Pytorch Convolutational Net Tutorial

TorchVision (TV) Built-in DataSets: https://pytorch.org/vision/stable/datasets.html

```
from torchvision import datasets
trainset = datasets.CIFAR10(root='./data', train=True,  download=True)
testset  = datasets.CIFAR10(root='./data', train=False, download=True)
```

Convolutional Data format: `BxCxHxW`

- Batch: determined by the training function/dataloader
- Dataset must formatted as CxHxW
- C: channels
    - For color input images: C = 3, for grayscale/BW input: C = 1
    - For convolutional output: C = number of kernels (aka filters)
- HxW: image height and width

Pytorch 2D Convolutional layer: https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html

```
Conv2D(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1,
groups=1, bias=True, padding_mode='zeros', device=None, dtype=None)
```

- Must define the Convolutional function in the .init of the network. The used in the forward of the network
- in_channels = number of channels in the input
- out_channels = number of kernels used
- kernel_size = (n,n) where n is an odd number
- stride = 1 by default.
    - Increasing stride reduced image HxW in the output
        - (H//s) x (W//s)
- padding=0 =>
    - no padding => image HxW reduced in output To pad so that the HxW
        - (H-N)x(W-N), N = n//2*2
    - To pad so that the image size in unchanged: padding = n//2
- Dilation=1 => filter is not dilated
    - Dilated filters are atrous convolutions.

Pytorch 2D Pooling layer: https://pytorch.org/docs/stable/nn.html#pooling-layers

torch.nn.MaxPool2d(kernel_size, stride=None, padding=0, dilation=1, return_indices=False, ceil_mode=False)

- Must define the Convolutional function in the .init of the network. The used in the forward of the network
- Kernel_size: typically (2,2)
- Stride: typically 2

Transformations via TV V2: https://pytorch.org/vision/stable/transforms.html
https://pytorch.org/vision/stable/auto_examples/transforms/plot_transforms_illustrations.html

```
from torchvision.transforms import v2
transforms = v2.Compose([                    #applies all the following
                v2.ToImage(),                #tensorize for PIL image
                v2.ToDtype(torch.float32), #tensorize images (instead of ToTensor())
                v2.Normalize(mean, std),     #normalize
                v2.Resize((H,W)),            #resize image to HxW
                ...)]
```

Realtime Data Augmentation:

- v2.Compose([transforms]) to apply a number of transforms together
- v2.RadomApply([transforms], p): apply randomly with a given probability (p)
    - Alternatively, use v2.Compose([random_transforms])
- Realtime: during training
    - Due to the randomness each batch is unique
- Make sure that that the transforms matches real data
- Method 1: apply transform to dataloader batch
- Method 2: write your own custom dataloader functions

Transformations via Albumentations: https://albumentations.ai/docs/api_reference/augmentations/
import albumentations as A
from albumentations.pytorch import ToTensorV2

- Albumentations are faster than other packages (even CV2!)