International Journal of Science and Research (IJSR)
ISSN: 2319-7064
SJIF (2022): 7.942

Object Detection in Images: A Survey

Sabyasachi Moitra[1], Sambhunath Biswas[2]

[1] Department of Computer Science & Engineering, Techno India University, West Bengal, India
moitrasabyasachi[at]gmail.com

[2] Department of Computer Science & Engineering, Techno India University, West Bengal, India
sambhunathbiswas17[at]gmail.com)
[2]Ex-Indian Statistical Institute, Kolkata, India

Abstract: ***Object detection in a static or dynamic scene, such as a video, is a well-known problem in the computer or machine vision community. Some techniques have already been developed so far. Such detected objects have applications in many different areas. The literature in this domain is vast; as a result it is really very difficult for a beginner to work in this domain. A guideline, therefore, is very helpful, particularly to the young researchers working in the area of computer vision. We have therefore felt the need of a review in this domain. With this objective in mind, we have written this article. We have studied some major existing state-of-the-art object detection methods and chronologically summarize each of them. Each summarization narrates a detailed description on motivation, work description, merits and demerits, followed by some discussion.***

Keywords: Computer vision, Convolutional neural network, Deep learning, Image classification, Image localization, Object detection.

1. Introduction

Humans can easily identify (or classify) and locate objects present in an image, *i.e*., different types of objects are present in an image, where they are located in that image, and how they interact with each other. The goal is to train a computer like humans to have understanding about a given image, *i.e*., what objects are present in the image and where are they located. With the availability of a large amount of data, faster GPUs, and improved algorithms, computers can be easily trained so that they can detect (identify and locate) objects present within a given image with a high degree of accuracy.

Image classification aids the classification of a single object within an image, whereas image localization specifies the exact location of the classified object within an image. Object detection is associated with both the class and location of multiple objects in an image (see Fig. 1). In computer vision, the most common method for localizing an object in an image is to draw a bounding box around the object with its coordinates.

Our review paper deals with some cutting-edge object detection methods and a comparative study among some of them.

Figure 1: Image Classification vs. Image Localization vs. Object Detection

2. Some Related Works

Detecting objects within an image can be accomplished using various techniques, each of which has better performance in some respects.

Object detection methods can be divided in two categories that use (i) classical computer vision techniques and the (ii) modern or deep learning-based techniques. A brief summarization can be found in Fig 2.

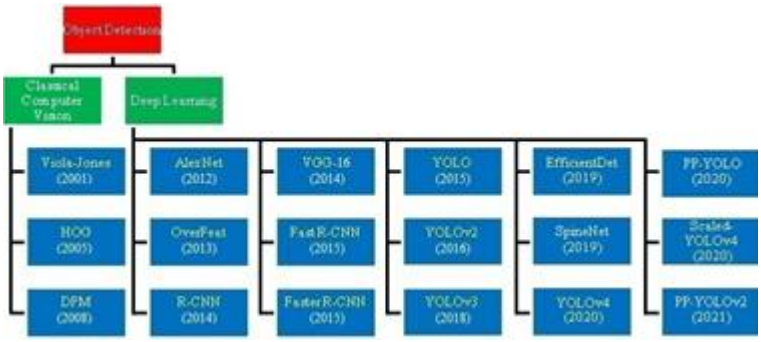

Figure 2: Classification of object detection methods

2.1 Viola-Jones

In 2001, P. Viola and M. Jones proposed an efficient algorithm [1] for face detection. They showed faces in real-time on a webcam feed. It was the most stunning demonstration of machine vision with high degree of potential during that time.

2.1.1. Motivation

Face detection as well as its recognition, now-a-days has 2 immense roles in our daily life, starting from school, office-attendance to different security measures used in surveillance. The human face is one of the most popular and significant area objects because it has numerous applications in security and entertainment. Faces have striking features that differentiate one from the other. However, it may not always be obvious to us.

2.1.2. Work description

Overview

The 3 main contributions in [1] are – (i) creating a new image from an existing image known as the integral image to compute the Haar features quickly, (ii) development of learning algorithm based on AdaBoost for extremely efficient face classifier from a reduced set of visual features, and (iii) cascading of classifiers for rejecting background regions of images and ensuring prominent object-like regions.

Haar features are similar to the convolutional kernels. Each feature yields a single value computed by deducting the sum of pixels in the white rectangle from that in the black rectangle. 3 kinds of Haar features that are used in [1] are – (i) 2-rectangle, (ii) 3-rectangle, and (iii) 4-rectangle features, and they are computed using an integral image [2], which is an intermediate representation of the original image,

$$II(x,y) = \sum_{x' \le x, y' \le y} I(x', y'), \quad (1)$$

where $II(x,y)$ and $I(x',y')$ are the integral and original (must be in grayscale) images, respectively.

For evaluating the Haar features in a given image, the method uses a 24 × 24 window. Within this window, approximately 160,000+ feature values are calculated, taking into account all possible feature parameters such as position, scale, and type. However, only a few sets of these features are useful to detect a face. AdaBoost [3] aids in the discovery of these useful features and builds a strong classifier as a linear combination of these useful features, which are also known as weak classifiers,

$$F(x) = \sum_{i=1}^{n} w_i f_i(x), \quad (2)$$

where $F(x)$ is the strong classifier, $f_i(x)$ is a weak classifier, and w_i is a weight associated with the weak classifier.

The final evaluation is carried out using a cascade classifier [1] composed of stages, each of which contains a strong classifier. Each of these stages determines whether a given window contains a face or not and immediately is discarded if it fails. For localization (if face), a bounding box is drawn with the help of the window coordinates within the image.

The architecture

The architecture of the Viola-Jones face detection model is shown in Fig. 3.

Figure 3: The Viola-Jones face detection model architecture

Training and testing

The Viola-Jones face detection model is trained [4] using frontal upright face images [1].

2.1.3. Merits, demerits and discussion

Viola-Jones does sophisticated feature selection and an invariant detector that locates scales [5]. Rather than scaling the image, the features can be scaled. Because it is a general detection scheme, it can be trained to detect other objects (*e.g.*, automobiles).

The method in [1] – (i) is less effective at detecting faces that are tilted/turned, (ii) is sensitive to illumination conditions, and (iii) suffers from multiple detections of the exact face due to overlapping sub-windows.

Viola-Jones is one of the most powerful methodologies of its time and sets its foundation in the facial detection field, and many modern technologies benefited from it.

2.2. HOG

In 2005, N. Dalal and B. Triggs proposed an efficient feature extraction algorithm [6] for detecting pedestrians in images.

2.2.1. Motivation

The need for a strong feature set that for a human was felt for long time. One of the requirements was to distinguish cleanly even in complex backgrounds with poor lighting. This was successfully done in their work.

2.2.2. Work description

Overview

The distribution of gradient directions is used as image features in the HOG feature descriptor. A small patch in the ratio of 1:2 (width: height) is cropped out of an input image of size $M \times N$ and resized to 64 × 128 [7], and the gradient magnitude (g) and direction (θ) of this resized patch are computed,

$$\begin{aligned} g &= \sqrt{(g_x^2 + g_y^2)}, \\ \theta &= arctan\frac{g_y}{g_x}, \end{aligned} \quad (3)$$

where g_x and g_y are the horizontal and vertical gradients, respectively. The resized patch is divided into 8 × 16 grid blocks with 8 × 8 cells per block. For each 8 × 8 patch, a histogram of gradients is created. Each patch's gradient has 2

values per pixel – (i) magnitude and (ii) direction. Each gradient magnitude is chosen and placed into the desired bin based on the corresponding value in the gradient direction. The final feature vector for the entire 64 × 128 image patch is computed by concatenating the 36 × 1 vectors (9 × 4 = 36 (9 → number of histogram bins, 4 → number of histograms in a 16 × 16 patch)) into a single vector of size 3780 × 1 (7 × 15 × 36 = 3780 (7 × 15 → number of feature vectors, 36 → size of each feature vector)). These extracted features are fed to a linear SVM trained for human/non-human classification to determine whether or not the image contains a human.

For detection of objects (pedestrians) at various locations in the image, a 64 × 128 window slides across the original image at all positions. For each sliding window crop, HOG + SVM is performed to check whether the object within the window is human or not and localize it (if human) by drawing a bounding box with the help of the sliding window coordinates (x, y, w, h) within the image. An image pyramid of different scales [8] is used to detect variable-sized objects at various locations within an image. Non-maximum suppression (NMS) is applied to suppress weak detections for final prediction.

The architecture

The complete human detection pipeline using HOG is shown in Fig. 4.

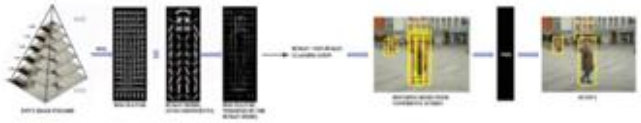

Figure 4: Human detection using HOG

Training and testing

The model is trained and tested on both MIT Pedestrian and INRIA Person datasets. 1239 positive and 1218 negative image samples [6] are used in training the model. During training, data is also augmented.

2.2.3. Merits, demerits and discussion

The HOG algorithm creates histograms of edge orientations from certain patches in an image, and even with a change in color, clothing, and other factors for a person (say), the general edges remain relatively constant, thus succeeding in human detection.

HOG works well, for a rigid body [6] but not for a non-rigid or deformed one.

HOG is a dense sampling based feature extraction technique. It significantly outperforms existing algorithms [9][10] in the task of human detection. The average precision of the detector on the INRIA Person dataset is 75% [11] and won the 2006 PASCAL object detection challenge [12].

2.3. DPM

In 2008, Felzenszwalb *et al*. proposed an improved version [12] of HOG.

2.3.1. Motivation

The requirement for such a model is to take care of that works well a non-rigid body (*e.g*., a human in various poses), intra-class variability (*e.g*., different shaped cars), variations caused by different perspectives and lighting on an object. The previous approach [6] could not handle these problems.

2.3.2. Work description

Overview

DPM is a star-structured part-based model based on a root filter, a set of part filters, and the associated deformation costs. The score of the model (M) at a particular location (l_0) and scale (s) within an image (I) is,

$$score(M, l_0, s, I) = \begin{matrix} score(F_0, l_0) \\ + \sum_{i=1}^{n} max_{l_i}[score(F_i, l_i) \\ - cost(l_i', l_i)] \end{matrix} \quad (4)$$

where $score(F_0, l_0)$ is the score of the root filter F_0 at the given location l_0, $score(F_i, l_i)$ is the score of the part filter F_i at the location l_i, and $cost(l_i', l_i)$ is the deformation cost measuring the deviation of the location of the ith part (l_i) from its ideal location relative to the root (l_i').

The complete configuration of the object hypothesis (the position of each part and the root width) is used in predicting a bounding box for the object [12]. For predicting the bounding box coordinates (x_1, y_1, x_2, y_2), four linear functions are learned by linear least-square regression on the training data. Like HOG, NMS is applied to suppress weak detections (per class) and, the final predictions are obtained.

The architecture

The complete human detection pipeline using DPM is shown in Fig. 5.

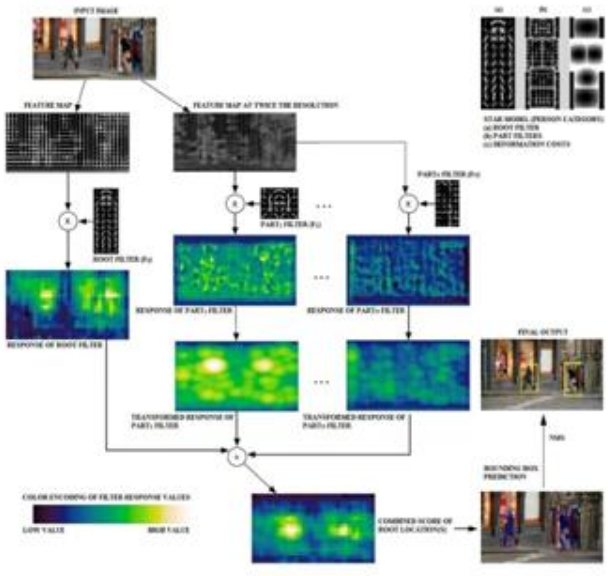

Figure 5: The human detection process at 1 scale [12]. The root and part filters' responses are computed at different resolutions and the transformed responses are added together

to produce a final score for each root location (assuming at most one object per root location).

Training and testing
A latent SVM [12][11] is constructed to learn the model parameters (root filter, part filters, and deformation costs) and must be able to determine the best values for the latent variables (part location).

The model is trained and tested on both INRIA Person and PASCAL VOC datasets.

2.3.3. Merits, demerits and discussion
The model works fine for a non-rigid or deformed body. But the process is a bit lengthy and complex. The manually designed feature extraction technique in this model leads to a lower object detection performance [13]. The object detection model relies on multiscale deformable part model mixtures. The model is heavily reliant on new methods for discriminative training of classifiers that use latent information and fast techniques for matching deformable models to images. In comparison to the previous methodology [6], the resulting system is both efficient and accurate.

A good survey paper [14] on pedestrian detection (using classical computer vision techniques) is also written by Gerónimo *et al*. The survey is helpful to all the researchers in the computer vision community. It includes some important comparative results in this domain.

2.4. AlexNet

In 2012, Krizhevsky *et al*. proposed an efficient deep learning algorithm [13] for image classification by building a deep CNN and marked the beginning of the deep learning era.

2.4.1. Motivation
Previous approaches [1][6][12] to object detection relied on small image datasets of tens of thousands of images. With datasets of this size, simple detection tasks are solved quite well. However, real-life objects show significant variability, so to learn to recognize them and improve performance, much larger training datasets with millions of images and a potent model with a high learning capacity are required.

Convolutional neural network (CNN) helps in classifying different objects, identify their boundaries, differences, and relations to one another from complicated sights (or scenes) with multiple overlapping objects. It is also a good feature extractor compared to the previously manually designed methods [1] [6] [12].

2.4.2. Work description

Overview
Random crops of size $M \times M$ are generated from an image of size $S \times S$ to feed the AlexNet for classification ($S = 256$, $M = 227$) [15]. The input image must be converted if it is not $S \times S$. Images are pre-processed by subtracting from each pixel the mean RGB value computed on the training set. For each crop, features are extracted using convolutional (CONV) layers and then passed to a series of fully-connected (FC) layers for classification. The final classification score is calculated, by averaging the network's classification layer's predictions, on the random crops.

The network architecture
The AlexNet contains 11 layers – 5 CONV, 3 MAXPOOL, 2 FC, and 1 SOFTMAX (classification) layers (see Fig. 6). The network takes an image of size 227 × 227 × 3 as input. The final CONV layer produces a 6 × 6 × 256 tensor, which is flattened and fed to a sequence of FC layers, producing 1000 outputs, *i.e*., the final classification scores. ReLU is applied after all of the CONV and FC layers, and a local response normalization (LRN) layer is added after the ReLU in the 1st and 2nd CONV layers. A dropout layer with a 0.5 rate is added after the first and second FC layers to avoid overfitting.

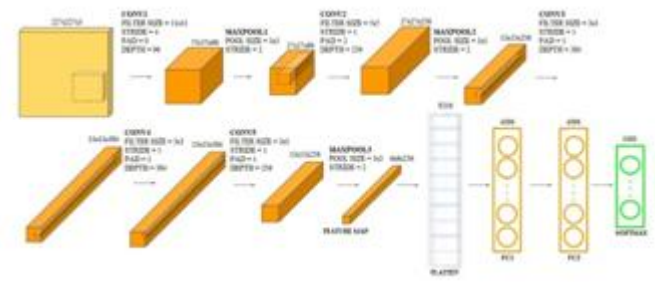

Figure 6: The AlexNet architecture

Training and testing
The network is trained on multiple GPUs (batch size: 128, number of epochs: 90, optimizer: stochastic gradient descent (SGD) with momentum (momentum = 0.9), learning rate: 0.01) [13]. To reduce overfitting, 2 types of data augmentation are used – (i) generating translations and horizontal reflections of the image and (ii) altering the intensities of the RGB channels of the image. The 1st type of data augmentation is used during both training and testing, while the 2nd type is only used during training.

The model is trained and tested on the ImageNet 1000-class dataset. The image classification using AlexNet is shown in Fig. 7.

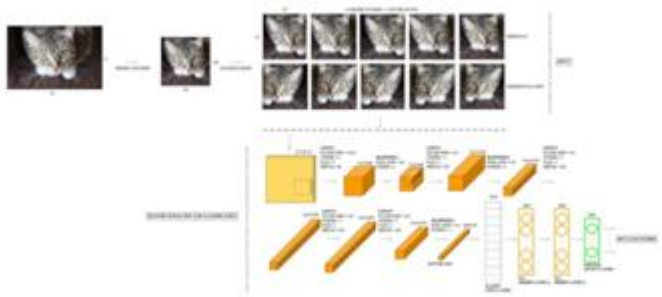

Figure 7: Image classification using AlexNet

2.4.3. Merits, demerits and discussion

Training on multiple GPUs reduces TOP-1 and TOP-5 error rates by 1.7% and 1.2%, respectively, compared to a single GPU and take slightly less time [13]. The 2nd form of data augmentation brings down the TOP-1 error rate by over 1%. The batch normalization reduces TOP-1 and TOP-5 error rates by 1.4% and 1.2%, respectively. The overlapping max-pooling lowers TOP-1 and TOP-5 error rates by 0.4% and 0.3%, respectively, compared to the non-overlapping max-pooling.

The approach uses a set of 10 views (4 corners and 1 center, with their respective horizontal flip) of fixed size (due to the fixed input size constraint) as input to the network, resulting in an excessive number of inputs [13]. This method also ignores many image regions (*i.e.*, it cannot detect objects of varying sizes at different locations in the image) and is computationally superfluous when views overlap.

In ILSVRC-2010, the model attains TOP-1 and TOP-5 test error rates of 37.5% and 17.0%, respectively, and a TOP-5 test error rate of 15.3% in ILSVRC-2012 using its variant [13]. When a single CONV layer is removed, the model's performance suffers, resulting in a loss of about 2% for the network's TOP-1 performance, indicating that the depth of a network is very important for achieving the results.

2.5. OverFeat

In 2013, Sermanet *et el*. proposed an integrated framework [16] for classification, localization, and detection of objects using CNN.

2.5.1. Motivation

The previous approach [13] used a set of 10 fixed-size views as input to the network, ending up with too many inputs. This approach also ignored many image regions, *i.e.*, it couldn't detect objects of varying sizes at different locations in the image and was computationally superfluous when views overlap. It was only used on a single scale, which may not be the scale at which the CNN will respond with the greatest confidence.

2.5.2. Work description

Overview

The OverFeat model removes the fixed-size input constraint [17][18] by converting the network's FC layers as convolution operations. An image pyramid of different scales is used to detect objects of different sizes at different locations in the image. Like AlexNet, features are extracted using CONV layers and then passed to a series of FC layers converted as CONV layers for classification and localization. Greedy merge [16][19] is used to get the final predictions.

The network architecture

OverFeat uses AlexNet as the base network [20]. The first 5 CONV layers are modified, and the 2 FC layers and 1 classification layer (SOFTMAX) are implemented as 3 CONV layers. An image pyramid of 6 different scales (245 × 245, 281 × 317, 317 × 389, 389 × 461, 425 × 497, and 461 × 569) is fed into the network, producing a C-dimensional vector as an output for each scale (C = 1000 (number of classes)) (see Fig. 8). A bounding box regression network replaces the classification network for localization.

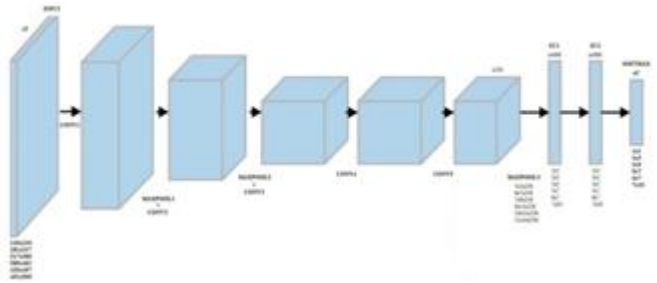

Figure 8: The OverFeat architecture

Training and testing

The classification is done through the classification network using the 6 different scales and their horizontally flipped versions. For each scale and flip, the final classification layer of the network generates a class score map that provides a confidence score that an object of class c (say) is present in the corresponding field of view. Following a series of steps [16], the final classification scores for an image are derived from these class score maps.

The regression network is trained using the same set of scales as the classification network. The regressor net's prediction at each spatial location is compared to the ground-truth bounding box. An L_2 loss between the predicted and ground-truth boxes is used to train the network. The network is not trained on boxes that have less than 50% overlap with the ground-truth. Instead of the class score map, the final regressor layer produces a set of bounding box predictions.

The model is trained and tested on the ImageNet dataset (batch size: 128, optimizer: SGD with momentum (momentum = 0.6), learning rate: 5×10^{-2} and is reduced by a factor of 0.5 after each (30, 50, 60, 70, 80) epoch). Object detection by the OverFeat model is shown in Fig. 9.

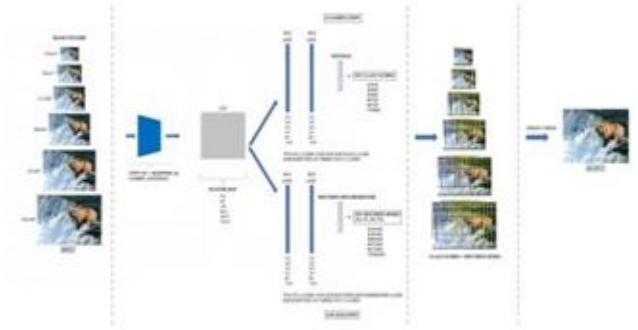

Figure 9: Object detection by OverFeat model.

2.5.3. Merits, demerits and discussion

The approach – (i) removes the fixed input size constraint by converting the FC layers as convolution operations, (ii) has no too many inputs to the network, (iii) is able to classify, localize, and detect objects of different sizes at different locations of an image due to the multi-scale, sliding window approach, (iv) uses CONV layers for classification, localization, and detection tasks instead of FC layers, and (v) uses an integrated pipeline that performs various tasks (classification, localization, and detection) while sharing a common base for feature extraction learned entirely from pixels.

For localization, the approach does not back-propagate through the entire network. Use of L_2 loss, rather than improving the IoU measure directly to get the bounding boxes. The multi-scale, sliding window approach for classification, localization and detection of objects is computationally expensive and time-consuming.

OverFeat uses a multi-scale, sliding window approach for classification with localization and detection of objects using ConvNets. The methodology uses an integrated pipeline that performs various tasks (classification, localization, and detection) while sharing a common base for feature extraction learned entirely from pixels. The accuracy of the OverFeat network is improved by lowering the network's resolution [21] from 36 to 12. As a result, it will be able to detect more objects. The model ranks 4th in classification (13.6% top-5 test error rate), 1st in localization (29.9% top-5 test error rate), and 3rd in detection (19.4% mAP) in ILSVRC-2013 competition [16].

2.6. R-CNN

In 2014, Girshick *et el*. proposed a region-based CNN model [22] for object detection.

2.6.1. Motivation

The previous method [16] took a dense sampling approach at different scales and their respective horizontal flipped versions to determine the presence and type of the object at every location, which was very time-consuming. The method also lowered the network resolution for better accuracy, which increased the computational cost. Use of L_2 loss, rather than improving the IoU measure directly to get the bounding boxes affected the model's accuracy.

2.6.2. Work description

Overview

R-CNN is a 2-stage object detector model – (i) the 1st stage – generates region proposals or RoIs, and (ii) at the 2nd stage – these regions are fed to a CNN for feature extraction and classification.

Using selective search [23], the object detection model generates around 2000 RoIs for an input image, each of which is warped to a fixed size (227 × 227) and fed to a CNN, which produces a 4096-d feature vector as output, which is then passed to an SVM for classification. Bounding-box regression is used to improve localization. NMS is applied to suppress weak detections (per class) for the final predictions.

The network architecture

R-CNN also uses AlexNet as the base network [24] containing 11 layers – 5 CONV layers, 3 MAXPOOL layers, 2 FC layers, and 1 classification layer. A 227 × 227 RGB image is fed into the network, producing output for *C* different classes (see Fig. 10). The FC layers are removed for localization, and the classification layer is replaced with a bounding box regression layer.

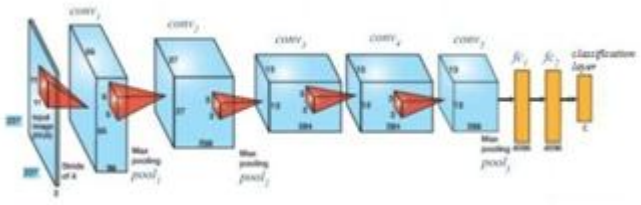

Figure 10: The R-CNN architecture

Training and testing

The model is trained and tested on PASCAL VOC dataset. The training needs mainly 4 different stages [25] – (i) supervised pre-training, (ii) domain-specific fine-tuning, (iii) object category classification, and (iv) bounding box regression.

The network is first trained on the ImageNet dataset so that it can learn the basic image features. For adapting to the detection task and the warped VOC windows, the ImageNet-specific 1000-way classification layer of the network is replaced with a 21-way classification layer (20 VOC classes plus background) and ultimately the model fine-tuned with this layer (batch size: 128 (32 positive and 96 negative), optimizer: SGD, learning rate: 0.001). The final classification layer is removed after fine-tuning, and a 4096-d feature vector is obtained for each of the 2000 RoIs. Next a linear SVM is trained for each class using the obtained feature vector, and the final output is a set of positive object proposals from the features of 2000 region proposals for each class.

Bounding box regression is performed using the $pool_3$ features to improve localization. As a result, an accurate and correct bounding box around the object is obtained for all the positive object proposals.

The object detection using R-CNN is shown in Fig. 11.

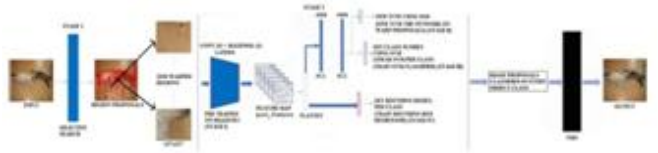

Figure 11: Object detection using R-CNN

2.6.3. Merits, demerits and discussion

The approach is faster and more accurate compared to the previous methods. It uses a region proposal algorithm to perform multiple object detection in one-shot instead of the sliding window technique at different scales.

The selective search region proposal method is a fixed technique; no learning occurs at that stage, which may result in poor candidate region proposals. R-CNN consumes both time and space because it computes the feature map for each RoI, and each feature map of each RoI must be saved, which requires a large amount of memory. Furthermore, the training task is difficult because it necessitates four stages of training.

On the VOC 2007, VOC 2010, and VOC 2011/2012 datasets, R-CNN achieves 58.5%, 53.7%, and 53.3% mAP, respectively [22]. This performance is achieved by combining traditional computer vision tools (region proposals using selective search) with deep learning (CNN). Supervised pre-training, domain-specific fine-tuning, object category classification, and bounding box regression contribute to this level of accuracy.

2.7. VGG-16

In 2014, K. Simonyan and A. Zisserman proposed a very large and deep CNN [26] for object detection.

2.7.1. Motivation

AlexNet, the base architecture of previous methods [16][22], had a lot of variations in its CONV and MAXPOOL layers, *i.e.*, all the CONV and MAXPOOL layers used different filter size, padding, and stride, which was very difficult to remember. Moreover, due to the fewer layers, the methods failed to achieve high accuracy.

2.7.2. Work description

Overview

The model uses the methodology of [13] at the time of training, and the methodology of [16] at the time of testing [26]. Both the training and testing are done using single and multiple scale images.

The network architecture

The VGG-16 contains 24 layers – 16 CONV, 5 MAXPOOL, 2 FC, and 1 SOFTMAX (classification) layers (see Fig. 12). The network takes an image of size 224 × 224 × 3 as input. The final CONV layer produces a 7 × 7 × 512 tensor, which is flattened and fed to a sequence of FC layers, producing 1000 outputs, *i.e.*, the final classification scores. ReLU is applied after all of the CONV and FC layers. LRN is not used. A dropout layer with a 0.5 rate is added after the 1st and 2nd FC layers to avoid overfitting. A bounding box regression network replaces the classification network for localization.

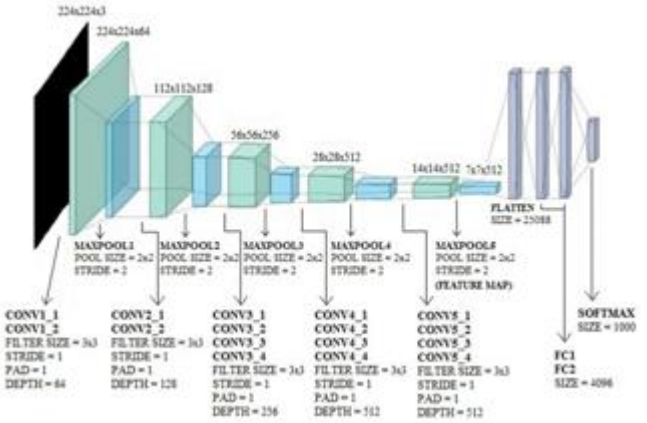

Figure 12: The VGG-16 architecture.

Training and testing

Random crops of size $M \times M$ are generated from an image of size $S \times S$ to feed the VGG-16 for classification training (S = {256, 384} for single-scale training and $S = [S_{min}, S_{max}]$ for multi-scale training (S_{min} = 256, S_{max} = 512), M = 224) [26]. The input image must be converted if it is not $S \times S$. Like AlexNet, images are pre-processed by subtracting from each pixel the mean RGB value computed on the training set. For each crop, features are extracted using CONV layers and then passed to a series of FC layers for classification. The final classification score is calculated by averaging the network's classification layer's predictions on the random crops. The network is trained by optimizing the multinomial logistic regression objective with a mini-batch of size 256. The optimizer and learning rate used for training is similar to that of AlexNet. The multi-scale models are trained by fine-tuning all the layers of the single-scale model with the same configuration, pre-trained with fixed S = 384. Localization training is similar to classification training, except that the objective is Euclidean loss rather than multinomial logistic regression, and the learning rate is 0.001. Multi-scale training is not used for localization.

At the classification test time, the image is first rescaled to $N \times N$, where N is set as – (i) $N = S$ for fixed S, and $N = 0.5 * (S_{min} + S_{max})$ for jittered $S \in [S_{min}, S_{max}]$ (for single-scale evaluation) (*e.g.*, if S = 256 then N = 256, and if S = [256, 512] then $N = 0.5 * (256 + 512) = 384$), and (ii) $N = [S_{min}, 0.5 * (S_{min} + S_{max}), S_{max}]$ for $S \in [S_{min}, S_{max}]$ (for multi-scale evaluation) (*e.g.*, if S = [256, 512] then $N = [256, 0.5 * (256 + 512), 512] = [256, 384, 512]$) [26]. Like OverFeat, the entire rescaled image is fed to the network, as the FC layers (FC1, FC2, and SOFTMAX) are converted to CONV layers. For each scale, the final classification layer of the network generates a class score map with the number of channels equal to the number of classes. The class score map is spatially averaged to get a fixed-size vector of class scores. The test set is also augmented by horizontal flipping. The final classification scores for an image are obtained by

averaging the predictions made by the network's classification layer on the original and flipped versions. Localization testing is similar to classification testing, except that the output of the last FC layer is a set of bounding box predictions rather than the class score map, as the classification layer (SOFTMAX) is replaced by the bounding box regression layer. Greedy merge [16][19] is used to get the final predictions. Multi-scale evaluation is not used for localization.

The model is trained and tested on the ImageNet dataset. Image recognition using VGG-16 is shown in Fig. 13.

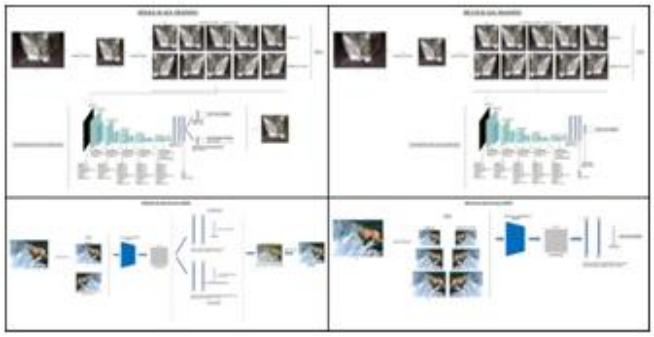

Figure 13: Image recognition using VGG-16.

2.7.3. Merits, demerits and discussion

The architecture of VGG-16 is simple, *i.e.*, all the CONV layers use the same filter size, padding, and stride, and all the MAXPOOL layers use the same filter size, padding, and stride, which is very easy to remember. It extracts more parameters from the images compared to the previous methods, which increases the model accuracy. The model also generalizes effectively to a variety of tasks and datasets, matching or surpassing more complex recognition pipelines based on shallower image representations.

VGG-16 is a heavier model and takes more training time. Like AlexNet and OverFeat, the model also takes too many inputs to the network.

In classification, the network achieves the TOP-1 and TOP-5 validation error rates of 25.5% and 8.0%, respectively, at a single test scale and 24.8% and 7.5%, respectively, at multiple test scales in ILSVRC-2014 [26]. In localization, the network achieves the TOP-5 test error rate of 25.3% in ILSVRC-2014. The network depth contributes to this level of accuracy, *i.e.*, as the network depth increases, so do the accuracy.

2.8. Fast R-CNN

In 2015, R. Girshick proposed an improved version [27] of R-CNN.

2.8.1. Motivation

Due to the fixed-size input constraint, the previous method [22] warped the input image to a fixed size, resulting in an unwanted geometric distortion affecting the accuracy. The training was a multi-stage process that included feature extraction, network fine-tuning, SVM training, and bounding-box regression. The method was slow because it extracted features by repeatedly applying the deep CNN to the warped regions per image. The features were saved to disc and took up a lot of memory space. The training algorithm was unable to update the weights below the FC layers, limiting the model's accuracy.

2.8.2. Work description

Overview

Fast R-CNN eliminates the network's fixed-size input constraint by inserting an RoI pooling layer between the last CONV layer and the 1st FC layer, which is similar to the SPP layer of SPP-net [28] with only one pyramid level. The layer employs max-pooling for converting the features within any valid RoI (as determined by mapping RoI to feature maps [29]) into a small feature map of fixed size $H \times W$. It divides the $h \times w$ RoI window into a $H \times W$ grid of sub-windows of approximately $h/H \times w/W$ size and then max-pools the values in each sub-window into the corresponding output grid cell [30]. The RoI pooling is independently applied to each feature map channel, resulting in a feature map of size $R \times H \times W \times D$ that is fed to a series of FC layers for classification and localization (R = number of RoIs, $H = W = 7$, $D = 512$) [31]. Like R-CNN, NMS is applied to obtain the final predictions.

The network architecture

Fast R-CNN uses VGG-16 as the base network [27]. An RoI pooling layer is added after the last CONV layer (replacing the last MAXPOOL layer) and before the 1st FC layer. The FC layers finally branch into 2 output layers – (i) the SOFTMAX classification layer and (ii) the bounding box regression layer (see Fig. 14).

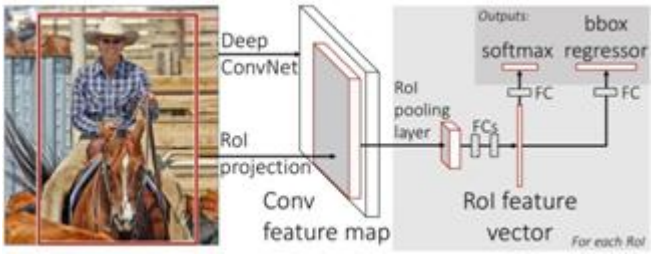

Figure 14: The Fast R-CNN architecture [27]

Training and testing

Like R-CNN, the network is pre-trained on the ImageNet dataset. The network is fed 2 sets of data – (i) a list of images and (ii) a list of RoIs within those images.

During training, SGD mini-batches of size B are hierarchically sampled, first by sampling I images, then by sampling B/I RoIs from each image, resulting in computation and memory sharing by RoIs from the same image in both the forward and backward passes ($B = 128$, $I = 2$ [27]). In addition, Fast R-CNN employs a smooth training process with a single fine-tuning stage that optimizes both the SOFTMAX classifier and the bounding-box regressor. Unlike previous methods, the entire network is trained via back propagation.

The loss function is a multi-task loss,

$$L(p, u, t^u, v) = L_{cls}(p, u) + \lambda[u \geq 1]L_{loc}(t^u, v), \quad (5)$$

where L_{cls} is the classification loss (log loss), L_{loc} is the localization loss (smooth L_1 loss), p is the predicted class, u is the actual class, $t^u = (t_x^u, t_y^u, t_w^u, t_h^u)$ is the predicted bounding box for class u, $v = (v_x, v_y, v_w, v_h)$ is the actual bounding box for the same class, $[u \geq 1]$ is a function that evaluates to 1 for the foreground and 0 for the background class, and λ is a hyperparameter for controlling the balance between the two losses.

For achieving a scale-invariant detection of objects, 2 methods are used – (i) single-scale approach (brute-force learning) and (ii) multi-scale approach. In a single-scale approach, the input images are resized so that the shortest side is *M* pixels and the longest side is no more than *N* pixels while maintaining the aspect ratio (*M* = 600 and *N* = 1000 [27]). In a multi-scale approach, an image pyramid of randomly sampled size (480, 576, 688, 864, and 1200 [27]) of input images is used.

The model is trained and tested on PASCAL VOC and MS COCO datasets. The object detection using Fast R-CNN is shown in Fig. 15.

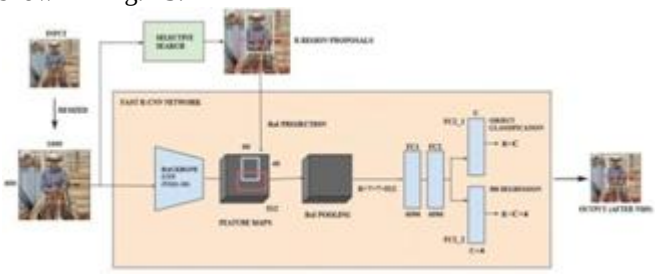

Figure 15: Object detection using Fast R-CNN

2.8.3. Merits, demerits and discussion

The Fast R-CNN has several advantages – (i) faster than R-CNN because it does not need to feed 2000 RoIs to the CNN every time; instead, the convolution operation is performed only once per image; (ii) higher detection quality than R-CNN; (iii) training is a single stage using a multi-task loss and updates all network layers' weights; and (iv) features are not stored on disc.

When comparison is done not using region proposals, Fast R-CNN performance during testing time significantly slows down [32].

On the VOC 2007, VOC 2010, and VOC 2012 datasets, Fast R-CNN achieves 70.0%, 68.8%, and 68.4% mAP, respectively [27]. This level of accuracy is achieved through multi-task training of the entire network. It processes images 9x and 45x faster than R-CNN at training and testing time, respectively. The detection time of the network is reduced by more than 30% when truncated SVD [27] is used, with only a 0.3 drop in mAP. On the COCO dataset, Fast R-CNN achieves a 19.7% mAP@[.5, .95].

2.9. Faster R-CNN

In 2015, Ren *et al*. also proposed an improved version [33] of Fast R-CNN.

2.9.1. Motivation

Methods in [22][27] used the selective search technique [23] to extract RoIs from an image but it is a time-consuming process (around 2s/image with CPU computation) and it degrades the network performance.

2.9.2. Work description

Overview

Faster R-CNN consists of 2 modules – i) RPN (RoI generator), and ii) Fast R-CNN (object detector).

Region proposal network (RPN) is a deep fully convolutional network (FCN) that accepts any size image as input and outputs a set of RoIs, each with an objectness score. RPN works on dedicated CONV layers, with the previous layers shared by Fast R-CNN.

The input image is resized in the same way that Fast R-CNN does so that the shortest side is *M* pixels and the longest side is no more than *N* pixels (*M* = 600 and *N* = 1000) and is fed to a backbone network producing a feature map of size $M' \times N' \times D$ (M' = 40, N' = 60, D = 512 (for VGG-16 [26]) and 256 (for ZFNet [34])). The output feature map is shared by both the RPN and Fast R-CNN networks.

RPN consists of 1 $P \times P$ CONV layer with D units or filters (P = 3) followed by 2 $Q \times Q$ CONV sibling layers (box classification and regression layers) with U and V units respectively (Q = 1, U = 18, V = 36) [35]. The output of the $P \times P$ CONV layer is subjected to ReLU. The RPN receives the backbone feature map. The network's 1st layer slides over this feature map, creating a D-dimensional feature at each sliding window location. This output feature is then passed to the 2 sibling layers, resulting in $M' \times N' \times U$ and $M' \times N' \times V$ outputs. At each sliding window location, the network learns about the presence of an object(s) in the input image with the help of a K set of anchors (K = 9).

The RoIs produced from the RPN module are passed to the Fast R-CNN module for detection.

The network architecture

Faster R-CNN architecture consists of 3 parts – i) CONV layers, ii) RPN, and iii) classification and bounding box regression layers (see Fig. 16). Faster R-CNN uses VGG-16 and ZFNet as the backbone network.

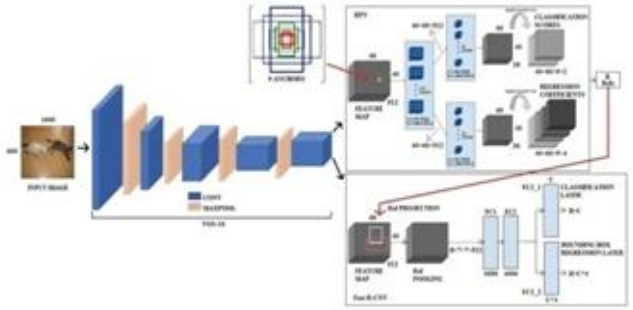

Figure 16: The Faster R-CNN architecture

Training and testing

A 4-Step Alternating Training [33] is used to train the entire model for object detection. This type of training mechanism assists the model in sharing the backbone ConvNet weights between RPN and Fast R-CNN.

During RPN training, an anchor is considered as a positive sample if it has an IoU > 0.7 with any of the ground-truth boxes and negative if it has an IoU < 0.3 with all the ground-truth boxes. The remaining anchors, which are neither positive nor negative, are discarded. A mini-batch of size 256 (128 positive and 128 negative samples from a single image) is used for training the RPN (optimizer: SGD with momentum (momentum = 0.9), learning rate: 0.001 (for 60k mini-batches), 0.0001 (for next 20k mini-batches)).

Like Fast R-CNN, the loss function for RPN is a multi-task loss,

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i, p_i^*) + \lambda\frac{1}{N_{reg}}\sum_i p_i^* L_{reg}(t_i, t_i^*), \quad (6)$$

where p_i is the predicted probability of the ith anchor being an object, p_i^* is the ground-truth label (1 if the ith anchor is positive and 0 if it is negative), t_i is the four parameterized coordinates of the predicted bounding box, t_i^* is the ground-truth box overlapped with the positive anchor, L_{cls} is the classification loss which is the log loss over binary classes (object vs. non-object), and L_{reg} is the regression loss which is the smooth L_1 loss [27]. These two terms (classification and regression) are normalized by N_{cls} (mini-batch size) and N_{reg} (number of anchor locations), respectively and the second term is multiplied by a hyperparameter λ.

The *R*k anchors from each image go through a sequence of post-processing steps [33] to get the final *r*k RoIs ($r < R$) from RPN and are passed to the object detector network for both training and testing.

The model is trained and tested on both PASCAL VOC and MS COCO datasets. The object detection using Faster R-CNN is shown in Fig. 17.

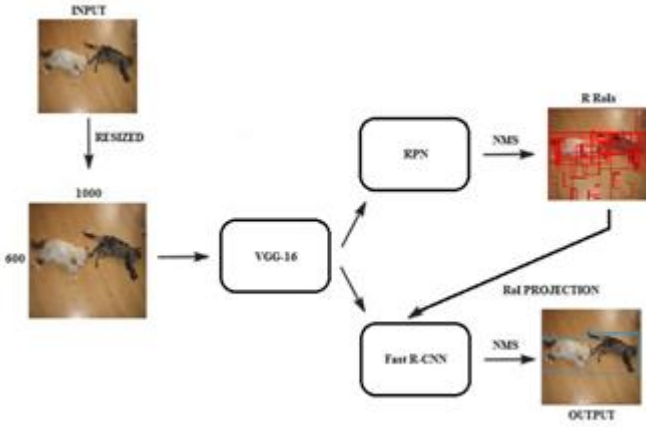

Figure 17 Object detection using Faster R-CNN

2.9.3. Merits, demerits and discussion

Faster R-CNN has several advantages – (i) introduction to RPN for region proposals makes it faster than its predecessors [32], (ii) introduction to anchors for object detection is cost-efficient than previous methodologies like image pyramid and filter pyramid, and (iii) sharing of ConvNet between the two network modules (RoI generator and object detector) makes it a single, unified model for object detection.

Each component of the Faster R-CNN model (RoI generator and object detector) is trained separately.

RPN is an efficient and accurate model for RoI generation. By sharing the ConvNet features with the object detector, the RoI generation step is nearly cost-free. RPN improves the RoI quality compared to other region proposal techniques (selective search and edge boxes [36]), thus improving the overall model accuracy. Faster R-CNN runs at nearly real-time frame rates.

With only 300 RoIs per image, Faster R-CNN achieved state-of-the-art object detection accuracy of 73.2% and 70.4% on VOC 2007 and VOC 2012 datasets, respectively, and 21.9% mAP@[.5, .95] on COCO dataset [33]. In the ILSVRC and COCO 2015 competitions, Faster R-CNN and RPN occupy the first place in various tracks.

2.10 YOLO

In 2015, Redmon *et al*. proposed an efficient unified, real-time, one-stage object detector model [37] that takes an entire image as input and concurrently learns the class label probabilities and bounding box coordinates of the object(s) present in that image.

2.10.1. Motivation

Previous object detection methods [22][27][33] used separate region proposal techniques [23][33] to generate all the possible bounding boxes in an image at first and then ran a classification and localization network on these RoIs for detection, *i.e*., a two-stage object detector model. Because each of these components was separately trained, the object

detection pipeline was very complex, slow, and difficult to optimize.

2.10.2. Work description

Overview

YOLO divides an image into $S \times S$ grid cells. Each of these grid cells predicts B bounding boxes and C class probabilities. If an object's (present within the image) center falls within a grid cell, then that grid cell is subject to detecting that object only, *i.e.*, each grid cell predicts only one object.

The predicted bounding box has 5 components (b_x, b_y, b_w, b_h, b_{conf}), where (b_x, b_y) is the box's center relative to the grid cell location (g_x, g_y), (b_w, b_h) is the box's dimensions relative to the image size (I_w, I_h), and b_{conf} is the box's confidence score, *i.e.*, how certain the model is that an object is contained within the box (the higher the confidence score thicker the box is),

$$\begin{aligned} b_x &= (x - g_x)/g_x,\ b_y = (y - g_y)/g_y, \\ b_w &= w/I_w,\ b_h = h/I_h, \\ b_{conf} &= P(object) * IoU(truth, pred), \end{aligned} \quad (7)$$

where (x, y) and (w, h) are the box's actual center (ground truths) and dimensions respectively, *P(object)* is the probability that the predicted box contains an object (1 if an object is present in that grid cell, otherwise 0), and *IoU(truth, pred)* is the IoU between the ground-truth and predicted boxes. Therefore, there are total $S \times S \times (B * 5)$ outputs associated with the bounding box predictions in an image.

The model predicts C conditional class probabilities per grid cell, despite the number of boxes B, *i.e.*, a total of $S \times S \times C$ class probabilities are predicted across an image,

$$P(class_i) = P(class_i \mid object), \quad (8)$$

where $P(class_i \mid object)$ is the probability that the object belongs to the ith class, given that the object is present in the grid cell. If no object is present in that grid cell, the loss function will not penalize it during training for incorrect class prediction.

The final score, *i.e.*, the category-specific confidence scores for each box, is obtained by combining the individual box's confidence score and the conditional class probabilities, producing $S \times S \times (B * 5 + C)$ outputs ($S = 7$, $B = 2$, $C = 20$) [37]. NMS is applied to suppress weak detections (per class) and, the final predictions are obtained.

The network architecture

YOLO makes use of the Darknet [37] network having 30 layers (24 CONV, 4 MAXPOOL, and 2 FC layers) (see Fig. 18). The network takes an image of size 448 × 448 × 3 as input. The final CONV layer produces a 7 × 7 × 1024 tensor [38], which is flattened and fed to a sequence of FC layers, producing 1470 outputs, which are then reshaped to produce a 7 × 7 × 30 tensor as the final output. The last layer (FC2) employs a linear activation function, while the remaining layers employ Leaky ReLU. A dropout layer with a 0.5 rate is added after the 1st FC layer to avoid overfitting.

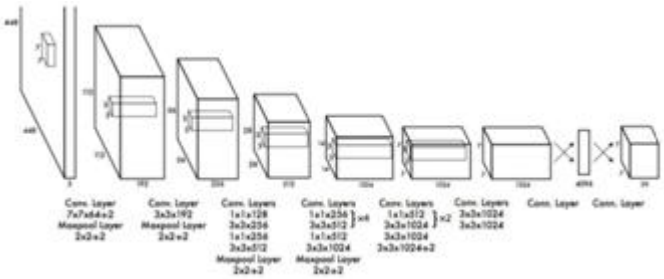

Figure 18: The Darknet architecture [37].

Training and testing

The network is first trained on the ImageNet dataset using the first 20 CONV layers followed by an AVGPOOL layer and an FC layer at an input resolution of 224 × 224 because the ImageNet images are of size 224 × 224. For detection, 4 CONV layers and 2 FC layers with randomly initialized weights are added to the pre-trained network, removing the AVGPOOL and FC layers. The network's input resolution is increased to 448 × 448. A mini-batch of size 64 is used for training the network (number of epochs: 135, optimizer: SGD with momentum (momentum = 0.9), learning rate: 0.01 (for first 75 epochs), 0.001 (for next 30 epochs), 0.0001 (for final 30 epochs)).

The loss function is a multi-part function,

$$\begin{aligned} &\left(\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_{ij}^{obj} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \right. \\ &+\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_{ij}^{obj} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 \right. \\ &\left. \left. + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \right) + \\ &\left(\sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_{ij}^{obj} \left(C_i - \hat{C}_i \right)^2 \right. \\ &\left. +\lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_{ij}^{noobj} \left(C_i - \hat{C}_i \right)^2 \right) + \\ &\left(\sum_{i=0}^{S^2} \mathbb{I}_{i}^{obj} \sum_{c \in classes} \left(p_i(c) - \hat{p}_i(c) \right)^2 \right), \end{aligned} \quad (9)$$

where the 1st part is the localization loss, the 2nd part is the confidence loss, and the 3rd part is the classification loss [39].

The model is trained and tested on PASCAL VOC dataset. The object detection using YOLO is shown in Fig. 19.

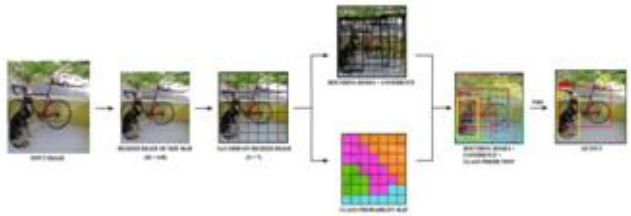

Figure 19: Object detection using YOLO

2.10.3. Merits, demerits and discussion

YOLO is faster compared to previously discussed object detectors due to its one-stage detector strategy, and good for real-time processing. The model is more generalized because it outperforms other methods when applied to domains other than natural images, such as artwork. Unlike the sliding window [16][26] and region proposal-based object detection approaches previously discussed, YOLO sees the entire image during training and testing and implicitly encodes contextual information about classes as well as their appearances. It produces fewer false positives in background areas.

YOLO is less accurate than the previously discussed two-stage object detectors. It imposes a strong spatial constraint on predicting bounding boxes since each grid cell predicts only 2 boxes and can have only 1 class (*i.e.*, each grid cell predicts only one object). Due to this constraint, the model predicts a limited number of adjacent objects (*e.g.*, if two objects (*i.e.*, their centers) accidentally fall within the same grid, YOLO can detect only one).

YOLO is a unified object detection model. It is simple to construct and trained directly on entire images. Unlike the object detection methods discussed previously, YOLO is fast at training and testing time as it requires a single network training and evaluation only. It achieves 63.4% and 57.9% mAP on VOC 2007 and VOC 2012 datasets, respectively [37].

2.11. YOLOv2

In 2016, J. Redmon and A. Farhadi proposed an improved version [40] of YOLO.

2.11.1. Motivation

As it struggles with tiny objects that appear in groups [37] and nearby objects, YOLO made a significant number of localization errors and had a low recall when compared to object proposal-based methods [22][27][33].

2.11.2. Work description

Overview

The FC layers responsible for bounding box prediction in YOLO are removed in YOLOv2, and anchor boxes are used to predict them instead. It shifts the class prediction mechanism from the cell to the anchor box level, which means that instead of predicting objectness and class for every cell, it predicts them for every anchor box within that cell. In contrast to [33], YOLOv2 finds good anchor boxes using the k-means clustering technique [40] on the bounding boxes of the training set and predicts the bounding box coordinates relative to the grid cell, like YOLO, using those anchor boxes.

YOLOv2 jointly trains on the classification and detection datasets to broaden the classes it can detect. The classification and detection dataset labels are combined to form a tree known as WordTree (see Fig. 20), with each child forming an is-a relationship with its parent (*e.g.*, a *jet* is an *airplane*).

To classify using WordTree, conditional probabilities for each child of that parent given that parent is predicted at each node using SOFTMAX, $P(class_{child_i} | class_{parent})$ [39]. For *e.g.*, at the *airplane* node of Fig. 20, the model predicts *P(biplane | airplane), P(jet | airplane), ..., P(stealth fighter | airplane)* conditional probabilities. The probability of a specific node (say *jet*) is calculated by following the path from the specified node to the root node and multiplying the conditional probabilities,

$$\begin{aligned} P(jet) = {} & P(jet \mid airplane) \\ & * P(airplane \mid air) \\ & * \ldots * \\ & * P(artifact \mid physical\ object) \\ & * P(physical\ object), \end{aligned} \tag{10}$$

assuming that an object is already detected (*P(physical object)* = 1).

In object detection, the value of *P(physical object)* equals the bounding box confidence score [39], which determines whether the box contains an object. YOLOv2 descends the tree, taking the most confident path at each split until it reaches a certain threshold and predicts the object class. Like YOLO, NMS is applied to suppress weak detections (per class) and, the final predictions are obtained.

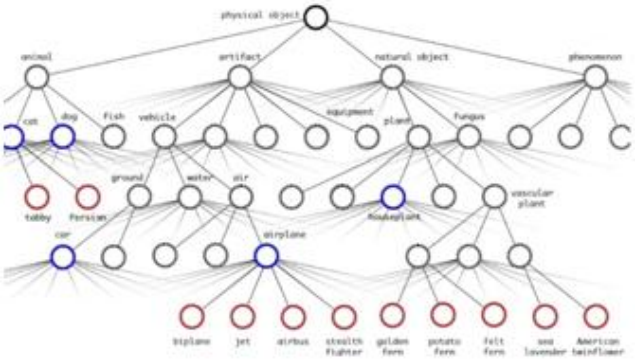

Figure 20: The WordTree [40].

The network architecture

YOLOv2 uses an FCN having 22 CONV layers, 5 MAXPOOL layers, and 1 passthrough layer (see Fig. 21). The network takes an image of size 416 × 416 × 3 as input instead of 448 × 448 × 3 in YOLO [39]. The final CONV layer predicts 5 bounding boxes, each with 5 coordinates and 20 classes per box. Instead of a dropout layer, batch normalization layers are added to all the hidden CONV layers. Like YOLO, the last layer employs a linear activation

function, while the remaining layers employ Leaky ReLU. YOLOv2 uses Darknet-19 [40] as the backbone network.

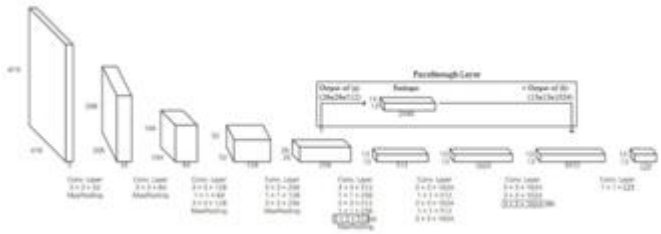

Figure 21: The YOLOv2 architecture

Training and testing

The base network is first trained on the ImageNet dataset (number of epochs: 160, optimizer: SGD with momentum (momentum = 0.9), learning rate: 0.1 at starting and decayed polynomially with a power of 4). The network initially trains with 224 × 224 images and then retunes with 448 × 448 (number of epochs: 10, optimizer: SGD with momentum (momentum = 0.9), learning rate: 0.001). For detection, the last CONV layer, the AVGPOOL layer, and the SOFTMAX layer of the pre-trained network are removed, and 3 3 × 3 CONV layers, each with 1024 filters, are added, followed by a final 1 × 1 CONV layer with 125 filters. A passthrough layer from the final 3 × 3 × 512 layer to the second last CONV layer is also added, as shown in Fig. 21 so that the network can use fine-grained features to detect small objects. The network's input resolution is decreased to 416 × 416. The detection network is trained for 160 epochs using SGD with momentum (momentum = 0.9) with a starting learning rate of 0.001 and then divides the rate by 10 at 60 and 90 epochs. The network is also trained on images with different scales [40].

The loss function is a multi-part function [41] like YOLO.

The model is trained and tested on both PASCAL VOC and MS COCO datasets. The object detection using YOLOv2 is shown in Fig. 22.

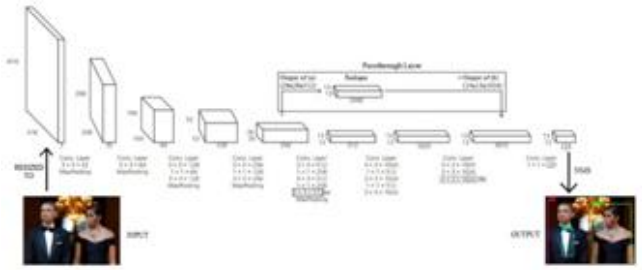

Figure 22: Object detection using YOLOv2.

2.11.3. Merits, demerits and discussion

YOLOv2 overcomes the localization errors and low recall of YOLO. The model performs well with the new varieties of a class not found in the COCO dataset because it can easily generalize their shapes from their parent classes.

YOLOv2 struggles with categories like "sunglasses" or "swimming trunks" [40] as the COCO dataset does not have annotations for any clothing.

YOLOv2 is a cutting-edge, real-time object detection system. It is faster, accurate, and more generalized than other object detection models (previously discussed) across a wide range of detection datasets. It achieves 78.6%, 73.4%, and 48.1% mAP on VOC 2007, VOC 2012, and COCO datasets, respectively [40][42].

2.12. YOLOv3

In 2018, J. Redmon and A. Farhadi proposed an improved version [43] of YOLOv2.

2.12.1 Motivation

Although YOLOv2 was the fastest and most accurate model, it was sometimes unable to detect small objects, losing out to models such as RetinaNet [44] and SSD [45] in terms of accuracy [42].

2.12.2. Work description

Overview

YOLOv3 predicts bounding boxes across 3 different scales by extracting features from those scales. The output is generated by convolving a $S \times S \times (B * 5 + C)$ detection kernel with the feature map ($S = 1$, $B = 3$ (number of boxes a cell on a feature map can predict), $5 \rightarrow$ box attributes (x, y, w, h, *confidence*), $C = 80$ (number of classes)) [43]. YOLOv3 uses anchor boxes for bounding box prediction and logistic regression for objectness score calculation and class prediction. NMS is applied to suppress weak detections (per class) and, the final predictions are obtained.

The network architecture

YOLOv3 uses an FCN having 106 CONV layers (see Fig. 23). CONV layers are used instead of MAXPOOL layers because they prevent the loss of low-level features, allowing the architecture to detect small objects. Like YOLOv2, the network also takes an image of size 416 × 416 × 3 as input. Like ResNet [46] and FPN [47], the network contains skipped connections and 3 detection heads, respectively. These 3 detection heads (82nd, 94th, and 106th layers) detect objects (large, medium, and small) at 3 different scales (13 × 13, 26 × 26, and 52 × 52) of the image [48]. YOLOv3 uses Darknet-53[43] as the backbone network.

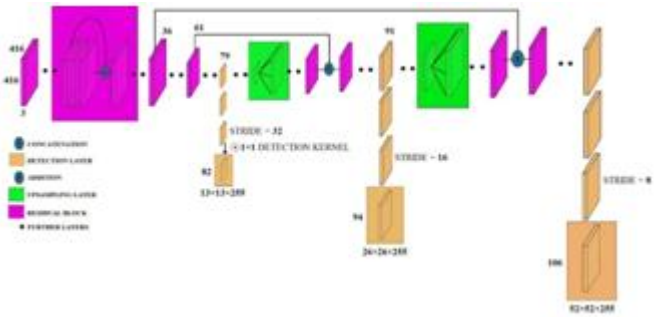

Figure 23: The YOLOv3 architecture

Training and testing

Like YOLOv2, the base network of YOLOv3 is first trained on the ImageNet dataset. For detection, the AVGPOOL layer and the SOFTMAX layer of the pre-trained network

are removed, and 53 more layers are added. The authors employed multi-scale training, extensive data augmentation, batch normalization, and other standard techniques like YOLOv2. YOLOv3 calculates the classification loss using binary cross-entropy loss rather than mean square error like YOLOv2.

The model is trained and tested on MS COCO dataset. The object detection using YOLOv3 is shown in Fig. 24.

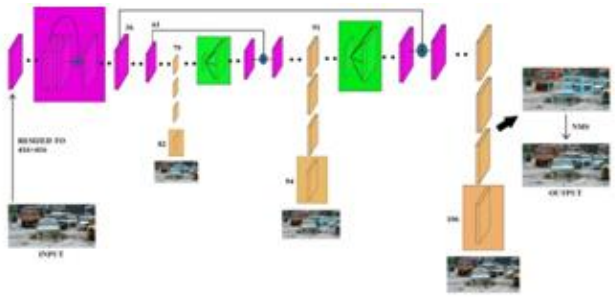

Figure 24: Object detection using YOLOv3.

2.12.3. Merits, demerits and discussion

YOLOv3 has several advantages – (i) increase in average precision for small objects, (ii) decrease in localization errors due to an increase in mAP [49], and (iii) addition of the feature pyramid method improved predictions at different scales for the same object.

YOLOv3 is comparatively slower than YOLOv2 due to its architecture and struggles to align the boxes perfectly with the object [43].

YOLOv3 can compete with the best two-stage object detection models in terms of speed and accuracy. YOLOv3 is used in object detection applications where speed is prioritized over accuracy. However, in the opposite case, it may be ineffective. On the COCO dataset, YOLOv3 achieves a 33.0% mAP@[.5, .95] [49].

2.13 EfficientDet

In 2019, the Google Brain team (Tan *et al.*) proposed a scalable object detection model [50] that followed the one-stage object detector paradigm and had superior accuracy and efficiency across a wide range of resource constraints.

2.13.1. Motivation

The brain team wanted to maximize the model accuracy under any given set of resource restrictions, since most of the previous works (like R-CNN family, YOLO family) only focused on a certain or narrow range of resource needs, but the variety of real-world applications of object detection, which are executed on several platforms, frequently needs various resource limits.

2.13.2. Work description

Overview

EfficientDet consists of 3 main components – i) backbone network, ii) feature network, and iii) detection head. Here, the backbone network is a pre-trained ConvNet for feature extraction. The feature network is used to collect feature maps from different backbone stages to build a feature pyramid. EfficientDet uses a weighted bidirectional feature pyramid network (BiFPN) [50] with cross-scale connections as shown in the feature network. The detection head is a class and box network, used to produce object class and bounding box predictions. It takes feature maps from the feature pyramid as input.

The authors use the compound scaling method [50] for object detection, which jointly scales up all dimensions of the input size, backbone network, feature network, and detection head to develop EfficientDet-D0 to D7 models.

The network architecture

EfficientDet uses EfficientNet [51] as the backbone network. The BiFPN takes level 3-7 features {P_3, P_4, P_5, P_6, P_7} from the backbone network and repeatedly applies top-down and bottom-up bidirectional feature fusion. These fused features are fed to the class and box network for final prediction (shown in Fig. 25). The class and box network weights are shared across all levels of features.

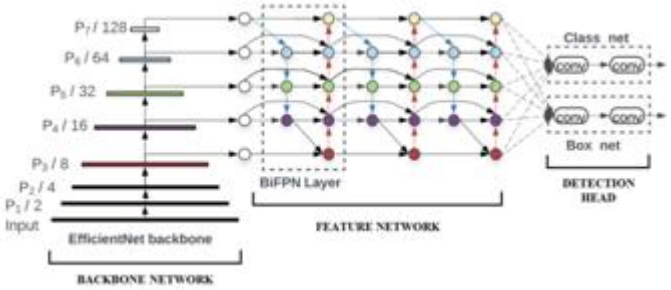

Figure 25: The EfficientDet architecture

Training and testing

The EfficientDet model is trained and tested on the MS COCO dataset (the backbone of the model is pre-trained on the ImageNet dataset). Each model (D0-D7) is trained using SGD optimizer with a momentum of 0.9. The learning rate is linearly increased from 0 to 0.16 in the first training epoch and then annealed down using the cosine decay rule. For D0-D6, each model is trained for 300 epochs on a total batch size of 128 and D7 for 600 epochs on 128. Like the previous works, data augmentation (horizontal flipping, scale jittering) is also applied during training. Soft-NMS [52] is used for evaluation.

2.13.3. Merits, demerits and discussion

EfficientDet has multiple advantages – i) smaller and lighter than previous object detectors, ii) scalable, iii) efficient and accurate across a wide variety of resource restrictions, iv) can be used in real-world object detection applications.

The model's accuracy degrades as the FPS increases [53].

EfficientDet achieves cutting-edge accuracy with much fewer parameters and FLOPs than previous object detectors (EfficientDet-D0: mAP@[.5, .95] – 33.8%, BFLOPs – 2.5B; YOLOv3: mAP@[.5, .95] – 33.0%, BFLOPs – 71B) [50].

On the COCO dataset, EfficientDet achieves a 55.1% mAP@[.5, .95].

2.14. SpineNet

Du *et al*. in the Google Brain team, in 2019 also proposed a scale-permuted backbone with cross-scale connections [54] for object detection.

2.14.1. Motivation

The scale-decreased backbone discards spatial information by down-sampling, which a decoder[1] attempts to retrieve [50]. But as the backbone layers get deeper, the features become more abstract and less localized, making it difficult for the decoder to retrieve the exact required features.

2.14.2. Work description

Overview

Unlike the previous architectures, the scales of feature maps can increase or decrease at any time in the architecture using permuting blocks, which enables the maintenance of spatial information. Moreover, the connections between feature maps are permitted to go across feature scales to perform multi-scale feature fusion.

Instead of handcrafted, Neural Architecture Search (NAS) [55] methodology is used to find an effective scale-permuted model with cross-scale connections in a given search space (scale permutations, cross-scale connections, and block adjustments) [54].

While performing cross-scale feature fusion, resampling operations in [54] are performed for dimension matching.

The network architecture

The architecture uses SpineNet, formed by first permuting the blocks of the ResNet [46] backbone and then by adding cross-scale connections using NAS. SpineNet-49 is the baseline model based on which SpineNet-49S/96/143/190 in [54] is constructed. The architecture of SpineNet is as shown in Fig. 26.

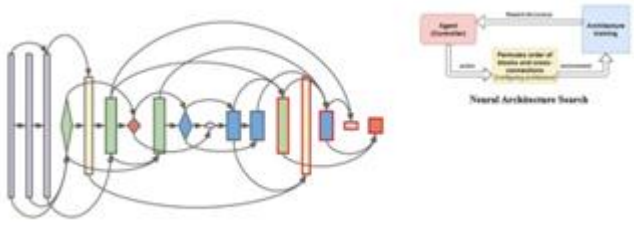

Figure 26: The SpineNet architecture [54].

Training and testing

SpineNet with RetinaNet [44] detector is used for object detection by the author(s), and the model is trained (from scratch) and tested on the MS COCO dataset. They have employed various protocols described in [54] for model training. Like the previous works, data augmentation (scale and aspect ratio augmentation, random cropping, horizontal flipping) is also applied during training. NMS is used for evaluation. The author(s) have also applied SpineNet for classification tasks. They have used ImageNet and iNaturalist classification datasets for it.

2.14.3. Merits, demerits and discussion

The use of scale-permuted blocks enables the maintenance of spatial information, and the use of cross-scale connections removes the need for a decoder.

Like EfficientDet, the model's accuracy degrades as the FPS increases [53].Using NAS to search for hyperparameters consumes more computing power.

SpineNet achieves a state-of-the-art accuracy of 52.1% mAP@[.5, .95] on the COCO dataset. It is also successful in achieving cutting-edge accuracy [54] on the image classification task.

2.15. YOLOv4

Bochkovskiy *et al*. introduced a new member in [53], in 2020, to the YOLO family that followed the design paradigm of EfficientDet.

2.15.1. Motivation

The motivation behind this work is to design for a fast operating speed of the object detection model (achieve FPS) in production systems and optimization for parallel computations, rather than the low computation volume theoretical indicator (BFLOP) [50][54].

2.15.2. Work description

Overview

Like EfficientDet, YOLOv4 also consists of 3 main components - i) backbone, ii) neck, and iii) head.

The backbone is a pre-trained ConvNet used for feature extraction. The neck is used to collect feature maps from different backbone stages to build a feature pyramid. The head is a dense prediction layer for detecting the bounding box coordinates and the class confidence score. It takes feature maps from the feature pyramid as input.

YOLOv4 uses Bag of Freebies[2] (BoF) and Bag of Specials[3] (BoS) for both the backbone and head components.

BoF for the backbone includes CutMix, Mosaic data augmentation, DropBlock regularization, and Class label smoothing. BoS for the backbone includes Mish activation, Cross-stage partial connections (CSP), and Multi-input weighted residual connections (MiWRC) [53].

[1] A network consists of a series of cross-scale connections that combine low-level and high-level features from a backbone to generate strong multi-scale feature maps [54].

[2] The methods that only change the training strategy or only increase the training cost [53].

[3] The plugin modules and post-processing methods that only increase the inference cost by a small amount but can significantly improve the accuracy of object detection [53].

BoF for the head includes CIoU-loss, CmBN, DropBlock regularization, Mosaic data augmentation, Self-Adversarial Training, Eliminate grid sensitivity, Using multiple anchors for a single ground-truth, Cosine annealing scheduler, Optimal hyperparameters, and Random training shapes. BoS for the head includes Mish activation, SPP-block, SAM-block, PAN path-aggregation block, and DIoU-NMS [53].

The network architecture

YOLOv4 uses CSPDarknet53 [56] as the backbone, SPP [28] additional module, PANet [57] path-aggregation neck, and YOLOv3 head (see Fig. 27).

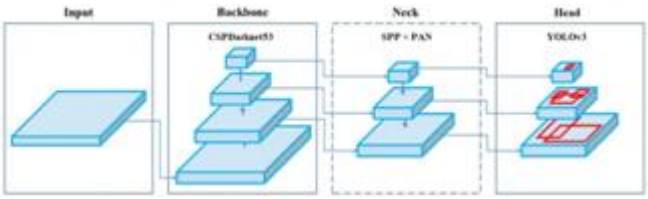

Figure 27: The YOLOv4 architecture [53].

Training and testing

The YOLOv4 model is trained and tested on the MS COCO dataset (batch size: 64, training steps: 500,500, optimizer: SGD with momentum (momentum = 0.9), learning rate: 0.01, and multiplied with a factor of 0.1 at the 400,000 and 450,000 steps, respectively). Like EfficientDet, the backbone of the YOLOv4 is also pre-trained on the ImageNet dataset. Like the previous works, multi-scale training is also employed.

2.15.3. Merits, demerits and discussion

YOLOv4 has several advantages – i) has faster FPS and is more accurate than available detectors, ii) can be trained and used on a conventional GPU enabling widespread adoption, iii) new features (BoF and BoS) improve model accuracy and may be used for other research projects.

Though YOLOv4 is considered one of the best models for speed and accuracy, it cannot top EfficientDet's largest model for overall accuracy.

YOLOv4 is superior to the fastest and most accurate detectors in terms of both speed and accuracy (YOLOv4: mAP@[.5, .95] – 43.5%, FPS – 62; EfficientDet-D2: mAP@[.5, .95] – 43.0%, FPS – 41.7 (on the COCO dataset)) [53].

2.16. PP-YOLO

In 2020, Long *et al*. proposed a modified version of YOLOv3 [58] by incorporating various tricks.

2.16.1. Motivation

To construct an object detection model with balanced effectiveness and efficiency that can be used immediately in practical applications, rather than proposing a novel one, exploring different backbone networks and data augmentation methods [53], and using NAS.

2.16.2. Work description

Overview

PP-YOLO also consists of 3 main components - i) backbone, ii) detection neck, and iii) detection head.

The backbone is a pre-trained ConvNet used for feature extraction. The detection neck is used to collect feature maps from different backbone stages to build a feature pyramid. The detection head is a dense prediction layer for detecting the bounding box coordinates and the class confidence score. It takes feature maps from the feature pyramid as input.

PP-YOLO uses various existing tricks to improve the overall performance of the model - i) Larger Batch Size, ii) Exponential Moving Average (EMA), iii) DropBlock, iv) IoU Loss, v) IoU Aware, vi) Grid Sensitive, vii) Matrix NMS, viii) CoordConv, ix) SPP, and x) Better Pre-train Model [58].

The network architecture

PP-YOLO uses ResNet50-vd-dcn [58] as the backbone, FPN [47] (with some modification(s)) as the detection neck that takes level 3-5 features C_3, C_4, C_5 from the backbone as input and output feature maps P_3, P_4, and P_5, respectively, forming a feature pyramid of level 3, and YOLOv3 detector (with some modification(s)) as the detection head that takes these feature maps for final prediction (see Fig. 28).

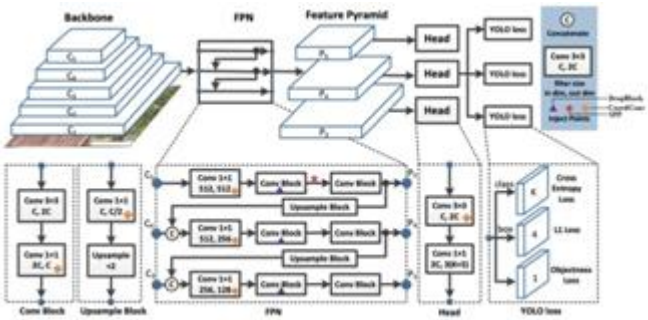

Figure 28: The PP-YOLO architecture [58].

Training and testing

PP-YOLO is trained and tested on the MS COCO dataset (batch size: 192, iterations: 250K, optimizer: SGD with momentum (momentum = 0.9), learning rate: 0.01, and divided by 10 at the 150K and 250K iterations, respectively). The backbone of the PP-YOLO is also pre-trained on the ImageNet dataset. Multi-scale training is performed, as in prior works, and only one data augmentation strategy (MixUp [59]) is used during training.

2.16.3. Merits, demerits and discussion

The tricks do not enhance the infer time, but they improve the model's overall performance and save developers' time of trial and error. PP-YOLO can be used immediately in practical applications.

Though PP-YOLO is considered one of the best models for speed and accuracy, it cannot also top EfficientDet's largest model for overall accuracy like YOLOv4.

PP-YOLO is faster and more accurate than other state-of-the-art detectors (PP-YOLO: mAP@[.5, .95] – 45.2\%, FPS – 72.9; YOLOv4: mAP@[.5, .95] – 43.5%, FPS – 62; EfficientDet-D2: mAP@[.5, .95] – 43.0%, FPS – 56.5 (on the COCO dataset) [58]).

2.17. Scaled-YOLOv4

In 2020, Wang *et al*. also proposed an improved version [60] of YOLOv4 by integrating model scaling methods.

2.17.1. Motivation

To develop a network scaling approach that modifies not only the depth, width, and resolution of the network [50] but also its structure, which can improve learning capability and hence increase accuracy while reducing the amount of computation and memory requirements.

2.17.2. Work description

Overview

YOLOv4 is re-designed to YOLOv4-CSP using the concepts laid out in [56], and then based on YOLOv4-CSP Scaled-YOLOv4 is developed (YOLOv4 → YOLOv4-CSP → YOLOv4-P5 → YOLOv4-P6 → YOLOv4-P7) using optimal network scaling techniques [60].

As the backbone (CSPDarknet53 [56]), the neck (PAN [57]) also uses CSP connections.

In contrast to YOLOv4, where just one network was trained for all resolutions, Scaled-YOLOv4 trains a distinct network for each resolution.

The author designed Scaled-YOLOv4 for general GPUs (YOLOv4-CSP), low-end GPUs (YOLOv4-tiny), and high-end GPUs (YOLOv4-large).

The network architecture

The architecture of Scaled-YOLOv4 is shown in Fig. 29.

Figure 29: The Scaled-YOLOv4 architecture [60]

Training and testing

The Scaled-YOLOv4 model is trained and tested on the MS COCO dataset. Unlike earlier detectors, ImageNet pre-trained models are not employed in this detector; instead, all Scaled-YOLOv4 models are built from scratch (optimizer: SGD, epochs: 300 (YOLOv4-CSP), 600 (YOLOv4-tiny), 450 (YOLOv4-large)). The authors also performed TTA (Test Time Augmentation).

2.17.3. Merits, demerits and discussion

Scaled-YOLOv4 has multiple advantages – i) increase in accuracy while decreasing computation and memory needs, ii) can be deployed on general, low-end, and high-end devices.

Scaled-YOLOv4 outperforms the fastest and most accurate detectors in terms of both speed and accuracy (Scaled-YOLOv4: mAP@[.5, .95] – 55.5%, FPS – ~16; EfficientDet: mAP@[.5, .95] – 55.1%, FPS – ~6 (on the COCO dataset) [60]) and overcomes the drawback of YOLOv4.

2.18. PP-YOLOv2

In 2021, Huang *et al*. proposed an improved version [61] of PP-YOLO.

2.18.1. Motivation

To achieve a better balance between effectiveness and efficiency.

2.18.2. Work description

Overview

PP-YOLOv2 only uses additional tricks (existing) over PP-YOLO - i) Mish Activation Function, ii) Larger Input Size, and iii) IoU Aware Branch [61].

The network architecture

PP-YOLOv2 follows the architecture of PP-YOLO but uses PAN [57] (with some modification(s)) instead of FPN as the detection neck (see Fig. 30).

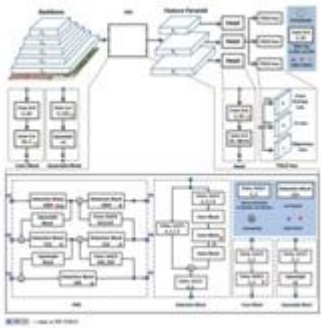

Figure 30: The PP-YOLOv2 architecture [61]

Training and testing

Like PP-YOLO, PP-YOLOv2 is trained and tested on the MS COCO dataset using the same strategy, but during multi-scale training, 320 to 768 pixels is applied instead of 320 to 608 pixels.

2.18.3. Merits, demerits and discussion

By combining multiple effective refinements, PP-YOLOv2 achieves a better balance between speed and accuracy than other famous detectors.

Though PP-YOLOv2 outperforms PP-YOLO in terms of accuracy, it cannot outperform in terms of speed.

PP-YOLOv2 achieves a mAP@[.5, .95] of 49.5% (+~4% than PP-YOLO, +~6% than YOLOv4) on the COCO dataset with an FPS of 68.9 (-~4 than PP-YOLO, +~7 than YOLOv4) [61].

A good survey paper [62] on object detection (using deep learning-based techniques) is also written by Zaidi *et al*. The survey is helpful to all the researchers in the computer vision community. It includes some important comparative results in this domain.

3. Discussion

Fig. 31 shows a comparative study among some of the reviewed cutting-edge object detection models in terms of their accuracy. According to the study, the YOLO family object detection models outperform the others and can be used in real-world object detection applications and other research projects.

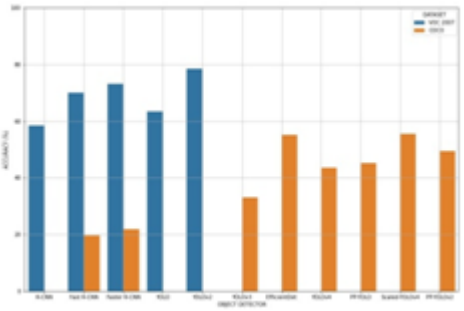

Figure 31: A comparative study among some of the reviewed cutting-edge object detection models

4. Conclusion

In this paper, we made some state-of-the-art reviews. The papers are analysed and the underlying concepts are highlighted and written concisely so that readers can obtain a brief summary of each of the articles. The merits of the article are also indicated. As a result, readers can focus to their work straightaway.

Acknowledgment
The authors would like to acknowledge Techno India University, West Bengal for its support to this work.

References

[1] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001, vol. 1, pp. I–I, 2001.

[2] R. Patil, "Viola jones face detection explained @ONLINE." https://www.youtube.com/watch?v=_QZLbR67fUU, May 2014.

[3] Y. Shakrina, "The viola-jones algorithm @ONLINE." https://www.youtube.com/watch?v=p9vq90NYHMs, Apr 2020.

[4] P. Irgens, C. Bader, T. L´e, D. Saxena, and C. Ababei, "An efficient and cost effective fpga based implementation of the viola-jones face detection algorithm," HardwareX, vol. 1, 03 2017.

[5] K. Aashish and A. Vijayalakshmi, "Comparison of viola-jones and kanade-lucas-tomasi face detection algorithms," Oriental journal of computer science and technology, vol. 10, pp. 151–159, 2017.

[6] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), vol. 1, pp. 886–893, 2005.

[7] S. Mallick, "Histogram of oriented gradients @ONLINE." https://www.learnopencv.com/histogram-of-oriented-gradients/, December 2016.

[8] Cogneethi, "C37 dalal & triggs object detection hog + svm computer vision machine learning evodn @ONLINE." https://www.youtube.com/watch?v=sDByl84n5mY&list=PL1GQaVhO4f_jLxOokW7CS5kY_J1t1T17S, Aug 2019.

[9] A. Mohan, C. Papageorgiou, and T. Poggio, "Example-based object detection in images by components," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 23, no. 4, pp. 349–361, 2001.

[10] P. Viola, M. J. Jones, and D. Snow, "Detecting pedestrians using patterns of motion and appearance," International Journal of Computer Vision, vol. 63, pp. 153–161, 2005.

[11] U. CRCV, "Lecture 18- deformable part models (dpm) -2014 @ONLINE." https://www.youtube.com/watch?v=2DihVLm8v38&list=PLd3hlSJsX_ImKP68wfKZJVIPTd8Ie5u-9&index=19&t=0s, Jan 2015.

[12] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 32, no. 9, pp. 1627–1645, 2010.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in Advances in Neural Information Processing Systems 25, pp. 1097–1105, Curran Associates, Inc., 2012.

[14] D. Gerónimo, A. M. López, A. D. Sappa, and T. Graf, "Survey of pedestrian detection for advanced driver assistance systems," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 32, no. 7, pp. 1239–1258, 2010.

[15] S. Nayak, "Understanding alexnet @ONLINE." https://www.learnopencv.com/understanding-alexnet/, June 2018.

[16] 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, 2014.

[17] Cogneethi, "C 5.2 convnet input size constraints cnn object detection machine learning evodn @ONLINE." https://www.youtube.com/watch?v=Mb79KKdluYI&list=PL1GQaVhO4f_jLxOokW7CS5kY_J1t1T17S, Aug 2019.

[18] Cogneethi, "C 5.4 overfeat intuition important-dont skip cnn object detection machine learning evodn @ON- LINE."

https://www.youtube.com/watch?v=t5PHp8uSMKo&list=PL1GQaVhO4f_jLxOokW7CS5kY_J1t1T17S, Aug 2019.
[19] C. Deledalle, "Mlip - chapter 5 - detection, segmentation, captioning @ONLINE." https://www.slideshare.net/ CharlesDeledalle/mlip-chapter-5-detection-segmentation-captioning, Mar 2019.
[20] Cogneethi, "C 5.6 overfeat network design important-dont skip cnn object detection evodn @ONLINE." https://www.youtube.com/watch?v=JKTzkcaWfuk&list=PL1GQaVhO4f_jLxOokW7CS5kY_J1t1T17S, Aug 2019.
[21] Cogneethi, "C 5.5 overfeat effective stride important-dont skip cnn object detection evodn @ONLINE." https://www.youtube.com/watch?v=50-PhoCJEOk&list=PL1GQaVhO4f_jLxOokW7CS5kY_J1t1T17S, Aug 2019.
[22] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2014.
[23] J. Uijlings, K. van de Sande, T. Gevers, and A. Smeulders, "Selective search for object recognition," International Journal of Computer Vision, 2013.
[24] Cogneethi, "C 6.3 rcnn network architecture cnn machine learning object detection evodn @ONLINE." https://www.youtube.com/watch?v=x_TGsU9_vcc&list=PL1GQaVhO4f_jLxOokW7CS5kY_J1t1T17S, Aug 2019.
[25] S. Ananth, "R-cnn for object detection - a technical paper summary @ONLINE." https://towardsdatascience. com/r-cnn-for-object-detection-a-technical-summary-9e7bfa8a557c, April 2019.
[26] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," CoRR, vol. abs/1409.1556, 2015.
[27] R. Girshick, "Fast r-cnn," in The IEEE International Conference on Computer Vision (ICCV), December 2015.
[28] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 37, no. 9, pp. 1904–1916, 2015.
[29] Cogneethi, "C 7.5 roi projection subsampling ratio sppnet fast rcnn cnn machine learning evodn @ONLINE." https://www.youtube.com/watch?v=wGa6ddEXg7w&list=PL1GQaVhO4f_jLxOokW7CS5kY_J1t1T17S, Aug 2019.
[30] S. Elfouly, "Introduction: Fast r-cnn (object detection) @ONLINE." https://medium.com/@selfouly/ part-2-fast-r-cnn-object-detection-7303e1988464, Jul 2019.
[31] S. Ananth, "Fast r-cnn for object detection - a technical paper summary @ONLINE." https://towardsdatascience.com/fast-r-cnn-for-object-detection-a-technical-summary-a0ff94faa022, Aug 2019.
[32] R. Gandhi, "R-cnn, fast r-cnn, faster r-cnn, yolo — object detection algorithms @ONLINE." https:// towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e, July 2018.
[33] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in Advances in Neural Information Processing Systems 28, pp. 91–99, Curran Associates, Inc., 2015.
[34] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in ECCV, 2014.
[35] S. Ananth, "Faster r-cnn for object detection - a technical paper summary @ONLINE." https:// towardsdatascience.com/faster-r-cnn-for-object-detection-a-technical-summary-474c5b857b46, Aug 2019.
[36] C. L. Zitnick and P. Doll´ar, "Edge boxes: Locating object proposals from edges," in Computer Vision – ECCV 2014, pp. 391–405, Springer International Publishing, 2014.
[37] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2016.
[38] M. Menegaz, "Understanding yolo @ONLINE." https://hackernoon.com/understanding-yolo-f5a74bbc7967, March 2018.
[39] J. Hui, "Real-time object detection with yolo, yolov2 and now yolov3 @ONLINE." https://medium.com/@jonathan_ hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088, March 2018.
[40] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," in The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), July 2017.
[41] P. Sought, "Yolo-v2 loss function understanding." https://www.programmersought.com/article/78354890194/.
[42] J. C. Redmon, "Yolo: Real-time object detection @ONLINE." https://pjreddie.com/darknet/yolov2/.
[43] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," ArXiv, vol. abs/1804.02767, 2018.
[44] T. Lin, P. Goyal, R. B. Girshick, K. He, and P. Doll´ar, "Focal loss for dense object detection," CoRR, vol. abs/1708.02002, 2017.
[45] 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.
[46] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," CoRR, vol. abs/1512.03385, 2015.
[47] T. Lin, P. Doll´ar, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie, "Feature pyramid networks for object detection," CoRR, vol. abs/1612.03144, 2016.
[48] A. Kathuria, "What's new in yolov3? @ONLINE." https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b, April 2018.
[49] J. C. Redmon, "Yolo: Real-time object detection @ONLINE." https://pjreddie.com/darknet/yolo/.

[50] M. Tan, R. Pang, and Q. V. Le, "Efficientdet: Scalable and efficient object detection," CoRR, vol. abs/1911.09070, 2019.
[51] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," CoRR, vol. abs/1905.11946, 2019.
[52] N. Bodla, B. Singh, R. Chellappa, and L. S. Davis, "Soft-nms – improving object detection with one line of code," 2017.
[53] A. Bochkovskiy, C. Wang, and H. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," CoRR, vol. abs/2004.10934, 2020.
[54] X. Du, T. Lin, P. Jin, G. Ghiasi, M. Tan, Y. Cui, Q. V. Le, and X. Song, "Spinenet: Learning scale-permuted backbone for recognition and localization," CoRR, vol. abs/1912.05027, 2019.
[55] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," CoRR, vol. abs/1611.01578, 2016.
[56] C. Wang, H. M. Liao, I. Yeh, Y. Wu, P. Chen, and J. Hsieh, "Cspnet: A new backbone that can enhance learning capability of CNN," CoRR, vol. abs/1911.11929, 2019.
[57] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia, "Path aggregation network for instance segmentation," CoRR, vol. abs/1803.01534, 2018.
[58] X. Long, K. Deng, G. Wang, Y. Zhang, Q. Dang, Y. Gao, H. Shen, J. Ren, S. Han, E. Ding, and S. Wen, "PP-YOLO: an effective and efficient implementation of object detector," CoRR, vol. abs/2007.12099, 2020.
[59] H. Zhang, M. Cissé, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond empirical risk minimization," CoRR, vol. abs/1710.09412, 2017.
[60] C. Wang, A. Bochkovskiy, and H. M. Liao, "Scaled-yolov4: Scaling cross stage partial network," CoRR, vol. abs/2011.08036, 2020.
[61] X. Huang, X. Wang, W. Lv, X. Bai, X. Long, K. Deng, Q. Dang, S. Han, Q. Liu, X. Hu, D. Yu, Y. Ma, and O. Yoshie, "Pp-yolov2: A practical object detector," CoRR, vol. abs/2104.10419, 2021.
[62] S. S. A. Zaidi, M. S. Ansari, A. Aslam, N. Kanwal, M. Asghar, and B. Lee, "A survey of modern deep learning based object detection models," Digital Signal Processing, vol. 126, p. 103514, 2022.