**Bounding boxes**

So far, we worked with image classification. In this video, we will talk about bounding boxes and object recognition.

**What is object recognition?**

Object recognition identifies objects in images. Think of a self-driving car. Its systems need to identify the location of all objects on the road such as other cars and pedestrians. This is typically achieved by drawing bounding boxes around the objects. All localized objects must then be identified with their class label. Object recognition is used in many applications such as surveillance, medical diagnosis, traffic management, or sports analytics. We'll be reviewing how to annotate image data with bounding boxes in this video while later videos will cover model evaluation and explore two different model structures for object recognition.

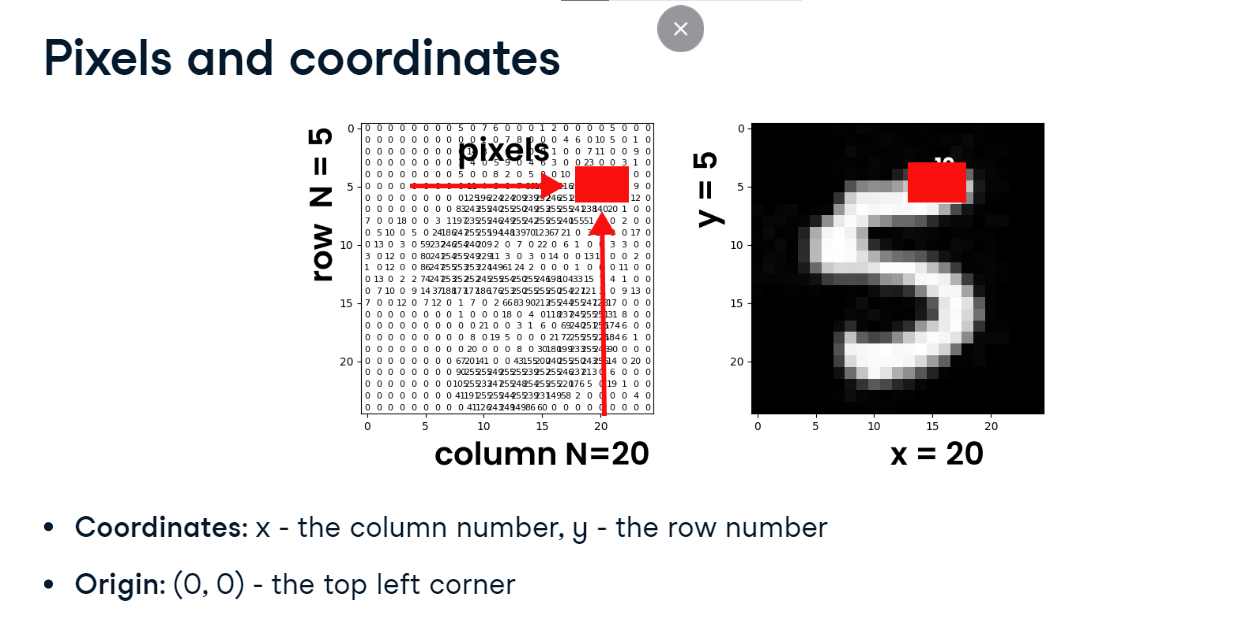
**Bounding box representation**

A bounding box, like this red rectangle around the cat, describes an object's spatial location within the image. Bounding boxes are used for annotating training data. They are also the outputs of object recognition models. A ground truth bounding box precisely outlines the location of an object within an image.

A bounding box is typically described by its top left and bottom right coordinates. These four numbers: x1, y1, x2 and y2 define each bounding box. Sometimes, the one and two are referred to as min and max, respectively, so that x1 is x\_min, x2 is x\_max, and

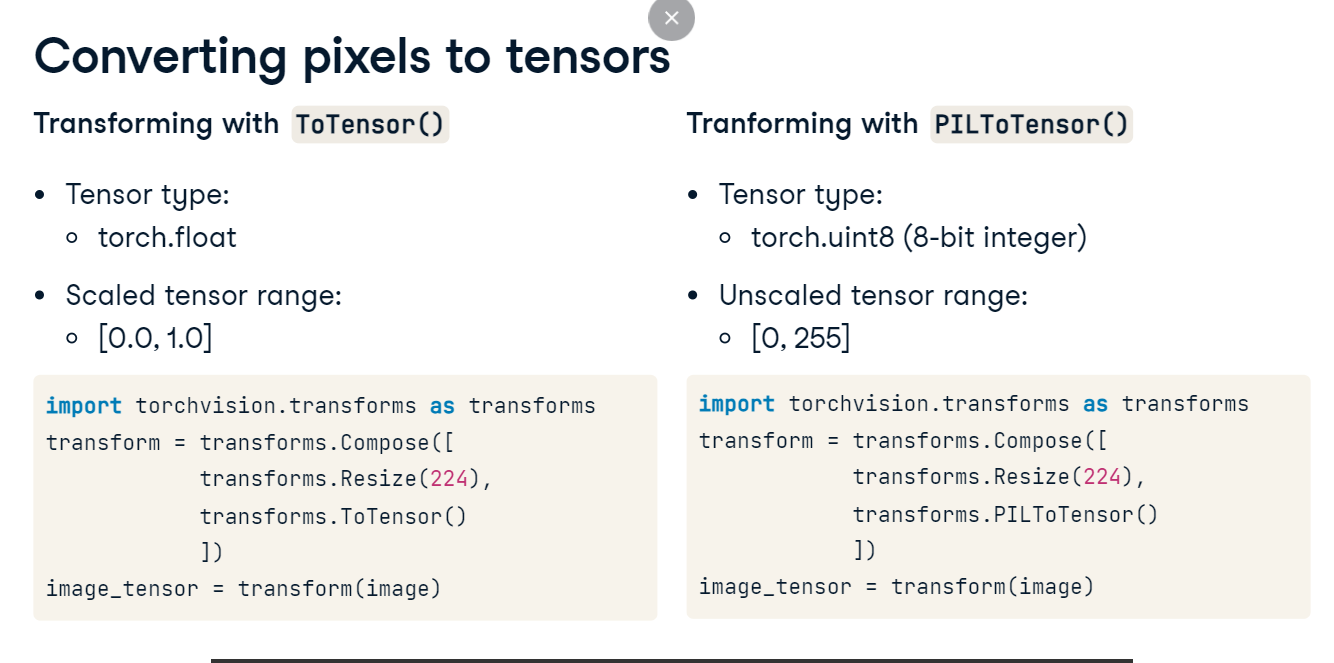
**Pixels and coordinates**

An image consists of pixels. Pixels provide a way to specify the location, size, and object boundaries within an image. Each pixel is defined by the column number or the x-coordinate and the row number or the y-coordinate. The origin of the image (the very first pixel) has the coordinates x equals zero and y equals zero and is at the top-left corner. For example, the pixel in the column twenty and row five corresponds to x and y equal to twenty and five, respectively.



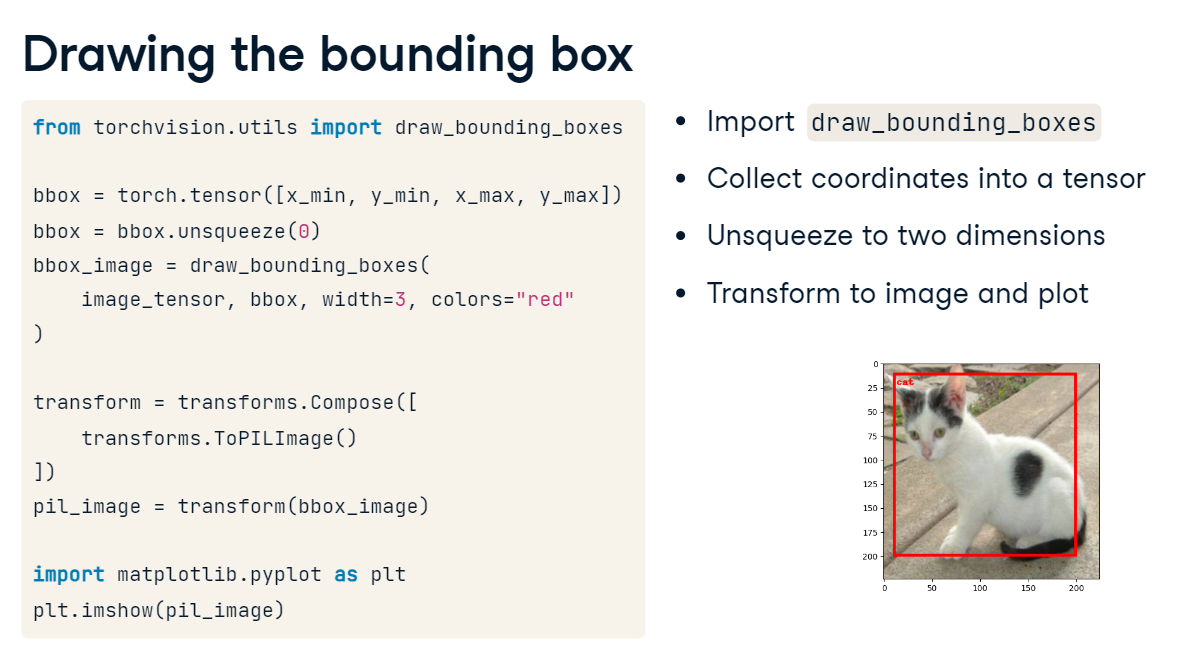
**Converting pixels to tensors**

**To be able to process images in PyTorch, we must convert pixel arrays to tensors.** There are two transforms, ToTensor() and PILToTensor() that produce different output formats. ToTensor converts pixels to float tensors, scaling values from zero to one. We import transforms from torchvision-dot-transforms and use transforms-dot-compose to combine transformations. Let's use transforms-dot-resize to set the image size to 224 and apply transforms-dot-ToTensor to create the float tensors. PILToTensor converts pixels to 8-bit integer tensors. Pixel values remain unscaled from zero to 255. We can apply PILToTensor in the same way, just updating the transform function. The PILToTensor transformation is useful for bounding boxes.



**Drawing the bounding box**

Let's see how to draw bounding boxes on top of images! We will use the draw bounding boxes function from torchvision-dot-utils. Assume we know the coordinates, perhaps as predicted by the object recognition model. We collect them into a tensor using the torch-dot-tensor method. Next, we pass the image tensor and the box coordinates to draw bounding boxes, setting line width to three and color to red. To display the box, we convert the tensor to an image using the ToPILImage transform and call imshow from matplotlib. We have a ground truth bounding box around the cat!

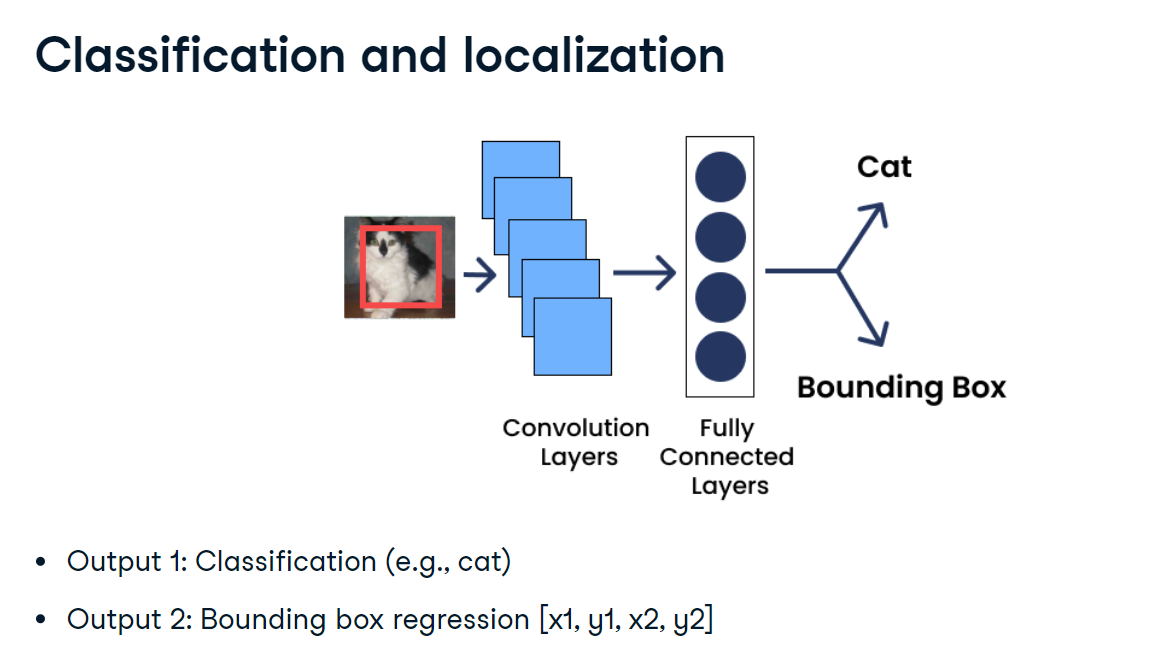


**Evaluating object recognition models**

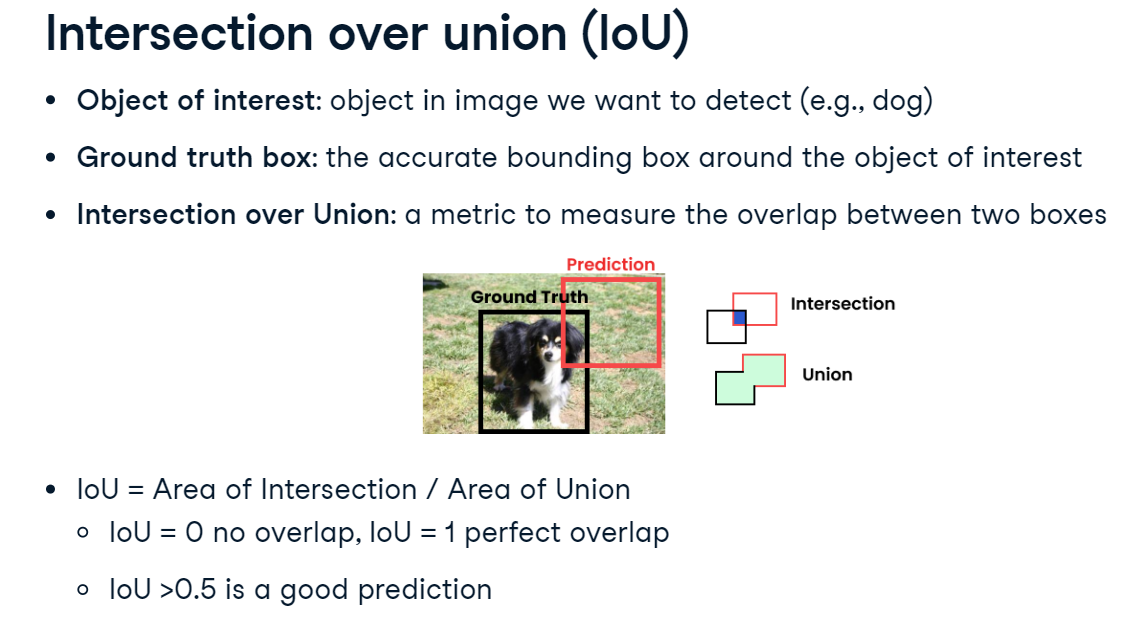
**Classification and localization**

Object recognition requires predicting two things: the class and location of an object in an image. The classification task is similar to the standard image classification we learned earlier.

Additionally, the model learns the bounding box coordinates to properly fit the target object. These coordinates are continuous values making this a regression task.

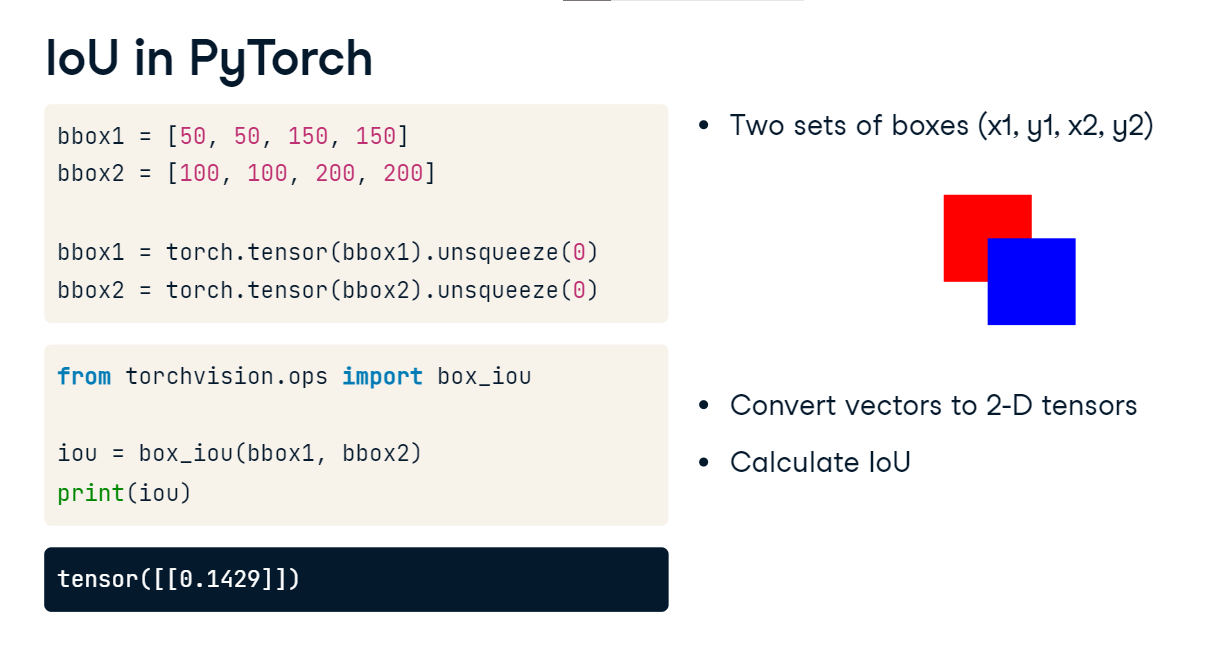


**Intersection over union (IoU)**

Suppose we are interested in detecting dogs - this is our object of interest. We annotated the test dataset with the accurate bounding boxes - these will be our ground truth boxes. Now, our object recognition model predicted a bounding box like this red one for a dog in the image. The ground truth box is colored black. We can see some overlap between these two boxes. How good is our prediction? Intersection over Union, or IoU, is a common metric in object recognition to evaluate the degree of overlap between two boxes. The overlap called an intersection is divided by the area of their union. IoU ranges from zero (no overlap) to one (perfect overlap). The common threshold is point five. Any prediction greater than point five is considered a good prediction.

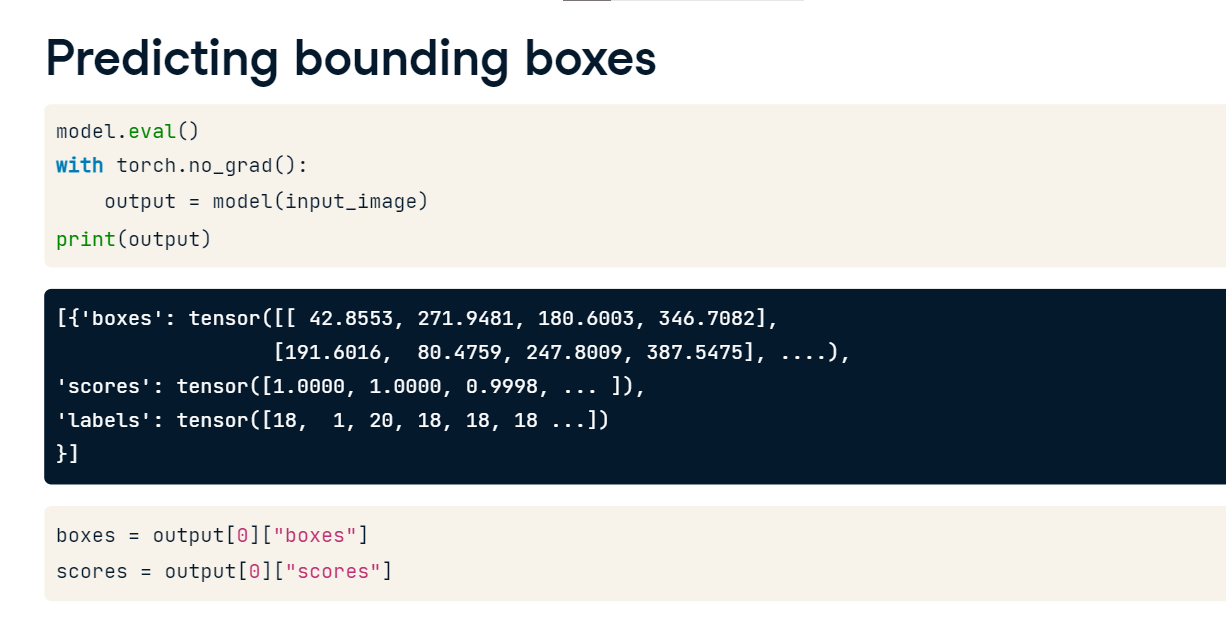
**IoU in PyTorch**

Let's see how we can calculate IoU with PyTorch. Consider this ground truth red box and the predicted blue box. We store their coordinates in two lists called bbox1 and bbox2. To calculate the IoU, we convert them into tensors and reshape them using the unsqueeze method. We import the box iou function from the torchvision-dot-ops module and pass our tensors as arguments. The result is point 14 which is less than the threshold of point five. So the predicted box is not very accurate.



**Predicting bounding boxes**

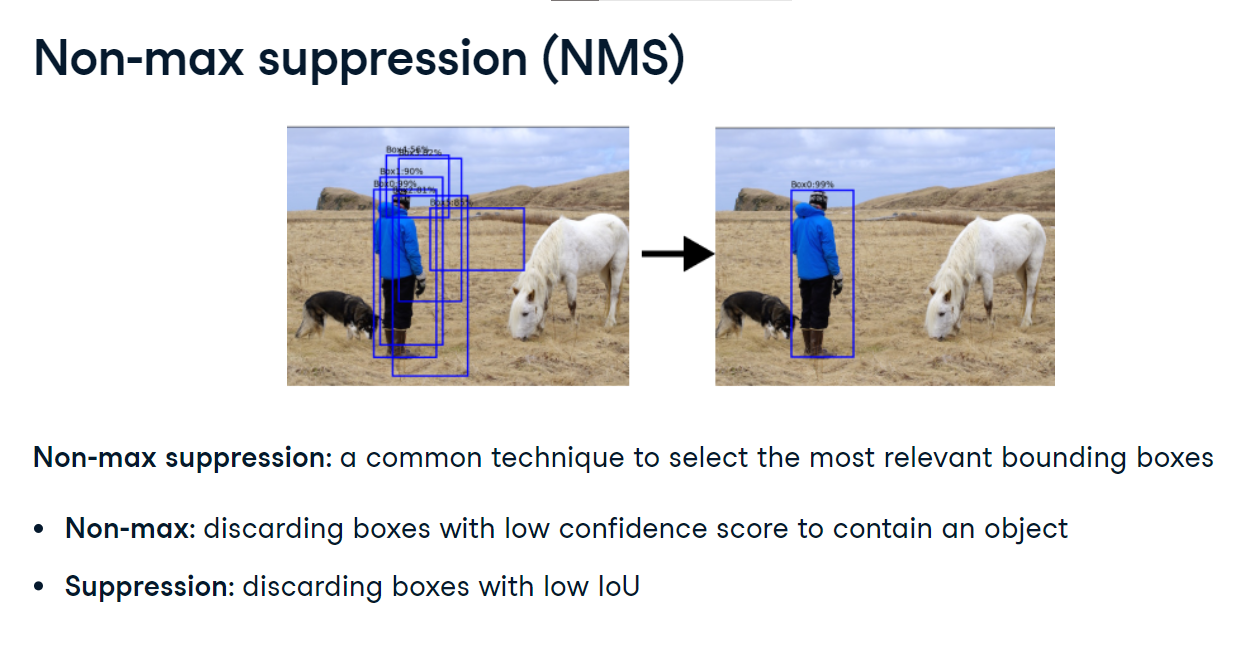
Let's see how to use a trained recognition model to predict bounding boxes. We will look at the model architecture in the next video. We switch the model to evaluation mode to use it for prediction and disable gradients calculation. Then, we pass the input image through the model to get the output predictions. The output is often a list of dictionaries with tensors containing bounding box coordinates of multiple boxes, their associated confidence scores indicating how confident the model is about each box, and predicted class labels for each box. Let's create a variable named boxes and extract coordinates from the output by accessing the first dictionary in the list with the key word boxes. Next, we create a variable named scores and extract confidence scores with the key word scores.



**Non-max suppression (NMS)**

As we just saw, object recognition models may generate many bounding boxes and some of them may be overlapping near-duplicates.

Our goal is to discard unnecessary boxes. Non-max suppression, or NMS, is a common technique in object recognition to select the most relevant box for our object of interest by discarding boxes with low confidence scores and below the IoU threshold.



**Non-max suppression in PyTorch**

Let's apply nms in PyTorch. We start with importing the nms function from torchvision-dot-ops. We then pass it three arguments: boxes, a two-dimensional tensor with the four bounding box coordinates for N boxes, scores, a one-dimensional tensor with a confidence score for each box, and the iou threshold, which we set to point five here. As a result, we get a list of the most relevant bounding boxes with no overlapping duplicates. The output of the nms function is a tensor containing the indices of the filtered boxes after non-maximum suppression. We can use these indices to filter our bounding boxes by retaining only the selected boxes.

