1. **(Deep) CNNs for Image Colorization**

   (a) This assignment uses a convolutional neural network for image colorization which turns a grayscale image to a colored image.[1] By converting an image to grayscale, we loose color information, so converting a grayscale image back to a colored version is not an easy job. We will use the CIFAR-10 dataset. Downolad the dataset from `http://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz`.

   (b) From the train and test dataset, extract the class birds. We will focus on this class, which has 6000 members.

   (c) Those 6000 images have $6000 \times 32 \times 32$ pixels. Choose at least 10% of the pixels randomly. It is strongly recommended that you choose a large number or all of the pixels. You will have between $P = 614400$ and $P = 6144000$ pixels. Each pixel is an RGB vector with three elements.

   (d) Run k-means clustering on the P vectors using $k = 4$. The centers of the clusters will be your main colors. Convert the colored images to k-color images by converting each pixel's value to the closest main color in terms of Euclidean distance. These are the outputs of your network, whose each pixel falls in one of those k classes.[2]

   (e) Use any tool (e.g., openCV or scikit-learn) to obtain grayscale $32 \times 32 \times 1$ images from the original $32 \times 32 \times 3$ images. The grayscale images are inputs of your network.

   (f) Set up a deep convolutional neural network with two convolution layers (or more) and two (or more) MLP layers. Use 5×5 filters and a softmax output layer. Determine the number of filters, strides, and whether or not to use padding yourself. Use a minimum of one max pooling layer. Use a classification scheme, which means your output must determine one of the $k = 4$ color classes for each pixel in your grayscale image. Your input is a grayscale version of an image ($32 \times 32 \times 1$) and the output is $32 \times 32 \times 4$. The output assigns one of the $k = 4$ colors to each of the 32×32 pixels; therefore, each of the pixels is classified into one of the classes $[1\ 0\ 0\ 0], [0\ 1\ 0\ 0], [0\ 0\ 1\ 0], [0\ 0\ 0\ 1]$. After each pixel is classified into one of the main colors, the RGB code of that color can be assigned to the pixel. For example, if the third *main color*[3] is [255 255 255] and pixel (32,32) of an image has the one-hot encoded class [0 0 1 0], i.e it was classified as the third color, the (32,32) place in the output can be associated with [255 255 255]. The size of the output of the convolutional part, $c_1 \times c_2$ depends on the size of the convolutional

[1]MATLAB seems to have an easy to use CNN library. `https://www.mathworks.com/help/nnet/examples/train-a-convolutional-neural-network-for-regression.html`

[2]Centers of clusters have been reported too close previously, so the resultant tetra-chrome images will be very close to grayscale. In case you would like to see colorful images, repeat the exercise with colors you select from `https://sashat.me/2017/01/11/list-of-20-simple-distinct-colors/` or `https://www.rapidtables.com/web/color/RGB_Color.html`. A suggestion would be Navy = (0,0,128), Red =(230, 25, 75), Mint = (170, 255, 195), and White = (255, 255, 255).

[3]Do not use the original CIFAR-10 images as the output. You must use the tetrachrome images you created as your output.

layers you choose and is a feature map, which is a matrix. That matrix must be *flattened* or *reshaped*, i.e. must be turned into a vector of size $c_1c_2 \times 1$, before it is fed to the MLP part. Choose the number of neurons in the first layer of the MLP (and any other hidden layers, if you are willing to have more than one hidden layer) yourself, but the last layer must have $32 \times 32 \times 4 = 4096$ neurons, each of which represents a pixel being in one of the $k = 4$ classes. Add a softmax layer[4] which will choose the highest value out of its $k = 4$ inputs for each of the 1024 pixels; therefore, the output of the MLP has to be reshaped into a $32 \times 32 \times 4$ matrix, and to get the colored image, the RGB vector of each of the $k = 4$ classes has to be converted to the RGB vector, so an output image will be $32 \times 32 \times 3$. Train at least for 5 epochs (30 epochs is strongly recommended). Plot training, (validation), and test errors in each epoch. Report the train and test errors and visually compare the artificially colored versions of the first 10 images in the test set with the original images.[5]

(g) Extra Credit (25 pts): Repeat the whole exercise with $k = 16, 24, 32$ colors if your computer can handle the computations.

[4]Compile the network with `loss = cross_entropy` .

[5]If you are using matplotlib, you may get a floating point error because to print an image, matplotlib either expects ints in range 0-255 or floats in range 0-1. You might be having, for example, 153.0 representation of 153 in your array and this is what makes matplotlib think that you are sending floats.

Wrap your array into `np.uint8()`. It will convert 153.0 into 153. NOTE that you cannot use `np.round` or `np.int` etc because matplotlib's requirement is unsigned int of 8 bit (think 0-255).