**Working with pre-trained models**

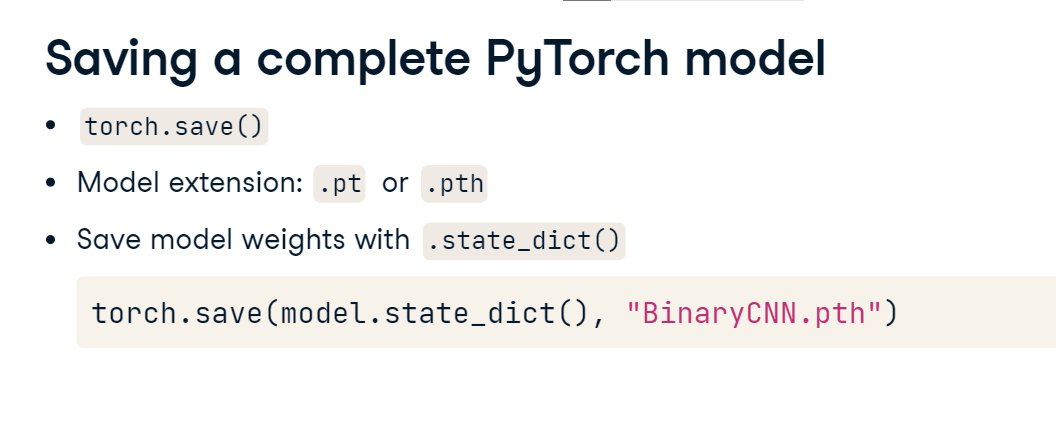
Welcome back! Working with pre-trained models in deep learning saves time and resources. We will be leveraging them throughout the course. Let's see how.

**Leveraging pre-trained models**

Training deep learning models from scratch is a long and tedious process, and it typically requires a lot of training data. Instead, we can often use pre-trained models, that is models that have already been trained on some task. Sometimes, we can directly re-use a pre-trained model if it can already solve the task we care for. We will see this many times throughout the course. On other occasions, we might need to adjust the pre-trained model to fit the new task. This is known as transfer learning and we won't cover it here. Pre-trained models can either be models that we have previously trained ourselves, or models publicly available on the Internet. To be able to leverage them, we will learn how to save and load models locally, and how to download models offered by torchvision.

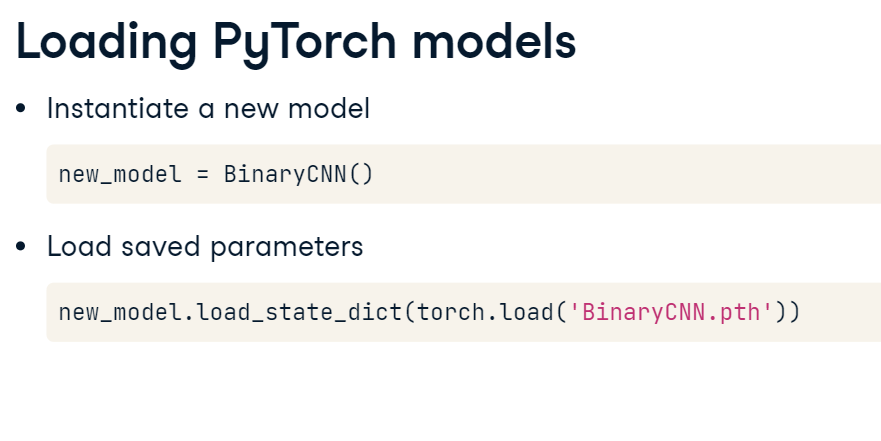
**Saving a complete PyTorch model**

To save a model, we can use torch-dot-save. A common file extension for torch models is either pt or pth. To save the model's weights, we pass model-dot-state-dict to torch-dot-save providing the output file name, for example BinaryCNN-dot-pth.



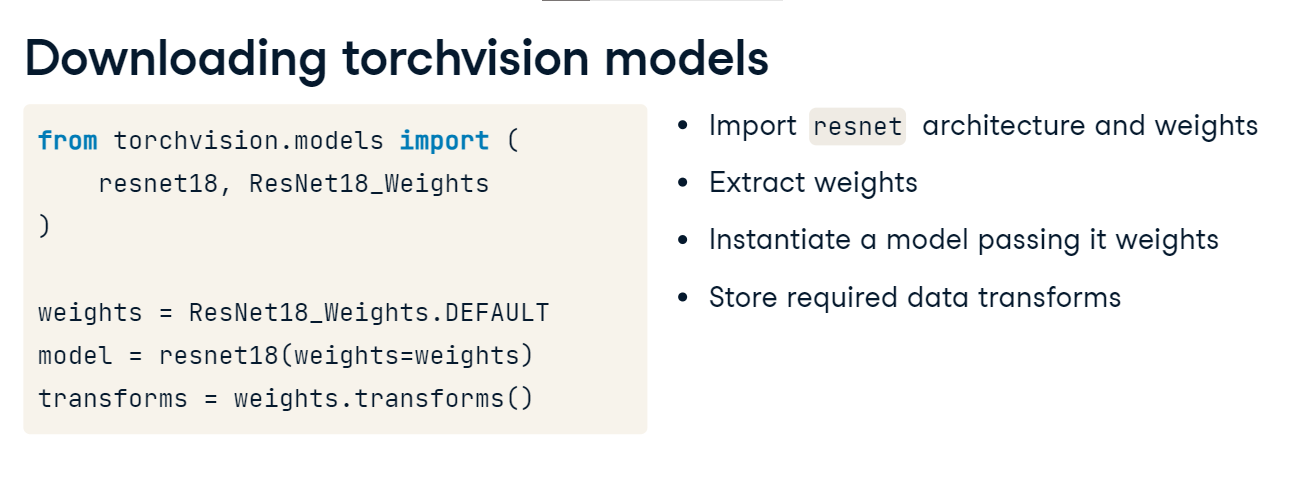
**Loading PyTorch models**

To load a saved model, we initialize a new model with the same architecture. We then use the load state dict method together with torch-dot-load to load the parameters to the new model.



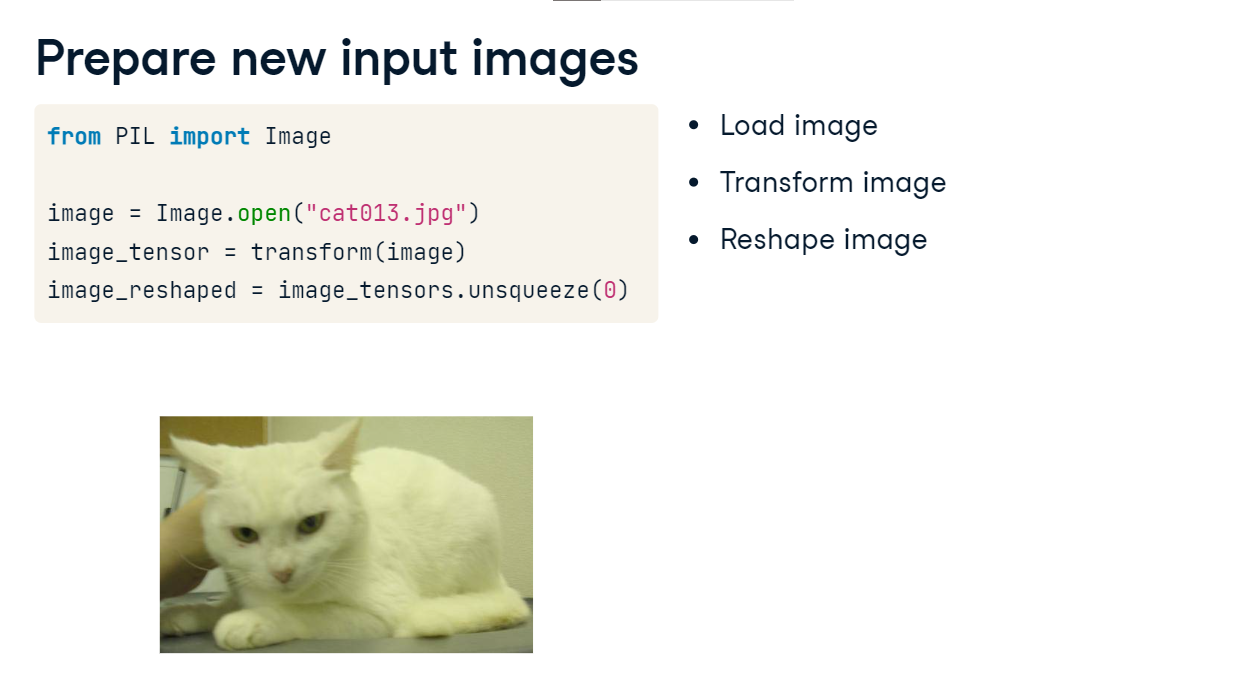
**Downloading torchvision models**

Torchvision provides a collection of pre-trained models for various image-related tasks. These models are pre-trained on large-scale image datasets and are easily available. Let's download one of them and use it to classify an image. We will use ResNet18, a popular convolutional network pre-trained for image classification. We start by import both the model architecture and its pre-trained weights from torchvision-dot-models. We extract the weights with resnet18 weights-dot-default and instantiate the model with the resnet architecture and weights as a parameter. The weights represent the model's knowledge. Finally, we store the data transformations used by the pre-trained model using weights-dot-transforms. We'll need to apply the same transformation to our data so that the input data fits what the model expects.



**Prepare new input images**

Let's prepare new input images for our resnet model. We load an image using the image-dot-open method from the PIL library. We preprocess the image using the ResNet transforms we saved earlier. Finally, we add an extra batch dimension using unsqueeze as required by PyTorch.



**Generating a new prediction**

We're now ready to predict with our model! We set the model to evaluation mode using model-dot-eval to prepare the model for inference. To classify our input image, we start with disabling gradients calculation for all layers using torch-dot-nograd. We pass the image to the model and remove the batch dimension from the output using squeeze. We apply softmax to the prediction output, use argmax to find the highest probability class index in the softmax output, and map that index to a label using the weights-dot-meta dictionary. The predicted class is Egyptian cat. Here, we used the pre-trained model along with its original labels - the model already learned to identify different cats, so we could directly re-use it to classify our cat image.

