# 1.1 Image as Data

**Introduction**

**Summary:** In this lesson, we'll learn how to work with tensors in PyTorch. We'll also explore the dataset for this project, focusing on how images are represented in tensors.

**Objectives:**

- Check important attributes of tensors, such as size, data type, and device.
- Manipulate tensors through slicing.
- Perform mathematical operations with tensors, including matrix multiplication and aggregation calculations.
- Download and decompress the dataset for this project.
- Load and explore images using PIL.
- Demonstrate how visual information is stored in tensors, focusing on color channels.

**New Terms:**

- Attribute
- Class
- Color channel
- Method
- Tensor

## Getting Ready

Before we dive into this lesson, there are two things we need to address. Refer to the notebook for more details.

## Working with Tensors in PyTorch

The term **tensor** comes from mathematics. It refers to an array of values organized into one or more dimensions.

In Python, there are several libraries for creating and manipulating tensors. In this program, we'll use PyTorch, which is built for deep learning. We'll build our computer visions with PyTorch.

In this section, we'll get familiar with tensors and what we can do with them. We'll begin making a 2-dimensional tensor from a nested list.

## Task 1.1.1

Use the nested list `my_values` to create the tensor `my_tensor`.

Print the dimensions and data type of `my_tensor`.

## Task 1.1.3

Print the device of `my_tensor`.

## Task 1.1.4

Change the device of `my_tensor` to `"cuda"`.

## Tensor Slicing

There are several ways to manipulate tensors. One important technique is **slicing**, where we use square brackets `[]` and indexing to select a subset of the values in a tensor.

Let's give it a try with `my_tensor`.

## Task 1.1.5

Slice `my_tensor`, assigning its top two rows to `left_tensor` and its bottom two rows to `right_tensor`.

## Tensor Math

Another way to manipulate tensors is to use mathematical operations. For example, we can perform addition using either the `+` operator or the `add()` method.

### Task 1.1.6

Use both the mathematical operator and the class method to add `left_tensor` to `right_tensor`. Assign the results to `summed_tensor_operator` and `summed_tensor_method`, respectively.

### Task 1.1.7

Use both the mathematical operator and the class method to multiply `left_tensor` to `right_tensor`. Assign the results to `ew_tensor_operator` and `ew_tensor_method`, respectively.

### Task 1.1.8

Use both the mathematical operator and the class method to perform matrix multiplication on `new_left_tensor` and `new_right_tensor`. Assign the results to `mm_tensor_operator` and `mm_tensor_method`, respectively.

### Task 1.1.9

Calculate the mean for all values in `my_tensor`.

### Task 1.1.10

Calculate the mean for each column in `my_tensor`.

## Explore Files

In this lesson, we'll focus on the multi-class training data. Let's define a variable for the directory that includes all the multi-class data and one for that contains the training data.

### Task 1.1.11

Following the pattern of `data_dir`, assign the path to the multi-class training data to `train_dir`.

### Task 1.1.12

Create a list of the contents of `train_dir`, and assign the result to `class_directories`.

### Task 1.1.13

Complete the `for` loop so that `class_distributions_dict` contains the name of each subdirectory as its keys and the number of files in each subdirectory as its values.

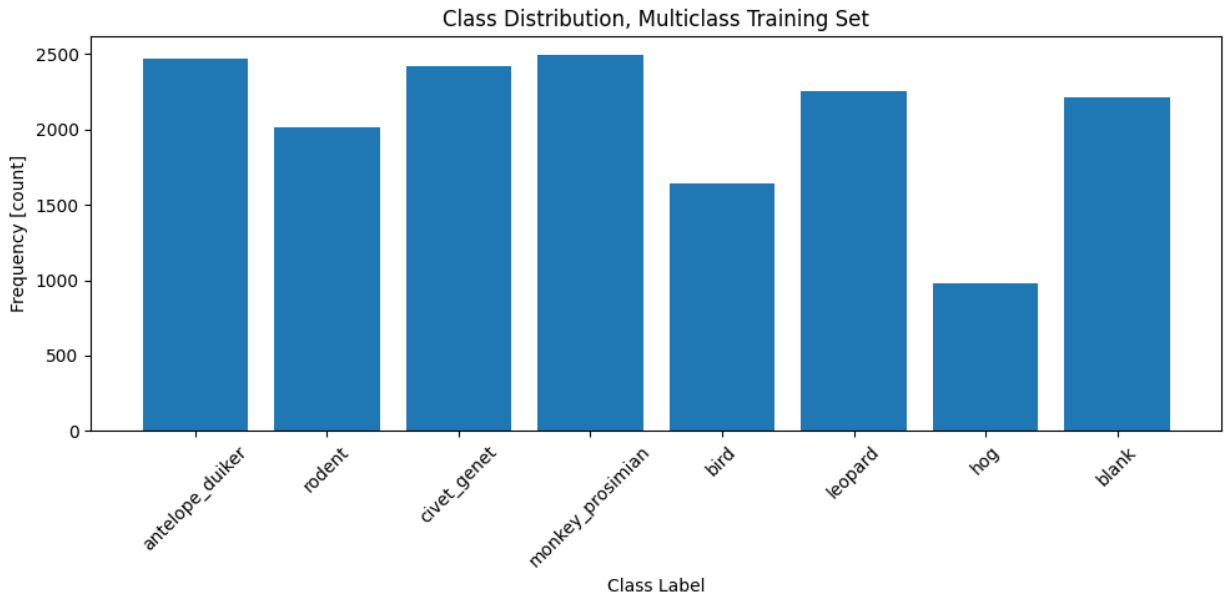
## Graded activities

### 1.1.14

**Create a bar chart from `class_distributions`**

Let's make a bar chart from `class_distributions`.

**Expected outcome:**




## Load Images

We know the distribution of our data, but what do the actual images look like? Let's select a couple to explore further. Here are the paths for a hog and an antelope. 🐷🦌

## Task 1.1.15

Use PIL to open `antelope_image_path`.

## Task 1.1.16

Get the `.size` and `.mode` attributes from `antelope_image_pil` and assign the results to `antelope_image_pil_size` and `antelope_image_pil_mode`, respectively.

## Load Tensors

We've loaded two image files using the Pillow library. For that reason, they're represented using the `JpegImageFile()` class. However, we'll need to represent them as tensors if we want to train a model.

The PyTorch community has created the `torchvision` library, which comes with lots of helpful transformation tools. We can use the `ToTensor()` class to convert `hog_image_pil` to a tensor.

## Task 1.1.17

Convert `antelope_image_pil` to a tensor and assign the result to `antelope_tensor`.

## Task 1.1.18

Complete the code below to plot the red, green, and blue channels of `antelope_tensor`.

## Task 1.1.19

Calculate the minimum and maximum values of `antelope_tensor` and assign the results to `max_channel_values` and `min_channel_values`, respectively.

## Graded activities

### 1.1.20

**Calculate the mean values of the separate color channels in `antelope_tensor` and assign the result to `mean_channel_values`**

Calculate average color channel values for antelope image.

**Expected outcome:**

Here is an example of your `mean_channel_values`

```
mean_channel_values class: <class 'torch.Tensor'>
mean_channel_values shape: torch.Size([3])
mean_channel_values dtype: torch.float32
mean_channel_values device: cpu
Mean channel values in antelope_tensor (RGB): tensor([0.2652, 0.3679, 0.3393])
```

---

You

**must always**:

- ✓ Give credit to WorldQuant University for the creation of this file
- ✓ Provide a link to the license