residual networks with 101 layers was trained for object detection and another ensemble of three used for region proposals with the RPN while being based on the Faster R-CNN framework. In addition to the ensemble, the winning entry also added box refinement, global context, and multi-scale testing to the Faster R-CNN.

The current leading method on MS COCO is an extension of the previously explained [12]. This method dubbed G-RMI on the MS COCO leaderboard [13] is an ensemble of five deep residual networks based upon ResNet [12] and Inception ResNet [14] feature extractors. No work has been published yet on G-RMI, however, a short explanation of the entry is included in a survey paper [15] from the winning authors. The approach was to train a large number of Faster R-CNN models with varying output stride, variations on the loss function, and different ordering of the training data. Based upon the collection of models, five were greedily chosen based upon performance on a validation set. While performance on the models were important, the models were also chosen such that they were not too similar. It should also be noted that apart from the ensemble of models, the entry did not include any extras such as multi-scale training, box refinement, or global context.

Another variant is that of MultiPath [16], placing second in MS COCO 2015. Which aimed to address the many small objects present in MS COCO by modifying Fast R-CNN. Firstly, rather than only having a single classification head, MultiPath has four. Each classification head observes different scaled regions around the RoI which aims to add context around the object. The output of each of the four are concatenated for classification and regression. Also MultiPath uses skip connections. The RPN and consequent RoI-pooling in Faster R-CNN and Fast R-CNN is only performed once at a number of convolutional layers. At which point the input image has been down-sampled a number of times, therefore, small objects are potentially not represented very well any more. With the use of skip connections higher-resolution features from earlier convolutional layers can be added giving the RPN and classier information about smaller objects.

Inside-Outside Net (ION) [17] also adds contextual and multi-scale information on top of Fast R-CNN. ION was the third place entry in MS COCO 2015. The multi-scale information is also added with skip connections. Whereas global context is added through the use of Recurrent Neural Networks (RNNs) passing information about the image both vertically and horizontally.

Global context has also been added to the Faster R-CNN framework in [18] with the use of semantic segmentation as a form of top-down information. A segmentation module is augmented onto the framework and is calculated using the same initial convolutional layers as Faster R-CNN. The segmented result is then added after the RPN and RoI-pooling is performed on both the convolutional layers and corresponding RoI segmentation area.

Additional work on adding finer details for smaller objects with Faster R-CNN was performed in [19] who aimed to improve skip connections with additional top-down information. While skip connections is useful method for finding higher-resolution features, the authors argue that with Top-down Modulation (TDM) features are taken from an appropriate lower layer. TDM is incorporated into the Faster R-CNN framework and can be trained along side it.

The use of hard example mining was conducted in [20]. In this work the problem of small objects in ILSVRC and MS COCO is also addressed. The authors present a method for training a Fast R-CNN object detector for these objects with Online Hard Example Mining (OHEM). Inspired by bootstrapping, with OHEM a modification is made to Stochastic Gradient Descent (SGD) training by selecting RoIs that the network currently has a high loss on and backpropagating accordingly.

The methods covered so far have all followed a region-based paradigm of first finding a selection of object proposals and second classifying these into one of the appropriate classes while also regressing bounding-boxes. These methods can be computationally expensive and therefore recent work has attempted to combine the two steps into a single feed-forward CNN. These methods can be denoted single shot object detectors, with he most recent approach is that of Single Shot Detector (SSD) [21] and is the first deep approach that does not resample pixels of features to perform object detection such as the convolutional feature

maps and RoI pooling in region-based methods [8] [11]. Rather convolutional feature layers are added to the end of a network and a small convolutional filter (predictor) is applied on these for simultaneous localisation and classification. The truncated layers become progressively smaller and allow SSD to find objects at multiple scales. The predictors used on these layers are of pre-determined size and aspect ratio similar to the anchor boxes used in Faster R-CNN [11]. The additional layers and predictors can be added to any classification-based CNN and SSD test using the standard VGG-16 network. On top of this the authors train two separate instances of the network, one for low-resolution input SSD300 (300×300) and one for high-resolution SSD512 (512×512). Overall the higher-resolution network performs best with 1-2% improvements in comparison to Faster R-CNN on PASCAL VOC 2007 and MS COCO test-dev 2015. In terms of speed there is a considerable difference, the authors found Faster R-CNN on average took 0.14 s/image, SSD300 0.02 s/image, and SSD512 0.05 s/image on PASCAL VOC 2007 testing with a Titan X GPU.

One of the original single shot CNN-based methods for object detection was OverFeat [22] with a sliding-window approach. Methods such as OverFeat have not recently been as popular due to deeper and more powerful networks being too computationally expensive to be run across the entire image at multiple scales. However, at the time OverFeat won the ILSVRC 2013 localisation challenge using an altered AlexNet [9] CNN. The main alteration was a regression layer for added accuracy in localisation.

A precursor to SSD was that of MultiBox [23] and the improved version in [24]. Again the goal was to directly predict the bounding-box of an object directly with a CNN for a given class. The MultiBox method was originally designed to prove that object proposals with a CNN could be an improvement of hand-engineered methods such as SelectiveSearch [7] in R-CNN [6] and Fast R-CNN [8]. MultiBox is similar to the RPN in Faster R-CNN [11] where object locations are predicted on a grid with a number of default predictions of different sizes. Additionally, MultiBox ranks the proposals according to a loss in relation to both the confidence of being an object and location of the bounding-box. With this ranking MultiBox is able produce and classify only 15 proposals per image while being competitive to other methods such as R-CNN at the time.

Another CNN method that only uses a single network is You Only Look Once (YOLO) [25] and it's successor YOLOv2 [26]. In the original YOLO method, a single shot approach was taken by producing bounding box locations and class scores from a fixed grid in an image. Various combinations of these grids are used as potential bounding-boxes. However, in YOLO the accuracy of the localisation was poor and this was addressed in YOLOv2 with a number of changes. Firstly, the RPN from Faster R-CNN [11] was adapted with this use of anchor boxes. But rather than using pre-determined anchor sizes k-means clustering is run on the training set to determine what appropriate sizes. Also to address the issue of small objects not being represented in the convolutional feature map in the RPN, additional features are added from an earlier convolutional layer. Other improvements to YOLOv2 include batch normalisation, multi-scale training, and high-resolution inputs. Overall the method produces competitive results against approaches such as Faster R-CNN and SSD. But the main improvement is in speed, where at inference time is up to 182× and 5× faster than Faster R-CNN and SSD respectively.

Recently a newer approach to region-based methods, such as Faster R-CNN, has been proposed with the use of Fully Convolutional Networks (FCNs). The authors argue that in region-based methods the act of cropping features from RoIs in the same layer adds an unnatural condition. There has been an issue in the two step pipeline in region-based methods, as the classification is translation-invariant, whereas detection in translation-variant. Due to this difference region-based methods have been adjusted towards the invariant properties of classification by pooling features and classifying them. However, [27] argue that translation-variant representations are important in object detection as the position of an object inside a RoI can provide meaningful information. Therefore, [27] present their fully convolutional approach with Region-based Fully Convolutional Network (R-FCN). The overall approach is similar to that used in region-based methods such as [6], [8] and [11]. First compute RoIs using a region proposal method and second perform classification on these regions. R-FCN uses the RPN from Faster R-CNN [11] for class-agnostic RoI computation. However, rather than extracting features with RoI-pooling, fully convolutional position-sensitive score maps are computed. The score maps are split up to represent a relative position in a $k \times k$ grid, with each cell presenting information relative to the spatial position of an object. For example, the upper-left cell represents scores that pixels are present at that relative position to the object. A bank for position-sensitive score maps are found for each class, generating a total of $k^2(C+1)$ where C is the number of classes plus a background class. After computing the bank, the R-FCN computes a position-sensitive RoI-pooling layer for each class. For each RoI found with the RPN each cell aggregates the response from the appropriate score map from the bank of maps. While the ordering and methodology of RoI-pooling is different in R-FCN to that of Faster R-CNN the same backbone CNN can be used. In this experiments conducted by the authors a ResNet-101 network is chosen. Overall R-FCN is an improvement on the Faster R-CNN approach on benchmarks such as PASCAL VOC and MS COCO. It is also competitive with the MS COCO 2015 winning entry [12], while not having any additions such as global context or iterative box regression. Additionally it is considerable faster in training and testing.

The use of FCNs are currently the leading method for segmentation both semantic [28] and instance [29]. With the latter, named Fully Convolutional Instance-aware Segmentation (FCIS) winning the 2016 MS COCO instance segmentation challenge and also is the current second place in detection. It uses a similar approach with position-sensitive score maps for pixel-level likelihood for an object category to produce bounding boxes. From these, instance segmentation is performed to produce the pixel-level classification. The main differences between R-FCN and FCIS is the addition of ensemble of ResNet models, multi-scale testing and training, and horizontal flipping.

3 Technical Analysis

3.1 Deep Learning

3.2 Residual Networks

After the impressive advances of various challenges with ResNets in 2015 the use of these have become a standard for object detection systems, with many of the entries in MS COCO and ILSVRC being based upon ResNets. The intuition of deep neural networks is that as the model becomes deeper richer representations of the original input are found. However, as shown in the work of ResNets [12], it is difficult to train deeper versions of commonly used CNN architectures. Before the appearance of ResNets one of the standard architectures were VGG nets [10], consisting of 16 to 30 layers. The intuition of deeper architectures lead to better networks was explored in [12] by stacking additional layers and creating a 56 layer CNN. It was found that stacked deeper networks have a degradation problem and the networks converge to a testing error higher than the corresponding shallower networks. A hypothesis could be made that this is due to a deeper network overfitting the dataset, however, it was determined that this was not the case as deeper models also exhibit higher training error. The solution to this degradation problem was to to construct deeper models using identity mapping and residuals, dubbed a deep residual learning framework. With this framework a given layer learns a residual mapping between the previous layer output and operations on the output. Using this reformulation the training error in the current layer should be no greater that the previous. This core concept of a residual block is visualised in Figure 3.1. The input x is passed into the block where a mapping is computed with two weight layers with convolutional operations with a Rectified Linear Unit (ReLU) operation between them, representing an alteration with $F(x)$. After this the original input (identity) is added through a shortcut connection to the mapping by $F(x)+x$. This formulation forces the convolutional layers to compute weights to learn the residual mapping $F(x)$.

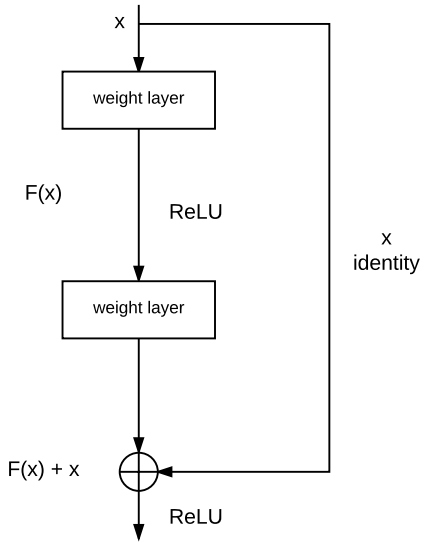


Figure 3.1: Core concept of residual blocks used in ResNets.

The formulation of $F(x) + x$ requires that the dimensions of F and x are equal. In situations where this is not the case a linear projection $W_s x$ is performed on x such that the dimensions match. The final formulation of a residual block is:

$$y = F(x, W_i + W_s x) \tag{3.1}$$

where W_i are the weights of the convolutional layers.

In the original work, experiments on a number of different architectures with residual blocks are conducted. Firstly, ImageNet classification is evaluated using naively stacked plain networks and ResNets. The plain networks are inspired by VGG nets [10] with two design criteria. Firstly, for a given output feature map size the number of filters must be equal. Secondly, if the feature maps size is halved the number of filters in the layer are doubled. The ResNets are variants of these plain networks but with residual connections in each block. In order to perform a fair comparison the linear projection performed in ResNets when dimensions are altered is done with zero-padding so that no extra parameters are added. Both sets evaluated are with 18 and 34 layers. An example of plain and ResNets can be seen in Figure 3.2.

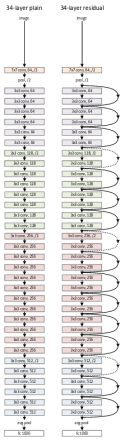


Figure 3.2: PLACEHOLDER. Overview of 34 layer plain and residual networks.

On top of the use of residual blocks, ResNets are also trained with scale augmentation and batch normalisation. During inference, multi-scale testing is conducted. Results on ImageNet validation top-1 error showed that the use of residual blocks aided in the optimisation of deeper architectures. Table 3.1 shows that the deeper plain networks exhibited troubles in optimisation with increased error with a deeper network. However, ResNets provided a decrease of 2.85% between the two respective architectures.

Table 3.1: Top-1 error(%) on ImageNet validation set.

	plain	**ResNet**
18 layers	27.94	27.88
34 layers	28.54	25.03

Haven shown that ResNets aid in optimisation of deep networks, the authors experimented with even deeper networks of 50, 101, and 152 layers. Due to concerns in the

Table 3.2: Results of various deep ResNet architectures on ImageNet validation set.

Method	**top-1 error (%)**	**top-5 error (%)**
ResNet-34	21.84	5.71
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	19.38	4.49

training time the residual blocks are altered in comparison to that shown in Figure 3.1. The new block shown in Figure 3.3, F consists of 3 convolutional layers of size $1 \times 1, 3 \times 3$, and 1×1. The two sets of 1×1 layers are used to reduce complexity by reducing the input to the 3×3 layer and restoring the resulting output.

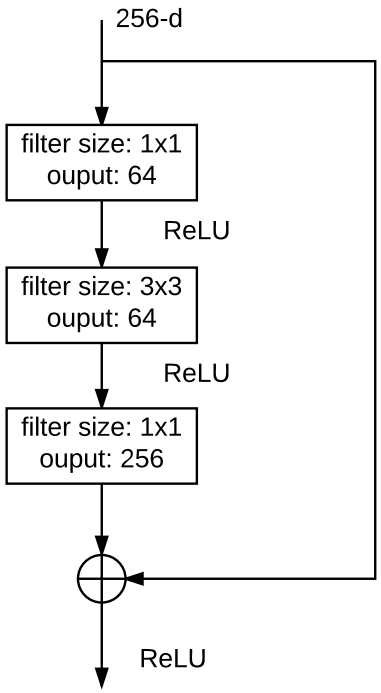


Figure 3.3: Residual block used in deeper ResNet architectures.

The deeper ResNets prove to give impressive results on the ImageNet validation sets as seen in Table 3.2, with the very deep ResNet-152 providing the lowest error.

3.3 Benchmark Datasets

This section will outline some of the commonly used object detection datasets. This will include their purpose and the general setup.

3.3.1 PASCAl Visual Object Classes Challenge

The PASCAL VOC challenge [1] was held yearly between 2005 to 2012 and provided datasets for benchmarking within vision tasks of visual object category recognition and detection. Between that time period is was considered the top benchmark for the respective challenges. While being an annual competition, PASCAL VOC evaluation in state-of-the-art literature is most often performed on data from the years 2007 and 2012. The competition saw a large shift in the former year as the number of classes increased from 10 to 20, in turn also significantly increasing the total amount of data. Additional data was added individually for the various recognition tasks between 2007 and 2012 and the performance metric was altered slightly between this time period, however, the overall ecosystem remained largely the same from 2007 until the competition's end. This section will be largely based upon the two retrospective papers, [30] and [31], published by authors who were involved in the challenge and for the most part will be in respect to the challenge after 2007.

Images were obtained from the website flickr [32] with the aim in mind to collect natural images for the recognition challenges. Ideally the dataset was to contain a significant level of visual variability in regards to object size, orientation, pose, illumination, position, and occlusion. A total of 20 classes were present in the 2007 challenges and these remained the same until 2012. The classes can be considered as a part of a taxonomy with 4 main branches, where each has finer-grain objects in sub-classes. The 20 classes and the branching taxonomy can be seen in Table 3.3.

Table 3.3: Taxonomy of the 20 classes introduced in VOC2007.

Vehicles	Household	Animals	Other
Aeroplane	Bottle	Bird	Person
Bicycle	Chair	Cat	
Boat	Dining table	Cow	
Bus	Potted plant	Dog	
Car	Sofa	Horse	
Motorbike	TV/Monitor	Sheep	
Train			

A total of 500,000 potential images were collected randomly based upon different combinations of queries for a given class. For example of class bird, potential queries were bird, birdie, birdwatching, nest, sea, aviary, birdcage, bird feeder, and bird table. Of these potential images the majority were discarded for potential annotation due to not meeting the considerations of visual variability mentioned earlier. The annotation process was completed by a team from the University of Leeds based upon strict guidelines. The aim was to ensure that the annotations resulted in a consistent, accurate, and exhaustive dataset. The annotations are stored in XML format which contains the following information:

- Class: one of the 20 shown in Table 3.3.
- Bounding box: axis-aligned bounding-box around the visual extent of the object.
- Viewpoint: viewpoint to the object.
- Truncation: whether or not object is truncated. An object is truncated when the

bounding-box does not cover the full extent of the object. Truncation can occur if the object extends outside the image or is partially occluded.
- Difficult: A subjective evaluation on if the object is difficult to detect. This is determined based on object size, illumination, or image quality.

An example of the XML format for the object chair can be seen in Code 3.1 and its corresponding image in Figure 3.4.

Code 3.1 Example of XML annotation for the object chair.

```
<object>
    <name>chair</name>
    <pose>Rear</pose>
    <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
                <xmin>263</xmin>
                <ymin>211</ymin>
                <xmax>324</xmax>
                <ymax>339</ymax>
        </bndbox>
</object>
```

Figure 3.4: Image from the PASCAL VOC 2007 dataset. The bounding box represents the annotated XML data shown in Code 3.1.

Of the 500,000 potential images, 9,963 were annotated for the VOC2007 challenges based upon the annotation guidelines. PASCAL VOC datasets is split into two subsets; `trainval`, consisting of training and validation data and `test`, consisting of the testing data. A histogram showing the frequency of an object class in an image and the total number of objects for the VOC2007 dataset can be seen in Figure 3.5.

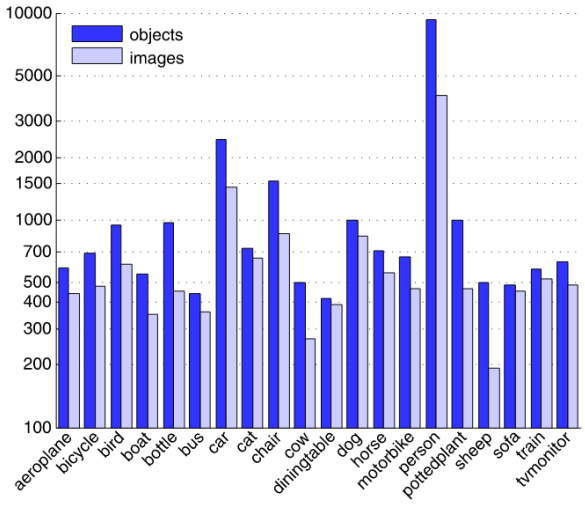


Figure 3.5: Image from the PASCAL VOC 2007 dataset. The bounding box represents the annotated XML data shown in Code **??**.

3.4 Evaluation Metrics

3.5 Object Detectors

This section will perform a technical analysis of the current primary CNN-based object detectors.

3.5.1 Faster Region-Convolutional Neural Network

A primary CNN-based object detector over the previous years has been Faster R-CNN [11] and it's predecessors, Fast R-CNN [8] and R-CNN [6]. As mentioned in Section *2.4 Related Work*, 15 of the 21 current entries in MS COCO is Faster R-CNN based. The general method of Faster R-CNN can be split into two parts, region proposals and region classification. Region proposals aims to reduce the amount of windows that need to be tested at inference time in comparison to the previously often used sliding window approach. Rather than testing a plethora of potential object window locations, scales and aspect ratios, region proposals finds a lower number of windows that are likely to contain an object. On top of reducing the total number of testing windows, it also allows for using more expensive classification techniques such as CNNs. There are a large number of different methods to compute region proposals, however, Faster R-CNN has one leading current methods being used in multiple other methods. The proposals are efficiently computed with a RPN, as proposals are found directly in the network, sharing convolutional layers with the classification step. The framework of the Faster R-CNN can be seen in Figure 3.6.

update faster rcnn framework figure

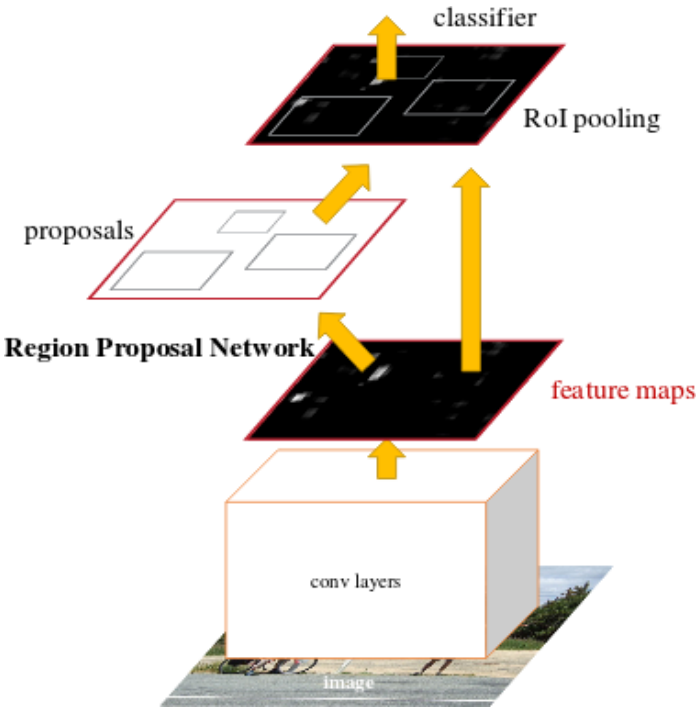


Figure 3.6: PLACEHOLDER. Faster R-CNN framework. A CNN computes a feature map from which a RPN finds region proposals. Given these proposals and the same feature map proposals are classed accordingly.

There a number of different possible CNN models that can be used the compute feature maps through the convolutional layers. In the original Faster R-CNN VGG nets [10] were experimented with. However, since then more efficient and accurate networks have come forward, with one of the most popular being the ResNet [12] architecture. Standard practice is to pre-train the network for classification on Imagenet followed by fine-tuning it towards object detection. Independent of the chosen network the RPN takes as input the last feature map from the convolutional layers of the network. A small network traverses the feature map which feeds the result into two sibling fully-connected layers, a box-classification layer and a box regression layer. The box-classification layer classifies the RoI into either an object or background, with a probability being associated to each. While the latter box-regression layer attempts to fit the bounding-box to the object of interest. In order to take into account different scales and aspect ratios of objects in the feature map the RPN uses a set of pre-defined RoIs at each sliding window location. These pre-defined RoIs are denoted as anchors. At each sliding window location a maximum of k possible region proposals can be computed based upon the k anchors. The anchors are user-defined into different sizes and aspect ratios. A default setting for the anchors is a total of $k = 9$, which corresponds to 3 scales and 3 aspect ratios. The computation of the k anchors at sliding window location followed by the sibling layers is visualised in Figure 3.7.

update rpn framework figure

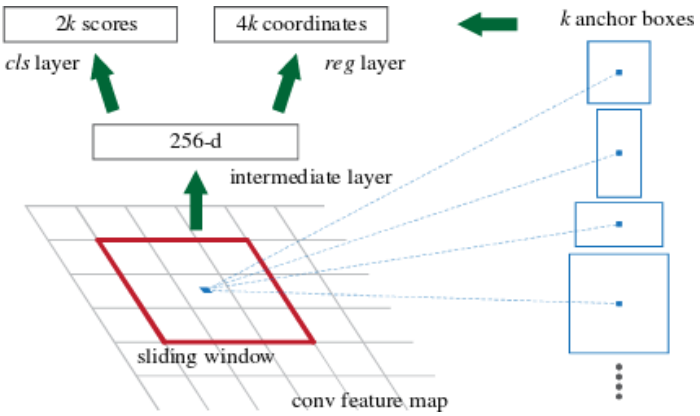


Figure 3.7: PLACEHOLDER. RPN framework. The k anchor boxes are placed at each sliding window location on the last feature map. The RPN uses two sibling layers to compute the classification of object or background and perform bounding-box regression.

Given the set of region proposals from the RPN, objects are classified in the $C+1$ categories based upon the same approach as in Fast R-CNN. Features are cropped for each proposal are their respective location from the same feature map given to the RPN. Features are computed using a RoI pooling layer that uses max pooling to convert the cropped area into a pooled map of fixed spatial extent ($H \times W$), where H and W are hyper-parameters. In order convert each RoI into a fixed max pooled size, the RoI of size $h \times w$ is split into a grid of $H \times W$, with each sub-window being of size $h/H \times w/W$. Max pooling is then applied at each sub-window and placed accordingly in the $H \times W$ pooled layer. Following the RoI pooling layer, two fully-connected layers feed sibling layers into a classification layer and a box-regression layer, similar to those in the RPN. However, in this instance the classification layer computes the probabilities for each of the $C+1$ classes.

3.5.2 Regional Fully-Connected Network

One of the current leading object detection methods is the R-FCN [27], which as mentioned in Section *2.4 Related Work*, takes a different approach to that of the region-based methods such as Faster R-CNN. The authors of R-FCN were inspired by the recent advances in FCN classification networks, such as ResNets, and argue that the addition of the RoI-pooling layer in the Faster R-CNN pipeline is unnatural and adds computational complexity. The authors hypothesise that the reasoning behind this addition is due to the trade-off between using a classification approach in an object detection pipeline. A defining factor in object detection is that the method should be able to respect translation variance, that translation of an object inside an object proposal should given a good indication as to how well the proposal fits the object. Whereas classification is more translation invariant, as the shifting of an object in an image does not effect how the system returns it's output. The use of the RoI-pooling layer placed in between convolutional layers means that any convolutions after this point are not translation invariant as it is not region specific. Rather than using this popular feature extractor, R-FCN uses position-sensitive score maps computed by a bank of convolutional layers. The maps add translation variance into the detection pipeline by computing scores in relation to position information with respect to the relative spatial position of an object. A RoI-pooling layer is added after the score-maps, however, no con-

volutional operations are done after this point ensuring translation variance.

The overall approach of the R-FCN also consists of the popular two-stages of region proposal and region classification. Region proposal is done using the RPN from Faster R-CNN followed by the position-sensitive score maps and RoI pooling for region classification. The overall architecture of the R-FCN can be seen in Figure 3.8.

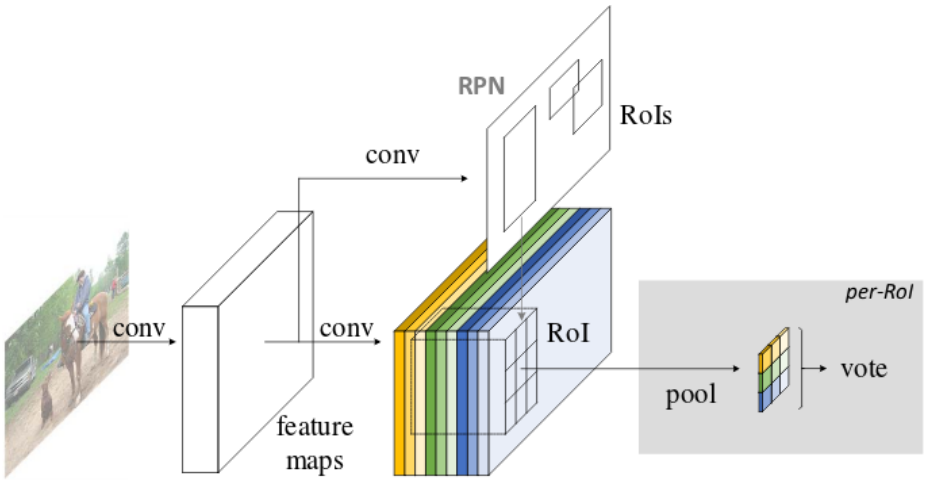


Figure 3.8: Architecture of R-FCN. Region proposals are found using the RPN followed by classification based on a bank of position-sensitive score maps.

The added translation variance post finding proposals with the RPN is done by producing a bank of k^2 score maps for each object category. Therefore, there are a total of $k^2(C+1)$ maps, where C is the number of object categories plus one for a background class. The number of k^2 maps is due to a $k \times k$ spatial grid representing relative positions. Typically $k = 3$, therefore, the nine score maps represent position-sensitive scores for a given object category.

update score maps figure

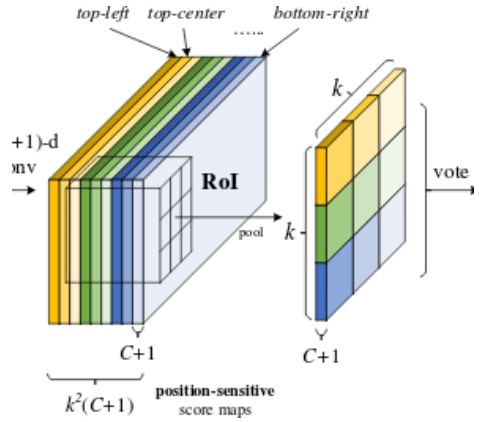


Figure 3.9: PLACEHOLDER. Score maps.

Once the bank of score maps have been computed, position-sensitive RoI-pooling is

found for region classification. Each individual $k \times k$ bin pools from its corresponding location in the relevant score map. For example, the top left bin pools from that position in the top-left score map and so on. The RoI-pool is computed using average pooling for each bin which can be seen in Figure 3.10. The final decision for a given class is determined by a vote where each of the bins are averaged, producing a $(C+1)$-dimensional vector for each RoI.

update score maps figure

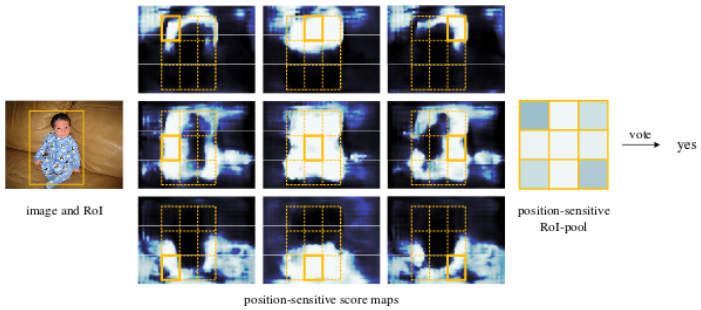


Figure 3.10: PLACEHOLDER. Position-sensitive RoI-pooling operation for a given class.

3.5.3 You Only Look Once

YOLOv2 [26] is one of the current best performing single shot detectors, with results on par with more commonly used object detectors while being considerably faster at test time. YOLOv2 uses a different approach than the common 2-step method of region proposal and region classification seen in Faster R-CNN and R-FCN by directly computing class probabilities on each RoI. Some of the main distinct difference between YOLOv2 and region-based methods is the use of directly predicting bounding boxes, using a modified model, and altering how the priors for anchor boxes are computed during region proposals with the RPN. The distinct differentiator for YOLO and YOLOv2 is that bounding boxes for a given object are predicted directly rather than predicting offsets to anchors with the RPN. This is done by splitting the image into $S \times S$ grid cells, with each cell predicting B bounding boxes. Each of the B boxes predict a total of 5 values: $[t_x, t_y, t_w, t_h, t_o]$. Where t_x, t_y are the coordinates of the centre of the given cell. The values t_w, t_h are the width and height relative to the entire image. Finally, t_σ is the confidence of how well the predicted box fits the ground truth. The location of the bounding box is determined by these values with respect to a given cell and the offset of the cell from the top left corner of the image (c_x, c_y) and the size of the anchor box is p_w, p_h. Then the bounding box predictions are calculated by:

$$\begin{aligned} b_x &= \sigma(t_x) + c_x \\ b_y &= \sigma(t_y) + c_y \\ b_w &= p_w e^{t_w} \\ b_h &= p_h e^{t_h}. \end{aligned} \tag{3.2}$$

Finally, the probability that the given bounding box fitting given the probability of their being an object is:

$$Pr(object) * IoU(b, object) = \sigma(t_o). \tag{3.3}$$

Each of the S^2 cells predicts C conditional probabilities of it containing a given class and also being object by $Pr(Class_i|Object)$. With the predicted bounding boxes and class probabilities calculated for each cell the final detections can be determined by adjusting a threshold based upon the calculation:

$$Pr(Class_i|Object) * Pr(object) * IoU(b, object). \tag{3.4}$$

This process of using grid cells, bounding box prediction, cell class probabilities and final detections can be seen in Figure 3.11.

update figure

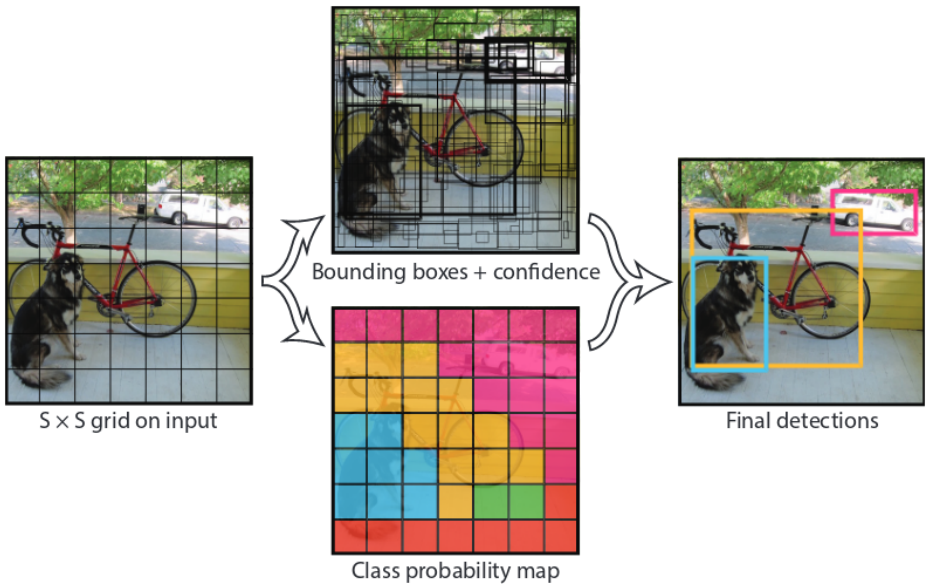


Figure 3.11: PLACEHOLDER. YOLO model.

As mentioned region proposals are found using the RPN from Faster R-CNN [11]. However, instead of using hand-picked priors for the anchor boxes, YOLOv2 proposed a method to learn more suitable sizes and aspect ratios. This is done by running k-means clustering on the annotated bounding boxes from the training set using a custom distance measurement. The custom measurement replaces Euclidean distance as these distances would create a bias due to more error on likely occurring on larger anchors. The custom distance measurement is designed for favourable IoU scores and is as follows:

$$d(box, centroid) = 1 - IoU(box, centroid) \tag{3.5}$$

where *box* is the ground truth bounding box from the training set and *centroid* is the predicted anchor box. By learning the priors YOLOv2 is able to use five anchor boxes at the same level of recall as the nine used in a typical RPN.

YOLOv2 also goes against the grain in comparison to other state-of-the-art object detectors in regards to the choice of classification model. Rather than using the common networks

such as VGG or ResNets, YOLOv2 propose their own 19 layer model dubbed Darknet-19. The model is of similar paradigm to VGG nets in that it uses mostly 3×3 convolutions and doubles the number of channels after pooling, which is also present in ResNets. But it is of considerably lower complexity than VGG-16 which consists of 15.3 billion Floating Point Operations (FLOPs), with only 5.58 billion FLOPs. The baseline model has competitive results on ImageNet which can be seen in Table 3.4. The baseline can be improved using standard data augmentations and also by initially training on 224×224 images followed by fine-tuning on 448, this is also shown as Darknet-19++ in Table 3.4.

Table 3.4: ImageNet classification results for the Darknet-19 model.

Model	top-1 error (%)	top-5 error (%)
Darknet-19	27.1	8.8
Darknet-19++	23.5	6.7

To aid in the detection of small objects the Darknet-19 model is also pre-trained on high-resolution images from ImageNet prior to training for object detection. Also fine-grained features are passthrough from an earlier layer when performing prediction. This gives features from a 26×26 feature map instead of the 13 size at the RPN. Finally multi-scale training is also performed.

3.5.4 Benchmark Results

This section will outline the results of the aforementioned CNN-based object detectors on leading benchmarks such as PASCAL VOC and MS COCO. This includes results on the methods with different combinations of CNN models, training data, and other bells and whistles such as multi-scale training. All results are taken from the respective authors papers.

PASCAL VOC

A summary of the results on the test set of PASCAL VOC 2007 can be seen in Table 3.5. The first column denotes which method is used while also stating the underlying CNN model, for example VGG-16 or ResNet-101. Improvements to some of the baseline methods are also included in the first column if relevant. The improvements are online hard example mining (OHEM), multi-scale training (MSTR), multi-scale testing (MSTE), box refinement (BR), and global context (GC). Training data used in the various methods include the train set of PASCAL VOC 2007 (07), train set of PASCAL VOC 2012 (12), and trainval set from MS COCO (COCO). In entries when COCO is included, the detector is first trained on COCO followed by fine-tuning on 07+12. The best mAP result for each combination of training data is shown in bold.

A clear initial improvement is the use of ResNet-101 in comparison to the VGG-16 network, both with Faster R-CNN and R-FCN. ResNet-101 gives a mAP improvement of 3.2% in this scenario, from 73.2% to 76.4%. This was improvement was clear to the authors of R-FCN and therefore the only network used in their work is ResNet-101 for object detection. A small improvement of 0.2% can also been seen for R-FCN over Faster R-CNN when using ResNet-101, both with and without OHEM. The best performing detector with

Table 3.5: PASCAL VOC 2007 results.

method	**training data**	**mAP (%)**
Faster R-CNN VGG-16 [11]	07	69.9
Faster R-CNN VGG-16 [11]	07+12	73.2
Faster R-CNN ResNet-101 [12]	07+12	76.4
Faster R-CNN ResNet-101 OHEM [12]	07+12	79.3
R-FCN ResNet-101 [27]	07+12	76.6
R-FCN ResNet-101 OHEM [27]	07+12	79.5
R-FCN ResNet-101 OHEM/MSTR [27]	07+12	**80.5**
YOLOv2 288 × 288 DarkNet-19 [26]	07+12	69.0
YOLOv2 544 × 544 DarkNet-19 [26]	07+12	78.6
Faster R-CNN ResNet-101 BR/GC/MSTE [12]	COCO+07+12	**85.6**
R-FCN ResNet-101 OHEM/MSTR [27]	COCO+07+12	83.6

07+12 training data is R-FCN with both OHEM and MSTR (80.5%), indicating that the addition of multi-scale training improves the result by 1%. YOLOv2 scores slightly lower that R-FCN and Faster R-CNN. The best performing YOLOv2 network is trained to inputs of resolution 544 × 544, resulting in a mAP of 78.6. When using the method of training on COCO followed by fine-tuning on 07+12, Faster R-CNN with ResNet-101 and BR/GC/MSTE scoring 85.6%. However, it is difficult to directly compare methods in this instance as the other method that uses the same training data, R-FCN with ResNet-101 and OHEM/MSTR, uses different variants of additions to the overall method. However, a general trend is that the training scheme of COCO+07+12 results in a significant improvement, with the comparable R-FCN method improving 3.1%.

Similar results can be seen on the PASCAL VOC 2012 testing set, shown in Table 3.6. A general standard for training on this test set is to include both the trainval and test from PASCAL VOC 2007 and 2012 trainval, denoted as 07++12. As in the 2007 test set, Faster R-CNN is improved with the deeper features from ResNet-101 by 3.4% when training with 07++12. The best result using the training set of 07++12 is with R-FCN with OHEM/MSTR, improving upon Faster R-CNN with ResNet-101 by 3.8%. However, again it is difficult to compare due to the additions of OHEM and MSTR. The high-

resolution version of YOLOv2 is again a number of percentage points behind resulting in 73.4%. The best result is again of Faster R-CNN with ResNet-101 and BR/GC/MSTE when using COCO+07++12 as the training data with 83.8%. R-FCN with ResNet-101 and OHEM/MSTR is similarly behind as in the 2007 test, scoring 1.8% lower.

Table 3.6: PASCAL VOC 2012 results.

method	**training data**	**mAP (%)**
Faster R-CNN VGG-16 [11]	12	67.0
Faster R-CNN VGG-16 [11]	07++12	70.4
Faster R-CNN ResNet-101 [12]	07++12	73.8
R-FCN ResNet-101 OHEM/MSTR [27]	07++12	**77.6**
YOLOv2 544×544 DarkNet-19 [26]	07++12	73.4
Faster R-CNN VGG-16 [11]	COCO+07++12	75.9
Faster R-CNN ResNet-101 BR/GC/MSTE [12]	COCO+07++12	**83.8**
R-FCN ResNet-101 OHEM/MSTR [27]	COCO+07++12	82.0

MS COCO

The final benchmark dataset to be compared is MS COCO. The results for this benchmark is more comprehensive than that shown in the PASCAL VOC challenge. mAP is calculated across a number of different IoUs. Also ground truths are split into three difference categories depending on the size of the object, denoted as either small, medium, or large. Training and testing data is done in two separate ways. Firstly, training is done on the train set of MS COCO, followed by testing on the validation set val. Secondly, training can be done on a combination of the aforementioned train and val (trainval), followed by testing on the test-dev set. The main results for the object detectors can be seen in Table 3.7. A final testing set is also used for the YOLOv2 method, denoted trainval35k. Which is made up of the same images in trainval, however, 5,000 are removed for other validation purposes.

As in the PASCAL VOC challenges, baseline R-FCN performs better than Faster R-CNN with ResNet-101, with Average Precision (AP)@.5 scoring 0.5% and AP@[.5, .95] 0.4% higher. R-FCN with MSTR gives the best result when using the training set of MS COCO only. Interestingly this best result is not consistent when comparing AP across the three object scales. For small object R-FCN with ResNet-101 is best at 8.9%, slightly above R-FCN with ResNet-101 and MSTR (8.8%). However, the later method is best for medium sized objects at 30.8%, 0.3% better than the next best method. Faster R-CNN with ResNet-101 performs best for large objects with 45.0%, 2.8% higher than the

Table 3.7: MS COCO test-dev results.

method	**training data**	**test data**	**AP @.5**	**AP @ [.5, .95]**	**AP small**	**AP medium**	**AP large**
Faster R-CNN VGG-16 [11]	train	val	41.5	21.2	-	-	-
Faster R-CNN ResNet-101 [12]	train	val	48.4	27.2	6.6	28.6	**45.0**
R-FCN ResNet-101 [27]	train	val	48.9	27.6	**8.9**	30.5	42.0
R-FCN ResNet-101 MSTR [27]	train	val	**49.1**	**27.8**	8.8	**30.8**	42.2
Faster R-CNN VGG-16 [11]	trainval	test-dev	42.7	21.9	-	-	-
Faster R-CNN ResNet-101 BR/GC/MSTE [12]	trainval	test-dev	**55.7**	**34.9**	**15.6**	**38.7**	**50.9**
R-FCN ResNet-101 [27]	trainval	test-dev	51.5	29.2	10.3	32.4	43.4
R-FCN ResNet-101 MSTR [27]	trainval	test-dev	51.9	29.9	10.8	32.8	45.0
R-FCN ResNet-101 MSTR/MSTE [27]	trainval	test-dev	53.2	31.5	14.3	35.5	44.2
YOLOv2 DarkNet-19 [26]	trainval35k	test-dev	44.0	21.6	5.0	22.4	35.5

next best result, despite being considerably worse performing in the small and medium sized objects. When using the trainval set for training and test-dev for testing the best performing method is again Faster R-CNN with ResNet-101 and BR/GC/MSTE across all AP modes. Again comparison is difficult as the methods do not all have the same additions to their baselines. R-FCN with ResNet-101 and MSTR/MSTE is competitive to the best result. According to the authors of the best method [12], the additions of box refinement gives roughly 2% improvement, while global context gives about 1%. This could account for the 2.5% difference in AP@.5. Finally YOLOv2 with DarkNet-19 performs considerably worse on MS COCO. This is especially present on smaller objects scoring 5.0% AP.

4 Implementation

4.1 Resolution-Aware Object Detection

Object detectors are commonly more accurate on objects that cover a larger number of pixels in an image. This is intuitive as objects with a lower resolution objectively have less details that can describe them. The poorer performance can be seen in Table 3.7, for all object detectors the AP is considerably lower for smaller objects in comparison to both medium and large. The best performing detector from [12], has an AP difference of 35.3%, from 50.9% for large objects to 15.6% for small. A potential method of tackling this issue is to train multiple detectors on separate partitions of the training data according to the size of the object. While deep-based CNN have millions of parameters to generalise from training to testing, the difference between small and large objects may skew the learning towards the latter. In order to test this hypothesis an initial test will be conducted on the PASCAL VOC dataset. However, PASCAL VOC does not have the same definition of objects sizes as in MS COCO. Therefore, the distribution of the bounding boxes from the training set must be analysed in order to determine an appropriate split of data based on object size. This was done by parsing all of the bounding box coordinates in the 07++12 training set and calculating the area. A histogram of the area can be seen in Figure 4.1. There is a clear tendency to smaller objects in the training set with a clear skew towards the left of the figure.

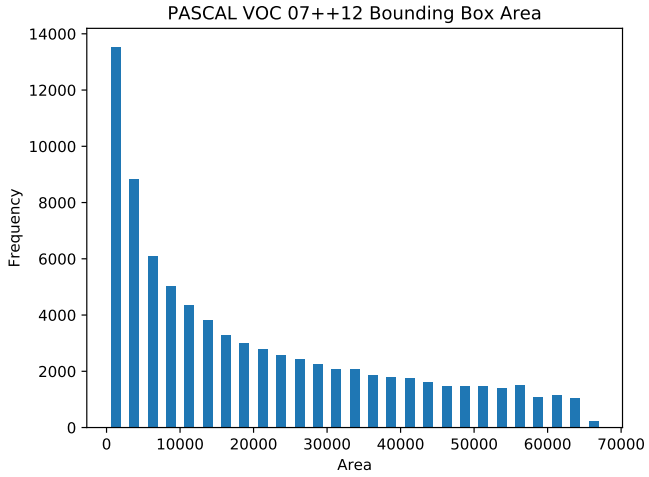


Figure 4.1: Histogram of the PASCAL VOC 07++12 bounding box area.

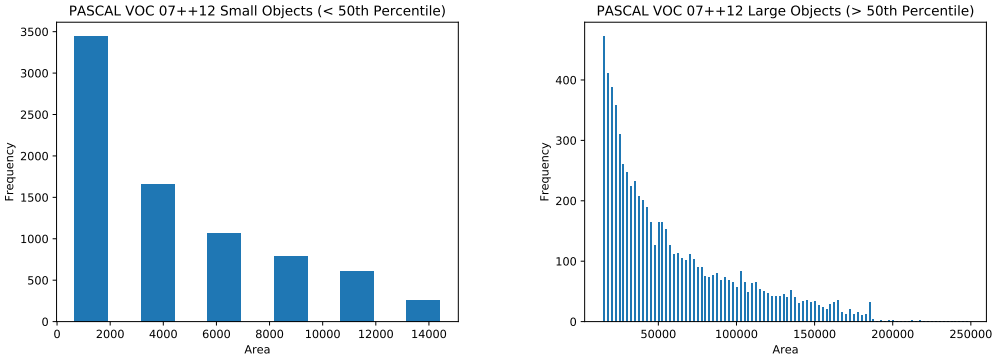
As there is no clear way to split the data into three sets with equal amounts of data, the split was instead done into into two sets, one for small objects and another for larger. The split was made at the median bounding box area which is 13,892 pixels. Therefore, the dataset with bounding box area less than 13,982 pixels is denoted $07{+}{+}12_{small}$ and objects larger make up the dataset $07{+}{+}12_{larger}$.

Identical R-FCN with ResNet-101 networks were trained on either sets of data. Additionally the exact same training strategies were used in both instances which are outlined in Table 4.1.

Table 4.1: Learning parameters for the set of resolution-aware R-FCN object detectors.

Parameter	**Value**
base learning rate	0.001
learning rate policy	step
gamma	0.1
stepsize	80,000
momentum	0.9
weight decay	0.0005

Testing was done on the 07 test set but also split according to the median threshold found in the training data. The two testing sets are denoted as 07Test$_{small}$ and 07Test$_{larger}$. The size of the two test sets are 7,827 and 7,149 ground truth bounding boxes respectively. Histogram for the sets can be seen in Figure 4.2a and Figure 4.2b.



(a) **(b)**

Figure 4.2

The results for the object detectors trained on the separate partitions of training data and a the baseline detector trained on all the data against the three sets of tests can be seen in Table 4.2. The table shows that the detectors trained on the same relative size of data perform best. In the instance of 07Test$_{small}$, its counterpart 07++12$_{small}$ gives a result 04 0.34 mAP. This is an improvement of 12% in comparison to the model trained on all of the data (07++12) which has a mAP of 22.16%; a relative improvement of 35%. The model trained on the data of larger objects, 07++12$_{larger}$, is very poor at detecting smaller objects scoring 3.08%. In this case of 07Test$_{larger}$, the model trained on 07++12$_{larger}$ data is the best performing at 76.60%. A similar improvement is mAP is present in this instance in comparison to the model trained on all of the data. Overall there is an increase of 12.36%, which is a relative improvement of 16%. Finally, the best performing on all of the 07 test data is the model trained on 07++12.

update following explanation

Ensemble of above networks proved not to be possible due to Caffe network architecture. Once networks have been trained as an end-to-end method, the networks were not able to take object proposals as inputs. Solution instead to train the network in 5 stages. Much

Table 4.2: Results

Train Data	Test Data	mAP (%)
$07{+}{+}12_{small}$	07Test_{small}	**34.10**
$07{+}{+}12_{larger}$	07Test_{small}	3.08
07++12	07Test_{small}	22.16
$07{+}{+}12_{small}$	07Test_{larger}	36.40
$07{+}{+}12_{larger}$	07Test_{larger}	**76.60**
07++12	07Test_{larger}	64.34
$07{+}{+}12_{small}$	07	62.65
$07{+}{+}12_{larger}$	07	59.65
07++12	07	**76.35**

slower. Result on VOC2007: 79.59 vs end to end: 76.35

While Table 4.2 shows that it is possible for a R-FCN network to be optimised to objects of a certain size, a more powerful approach is to combine the two networks in an ensemble during testing. Given a potential object, the ensemble should weight one of the networks outputs accordingly. A simple example in this case would be to weight one of the outputs by 100%, if the potential object is below or above the threshold found in the training data. The results gathered in Table 4.2 are much more naive than this, simply splitting the testing data and only using one trained network at a time. Object proposals will needed to be known during inference to ensure that the outputs of the networks in an ensemble are performing classification in the same locations. On top of this, new networks will needed to be trained on subsets of smaller and larger data due to the internal architecture of Convolutional Architecture for Fast Feature Embedding (Caffe) networks. The networks in the ensemble will need to take object proposals as inputs, however, the above mentioned networks are trained based upon the end-to-end versions of the R-FCN. These do not take proposals as inputs but rather have an internal RPN. Therefore, the new networks are trained using 4-step alternating training used in the original Faster R-CNN paper [11]. In this approach the overall network in trained in multiple steps rather than an end-to-end method. In step one a RPN is trained to determine region proposals, the RPN is initialised from a pre-trained ImageNet model and fine-tuned to the proposal task. Next a R-FCN is trained based upon the proposals found in the previous step. This network is also initialised with a pre-trained ImageNet model. In step three, another RPN is trained but initialised using the R-FCN from step two. In this step the convolutional layers that are shared between the R-FCN and RPN are fixed and only the layers unique to the RPN are updated. In this final step, the shared layers between the two networks are again kept fixed and the layers unique to the R-FCN are updated. By training a model with this approach a testing image is able to run through the same steps as a R-FCN trained end-to-end, however, as the networks are split into different models it is also possible to use the stages of the method individually. Creating a solution for finding region proposals with an RPN and having a R-FCN that can take the proposals as inputs.

A potential shortcoming of using RPN proposals as inputs to training a R-FCN rather than using ground truth bounding boxes is that a RPN likely finds many more examples of

objects that actually exist in an image. For a given image, an RPN may find hundreds of potential objects despite an image only containing a couple of true positive examples. This can be solved by setting the proposals with the highest confidence as the ground truth examples and labelling the remaining proposals as the background class. This is a stark comparison to the end-to-end training approach as there are now many more training examples and a large skew towards the background class. Whereas the end-to-end approach does not train on any background examples. With this approach, the total number of training examples is increased from 80,116 to 9,979,345. The median of the almost 10 million proposals is 4,684 pixels, significantly smaller than the threshold of 19,205.5 determined using only ground truth boxes. This large increase in training examples and skew in data poses a question on how to split the RPN proposals such that two networks can be trained towards small and large objects respectively. As mentioned earlier, a pre-requisite in creating subsets of data is that the multiple sets should be roughly equal in size in order for training to be conducted fairly. If the subsets were split by the median of the RPN proposals (4,684) the two sets of data would have equal numbers of examples. However, upon inspection there seems to be a large skew in RPN proposals to smaller objects as there are significantly more true positive examples in the subset of data containing larger objects. This can be seen in Table 4.3, where despite there being an almost even split in data subsets there are significantly more ground truth annotations in the RPN_{larger} subset.

Table 4.3: My caption

Data	**RPN**$_{small}$	**RPN**$_{larger}$
Ground Truth	19,992	60,116
Background	4,969,369	4,929,297
Total	4,989,361	4,989,413

Another option is to use the threshold of 19,205.5 found on only ground truth boxes used in the initial test. The data distribution based on this threshold can be seen in Table 4.4. In this instance there is significantly more data in the RPN_{larger} subset, however, the skew is solely due to the many more background examples. The ground truth annotations are shared equally with 40,058 in each.

Table 4.4: My caption

Data	**RPN**$_{small}$	**RPN**$_{larger}$
Ground Truth	40,058	40,058
Background	3,528,370	6,370,859
Total	3,568,428	6,410,917

As the overall goal of object detectors is to find objects within the C classes, the decision was made to use the threshold of 19,205.5 to create the split in data. Despite there being significantly more background examples.

A baseline 4-step R-FCN with ResNet-101 model was trained on all the data (07++12).

Table 4.5: My caption

Train Data	AP
Small	46.74%
Large	56.91%
All	79.59%
Fusion	**81.81%**

Table 4.6: My caption

Train Data	AP
Small	**55.00%**
Large	10.86%
All	43.80%

4.2 Image Quality Assessment

To evaluate the amount of distortions in the dataset a method for Image Quality Assessment (IQA) is needed. A recent state of the art method is that of Deep IQA [33]. Deep IQA is a CNN-based No-Reference (NR) IQA method.

4.2.1 Training Deep IQA

As per the original authors [33], the models are trained for each distortion type in the Laboratory for Image & Video Engineering (LIVE) IQA second release dataset [34] [35].

Matches across various subsets of data.

Table 4.7: My caption

Train Data	**AP**
Small	21.28%
Large	**80.10%**
All	75.14%

Table 4.8: Gaussian Blur

Model	Best Epoch	LCC	SROCC	Mean Difference
1	5	0.9879	0.9825	2.218
2	1	0.9552	0.9547	2.395
3	1	0.9872	0.9750	2.499
4	6	0.9767	0.9772	2.266
5	23	0.9864	0.9913	2.103
6	6	0.9633	0.9216	2.875
7	10	0.9914	0.9807	2.785
8	7	0.9686	0.9756	2.188
9	6	0.9807	0.9807	4.013
10	2	0.9528	0.9415	3.389

Table 4.9: White Noise

Model	Best Epoch	LCC	SROCC	Mean Difference
1	50	0.9958	0.9900	1.387
2	44	0.9935	0.9817	0.862
3	93	0.9940	0.9875	1.2946
4	8	0.9952	0.9862	1.312
5	96	0.9953	0.9886	1.3604
6	5	0.9953	0.9896	1.5867
7	5	0.9976	0.9952	0.9822
8	3	0.9953	0.9856	1.728
9	129	0.9981	0.9897	1.412
10	286	0.9968	0.9945	1.196

Table 4.10: JPEG

Model	Best Epoch	LCC	SROCC	Mean Difference
1	4	0.9861	0.9584	3.025
2	44	0.9766	0.9523	2.094
3	14	0.9766	0.9374	3.067
4	46	0.9838	0.9565	2.716
5	8	0.9783	0.9304	2.415
6	31	0.9761	0.9560	1.986
7	14	0.9927	0.9553	3.023
8	10	0.9771	0.9728	2.981
9	6	0.9834	0.9500	3.165
10	16	0.9732	0.9539	2.580

Table 4.11: JP2K

Model	Best Epoch	LCC	SROCC	Mean Difference
1	36	0.9823	0.9624	3.819
2	32	0.9879	0.9703	2.822
3	1	0.9658	0.9525	3.243
4	26	0.9861	0.9775	4.011
5	17	0.9790	0.9792	3.552
6	48	0.9885	0.9804	2.720
7	25	0.9836	0.9675	2.916
8	51	0.9871	0.9654	3.345
9	25	0.9714	0.9447	2.953
10	4	0.9567	0.9597	2.265

Table 4.12: Fast Fading

Model	Best Epoch	LCC	SROCC	Mean Difference
1	2	0.9611	0.9379	2.585
2	17	0.9528	0.9343	2.740
3	10	0.9692	0.9521	2.341
4	2	0.9627	0.9585	3.004
5	3	0.9764	0.9487	3.620
6	39	0.9848	0.9748	3.121
7	8	0.9504	0.9376	3.445
8	1	0.9772	0.9668	4.188
9	59	0.9747	0.9477	2.578
10	2	0.9697	0.9467	3.105

Table 4.13: Average Results

Distortion Type	LCC	SROCC
Gaussian Blur	0.9750	0.9681
White Noise	0.9957	0.9887
JPEG	0.9805	0.9523
JP2K	0.9788	0.9600
Fast Fading	0.9679	0.9505

Table 4.14: Lower

	White Noise	Gaussian Blur	JPEG	JP2K	Fast Fading
White Noise		3511 / 42.42%	2918 / 35.25%	2263 / 27.34%	3722 / 44.97%
Gaussian Blur	3511 / 42.42%		6136 / 74.14%	5858 / 70.78%	6879 / 83.12%
JPEG	3511 / 42.42%	6136 / 74.14%		6672 / 80.62%	6454 / 77.98%
JP2K	2263 / 27.34%	5857 / 70.78%	6672 / 80.62%		6021 / 72.75%
Fast Fading	3722 / 44.97%	6879 / 83.12%	6454 / 77.98%	6021 / 72.75%	

Table 4.15: Upper

	White Noise	Gaussian Blur	JPEG	JP2K	Fast Fading
White Noise		3510 / 42.17%	2917 / 35.25%	2262 / 27.34%	3721 / 44.96%
Gaussian Blur	3510 / 42.17%		6135 74.14%	5857 / 70.78%	6878 / 83.11%
JPEG	2917 / 35.25%	6135 / 74.14%		6671 / 80.62%	6453 / 77.98%
JP2K	2262 / 27.34%	5857 / 70.78%	6671 / 80.62%		6020 / 72.75%
Fast Fading	3721 / 44.96%	6878 / 83.11%	6453 / 77.98%	6020 / 72.75%	

Table 4.16: Lower / Upper

	White Noise$_{upper}$	Gaussian Blur$_{upper}$	JPEG$_{upper}$	JP2K$_{upper}$	FF$_{upper}$
White Noise$_{lower}$	0%	57.57%	64.74%	72.66%	55.03%
Gaussian Blur$_{lower}$	57.58%	0%	25.86%	29.22%	16.88%
JPEG$_{lower}$	64.74%	25.86%	0%	19.38%	22.02%
JP2K$_{lower}$	72.66%	29.22%	19.38%	0%	27.25%
Fast Fading$_{lower}$	55.03%	16.88%	22.02%	27.25%	0%

5 Discussion

6 Conclusion

Bibliography

[1] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results," http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[2] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[3] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, *Microsoft COCO: Common Objects in Context*. Cham: Springer International Publishing, 2014, pp. 740–755. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-10602-1_48

[4] X. Zhang, Y.-H. Yang, Z. Han, H. Wang, and C. Gao, "Object class detection: A survey," *ACM Comput. Surv.*, vol. 46, no. 1, pp. 10:1–10:53, Jul. 2013. [Online]. Available: http://doi.acm.org/10.1145/2522968.2522978

[5] F. Schroff, *Semantic Image Segmentation and Web-supervised Visual Learning*. University of Oxford, 2009. [Online]. Available: https://books.google.dk/books?id=4EqZYgEACAAJ

[6] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[7] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders, "Selective search for object recognition," *International Journal of Computer Vision*, vol. 104, no. 2, pp. 154–171, 2013. [Online]. Available: http://dx.doi.org/10.1007/s11263-013-0620-5

[8] R. Girshick, "Fast R-CNN," in *Proceedings of the International Conference on Computer Vision (ICCV)*, 2015.

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[10] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *ICLR*, 2015.

[11] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Neural Information Processing Systems (NIPS)*, 2015.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[13] M. COCO. (2017) Ms coco detections leaderboard. [Online]. Available: http://http://mscoco.org/dataset/#detections-leaderboard

[14] C. Szegedy, S. Ioffe, and V. Vanhoucke, "Inception-v4, inception-resnet and the impact of residual connections on learning," *CoRR*, vol. abs/1602.07261, 2016. [Online]. Available: http://arxiv.org/abs/1602.07261

[15] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy, "Speed/accuracy trade-offs for modern convolutional object detectors," *CoRR*, vol. abs/1611.10012, 2016. [Online]. Available: http://arxiv.org/abs/1611.10012

[16] S. Zagoruyko, A. Lerer, T. Lin, P. H. O. Pinheiro, S. Gross, S. Chintala, and P. Dollár, "A multipath network for object detection," *CoRR*, vol. abs/1604.02135, 2016. [Online]. Available: http://arxiv.org/abs/1604.02135

[17] S. Bell, C. L. Zitnick, K. Bala, and R. B. Girshick, "Inside-outside net: Detecting objects in context with skip pooling and recurrent neural networks," *CoRR*, vol. abs/1512.04143, 2015. [Online]. Available: http://arxiv.org/abs/1512.04143

[18] A. Shrivastava and A. Gupta, *Contextual Priming and Feedback for Faster R-CNN*. Cham: Springer International Publishing, 2016, pp. 330–348. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-46448-0_20

[19] A. Shrivastava, R. Sukthankar, J. Malik, and A. Gupta, "Beyond skip connections: Top-down modulation for object detection," *CoRR*, vol. abs/1612.06851, 2016. [Online]. Available: http://arxiv.org/abs/1612.06851

[20] A. Shrivastava, A. Gupta, and R. B. Girshick, "Training region-based object detectors with online hard example mining," *CoRR*, vol. abs/1604.03540, 2016. [Online]. Available: http://arxiv.org/abs/1604.03540

[21] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg, "SSD: single shot multibox detector," *CoRR*, vol. abs/1512.02325, 2015. [Online]. Available: http://arxiv.org/abs/1512.02325

[22] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "Overfeat: Integrated recognition, localization and detection using convolutional networks," *CoRR*, vol. abs/1312.6229, 2013. [Online]. Available: http://arxiv.org/abs/1312.6229

[23] D. Erhan, C. Szegedy, A. Toshev, and D. Anguelov, "Scalable object detection using deep neural networks," in *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, ser. CVPR '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 2155–2162. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2014.276

[24] C. Szegedy, S. E. Reed, D. Erhan, and D. Anguelov, "Scalable, high-quality object detection," *CoRR*, vol. abs/1412.1441, 2014. [Online]. Available: http://arxiv.org/abs/1412.1441

[25] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *CoRR*, vol. abs/1506.02640, 2015. [Online]. Available: http://arxiv.org/abs/1506.02640

[26] J. Redmon and A. Farhadi, "YOLO9000: better, faster, stronger," *CoRR*, vol. abs/1612.08242, 2016. [Online]. Available: http://arxiv.org/abs/1612.08242

[27] J. Dai, Y. Li, K. He, and J. Sun, "R-FCN: object detection via region-based fully convolutional networks," *CoRR*, vol. abs/1605.06409, 2016. [Online]. Available: http://arxiv.org/abs/1605.06409

[28] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," *CoRR*, vol. abs/1411.4038, 2014. [Online]. Available: http://arxiv.org/abs/1411.4038

[29] Y. Li, H. Qi, J. Dai, X. Ji, and Y. Wei, "Fully convolutional instance-aware semantic segmentation," *CoRR*, vol. abs/1611.07709, 2016. [Online]. Available: http://arxiv.org/abs/1611.07709

[30] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, 2010. [Online]. Available: http://dx.doi.org/10.1007/s11263-009-0275-4

[31] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes challenge: A retrospective," *International Journal of Computer Vision*, vol. 111, no. 1, pp. 98–136, 2015. [Online]. Available: http://dx.doi.org/10.1007/s11263-014-0733-5

[32] flickr. flickr. [Online]. Available: https://www.flickr.com/

[33] S. Bosse, D. Maniry, K. Müller, T. Wiegand, and W. Samek, "Deep neural networks for no-reference and full-reference image quality assessment," *CoRR*, vol. abs/1612.01697, 2016. [Online]. Available: http://arxiv.org/abs/1612.01697

[34] H. R. Sheikh, M. F. Sabir, and A. C. Bovik, "A statistical evaluation of recent full reference image quality assessment algorithms," *IEEE Transactions on Image Processing*, vol. 15, no. 11, pp. 3440–3451, Nov 2006.

[35] H. R. Sheikh, M. F. Sabir, and A. C. Bovik. Live image quality assessment database release 2. [Online]. Available: http://live.ece.utexas.edu/research/quality

Appendices

A Appendix A

A.1 Resolution-Aware Object Detection

Table A.1: Results Test07$_{small}$

Train Data	Test Data	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
07++12$_{small}$	07Test$_{small}$	**34.10**	**28.32**	**31.47**	**50.08**	**52.30**	**46.47**	**17.04**	**50.08**	**13.36**	**45.58**	**63.96**	0.38	**16.54**	**15.72**	**34.19**	**55.16**	**31.02**	**65.25**	**2.52**	**9.07**	**53.46**
07++12$_{larger}$	07Test$_{small}$	3.08	1.54	4.91	4.95	2.96	1.42	2.90	3.44	1.04	5.58	2.79	0.34	2.68	2.50	3.42	3.02	2.20	5.74	0.94	4.22	4.98
07++12	07Test$_{small}$	22.16	21.49	20.70	28.18	30.57	35.72	12.04	35.48	5.83	39.40	41.18	**0.52**	10.56	10.08	15.07	28.92	20.93	44.11	1.58	8.76	32.13

Table A.2: Results Test07$_{larger}$

Train Data	Test Data	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
07++12$_{small}$	07Test$_{larger}$	36.40	53.91	46.30	29.28	12.45	13.91	62.50	35.16	71.09	4.57	22.06	43.96	61.90	56.13	44.99	17.84	5.09	17.14	52.30	64.13	13.35
07++12$_{larger}$	07Test$_{larger}$	**76.60**	**87.49**	**78.95**	**76.57**	**66.74**	**67.98**	**86.49**	**86.40**	**88.44**	**50.11**	**82.56**	74.15	**86.59**	**86.16**	**82.57**	**84.00**	**46.13**	**67.78**	**78.31**	**84.39**	**71.11**
07++12	07Test$_{large}$	64.34	78.18	71.17	58.76	43.91	36.72	82.71	73.40	85.64	25.73	54.67	**74.47**	82.66	82.25	77.02	63.90	31.53	46.35	77.67	84.07	54.92

Table A.3: Results 07

Train Data	Test Data	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
07++12$_{small}$	07	59.65	60.70	68.59	58.54	46.91	22.21	76.58	56.20	84.76	39.22	51.84	66.49	79.93	79.00	68.58	54.64	27.72	47.86	75.42	78.99	48.95
07++12$_{larger}$	07	62.65	75.46	69.93	69.03	50.15	51.35	74.68	80.54	77.36	40.38	75.05	40.65	73.59	69.38	67.94	59.37	31.35	68.72	54.13	69.66	54.23
07++12	07	**76.35**	**79.14**	**80.16**	**76.97**	**68.68**	**57.11**	**86.36**	**85.95**	**88.34**	**60.06**	**86.85**	**67.12**	**87.91**	**86.51**	**80-36**	**78.14**	**45.67**	**77.74**	**76.98**	**83.75**	**73.30**

Notes