**Introduction to image segmentation**

**Image segmentation**

Image segmentation is a computer vision task that involves partitioning the image into multiple segments, or areas withing the image, on the pixel level. This means that each pixel in an image is assigned to a particular segment. There exist three types of segmentation: semantic, instance, and panoptic and each of them requires a different model architecture. Let's discuss them one by one.

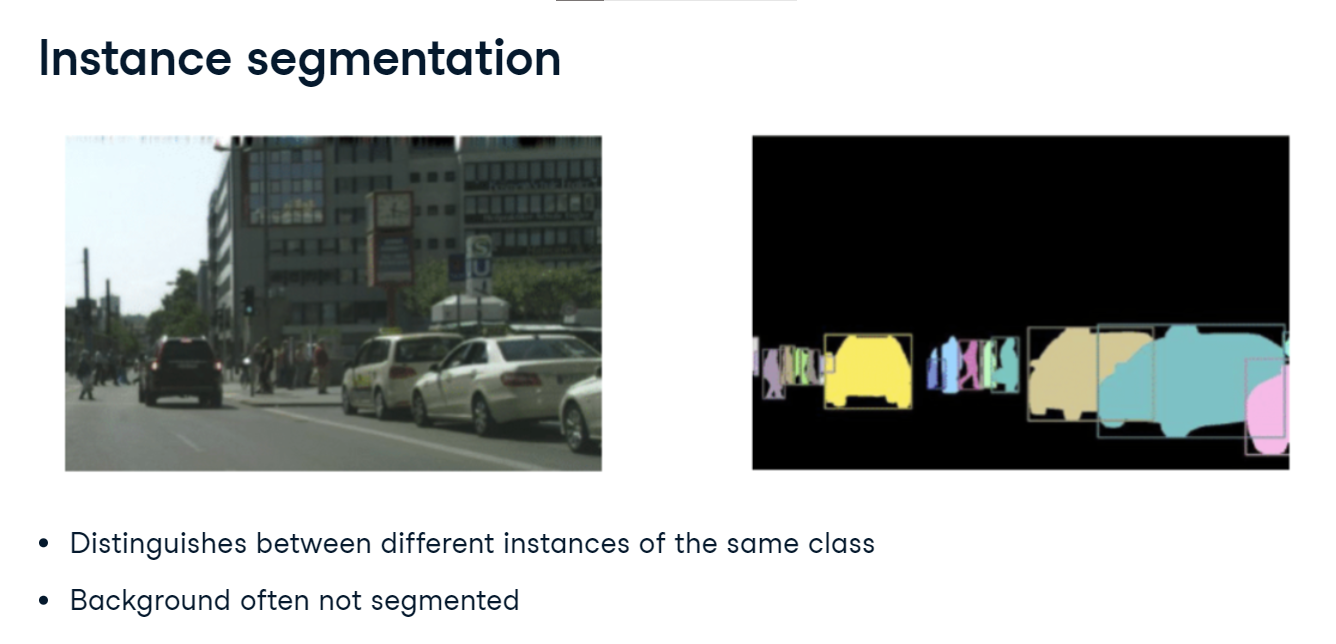
**Semantic segmentation**

In semantic segmentation, each pixel in the image is classified into a predefined class or category. All pixels belonging to the same class are treated equally, and there is no distinction between different instances of the same class. In a street scene, all pixels belonging to cars are marked as dark blue, all pixels belonging to roads are marked as purple and so on, without distinguishing between individual cars or road sections.



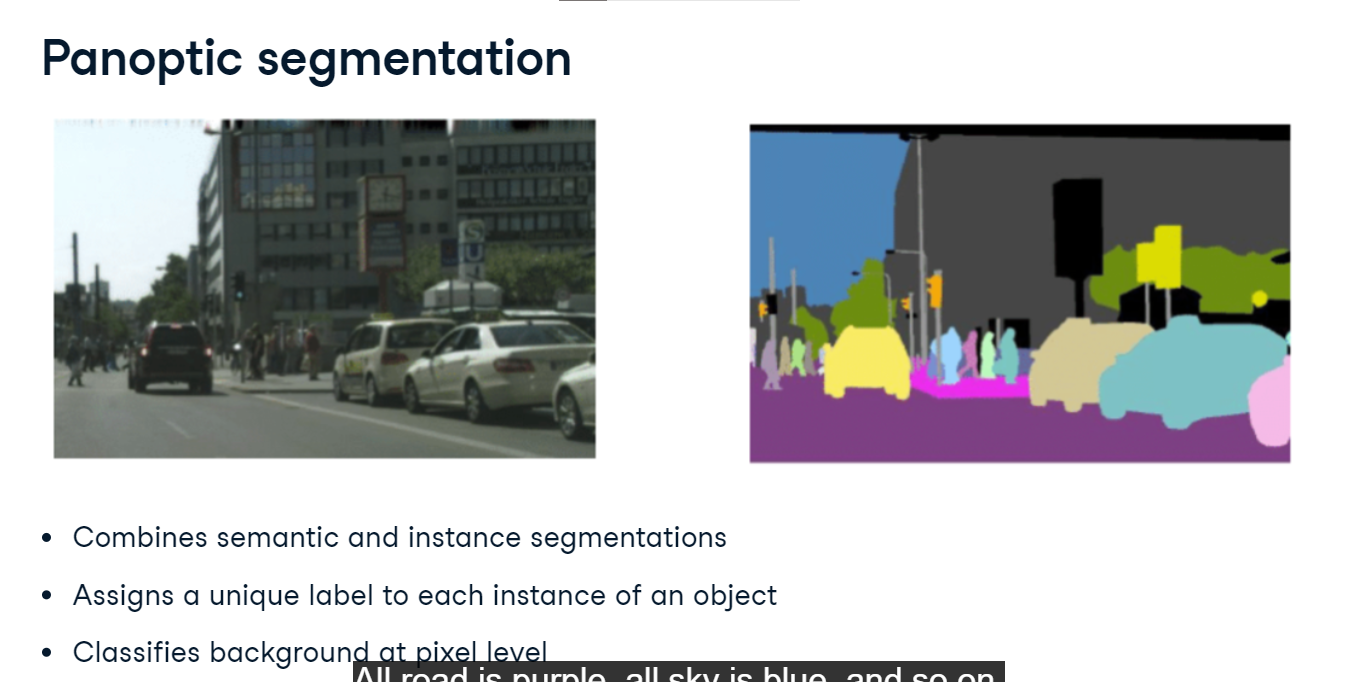
**Instance segmentation**

Instance segmentation goes a step further than semantic segmentation by not only classifying each pixel but also distinguishing between different instances of the same class. In the same street scene, each car would be assigned a unique label shown with a unique color, so that different cars can be differentiated from each other, even though they belong to the same class "car". Since the primary focus of instance segmentation is on identifying and segmenting individual object instances in the image, the background such as road or sky is typically not segmented.



**Panoptic segmentation**

Panoptic segmentation combines the concepts of semantic segmentation and instance segmentation. It assigns a unique label to each instance of an object while also classifying background regions (such as sky, road, or grass) at the pixel level. In the street scene, each car would get a unique label like in instance segmentation (each car is shown in different color). At the same time, areas like the road, sky, and trees are labeled at the pixel level without instance differentiation like in semantic segmentation. All road is purple, all sky is blue, and so on.



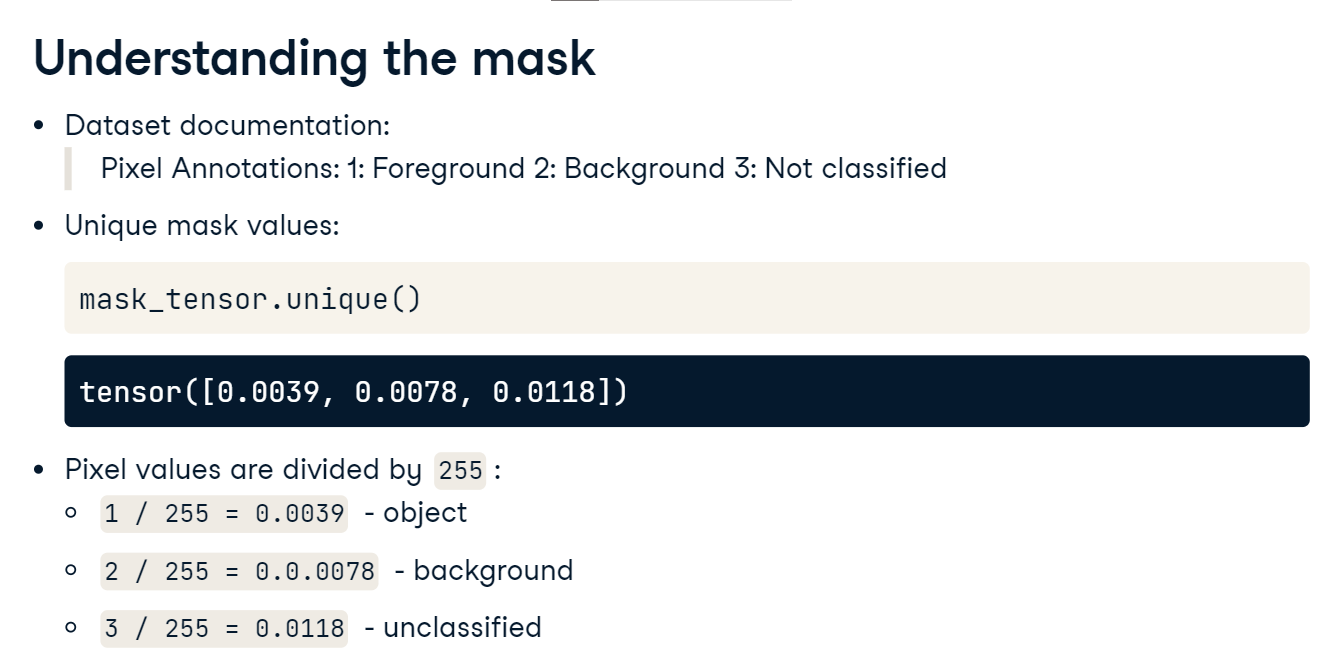
**Data annotations**

Let's take a look at data annotations for segmentation tasks. We load two image files. image is the picture of this British Shorthair cat sitting on a sofa. mask is the corresponding data annotation. The mask tells us which pixels are part of the cat, and which are not. Let's convert both PIL images to PyTorch tensors and print their shapes. The image is 333 by 500 pixels and has three color channels. The corresponding annotation is of the same height and width, but has only value for each pixel, describing its segment.



**Understanding the mask**

In the dataset documentation, we read that the annotations can only take three values: 1 for the object, 2 for the background, and 3 for unclassified. But when we print the unique mask values, we see three different numbers! This is because the ToTensor transform has divided the pixel values by 255. In our case, 1 over 255 which equals 0.0039 denotes the foreground, or the object of interest. A similar calculation is done for background and unclassified.



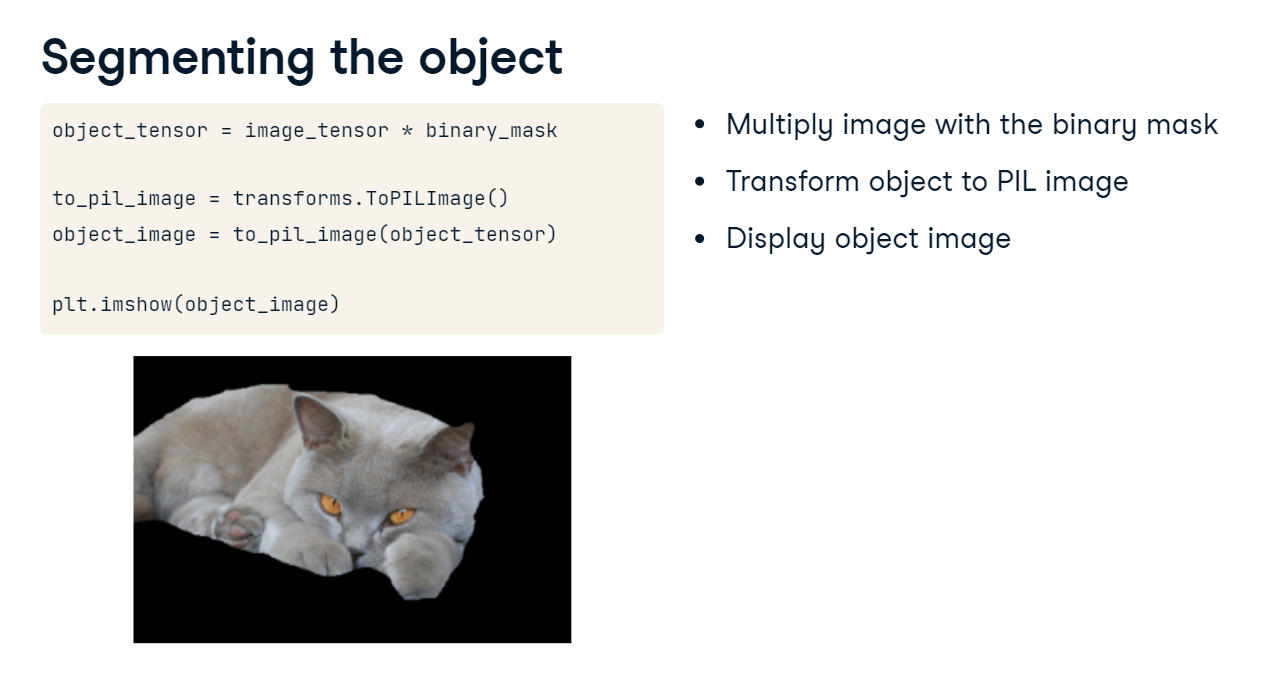
**Creating a binary mask**

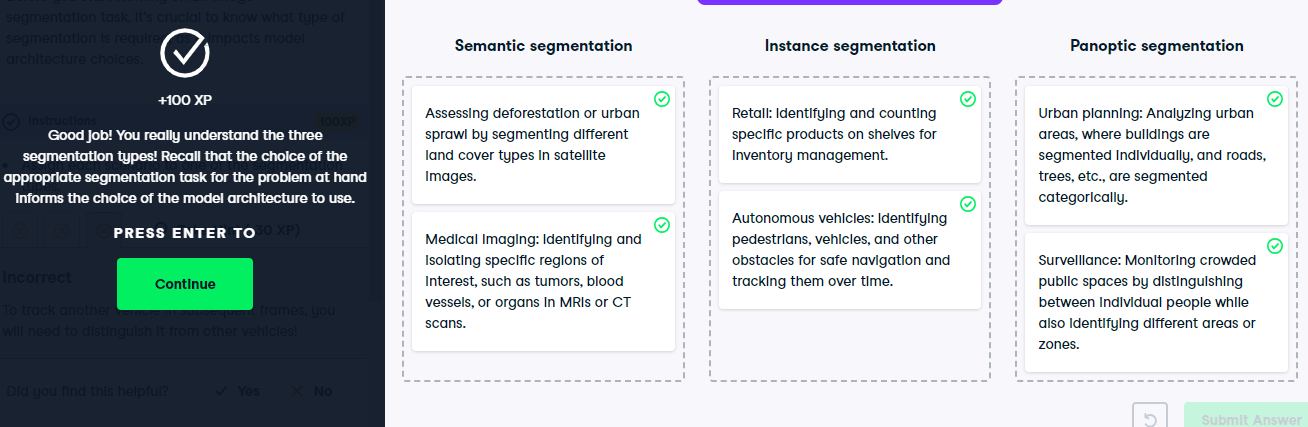
Let's create a binary mask, where 1 corresponds to the object and 0 to everything else. We will use the torch-dot-where function to do so. It takes three arguments. First, the condition to check: whether the pixel value represents the object. Then, the value to use when the condition is met, here 1, followed by the value to use otherwise, here 0. Let's take a look at our binary mask. We convert the mask tensor back to a PIL image and display it. The cat's shape is clearly visible!



**Segmenting the object**

Now, let's segment our cat out of the picture. To create the object tensor, we multiply the image with the binary mask. Next, we proceed just like we did with the mask: we transform the object to a PIL image and display it. The cat has been segmented out and the sofa in the background is gone!





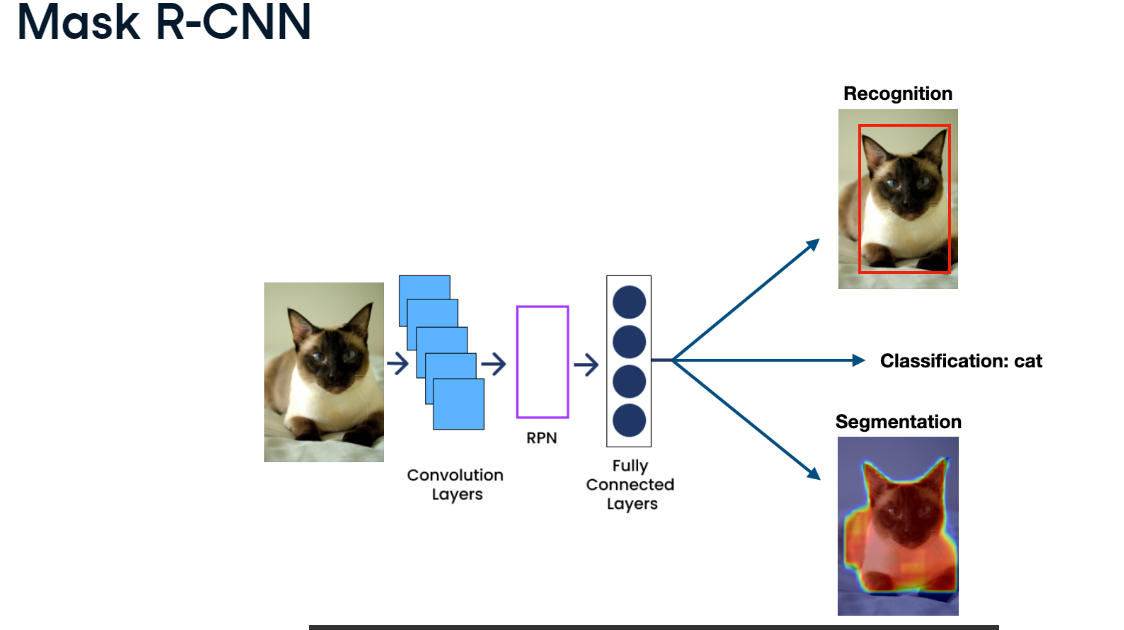
**Instance segmentation with Mask R-CNN**

**Faster R-CNN**

We've previously covered Faster R-CNN for object recognition. Given the image, it would predict its class and the bounding box around the object.

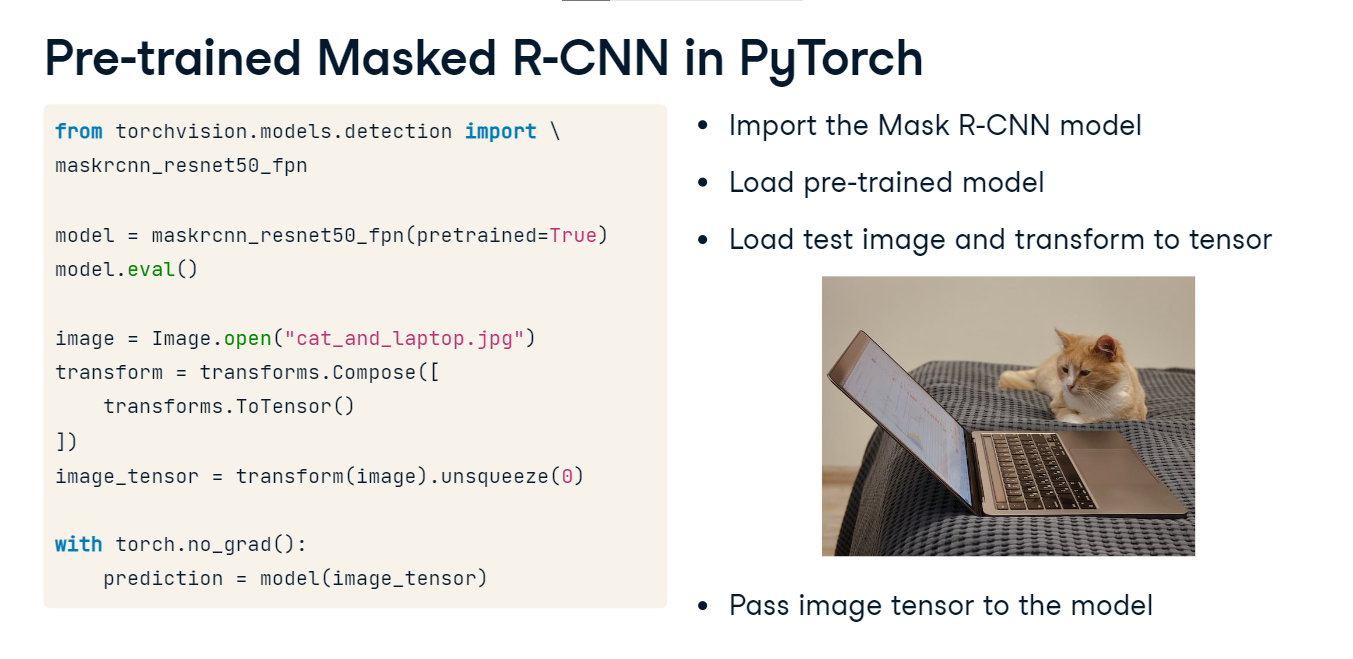
**Mask R-CNN**

Mask R-CNN extends Faster R-CNN by adding instance segmentation, retaining a nearly identical architecture with convolutional layers, a Region Proposal Network, and fully connected layers. Mask R-CNN introduces a third model branch that predicts a pixel-to-pixel segmentation mask. This enables accurate instance segmentation.



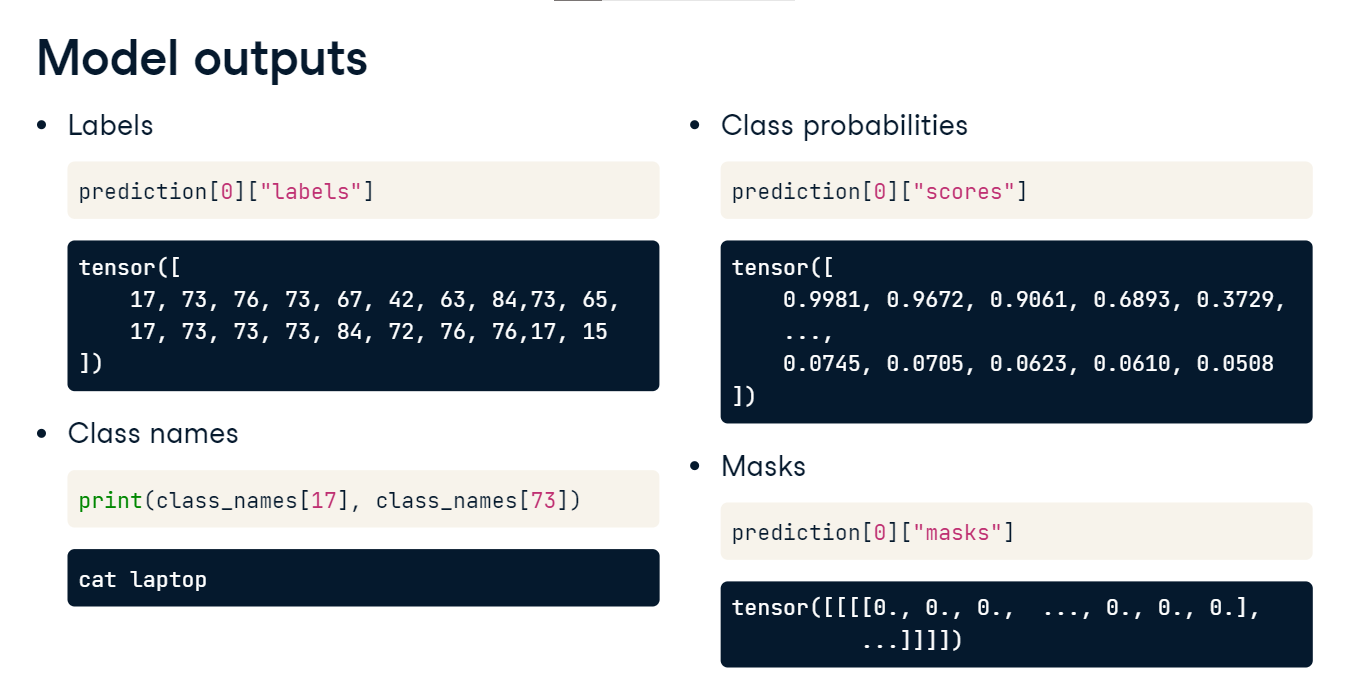
**Pre-trained Masked R-CNN in PyTorch**

Let's explore using a pre-trained Mask R-CNN model for instance segmentation. We start by importing the maskrcnn-resnet50-fpn from torchvision-dot-models-dot-detection. Next, we load the model with pre-trained weights. Then, we load and convert the test image into a tensor. We will use a photograph of a cat sitting next to a laptop, and we want to detect these two objects. Since the model is pre-trained on the COCO dataset, which includes common objects like animals and computers, it should detect our objects without requiring fine-tuning. Finally, we pass the image tensor to the model to run the inference, saving the result in a variable called "prediction".



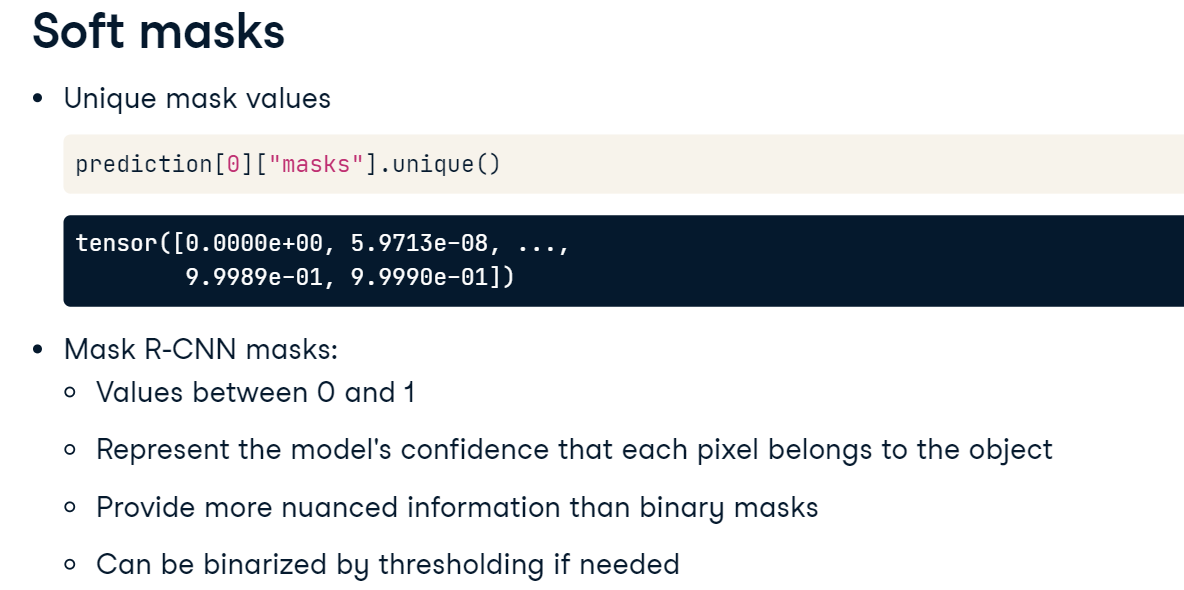
**Model outputs**

Examining the Mask R-CNN outputs: "prediction" is a list of length one since we only passed one image to the model. This single list element is a dictionary with a couple of keys. "labels" contains the class IDs of recognized objects. These IDs correspond to the COCO dataset classes which we have stored in the variable class\_names. These class names are available on the COCO dataset website. We can see that the top two predicted classes with indices 17 and 73 correspond to a cat and a laptop, respectively. The scores key stores the class probabilities. We can see that the cat was detected with a probability larger than 99% - that's the first value in the tensor - and the laptop with more than 96%. The following values correspond to other, less probable classes. Finally,the masks key stores instance segmentation masks which we will look at next. Additionally, the Mask R-CNN prediction also contains bounding boxes, but we are not interested in them when discussing segmentation.



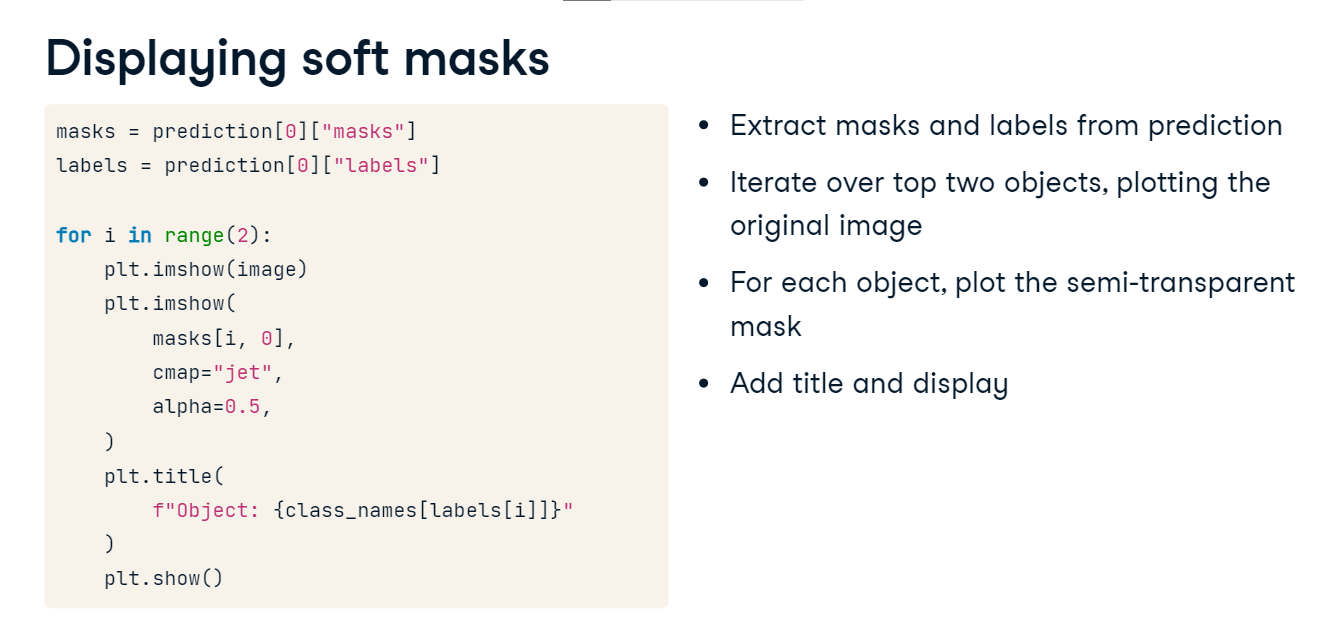
**Soft masks**

Let's print the unique values of the predicted masks. The values in the segmentation masks produced by Mask R-CNN are not binary (0s and 1s) but are instead floating-point values ranging from 0 to 1. These values represent the model's confidence that each pixel belongs to the object being segmented. These continuous values produce what is known as a "soft mask". Soft masks can provide more nuanced information than binary masks, especially at the boundaries of objects where there might be ambiguity. If we need a binary mask, we can apply a threshold to the soft mask. For example, we might decide that any value above 0-point-5 should be considered part of the object (set to 1), and any value below 0-point-5 should be considered background (set to 0).



**Displaying soft masks**

Lets' see how to display a soft mask overlaid on top of the image. We first extract the masks and labels from the prediction. Next, we iterate over the top two predicted objects. For each, we display the original image, and then the mask, setting the color map to "jet" and alpha to 0-point-5 to make the mask semi-transparent so that it does not obscure the image. Finally, we add class labels to the title and display.



**Displaying soft masks**

The cat mask is very accurate. The one for laptop slightly less so, although the high-confidence red regions are still pretty good.

