A Brief Report on the Image Colorization Challenge

- As instructed, the dataset was split into train and validation for the basic model.
- As instructed, the data loader, loss function, optimizer, training loop, and validation loop have been completed.
- An inference script named "inference_script.py" has been developed that takes as input grayscale image, model path and produces a color image.
- The following arguments can be passed from the command line for training:
    1. Images directory
    2. The train:validation split of the dataset
    3. Number of epochs
    4. Whether or not to save the images during validation
    5. Learning rate for training
    6. Weight decay for Adam optimizer
    7. Whether or not to save the model after training
    8. Choice of loss function: MAE or MSE
    9. Batch size for training
- The following arguments can be passed from the command line for inference:
    1. Model path
    2. Image path

Q. Determine model performance using appropriate metric. Describe your metric and why the metric works for this model?

A. Since the task at hand is a regression of pixel values, I have implemented both Mean Squared Error (MSE) loss function and Mean Absolute Error (MAE) loss function to determine the model performance. The MSE/L2 loss minimizes the squared distance between the color value we try to predict, and the true (ground-truth) color value, whereas the MAE/L1 loss minimizes the absolute distance between the same. Both the metrics consider that a pixel value close to the true value corresponds to better performance. Although this does not hold true while predicting the color of entities that could have several 'true' colors, this generally works. For example, the color corresponding to the sky is most likely blue and that of the trees is most likely green. Even if the true color of the sky or the tree were orange, due to sunset or autumn, the model will succeed to produce a natural image even if it learns to predict blue and green respectively.

For this model, **MAE/L1** Loss is preferred over L2/MSE loss because it reduces the effect of producing grayish images. With L2 loss, the model still learns to colorize the images but it is conservative and most of the time uses colors like "gray" or "brown" because when it doubts which color is the best, it takes the average and uses these colors to reduce the L2 loss as much

as possible. For example, if a gray house could be red or blue, and our model picks the wrong color, it will be harshly penalized. As a result, our model will usually choose desaturated colors that are less likely to be "very wrong" than bright, vibrant colors.

Q. The network available in model.py is a very simple network. How would you improve the overall image quality for the above system? (Implement)

A. I have implemented the following to improve the overall image quality of the system:

1. **Lab Color Space instead of RGB**: An RGB image is a rank 3 array with the last axis containing the color data which consists of 3 numbers for each pixel indicating how Red, Green, or Blue the pixel is. In contrast, the Lab color space has three numbers for each pixel which encodes L - Lightness of each pixel, and ‘a’ and ‘b’ - how green-red and yellow-blue each pixel is. The reason for using Lab color space is:
    a. The input is a grayscale image that already contains the Lightness (L) of the output. The model only needs to predict the other two channels ‘a’ and ‘b’.
    b. Predicting 3 numbers for an RGB output is a more unstable and difficult task compared to 2 numbers.
    c. Needs lesser computing power and takes lesser time to achieve comparable results.

    The implementation of Lab color space can be found in train.py and colorize_data.py.
2. **Transforms to introduce generalization**: It is important to make sure that the model does not overfit the training images so as to perform well on a test image. Adding RandomResizedCrop and RandomHorizontalFlip transforms introduce noise to the training data so that the model learns to adapt to randomness, which makes the model a better predictor of randomness in test images.
    a. **RandomResizedCrop** crops the 256x256 image to a random 224x224 portion of the image so that the model receives different parts of the image every time it encounters that image during training.
    b. **RandomHorizontalFlip** mirrors the image on a horizontal axis so that the model learns to colorize images regardless of the orientation of contents in the image.

The following are some more ideas for improving the model further:

3. **Data Augmentation**: Data augmentation increases the diversity and amount of training data by applying random (but realistic) transformations. For example, Image resizes, Image rotation, Image flip, Color jitter, and many more. The existing images are augmented and added to the training dataset as new images. The model thereby learns to adapt to these noises in input images and thereby creates more robust outputs at test time.
4. **Generative Adversarial Network**: Using a GAN is a better performing, yet compute-heavy solution to the colorization task. A GAN has a generator and a discriminator model which learn to solve a problem together. The generator model takes a grayscale image and produces a 2-channel image, a channel for ‘a’ and another for ‘b’ in

the Lab color space. The discriminator, takes these two produced channels and concatenates them with the input grayscale image, and decides whether this new 3-channel image is fake or real. The discriminator looks at some real images (3-channel images again in Lab color space) that are not produced by the generator and should learn that they are real. A GAN would be better than a CNN for this task as the task at hand does not have a well-defined correct output. The CNN model assumes that the target image is the only true image for an input. At the same time, in a GAN, as the generator is constantly trying to create an image that fools the discriminator, it will eventually learn to generate images that look realistic. A great implementation of GAN using U-Net can be found here.

5. **Improved loss functions:** A more complex loss function can be adopted which corresponds to the perceptual quality of the image better than a simple L1 or L2 loss.
    a. VGG Loss: The VGG loss is a content loss function, which is applied over generated images and real images. VGG19 is a very popular deep neural network that is mostly used for image classification. VGG19 was introduced by Simonyan and Zisserman in their paper titled Very Deep Convolutional Networks for Large-Scale Image Recognition, which is available at https://arxiv.org/pdf/1409.1556.pdf. The intermediate layers of a pre-trained VGG19 network work as feature extractors and can be used to extract feature maps of the generated images and the real images. The VGG loss is based on these extracted feature maps. It is calculated as the Euclidean distance between the feature maps of the generated image and the real image.
    b. L1 + L2 Loss: The L1 and L2 loss can be combined to give a more robust loss function that suffers less from the graying effect of L2 loss. More about this combined loss is mentioned in this paper https://arxiv.org/abs/1611.07004.

Q. Bonus: You are tasked to control the average mood (or color temperature) of the image that you are colorizing. What are some ideas that come to your mind? (Implement)

A. The color temperature of the image can be controlled during training or during inference.

1. During training: According to the required color temperature, the model can be taught to learn that the true image has a certain color temperature. This is carried out by applying a transform to every training image using the adjust_hue function as follows:

    `torchvision.transforms.functional.adjust_hue(img, hue_factor)`

    Here, the hue_factor is a float number in the range [−0.5, 0.5]. 0 gives a solid gray image, while -0.5 and 0.5 will give an image with complementary colors.
    This has been implemented in the files colorize_data_hue_control.py and train_hue_control.py. Some inference results are shown below:

Input grayscale image | Colorized image with warm temp | Colorized image with cool temp

2. During inference: The hue of output image can be manipulated after the model has completed training. Once the inference image is passed through the pretrained model, the output can be transformed using the same adjust_hue function of torchvision to change the average mood of the output.

RESULTS