Seminar Computer Vision by Deep Learning Classification with Localization and Single-stage object detectors

E. Geenjaar 4373480
P. de Rijk 4375017
S. Tak 4975154

Abstract

This chapter aims to establish an understanding of classification with localization, and extend that understanding to single-stage detectors. The first part of this chapter will cover the concept of classification as performed by contemporary computer vision systems, and demonstrate how this can be extended to include localization. Additionally, the concept of a sliding window detector will be elaborated on. Building upon this knowledge, single-stage detectors are discussed with a focus on YOLO. The basic principle of YOLO is to approach object detection as a single regression problem, where image pixels are translated to bounding box coordinates and class probabilities in one stage. During testing, only the best predictions are kept using non-maximal suppression. The limitations of YOLO are discussed after which solutions to these limitations are examined. The solutions include anchor boxes and usage of multiple feature map scales. Finally, real-world applications and controversies surrounding YOLO are covered.

1 Classification with Localization

We are already familiar with the concept of object detection, introduced in the previous chapter (5.1). There are however two main tasks that precede object detection: **classification** and **classification with localization**. The distinction between these tasks is illustrated in figure 1 and can be formulated as follows: In a classification task the central question that needs answering is: *Is there a dog in the image?* The classification with localization task extends this question to *Is there a dog and wherein the image is it located?* Finally, with object detection, the question becomes: *Are there any objects and wherein the image are they located?* This section will focus on answering the first two questions, by first explaining how we can achieve a model that performs image classification, and subsequently how this model can be extended to include localization. These concepts will form the basis for section 2, which discusses state-of the art single-stage object detection methods.

TU Delft Seminar Computer Vision by Deep Learning (2020).

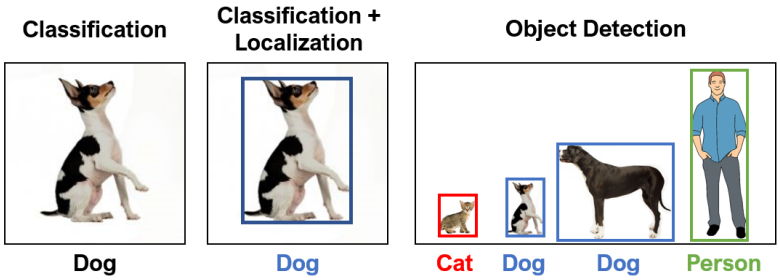


Figure 1: Overview comparison between Classification, Classification with localization and Object Detection

1.1 Classification

Image classification is one of the core tasks of computer vision. It forms the basis for other computer vision tasks such as localization, detection, and segmentation [7]. Classification can be described as:

"The mapping from an input x to an output class in y, which consists of k categories, through a function $y = f(x)$."

This definition is exemplified using figure 2: The input image $\boldsymbol{x}$ is represented as a $100 \times 100 \times 3$ array, where the first two dimensions are the height and width, and the third dimension represents the three color channels (red, green and blue) respectively. The array contains values for the pixel intensities that range from 0 to 255. The output $\boldsymbol{y}$ is represented as a vector containing the probabilities for one of $k = 4$ categories: $\{dog, cat, horse, car\}$. A good classifier $\boldsymbol{f}$ will be able to map the values in the input image to the correct label for that image.

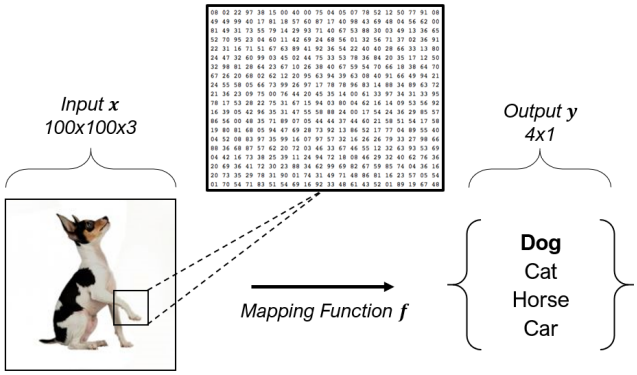


Figure 2: A good classifier can map an input image $\boldsymbol{x}$ in the form of an array to a correct label contained in the output vector $\boldsymbol{y}$. Inspired by Karpathy [7]

Even though this classification task may seem trivial to a human, it can pose quite a lot of challenges for a computer vision system. The algorithm only "sees" a large array of numbers, and it needs to translate this array into the *semantic* idea of a, for example, a dog. Complications discussed in chapter 1.1, such as viewpoint-dependent object variability and the high in-class variability of object types [3] make it hard for a classifier to *robustly* map each image of a certain object to its correct label: There are many types of dogs; small dogs, large dogs, white dogs, brown dogs, etc. Dogs can also be captured on an image from many viewpoints. All these variations make it hard for a classifier to map the image array it receives as an input to the semantic label "dog". This problem of translating a numerical array to the semantic classification of an object is also referred to as the **semantic gap** [7].

1.1.1 Closing the Semantic Gap

So how do we close the semantic gap? Traditionally, a dual-stage approach was used, where handcrafted features were extracted from images using *feature descriptors* such as edge detectors and corner detectors. These features were then used as the input to a trainable classifier [13], as shown in figure 3. This approach suffered from the fact that the accuracy was highly dependent on the design of the feature extraction stage, which in itself has shown to be a very challenging task [9].

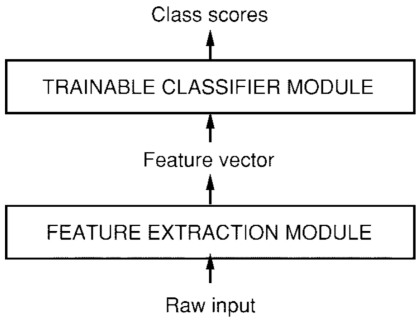


Figure 3: Traditional pattern recognition is performed with two modules: a fixed feature extractor and a trainable classifier. Taken from LeCun et al. [9]

The increase in computing power and the advent of larger datasets have enabled the adoption of convolutional neural networks (ConvNets) in the field of computer vision. These ConvNets have driven progress on classification tasks. The most significant advancement was achieved in the 2012 ImageNet Large Scale Visual Recognition Challenge [18]. A new ConvNet proposed by Krizhevsky et al. [8] beat the competition by an error margin of over 10%, establishing the dominance of ConvNets on classification tasks for the coming decade.

1.1.2 Feature maps

So how do ConvNets differ from the traditional methods which used manually crafted features to classify an image? The main benefit arises from the fact that ConvNets can learn this feature map by training on labelled data. **Feature maps** in ConvNets are 3-dimensional arrays, where the first two dimensions are spatial dimensions and the third dimension contains the features for each spatial location in the feature map. An example of a feature map is shown in figure 4. The network that is used to produce the feature map is called a **backbone network**. The features in a feature map are learned in such a way that they can be used to distinguish between the different class labels in a classification task.

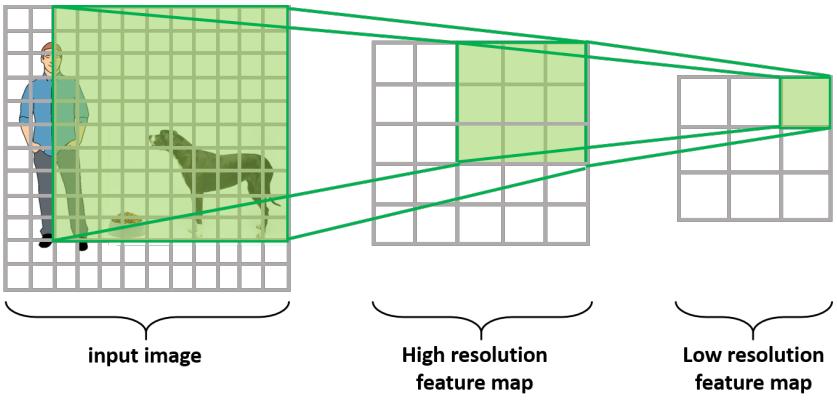


Figure 4: An example of a feature map, a simplified backbone network and the receptive field of the feature map

Throughout this chapter, we will refer to the spatial locations in a feature map as **pixels** in the feature map. A high spatial resolution of the feature map indicates a large number of pixels in the feature map and a low spatial resolution to a small number of pixels in the feature map. A high feature resolution refers to a large number of features (the third dimension) of a feature map, a small feature resolution refers to a small number of features in the feature map. The feature map has a lower spatial resolution than the input image and thus fewer pixels. These feature map pixels do however extract information from a very large part of the image. The part of the input image a pixel in the feature map can 'see' is referred to as the **receptive field** of that feature map pixel. The receptive field of a feature map pixel is often just shy of the whole image. Figure 4 shows an example of a feature map's receptive field. Two important factors that determine the receptive field of a feature map pixel are the *number of layers* in a backbone network and the *dilation rate* used in those layers. The backbone networks used for computer vision classification tasks are often ConvNets. An example of a feature map, a simplified backbone network and the receptive field of the feature map is shown in figure 4

To create classifications out of the feature map, a fully connected layer is used. A flattened version of the feature map is passed through a fully connected layer, which learns the mapping from the feature map to the class labels. The fully connected layer will push feature maps to contain information that is important for the classification of the image and these two parts can be trained end-to-end. **End-to-end** training is where all parameters of the network are trained in conjunction. The backbone network and the fully connected layer together are the classifier that can map an input x to an output class y. The number of output neurons of the fully connected layer is equal to the number of classes k we want to predict. Each neuron corresponds to a certain class label and will predict an unnormalized score for that class label. Figure 5 illustrates an example of the classification pipeline.

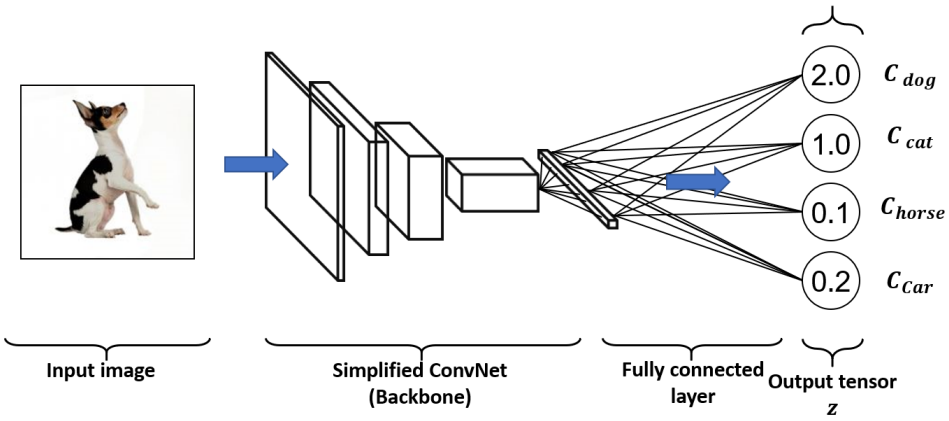


Figure 5: An example of the classification pipeline. The input image is fed through a backbone network containing several convolutional layers, which are followed by a flattening layer and a fully connected layer resulting in the class scores $\boldsymbol{z}$

1.1.3 Measuring Classification Performance

The problem with the fully connected layer in our classifier is that it will provide predictions for each class that are not normalized and can range between $[-\infty, \infty]$. These unnormalized predictions are also referred to as **logit scores** and are the outputs of the fully connected layer. To make the outputs of the classifier more intuitive, we want to map them to a probability for each class. In order to do that we can use the **Softmax** function, which is defined as follows:

$$Softmax(\mathbf{z}) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} = \hat{\boldsymbol{y}} \tag{1}$$

Where $\boldsymbol{z}$ is a vector of logit scores for each class, i is the index of the class the classifier is predicting. The sum in the denominator sums over each logit score to make sure that the probabilities for each class sum to 1. We refer to the output of the softmax function as $\hat{\boldsymbol{y}}$. Figure 6 demonstrates an example of a mapping between the logit scores and the probabilities that are assigned to each class.

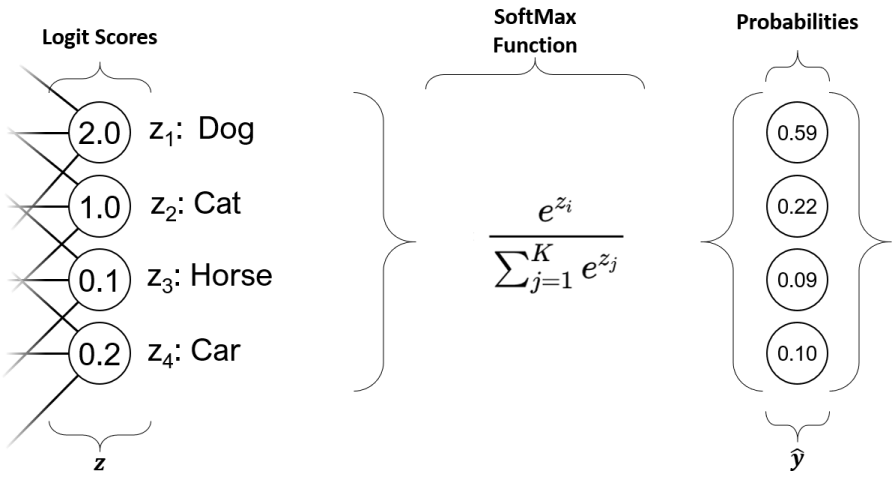


Figure 6: An example of how logit scores (left) are mapped to normalized probabilities (right) for each class using the softmax function (middle)

The loss function for classifiers can subsequently be defined by taking the difference between the ground truth class label y_i for an image and the discrete probability distribution that is given by the softmax. This is generally done using the cross entropy loss, which measures the difference between two probability distributions:

$$\mathcal{L}_{\text{cross-entropy}}\left(\hat{\mathbf{y}}, \mathbf{y}\right) = -\sum_{i} y_i \log\left(\hat{y}_i\right) \tag{2}$$

Figure 7 illustrates the cross-entropy loss applied to our example network.

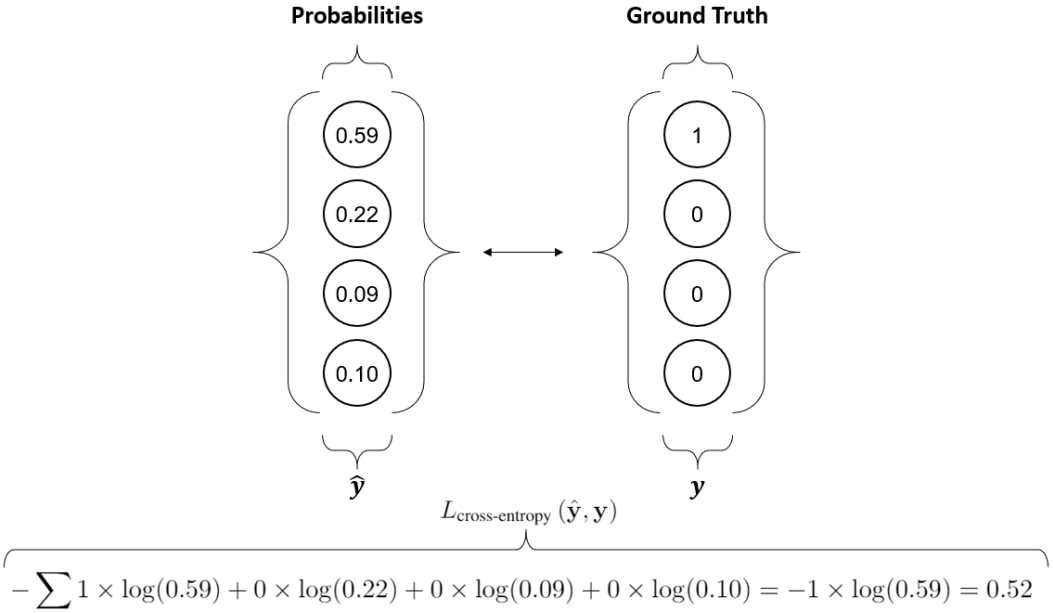


Figure 7: An example of how the cross-entropy loss is calculated on normalized probabilities for each class (left) and the ground truth class label (right)

The normalized probabilities that are predicted by the classifier are a discrete probability density function. The ground truth label is also a probability probability density function. The larger the difference between the predicted probability density function and the ground truth probability density function, the larger the gradients that flow back into the classifier.

1.2 Extending to Localization

In the previous subsection, we discussed image classification, where an image label is assigned to an input image. Assume we do not only want to classify an image as a "dog", but also want to know where it is located, we need to extend the classifier to be able to localize the dog. It turns out that the simplest way to localize an object in an image is by constructing a **bounding box** around the object. The bounding box is defined by 4 variables: b_x, b_y, b_w, b_h. These variables respectively represent the x and y position of the centre of the bounding box, the width, and the height. To make it easier to talk about the location of a bounding box in an image, we normalize the width and height of an image to [0, 1]. The left uppermost location of an image becomes $(0, 0)$ and the right bottom part becomes $(1, 1)$. Figure 8 illustrates an example of a bounding box.

Note that we are still only discussing classification and localization of *one* object in an image.

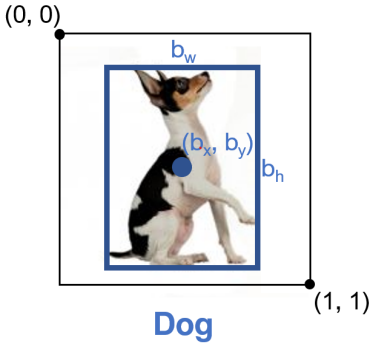


Figure 8: Bounding box around the object 'dog' including its centre and dimensions

So how do we add this additional localization task to our model? It turns out that a lot of the architecture used in the image classification model can be reused: Again, we feed the input image through our backbone network which in turn generates a feature map that summarizes the content of the image. We also use a fully-connected layer, but we add 5 output neurons that will learn to predict b_x, b_y, b_w, b_h and P_c. Where P_c is the probability that there is an object in the image. This probability is referred to as the **object score**. From now on we will refer to the classes that we can detect as C_i, where i indexes the k classes. Figure 9 illustrates what our example network would look like when it is extended to also handle localization.

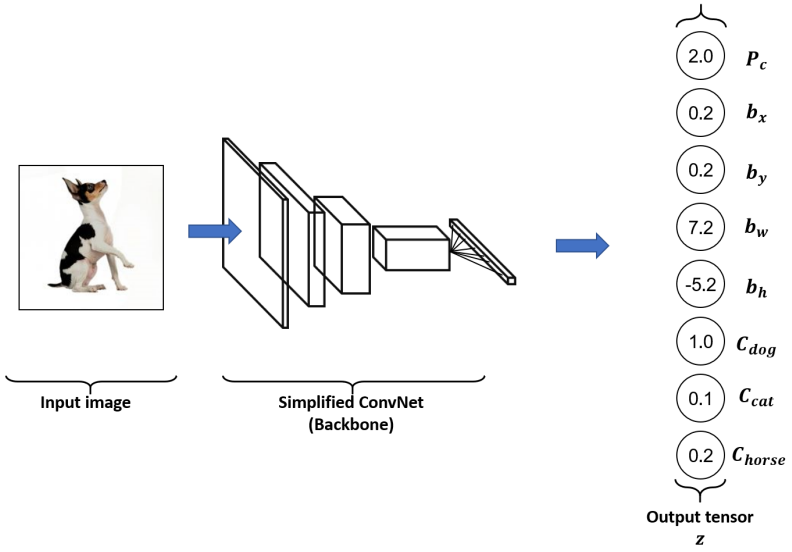


Figure 9: The classification with localization pipeline. The input image is fed into a backbone network and mapped onto an output tensor z by a fully-connected layer

1.2.1 Measuring Localization Performance

Classifiers are generally evaluated on their classification accuracy, which is the number of correctly classified samples divided by the total number of classified samples. This is not an adequate metric for classification and localization, because the metric does not measure how well the object detector localizes the object. It also does not take into account that there could be multiple objects in an image. **Intersection over Union** (IoU) $\in [0, 1]$ is a metric that can be used to determine the quality of the bounding box in localizing the object. The IoU is defined as follows:

$$\text{IoU} = \frac{\text{Pixels in ground truth bounding box} \cap \text{Pixels in predicted bounding box}}{\text{Pixels in ground truth bounding box} \cup \text{Pixels in predicted bounding box}} \tag{3}$$

Where $\cap$ indicates the intersection. Hence, the numerator in equation 3 refers to the intersection between the pixels in the ground truth bounding box and the pixels in the predicted bounding box. $\cup$ refers to the union, so the denominator in equation 3 refers to the union of the pixels in the ground truth bounding box and the pixels in the predicted bounding box. The formula is visualized in figure 10

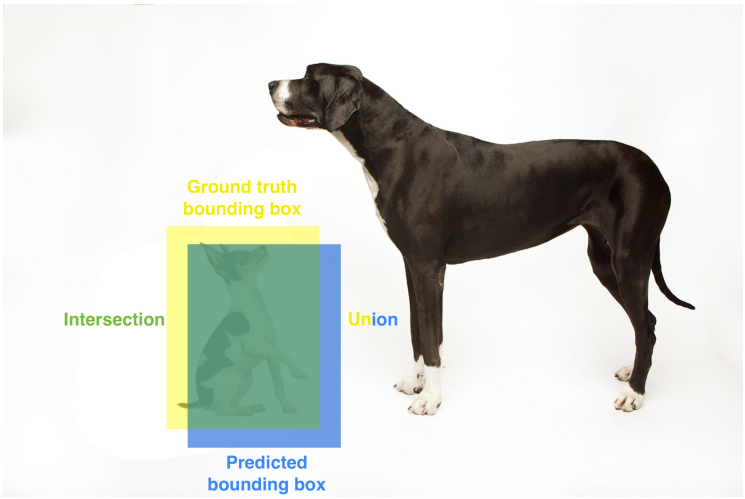


Figure 10: An example of the intersection over union metric. The ground truth bounding box is displayed in yellow, the predicted bounding box is shown in blue. The intersection between the two boxes is shown in green and the union is the area of the yellow, the blue and the green box

The IoU metric can then be used to determine whether a predicted bounding box is correct using a threshold $\tau \in [0, 1]$. A predicted bounding box is then considered correct if IoU $\geq \tau$. A common value for τ is 0.5, which means that at least 50% of the pixels of both the predicted bounding box and the ground truth bounding box have to overlap.

1.2.2 Defining the Loss

The localization of the bounding boxes can not be evaluated using a softmax activation on the fully-connected layers and a cross-entropy loss, as defined in section 1.1.3. We thus need to use an alternative loss function for the prediction of the bounding box. The fully-connected layer will provide logit scores for the centre, width, and height of the bounding box. These logit scores are however again unconstrained and not between 0 and 1. The values that represent the bounding box are therefore mapped between 0 and 1 using a sigmoid function to mitigate this. The sigmoid function is defined as follows.

$$S(z_i) = \frac{1}{1 + e^{-z_i}} \tag{4}$$

Where z is a vector with the logit scores for b_x, b_y, b_w and b_h. $S(z_i)$ is the value for those logit scores normalized to one, where i indexes each logit score. Figure 11 illustrates how a sigmoid layer is applied on our example network.

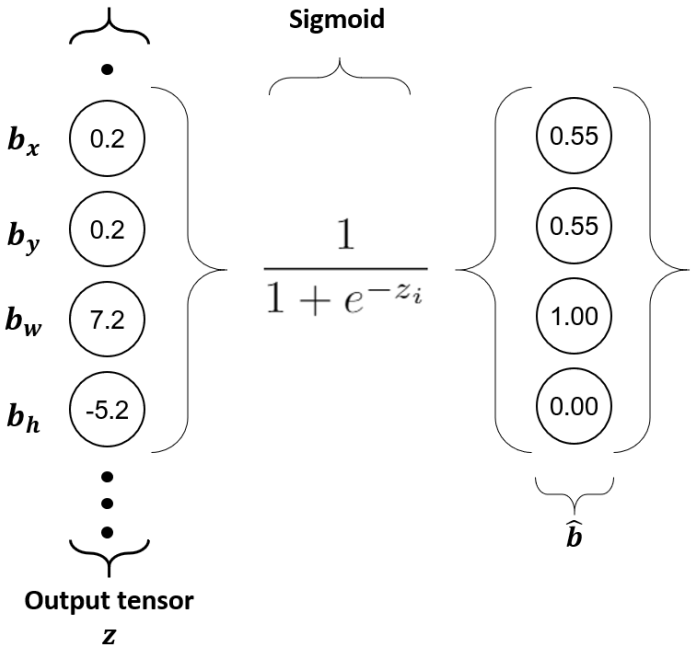


Figure 11: An example of how a sigmoid layer is used to transform b_x, b_y, b_w and b_h to remain within image bounds. Logit scores (left) are mapped to values 'squashed' between (0,1).

After applying the sigmoid function, we end up with the networks estimated values for $\hat{b}_x, \hat{b}_y, \hat{b}_w, \hat{b}_h$, which we will call the $\hat{\mathbf{b}}$ vector. Using $\hat{\mathbf{b}}$ and the ground truth vector $\mathbf{b}$ we can now calculate the difference between those vectors. Some conventional functions used to accomplish this are the L1 Loss, L2 loss, or the smooth L1 loss, defined in equations 5a, 5b and 5c respectively.

$$\mathcal{L}_{\text{L1}}(\hat{\mathbf{b}}, \mathbf{b}) = \sum_{i=1}^{4} \left| b_i - \hat{b}_i \right| \tag{5a}$$

$$\mathcal{L}_{\text{L2}}(\hat{\mathbf{b}}, \mathbf{b}) = \sum_{i=1}^{4} \left(b_i - \hat{b}_i \right)^2 \tag{5b}$$

$$\mathcal{L}_{1,smooth} = \begin{cases} \left| b_i - \hat{b}_i \right| & \text{if } \left| b_i - \hat{b}_i \right| > 1 \\ \left(b_i - \hat{b}_i \right)^2 & \text{if } \left| b_i - \hat{b}_i \right| \leq 1 \end{cases} \tag{5c}$$

A much more common measure for single-stage detectors however is to use the earlier explained IoU as a loss function. The IoU is scale invariant, unlike L1 and L2 loss. We can use network prediction $\hat{\boldsymbol{b}}$ and the ground truth bounding box $\boldsymbol{b}$ to calculate the IoU, and consequently calculate the loss as follows:

$$\mathcal{L}_{\text{IoU}} = 1 - \text{IoU} \tag{6}$$

To calculate the total loss of the network, we calculate the cross-entropy loss for P_c and our class scores C_i and add them to the IoU loss as follows:

$$\mathcal{L}_{\text{Total}} = \mathcal{L}_{\text{IoU}} + \mathcal{L}_{\text{CE,P}_\text{c}} + \sum_{i=1}^{k} \mathcal{L}_{\text{CE,C}_\text{i}} \tag{7}$$

1.2.3 Overall performance

The performance can now be generalized to the localization of each object in image by introducing two more metrics, as follows:

$$\text{Precision} = \frac{\text{Number of correctly predicted bounding boxes}}{\text{Number of ground truth bounding boxes}} \tag{8a}$$

$$\text{Recall} = \frac{\text{Number of correctly predicted bounding boxes}}{\text{Number of predicted bounding boxes}} \tag{8b}$$

Where **precision** measures the fraction of correctly predicted bounding boxes relative to the number of ground truth boxes. **Recall** measures the fraction of correctly predicted bounding boxes relative to the total number of predicted bounding boxes. A smaller number of ground truth bounding boxes will increase the precision. A smaller number of predicted bounding boxes by the object detector will increase the recall. This creates a trade-off between the precision and the recall of an object detector. This is why these two metrics are taken together and plotted against each other, like in figure 12.

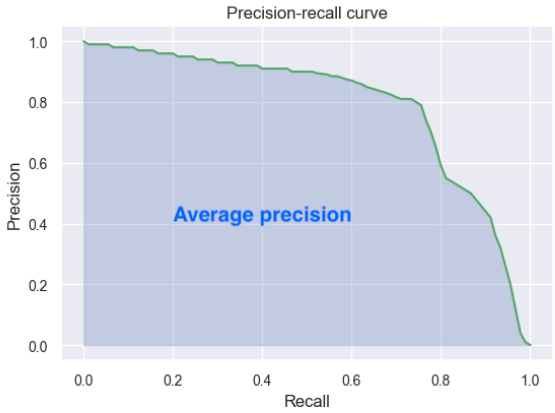


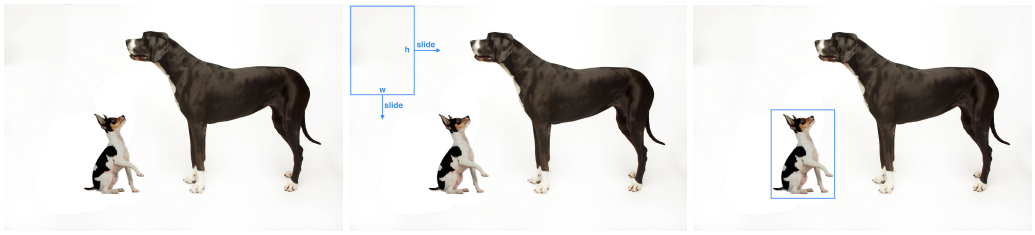
Figure 12: An example of a possible precision-recall curve. The precision is on the y-axis and the recall is on the x-axis. The precision-recall line is the green line in the figure and the transparent blue area under the green curve is the average precision

The area under the curve in figure 12 is called the **Average Precision** (AP) and is often used as a metric for object detectors. It takes the trade-off between the precision and recall into account. The AP only measures the performance of one class of the object detector. A version of the AP that is often used is the **mean Average Precision** (mAP), which is the average AP for all the classes the object detector can detect.

1.3 Sliding window detection

The original concept of classification and localization can only detect one object in an image, even if there are multiple objects present in that image. One way of trying to detect multiple objects in one image is to use a sliding window detector.

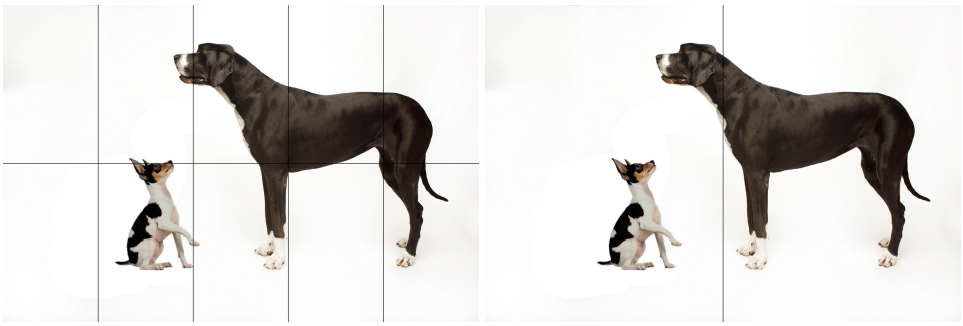
A sliding window detector uses the idea that a window with a certain shape and size can be slid over an image to detect objects in that image. Let's start simple, we have decided we want to detect the small dog in figure 13a. In this example, we again assume that we have a classifier $f(x)$ that can detect a dog in an image. We define a **sliding window** with a width w and a height h and place it in the top left corner of the image. The size of the window is generally *fixed*. The window 'slides' over the image to the right, with a step size dw until it hits the border on the right of the image. The window is then moved down with a certain step size dhs. The window is then slid horizontally again until it hits the border and this process is repeated until the window has been slid over the entire image. Each location of the window can be interpreted as a separate image that can be provided as input to the classifier. We get a predicted class and location for each window location and can thus classify and localize multiple objects in an image.



(a) Original image (b) Sliding window (c) Detection

Figure 13: A sliding window detection example. (a) is the original image, (b) shows a sliding window and (c) shows a location of the sliding window that allows us to detect the smaller dog

There are a few disadvantages to sliding a fixed size window over an image, however. First, a small step size in the horizontal and vertical direction can lead to many different window locations. This is computationally expensive. Small step size can also lead to the classification and localization of one object across many different window locations. The step sizes dh and dw are therefore generally chosen to be the same as the width (w) and height (w) of the sliding window. This leads to a grid that is placed over the image, where each cell is a window, like in figure 14a. The size of the window in figure 14a makes it hard to detect the larger dog. If we increase the window size, however, like in figure 14b, we can detect the large dog, but it has now become impossible to detect more than 2 dogs because we only have 2 windows.



(a) (b)

Figure 14: Windows as a grid, where the step size horizontally is equal to the width of the window. The vertical step size is equal to the height of the window. In image (a), a smaller window size is used, this may make it hard to detect the larger dog. In figure (b), a larger window size is used, but it is now impossible to detect more than 2 dogs in an image

More objects in the picture at different scales would mean that we need to place multiple window sizes over the image to detect each of the objects. We can keep the window size fixed and slide it over downsampled versions of the image to do that. This would mean, however, that we have to run the detection algorithm over multiple windows, over multiple scales of the image. There are some obvious downsides to this approach and the main disadvantages have been summarized in the following bullet points.

- Sliding a window over an image, even if multiple windows are slid over an image in parallel, is computationally inefficient;
- The size(s) of the windows relative to the image size and the number of different window sizes have to be determined using prior knowledge;
- Objects with different types of shapes can not be detected using the same window, for example a wide object or a tall object;
- A sliding window may not always line up well with a or multiple objects in an image, which may lead to that object not being detected.

2 Single-Stage Object Detectors

The previous section finished with a description of a sliding window detector, which is one of the ways in which a classifier can detect multiple objects in an image. It turns out however that the sliding window detector faces several challenges, amongst others that it is inefficient and might not be feasible for all applications. One method to mitigate these challenges is to predict regions of interest in the image and apply a classifier to this region. This is called a two-stage method. Stage one is to predict the region of interest and stage two is to classify and localize objects within this region. Two-stage methods are discussed in-depth in chapter 5.1.

Another method to mitigate these challenges is by making each feature map pixel responsible for the classification and localization of a local region in the input image. The localization and classification can be done using a single forward pass through the network, which is why it is called single-stage object detection. Examples of single-stage detectors include YOLO, SSD [11], DSSD [5], RetinaNet [10], M2Det [22], RefineDet [20]. Out of these detectors, YOLO is the most well known. We will explain and explore this single-stage detector in the next section. A lot of the concepts that are true for YOLO will generalize to the other single-stage detectors, but these other single-stage detectors are outside of the scope of this chapter. We encourage you to take a look at the papers that are referenced.

2.1 YOLO

This subsection will elaborate on YOLO, or You Only Look Once, which was presented as a novel approach to object detection by Redmon et al. [16]. Two official YOLO iterations have followed since, namely YOLO9000 [14] and YOLOv3 [15]. These iterations were all created by the original YOLO author, Joseph Redmon. YOLOv4 [1] is the most recent iteration and was not created by the original author. Even though YOLOv4 offers a significant improvement over YOLOv3, the methods that are used to improve upon YOLOv3 are outside the scope of this chapter. This is why we have decided against discussing its improvements here.

YOLO is one of the fastest object detectors in terms of inference time. The inference time is the time it takes to do one forward pass. It approaches object detection as a single regression problem, where image pixels are translated to bounding box coordinates and class probabilities in one stage.

2.1.1 Basic principle

Single-stage detectors such as YOLO approach object detection based on feature maps. They map a 2-dimensional grid over the input image corresponding to the spatial dimensions of the feature map. Take for example a 3-dimensional feature map with a width and height of 3 pixels, and a depth of 1024 features. In this case, a 3 by 3 grid is mapped onto the input image. Each pixel in the feature map corresponds to a grid cell, which is a *rectangular area of pixels* in the input image. A visual understanding of this notion is given in figure 15. This mapping is direct, so the right upmost pixel in the feature map corresponds to the right upmost grid cell in the input image, etc.

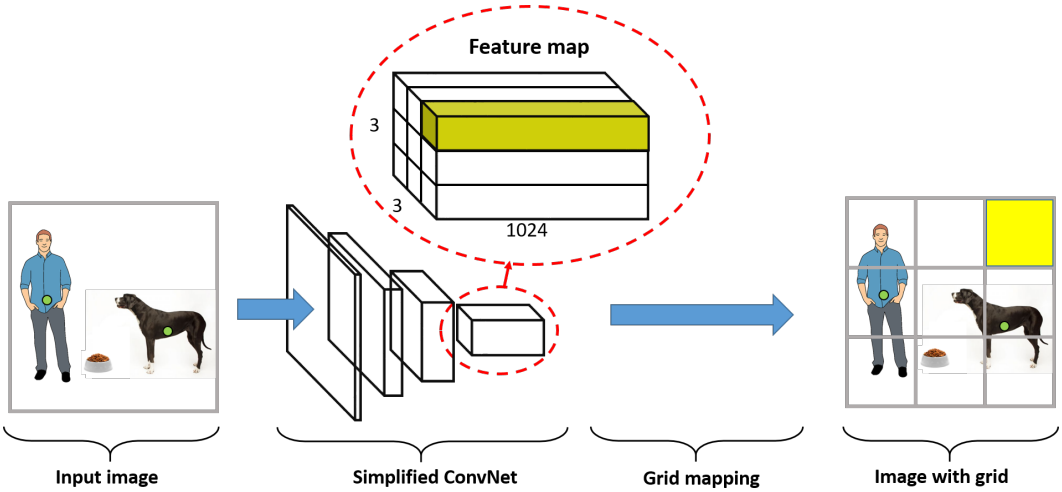
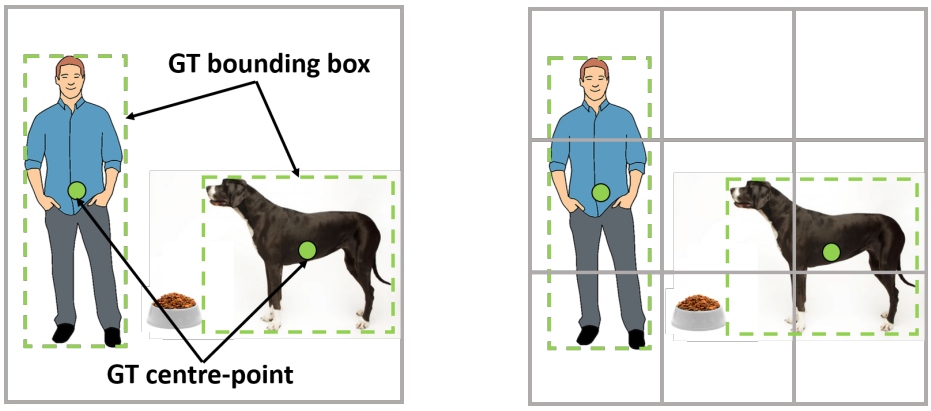


Figure 15: Input image is fed into a simplified ConvNet backbone which outputs a 3-dimensional feature map, which is 3x3x1024 in this example. The spatial resolution of the feature map is mapped as a 3x3 grid onto the original input image. The yellow parts show the correspondence of one of the feature map pixels its image grid cell

Each feature map pixel has a number of features attributed to it. These features are learnt from a rather large receptive field (section 1.1.2) of the input image. This receptive field exceeds the size of the grid cell in the input image that corresponds to this pixel. A feature map pixel is made *responsible* for the detection of objects in its corresponding grid cell. The features it uses can, however, contain information from a large part of the image to provide context. This is an important property because an object can span multiple grid cells. An example is shown in figure 16, where the human covers three grid cells.

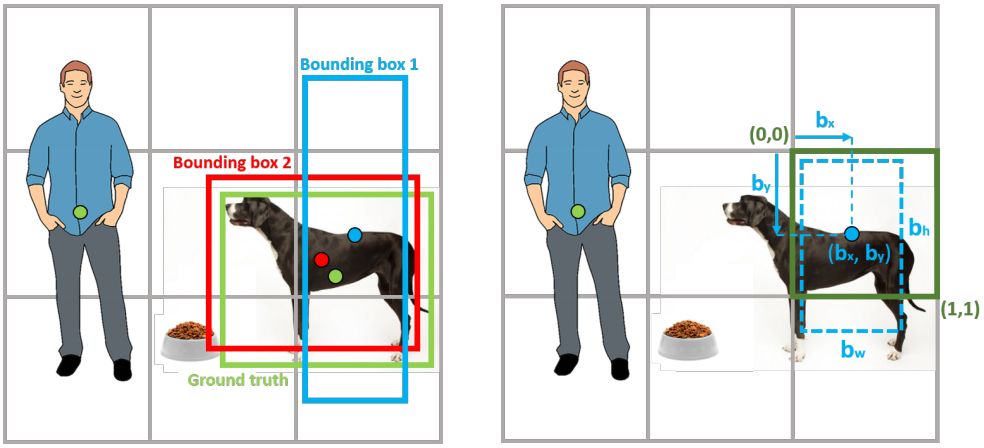


(a) Ground truth (GT)

(b) Input image divided into $3x3$ grid

Figure 16: Input image with 2 identifiable objects including their ground truth (GT) is subdivided into an $S \times S$ grid. In this case the grid is 3×3

But how do we determine which feature map pixel is made responsible for an object that covers multiple grid cells? During training, YOLO deals with this by making the grid cell that contains the centre point of the object responsible for detecting that object. The grid cells on the input image are a direct mapping to and from the feature map pixels. The feature map pixel that corresponds to the grid cell with the centre of the object in it, will thus be responsible to detect that object. In figure 16b, the left-middle grid cell contains the centre of the human, the corresponding left-middle feature map pixel is thus made responsible for detecting this object. YOLO splits up the detection of objects into grid cells and assigns a responsibility to the feature map pixels to penalize the network when it detects a bounding box for an object in another cell. It also makes it easier for the network to converge on predictions because it splits the problem up into smaller parts. This allows the network to focus on object predictions for smaller sections of the image, which allows each part of the object detector (each feature map pixel) to focus on a small number of predictions. Figure 17a illustrates 2 example bounding box predictions for one grid cell. The green bounding box represents the ground truth and the red and blue bounding boxes are predictions.



(a) ground truth bounding box and centre point (green) and predicted boxes and centre points (blue & red)

(b) Bounding box parameterization of centre point and dimensions

Figure 17: Bounding box predictions and parameterization

The final link in understanding the underlying principle of YOLO is the fully-connected layer that is used to map the information of the feature map to an output tensor. The **output tensor** is an extended version of the tensor introduced in section 1.2, and contains the class probabilities, object score and bounding box predictions for each grid cell in a vector format. Because the feature map is trained in conjunction with the fully-connected layer, each feature map pixel will learn which parts of the image are important as context for the cell it is responsible for. This allows YOLO to make predictions of multiple objects located around the entire image in one forward pass; hence the name You Only Look Once.

Summarizing the most important properties of YOLO:

1. The backbone network outputs a 3-dimensional feature map with a certain spatial and feature resolution, which captures important features in the input image;
2. A 2D grid is mapped onto the input image that is equivalent to the spatial resolution of the feature map;
3. During training, a feature map pixel that corresponds to a grid cell containing the ground truth centre coordinates of an object is made responsible for detecting that object;
4. Contextual information around a grid cell is also encoded in the corresponding feature map pixel due to the large receptive field the feature map has over the input image;
5. The feature map is trained in conjunction with the fully-connected layer and hence, each pixel in the feature map will learn which parts of the image are important as context for the cell it is responsible for.

2.1.2 Output tensor

For each feature map pixel, YOLO predicts an output label y. This label encodes information about the predicted bounding boxes and class corresponding to its grid cell. The length of the label mainly depends on the number of bounding boxes B predicted per pixel; each label has shape $B \cdot 5 + C$ and contains the following variables **per bounding box**; the object score P_c, the centre-point coordinates of the object, b_x and b_y, the width and height of the bounding box, b_w and b_h. Additionally, the output label contains a set of conditional class probabilities C **per grid cell**.

The meaning of these variables has been discussed previously in section 1.2. However, there are some important things to note; instead of the image spatial dimensions being normalized to [0, 1] as discussed in section 1.2.2, the spatial dimensions of the grid cells are all normalized to [0, 1]. This is illustrated in figure 17b. This normalization constrains the prediction of the centre of a bounding box to a certain grid cell. If we would use a sigmoid for the width and height, as discussed in section 1.2.2, the width, and height of a bounding box would be constrained to the width and height of that grid cell as well (between 0 and 1). YOLO thus uses the exponential function $\exp z$ to map the logit scores for the width and height of a bounding box to $[0, \inf]$. Where z corresponds to the logit scores for the width and height at the output of the fully-connected layer. This allows a bounding box to be larger than the grid cell that contains the centre of the object.

The final output tensor is a concatenation of all output labels and consequently has the shape $S \times S \times (B \cdot 5 + C)$, where SxS is the spatial resolution of the feature map, B the number of predicted bounding boxes per pixel and C the total number of classes. The notation of this output tensor differs between literature and is sometimes denoted as $S \times S \times B \times (5 + C)$. We will stick to the first notation. Figure 18 shows the output tensor for a YOLO algorithm that has a spatial feature map resolution of 3x3, predicts 2 bounding boxes per feature map pixel, and can predict a total of 3 different classes.

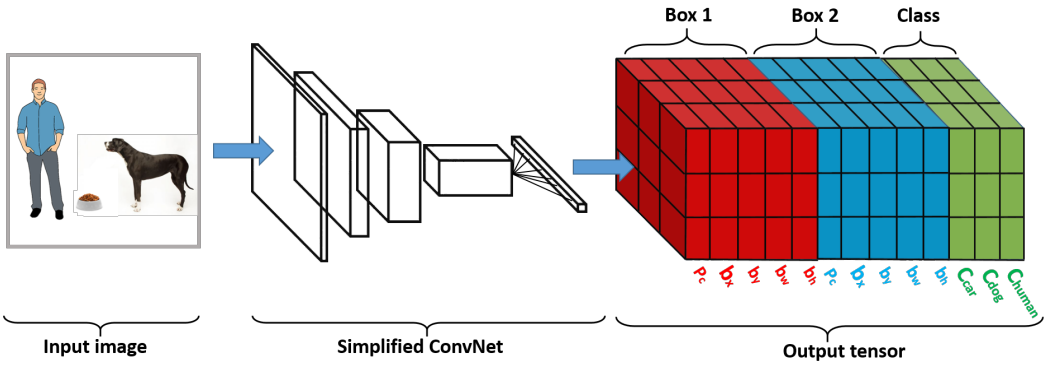


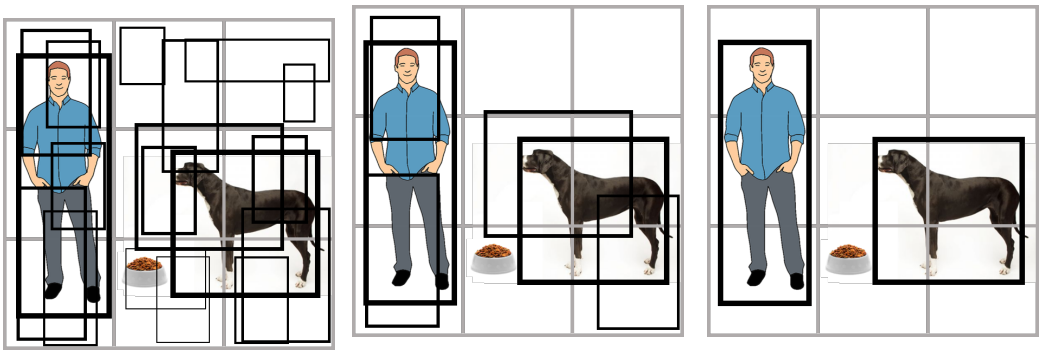
Figure 18: Simplified illustration of YOLO pipeline. Right: input image of a dog. Middle: simplified CNN. Left: YOLO output tensor of shape $S \times S \times (B \cdot 5 + C)$ corresponding to 3x3 gridcells S, 2 bounding boxes per cell B and 3 class labels C

2.1.3 Non-Maximal Supression

Assume that we have an input image onto which a 3x3 grid is mapped. We will predict 2 bounding boxes per grid cell and thus per feature map pixel. This is illustrated in figure 19a, where the thicker bounding boxes reflect predictions with a higher confidence score. As you can see, multiple bounding boxes are trying to predict the same object. *During testing* (inference), we would like to end up with only one bounding box per object, which is illustrated in figure 19c. In order to get rid of the redundant bounding boxes we will use **Non-Maximal Suppression (NMS)**. The 3 steps for NMS are provided below.

1. Discard all bounding boxes with a confidence score below a certain threshold ϵ; $P_c \leq \epsilon$;
2. Pick the bounding box with the largest confidence score and use that as predicted bounding box B_{pred};
3. Remove any remaining bounding box B_{remain} that has an IoU with the previously predicted bounding box (B_{pred}) that is above a certain threshold τ; $IoU(B_{pred}, B_{remain}) \geq \tau$.

Step 2 and 3 are performed iteratively until there are no more remaining bounding boxes. The thresholds ϵ and τ can be set to any desired value.



(a) All predicted bounding boxes

(b) All bounding boxes below a certain threshold are discarded

(c) Final bounding box predictions after running NMS

Figure 19: Removing redundant bounding box predictions using Non-Maximal Suppression

2.2 Limitations of YOLO

The ideas described in the previous subsections were used in YOLO, but there are a few downsides to this method. This motivated the next iterations YOLO9000 [14] and YOLOv3 [15]. The most important limitations of YOLO are the following.

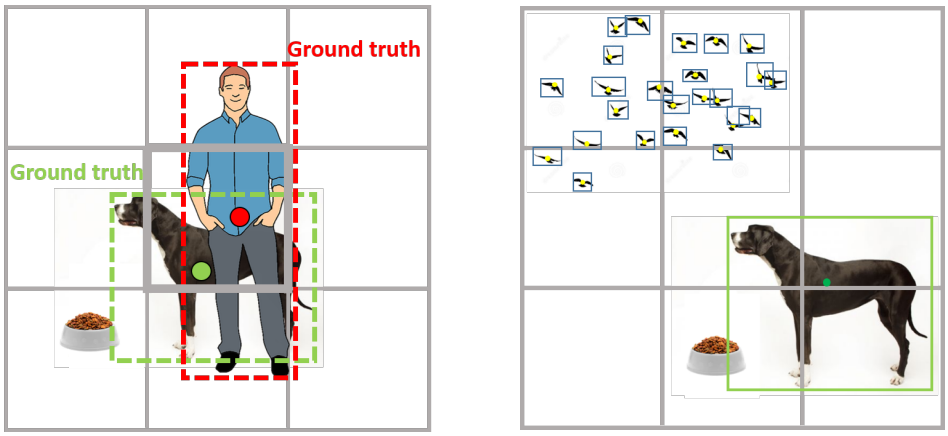
- **Each grid cell can only be assigned one class**, which is unrealistic because a grid cell can contain multiple different types of objects. The problem is visualized in figure 20a, where the centre of the human is in the same grid cell as the centre of the dog. The problem is caused by the output tensor, because the output tensor only encodes the classes on a grid cell level, instead of encoding the classes for each bounding box.;
- **Generalization to unusually shaped objects** is also a problem for the original YOLO network. The network has a hard time converging to bounding boxes with extreme aspect ratios, probably because the large deformation can take a long time to learn. Most bounding boxes will have a certain shape, so the deformations from the common bounding boxes to the ones needed for the outliers could be hard to learn;
- **Detection of small-scale objects** is a problem that is visualized in figure 20b. An important reason for this is that the feature map for YOLO is not fine enough ($7x7$). Therefore, each feature map pixel is responsible for a relatively large area in the input image, increasing difficulty in detecting small-scale objects.

2.3 YOLO9000 and YOLOv3

The limitations of the original YOLO have been tackled by more recent YOLO iterations. We will discuss two of the most prominent improvements introduced in YOLO9000 and YOLOv3, namely anchor boxes and different feature map scales.

2.3.1 Anchor boxes

In YOLO9000 [14] the concept of **anchor boxes**, or **priors**, was introduced. This improved performance with regards to generalization to unusually shaped objects and introduced multiple class predictions per grid cell. Anchor boxes are a set of predefined bounding boxes with a certain height and width. The shapes of the anchor boxes are defined to capture scale and aspect ratios of specific object classes within a dataset, e.g. a human is generally a tall shaped object and a car is generally a widely shaped object. We can incorporate this knowledge into YOLO by predefining anchor boxes corresponding to common shapes in the dataset. This makes it easier for the network to learn better predictions, as each anchor box specializes in detecting objects of a specific size and aspect ratio. Each anchor box corresponds to a certain class label, so the network can now learn multiple classes in a grid cell. Determining suitable shapes for the anchor boxes can be done by hand, but a more



(a) Centre-points of multiple objects of a different class share the same grid cell.

(b) Many small objects, or birds, sharing the same grid cell

Figure 20: Removing redundant bounding box predictions using Non-Maximal Suppression

clever approach is preferred. YOLO9000 and YOLOv3 use k-means clustering on the ground truth bounding boxes in the training set to determine a good set of anchor boxes. The number of means that are optimal to use for YOLO9000 (5) is different from YOLOv3 (9). The clustering can be done before training the network.

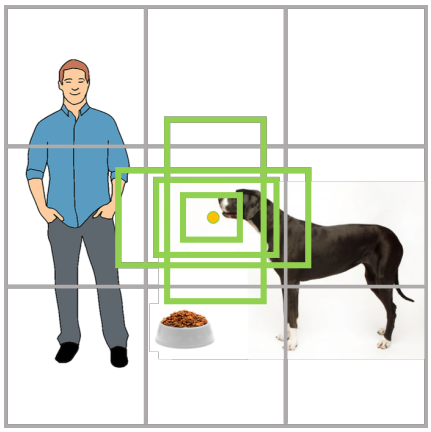


Figure 21: 4 predefined anchor boxes with different aspect ratios. Although not illustrated these anchor boxes are present in each of the grid cells

With anchor boxes, YOLO no longer predicts the centre coordinates of a bounding box or its height and width, but rather the offset from the predefined anchor box location and size to the predicted bounding box location and size. The anchor box is thus used as a prior for the shape of the object classes the object detector is trying to predict. Each anchor box can be assigned one object of any class. That means that each grid cell can detect as many different objects as there are anchor boxes. Essentially, the classification level has moved from the grid cell level to the bounding box level.

2.3.2 Different feature map scales

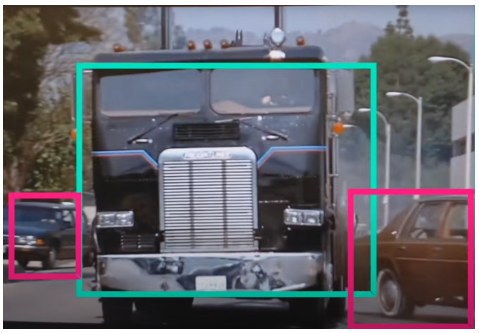
To overcome the problem of not being able to detect small-scale objects, YOLOv3 uses intermediate outputs of the backbone network as additional feature maps. Specifically, it uses feature maps produced at three locations in the network, which have a higher resolution. Because the intermediate feature maps are coming from lower layers, the receptive fields are smaller. To have meaningful intermediate feature maps, the number of layers in YOLOv3 is increased from 19 layers in YOLO9000 to 53 layers in YOLOv3. The 3 feature maps with different sizes are each used to map a grid onto the input image. This results in 3 different grids; one very fine grid, a coarse grid, and an intermediately

coarse grid. The network uses 3 anchor boxes per grid cell to make predictions in each of the 3 grids. YOLO9000 does not use multiple feature maps and only uses 5 anchor boxes per grid cell. This results in an increase in inference time for YOLOv3, because of the number of anchor boxes (9 vs 5) it has to predict and the increase in network size (53 vs 19 layers).

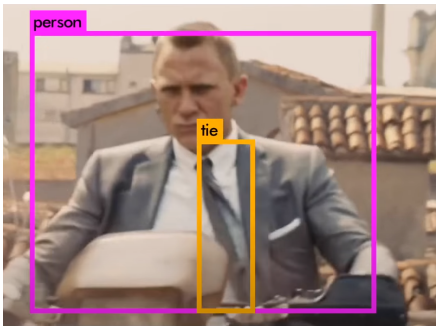
2.4 Real world usage

So what does YOLO look like when used on a real-world video? Along with the paper, the authors released a demonstrative showcase of YOLO on several videos and well-known movies. Additionally, we can use a very recent implementation of YOLOv4 on the Deepdrive PL Dataset by Karol Majek [12]. All videos are linked here on the PDF version: YOLO, YOLO9000, YOLOv3, YOLOv4.

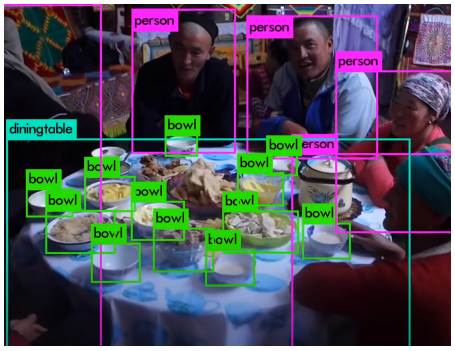
Figure 22 showcases selected frames of the YOLO, YOLO9000, YOLOv3 and YOLOv4 demonstration videos. There is a clear increase in performance with each iteration producing more accurate detection and being able to capture more detail in the image.



(a) YOLO: The Algorithm is able to detect multiple separate vehicles



(b) YOLO9000: The algorithm is able to detect attributes that a person is wearing



(c) YOLOv3: the algorithm is able to detect several varying, complicated objects on top of each other

(d) YOLOv4: The algorithm is able to detect vehicles that are far away and even people present in the vehicle in front

Figure 22: Showcase of the current YOLO implementations

2.4.1 Current applications

The advantage of YOLO over two-stage systems is that it looks at the whole image at test time, ensuring that its predictions are informed by a global context. Furthermore, as it only uses just one network evaluation to make predictions it is much faster (1000x faster compared to RCNN and 100x Faster compared to Fast R-CNN [15]. It is therefore perfectly suited for applications that require speed such as autonomous cars [2]. The size of the network and the low inference time also makes it possible to translate a YOLO version to an embedded environment, like YOLO NANO [19] and YOLO-LITE [6].

2.4.2 Controversy

YOLO-based algorithms are known to be actively used in military applications, such as autonomous **unmanned aerial vehicles** (UAV's) [17] and **surveillance tracking** by governments [21]. As a result, it even caused the original creator of the algorithm, Joe Redmon, to stop working in the computer vision field, because of his concerns regarding the usage of his work. As stated by him in a Twitter post: "I stopped doing CV research because I saw the impact my work was having. I loved the work but the military applications and privacy concerns eventually became impossible to ignore."

Reference image

The image that is used as a reference image throughout this chapter [4] was taken from Psychology Today. The official source is SC Psychological Enterprises Ltd.

References

[1] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020.

[2] J. Choi, D. Chun, H. Kim, and H.-J. Lee. Gaussian yolov3: An accurate and fast object detector using localization uncertainty for autonomous driving. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 502–511, 2019.

[3] D. C. Ciresan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[4] S. Coren. Behavior differences between smaller and larger dogs, Jul 2015. URL `https://www.psychologytoday.com/us/blog/canine-corner/201507/behavior-differences-between-smaller-and-larger-dogs`.

[5] C.-Y. Fu, W. Liu, A. Ranga, A. Tyagi, and A. C. Berg. Dssd: Deconvolutional single shot detector. *arXiv preprint arXiv:1701.06659*, 2017.

[6] R. Huang, J. Pedoeem, and C. Chen. Yolo-lite: a real-time object detection algorithm optimized for non-gpu computers. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 2503–2510. IEEE, 2018.

[7] A. Karpathy. Cs231n: Convolutional neural networks for visual recognition. https://cs231n.github.io/classification/, 2016.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

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

[10] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[11] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[12] K. Majek. Deep drive pl warsaw. https://archive.org/details/0002201705192, 2017.

[13] W. Rawat and Z. Wang. Deep convolutional neural networks for image classification: A comprehensive review. *Neural computation*, 29(9):2352–2449, 2017.

[14] J. Redmon and A. Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[15] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[16] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[17] Á. Rocha and T. Guarda. *Developments and Advances in Defense and Security: Proceedings of the Multidisciplinary International Conference of Research Applied to Defense and Security (MICRADS 2018)*, volume 94. Springer, 2018.

[18] 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)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

[19] A. Wong, M. Famuori, M. J. Shafiee, F. Li, B. Chwyl, and J. Chung. Yolo nano: a highly compact you only look once convolutional neural network for object detection. *arXiv preprint arXiv:1910.01271*, 2019.

[20] S. Zhang, L. Wen, X. Bian, Z. Lei, and S. Z. Li. Single-shot refinement neural network for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4203–4212, 2018.

[21] X. Zhang, X. Hao, S. Liu, J. Wang, J. Xu, and J. Hu. Multi-target tracking of surveillance video with differential YOLO and DeepSort. In J.-N. Hwang and X. Jiang, editors, *Eleventh International Conference on Digital Image Processing (ICDIP 2019)*, volume 11179, pages 701 – 710. International Society for Optics and Photonics, SPIE, 2019. doi: 10.1117/12.2540269. URL `https://doi.org/10.1117/12.2540269`.

[22] Q. Zhao, T. Sheng, Y. Wang, Z. Tang, Y. Chen, L. Cai, and H. Ling. M2det: A single-shot object detector based on multi-level feature pyramid network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 9259–9266, 2019.