**Binary and multi-class image classification**

We will learn about object detection models which identify objects in images by drawing a box around them.We will also apply image segmentation models to segment images into meaningful areas.

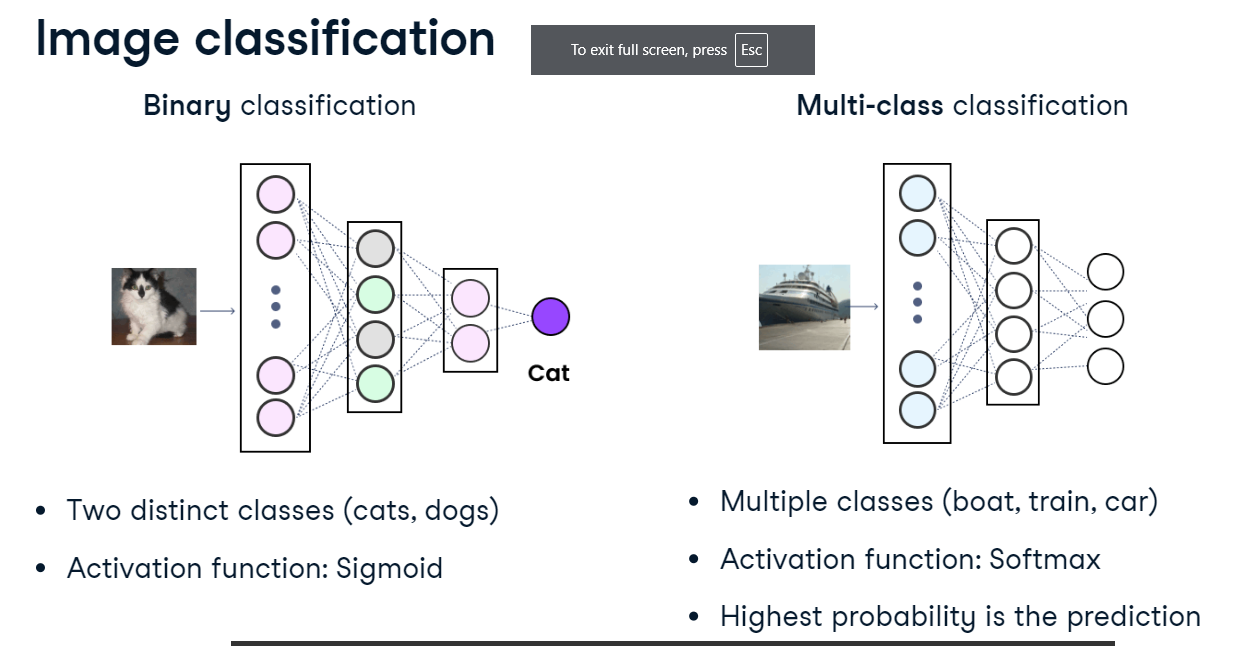
Finally, we will create new images based on learned patterns using image generation models.

Before starting, you should already be familiar with Convolutional Neural networks, including how they work and how to construct them in PyTorch, as well as with PyTorch model training in general, as taught in this prerequisite course.

We will use TorchVision throughout this course. It is a PyTorch image library that provides useful tools, including transformations for image pre-processing,pre-trained CNN models, and labeled image datasets for training and testing.

**Image classification**

Let's begin with image classification, commonly categorized into two types. The first type is a binary classification with two distinct classes, for example, cats and dogs. We use the sigmoid activation function to produce get the probability of either class. The second type is multi-class classification. Here, we deal with more than two classes, for example, boat, train, and car. We use the softmax activation to get the probability of each class. The class with the highest probability is the final prediction.



**Convolutional Neural Network model**

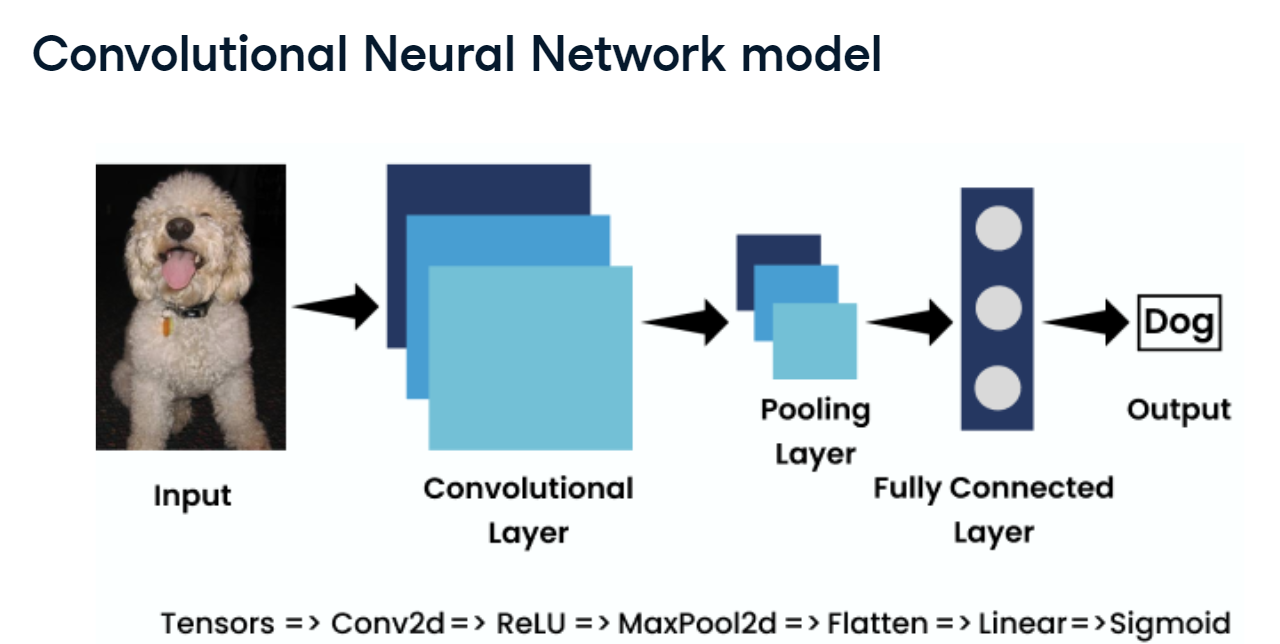
Let's now revisit the CNN model. First, we load a dataset (for example, pet images) and transform it into tensors.

We pass tensors through the convolutional layer, where the network learns image features and generates feature maps. Then, we apply a non-linear activation function, for example, ReLU.

In the pooling layer, we reduce the size of feature maps to decrease the computational workload.

Then, we flatten multi-dimensional tensors into a one-dimensional vector and pass it into the fully connected layer.

Finally, we apply the Sigmoid or Softmax activation function to generate class probabilities.



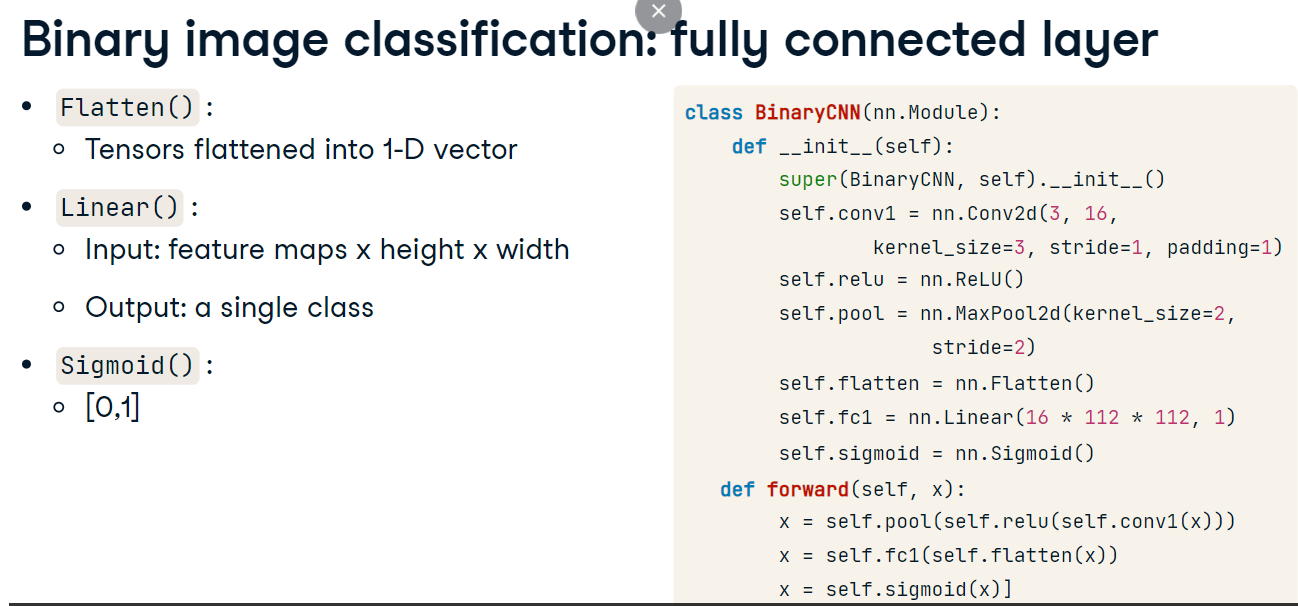
**Datasets: class labels**

Suppose we have pets dataset with separate directories for each class. This is a common format in image classification. We import the datasets and transform modules from torchvision. The training data directory is located in the data train subfolder. To load our dataset into PyTorch we use the ImageFolder class, passing it two arguments: root is the data path, and transform is the transformation to apply to the upon loading, here: conversion to tensors. We assign the dataset to train\_dataset. Now, we can access the class labels from the train dataset using dot-classes. We have two labels, cat and dog. The class to idx attribute maps class labels and their indices. Cat is zero and dog is one.

**Binary image classification: convolutional layer**

Let's build the binary CNN model. The Conv2d layer has three input RGB channels for red, green, and blue, sixteen output channels, and a three-by-three kernel that moves one stride, or step, at a time. One-pixel padding is added around the image border. We also define the ReLU activation function and the MaxPool2d layer with a two-by-two kernel size and stride of two.

**Binary image classification: fully connected layer**

The flatten layer reshapes tensors into a one-dimensional vector. This vector is passed to the linear layer with input features equal to the number of feature maps times their height and width. The output is a just one value, which we pass to a sigmoid activation. Finally, in the forward method, we pass the input through subsequent layers and return the output.

**Multi-class image classification with CNN**

For the multi-class model, we adjust the final layer output by specifying the number of classes. We also modify the activation function to softmax. We use dim equal one as this dimension stores classes.

