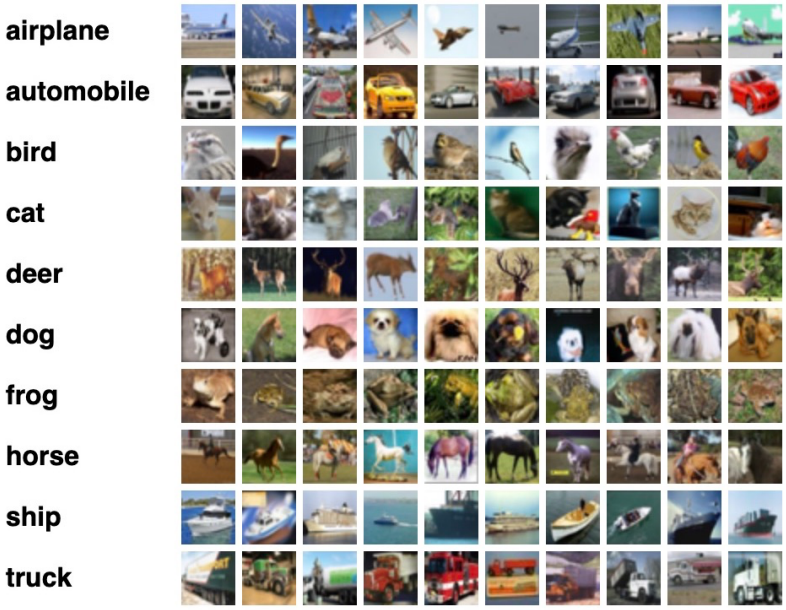
INFO4150E Mini Project 2 Instructions - Deep Learning - CNN

Objective

To build a Convolutional Neural Network model for 10 classes of images in the Cifar10 dataset. The model will be built from scratch, trained, and used for prediction based on new data. The complete MP2 will be done using the PyTorch framework in the Jupyter Notebook template provided.

Cifar10 dataset

The **CIFAR-10 dataset** (Canadian Institute for Advanced Research) is a collection of images that are commonly used to train machine learning algorithms. It is one of the most widely used datasets for machine learning research. The CIFAR-10 dataset contains 60,000 32x32 color images in 10 different classes. 50000 are used for training and 10000 for testing. The 10 different classes represent airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. There are 6,000 images of each class. Here is a picture showing the 10 classes and some sample images. The 10 classes are encoded as numbers in the dataset.



The data set is provided to you as csv files, **'cifar10.csv'** (used for training) and **'cifar_test.csv'** (used for testing). The images are in color and have a size of 32x32 and so the shape is 32,32,3. Each row of the dataset carries the pixel values of an image for the red, blue, and green channel. In other words, each row of the csv file has 32x32x3 = 3072 columns. The last column contains the labels of the images encoded in numbers.

Labels:

- airplane: 0
- automobile: 1
- bird: 2
- cat: 3
- deer: 4
- dog: 5
- frog: 6
- horse: 7
- ship: 8
- truck: 9

The end product of the project is a predictive image classification model which should be able to predict, given a new image, what the object is.

Guidelines and Tasks to be completed

The guidelines are as follows:

1. The project will be completed in a Jupyter Notebook provided to you as a template.
2. The template will have some pre-populated cells and some cells to be worked upon by you.
3. The cells and areas where you must complete the code will be clearly marked as **# To Do** in the notebook.
4. Explanations about what every cell of code is doing in the overall application will be given and so you are expected to have an overall understanding of how the final application works.
5. The tasks to be completed and the instructions given below to complete them should be read carefully.
6. You will complete the project, including answering the questions below in a last markdown cell, and submit it with a file naming convention "**First_LastName_MP2.ipynb**".

Tasks:

1. The two main tasks you will complete in this MP2 are:
    a. Assigning to the variable data_path, the location where your 'cifar' directory or folder is. Please make sure that you create a directory or folder called cifar and have all your files in that. The cell shows an example for my directory location, and you must change that to yours. Do not change the variable name 'data_path'.

    b. Completing the PyTorch model building Class to build a CNN model (see information below on how to go about it). In building the model you will use the following parameters:

        i. Kernel or filter of size 3
        ii. Padding of 1
        iii. Max_Pooling kernel size of 2.
        iv. Stride is 1 as a default.
        v. Wherever you need an activation function – you will use ReLu.

vi. For this project just create a max of 3 convolution layers and stick to output channels of 32,64 and 128. The reason I suggest this is to be within the limits of your computer's capability.
vii. Use of dropout layers is also a choice you have make.

2. Do not copy and paste from elsewhere as there is no standard solution and the rest of the code will mostlikely not run.

3. If the tasks are implemented correctly, the rest of the pre-populated template will run the complete application and generate the required results.

4. In addition to completing these tasks you will answer the following questions in a Markdown cell at the end of the Jupyter notebook:

    a. Explain in detail the strategy used in implementing the Dataset class. The answer should clearly describe how the data is being accessed in the __init__ function, and how it is delivered to the DataLoader It should also explain the processing I am doing on the images in the __get_item__ method.

    b. Describe in brief, the 3 main things that this complete application is doing, and the parameters one can tweak to build a good final model.

    c. How accurate were your predictions on percentage basis?

    d. What were the challenges you faced while completing this HW and what helped you in putting it together?

Important information to complete the Model building Class

1. The model to be built, like we saw in the MNIST example in class, is a CNN model consisting of convolution layers and fully connected layers, activation functions augmented by MaxPool and perhaps Dropout layers.
2. We also define the kernel sizes, stride, and padding values in the convolution layers.
3. Here is a ready reference to how a typical convolution layer architecture is built (please also refer to PyTorch's documentation online as it is very informative).
    a. Input conv layer: **nn.conv2d(inp_channels , out_channels, kernel, stride, padding)** where inp_channels would be the number of channels that the image has (1 in our case). The out_channels tell us how many kernels we want to implement (build up as multiples of 32 per conv layer as a general guideline).
    b. Activation function: **F.ReLU()**
    c. MaxPool Layer: **F.MaxPool2d (2)** , with kernel size defined.
4. Fully Connected Layer (FC) – These are the fully flattened layers after the Convolution layers and typically there are 3 to 4 of them. So, if the total number of Neurons after unraveling / flattening the last convolution layer is N, then the first FC layer will have N,1024 or typically some such connection. The next FC layer could have 1024,512 and the output layer could have 512, num of classes. In our case the output layer in this example would be 512,10.

In between one of the FC layers you can also throw in a Dropout layer with a probability of 0.2 as a typical value.

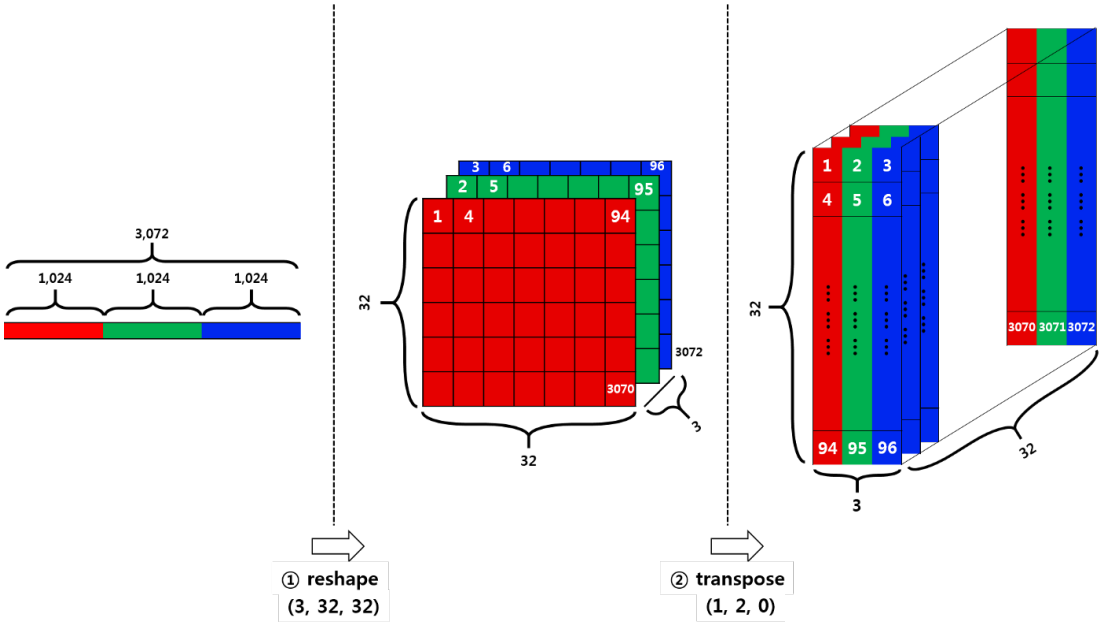
5. You will notice that ReLU, MaxPool etc. come from the "**import torch.nn.functional as F**" module.
6. The final image size after the last convolution layer must be computed. Remember that the kernel, the stride, the padding and the MaxPooling operators change the size of the original image (32x32) and is governed by the formulas S = (W – K +2P)/Stride. + 1 and the output of this gets divided by the MaxPool kernel size (if MaxPooling is done after any convolution layer). Let us look at an example for a color image starting from the input layer:

    a. ***x = Conv2D(inp_ch=3, out_ch=32,ker=3,stride=1,padding=1)***
    b. ***x = Activation(F.ReLU())***
    c. ***x = Conv2D(inp_ch=32, out_ch=64, ker=3, stride=1, padding=1)***
    d. ***Pool = MaxPool2D(2,2)***
    e. The computation for image size, starting with input image size of 32x32, at the end of the convolution layers will be:
        i. Conv1: **(32-3+2*1)/1 +1 = 32** (Our input image has a symmetrical shape, so we don't need to treat width and breath separately).
        ii. Conv2: **(32-3+2*1)/1 + 1 = 32** (Notice because of the padding the image size does not shrink).
        iii. Pool: **(32/2) = 16** (Note that the pooling layer reduces the image size by the pooling kernel size).
        iv. So now we know that the width & height values are 16x16. The final convolution layer will therefore have 16*16*64 neurons and this needs to get connected to the FC layers.
        v. Important to note is that the first dimension through this model building process is always the batch size and the Class takes care of that. But we must reshape our data at this point to provide for that. The equivalent of reshape in the tensor world is the operator **'view'**.
        vi. x = x.view(-1,16*16*64) and so the first dimension will take whatever the batch value is.

<u>Important information to about the dataset</u>

1. **Why it takes the preprocessing to extract the color image from the Cifar10 dataset provided is explained in the image below -**

    The 3072 pixels in each row of data is arranged such that the first 1024 represents the Red color, the next 1024 the Green and the final 1024 the Blue. Reconstructing is not possible and hence, we need to rearrange it.



We first must create the 3 RGB channels by reshaping the row of data.

Also, PyTorch is expecting the data in a certain form and so the order of each axes needs to be changed.

Also, finally we want to work with just a simple gray scale image.

Running the model

1. When you have completed Model Class, the template has ways to test if it is working which I will highlight in the template.
2. The epochs are set to 1 by default. Once everything runs, you can experiment by increasing the number of epochs to 3 or 4 depending on your hardware capabilities. However, for evaluation of your work it should run at least for one full epoch.
3. My model for this project gave me a max accuracy of about 65% after a couple of epochs. But the accuracy is not the important criteria in this MP2.

Predicting with the model

When you run the whole program, a function has been coded in the end which loads the saved, trained model files and predicts on new pictures which are in the directory. If your code runs flawlessly after you build the model, you should get reasonable predictions of 80% or more accuracy. Feel free to increase the number of epochs up to 4 or 5 if you want to improve the accuracy. Please keep in mind that, depending on your computer, this training process can take 5 to 15 mins.

Make sure that the test images I upload are in the same directory as the. ipynb file and do not change their names either.

The project submission deadline is July 21st, 2022, midnight. Please start on it as early as you can. No extensions will be given as we have 2 more assignments to do after this.