**Handling images with PyTorch**

**Clouds dataset**

We will work with the clouds dataset from Kaggle containing photos of seven different cloud types. We'll build an image classifier to predict the cloud from an image. But first - what is an image?

1. 1 https://www.kaggle.com/competitions/cloud-type-classification2/data

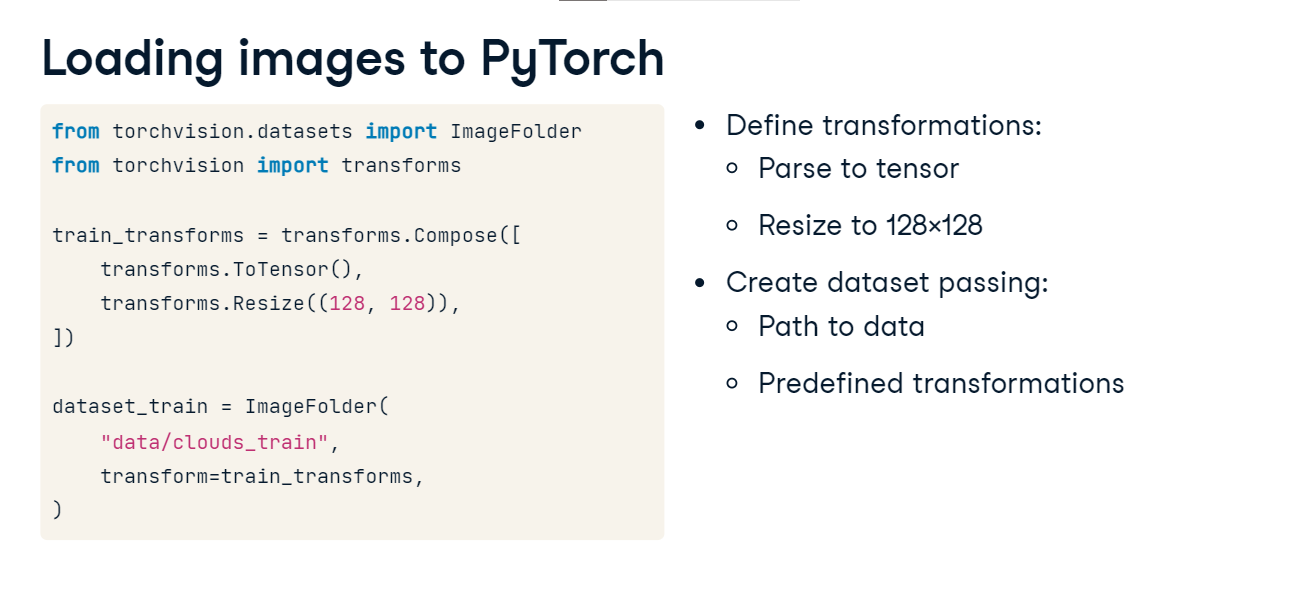
**What is an image?**

Digital images are comprised of pixels, short for "picture elements". A pixel is the smallest unit of the image. It's a tiny square that represents a single point. If we zoom into this cloud picture, we can see the pixels. Each pixel contains numerical information about its color. In a grayscale image, each pixel represents a different shade of gray, ranging from black to white which would be an integer between 0 and 255, respectively. A value of 30, for example, represents the following shade of gray. In color images, each pixel is typically described by three integers, denoting the intensities of the three color channels: red, green, and blue. For example, a pixel with red of 52, green of 171, and blue of 235 represents the following shade of blue.

**Loading images to PyTorch**

Let's build a PyTorch Dataset of cloud images. This is easiest with a specific directory structure. We have two main folders called cloud\_train and cloud\_test. Within each, there are seven directories, each representing a cloud type, or one category in our classification task. We have jpg image files inside each category folder.

With this directory structure, we can use ImageFolder from torchvision to create a Dataset. First, we need to define the transformations to apply to an image as it is loaded. To do this, we call transforms.Compose and pass it a list of two transformations: we parse the image to a torch tensor with ToTensor and resize it to 128 by 128 pixels to ensure all images are the same size. Then, we create a Dataset using ImageFolder, passing it the training data path and the transforms we defined.



**Displaying images**

dataset\_train is a PyTorch dataset just like the WaterDataset we saw before. We can create the DataLoader from it and get a data sample. Notice the shape of the loaded image: 1 by 3 by 128 by 128. 1 corresponds to the batch size of 1, 3 - to the three color channels, and 128 by 128 represents the image's height and width. To display a color image like this, we must rearrange its dimensions so the height and width come before the channels. We call squeeze on the image to eliminate the 1-dimensions of the batch size, and then permute the order by replacing the original order of dimensions, 0-1-2, with 1-2-0: this way, we place the channel dimension at the end. For grayscale images, this permutation is not needed. This lets us call plt.imshow from matplotlib followed by plt.show to display the image.



**Data augmentation**

Recall the dataset building code. We said that upon loading, one can apply transformations to the image, such as resizing. But many other transformations are possible, too. Let's add a random horizontal flip, and rotate by a random degree between 0 to 45. Adding random transformations to the original images is a common technique known as data augmentation. It allows us to generate more data while increasing the size and diversity of the training set. It makes the model more robust to variations and distortions commonly found in real-world images, and reduces overfitting as the model learns to ignore the random transformations. Here's a sample of augmented images using rotation.

