CNN implementation

In this task you will implement a classifier for the CIFAR-10 dataset. The objective is to classify the input images correclty in one of the 10 classes of the dataset.

E0 Download and extract the python version of the dataset from the CIFAR website. Take a moment to look at the description of the data inside the new folder

E1 Create a Dataset class to read the data. When initialized, this class should take as arguments the path to the data, the transformation to be applied to each image (see TIP2) and if the dataset is train or test. If train you should load all the 5 batches that composed the whole CIFAR training set. [2.0 pts]

TIP1 The code to load the CIFAR dataset in batches can be found in the main page of the dataset. Note that data are loaded with bytes encoding so to access elements in the returned *dict* you should use *dict[b'key']* (The only difference is the *b* before the string of the key).

TIP2 We want to normalize the images in [-1,1]. To do so you can use the torchvision package and the transforms module:

```
import torchvision.transforms as transforms
transform = transforms.Compose([
    # Convert images to tensors
    transforms.ToTensor(),
    # Normalize to range [-1, 1]
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
    ])
tranform_img = transform(img) #img is one image in the ds
```

TIP3 If you are curious and want to visualize some image you can use the following function:

```
import matplotlib.pyplot as plt
def imshow(img):
    img = img / 2 + 0.5     # unnormalize
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()
```

Me

E2 Build a CNN model to predict a class from the input image (you can use the Conv2D module and one of the plenty pooling layers already implemented). Which are the main hyperparameters you should set to build the main model? Good practice is to build the model class as general as possible, and specify the hyperparaemeters when the class is called. [2.0 pts]

E3 Train and test your model with the following hyperparameters configuration:

- **layer 1**: Convolutional + activation with 6 output channels and 5 kernel
- **layer 2**: Pooling (which pooling is better?) with kernel size of 3 and a stride of 2
- **layer 3**: Convolutional with 16 output channels and 5 kernel
- **layer 4**: Pooling
- **layer 5**: A fully connected layer (Linear + activation) with output size 120 (which is the input size?)
- **layer 6**: A fully connected layer with output size 84
- **layer 7**: The output layer

Use the SGD optimizer with learning rate=0.001 and momentum=0.9 (it works better than Adam for image classification task, if you are interested to know why you can have a look at this paper) and choose the proper loss and activation functions and the batch size (how to choose it? which factor should you take into consideration?). Plot the loss functions, how many epochs are needed to achieve a good performance? If you want you can play with the parameters to see if you can improve the performances [2.0 pts]

TIP4 Remember that between the last convolutional layer and the first fully connected layer you should flat the data in a 1D vector. You canuse the flatten function of Pytorch to flat all the dimension but the batch one.

E4 Properly evaluate your model: remember that a proper visualization of what is happening inside the model is as crucial as the model itself. For example, you can evaluate the accuracy and precision of your model for each class and the mean ROC AUC. [2.0 pts]

E5 Now try to build and train a MLP model for the same task (flatting the image in a 1D vector), can you do that? Does this new model achieve the same results as the CNN? In the same amount of time? Why? [2.0 pts]