Project Check In Report

CPSC 599, Deep Learning For Vision

Deliar Mohammadi, 30072994
John Zheng,30125258
Xinzhou Li, 30066080

Introduction

For this project we are tackling the problem of image colorization. Given an image that is grayscale, we would like to take that image and create a plausible colored version of that image. This topic has been researched by many computer vision experts in the past due to its wide range of applications. These applications include video and image editing, colorization of surveillance footage, restoration of historic photographs, and medical imaging. Before the era of machine learning, grayscale photos had to be painted by humans in order to be brought back to life. However, this approach is inefficient for many reasons. First, there are many interpretations of a photograph, and different artists may have vastly different color choices when it comes to image colorization. Another big issue with this problem is that it takes a lot of time. An artist can only colorize a single painting at a time, which makes many problems such as video colorization and medical colorization infeasible. However, neural networks solve both of the problems discussed above. They are able to generate a plethora of colored images from a single grayscale photo, allowing us as humans to decide the best color palette. Additionally, computers are much faster than humans, so many images can be colorized in parallel. This massive difference in speed also opens the door to things like video colorization, which was infeasible when done by hand.This problem has been approached many times as previously mentioned, and the oldest approach in the book was as follows:

1. Take an RGB image and feed the corresponding grayscale to a CNN
2. Have the CNN predict an output RGB image
3. Compare the predicted RGB image to the ground truth image (also known as original)
4. Calculate the loss as the difference in RGB from each predicted pixel to each true pixel
5. Update weights with the loss from step 4 and repeat this until the model is sufficiently trained.

Although the classic approach mentioned above is intuitive, it's not as effective as the method we will be using, which is the same method found in "Colorful Image Colorization". Instead of using the RGB color space, we can first convert the image into the LAB color space. Where L represents the grayscale image, A represents the red/green value and B represents the blue/yellow value. L falls within the range [0,100] whereas A and B fall between [-128, 127] inclusive. Once the image is converted to LAB we can feed the L dimension into a CNN with a series of convolutional blocks. At the end of the network, we can make a prediction of the A and B channels and concatenate that with the original L channel to form the full LAB image. The reason this method is more effective is because we are reducing the number of values our model needs to output. With the original RGB method, our model would produce an output that has 3 channels, one for red, one for blue and one for green, meaning that for each and every pixel we predict 3 different values that are between 0 and 255. It follows that there are a total of (256^3) = 16777216 combinations for each pixel. However with the LAB approach, our model only predicts the A and B value for each pixel, each of these fall between [-128,127] so there are a total of 256 possibilities for each of the 2 values, meaning there are a total of (256x256) / 2 = 32768 combinations. As we can see, the LAB approach drastically reduces the output space for our problem and makes it much more feasible for us to predict a plausible color

of a grayscale image. In addition to reducing the prediction space, the LAB color space is *"designed for perceptual uniformity which makes it ideal for computer processing"*. In other words, the LAB color space is more perceptually linear, meaning that a change of the same amount in a color value should produce a change of about the same visual importance.

In addition to colorizing images, there are several advancements within this sub topic that allows models to perform even better. One of these is color hinting, as proposed in this paper. Color hinting allows a user to select colors for any pixels within the image, and then allow the model to fill in the rest of the image based on what the user inputted. *Fig 1* demonstrates this feature.

Fig. 1. Our proposed method colorizes a grayscale image (left), guided by sparse user inputs (second), in real-time, providing the capability for quickly generating multiple plausible colorizations (middle to right). Photograph of *Migrant Mother* by Dorothea Lange, 1936 (Public Domain).

Goal

Our goal for this project remains largely unchanged from the proposal. We hope to be able to convert a grayscale image to a plausible colored image. With that being said, there are many potential outputs for a given object. Take for example an apple, if we were to feed a grayscale image of an apple to our network, the apple could be red,green or even yellow. All of these answers would be correct, and by our standards would be acceptable output from our network. However if the color of the apple were to be purple, this would mean the model is not properly trained yet as purple is not a viable color option for an apple. Apart from being able to achieve this task, we would also like to explore various model architectures for learning purposes, and to determine which model performs best. Up to date we have produced the eccv16 model that from *colorful colourization* , and the CUNet architecture from *Image Colorization Algorithm Based on Deep Learning*. These two models both seem promising as they have worked for previous iterations, however we would like to expand on these networks and potentially add more trainable parameters so that the models become more robust.

Data

For this project we are using two data sources. The first data source is the *MIRFLICKR dataset*. This dataset contains 25000 images, totaling almost 3GB of data. Each of the images vary in height and width, so we will need to resize the images before passing them to our network. We plan on using this dataset entirely for training, the reason being is that we will use a different dataset for validation, which will be discussed in the next paragraph. See figure 2 for some sample images from this dataset.

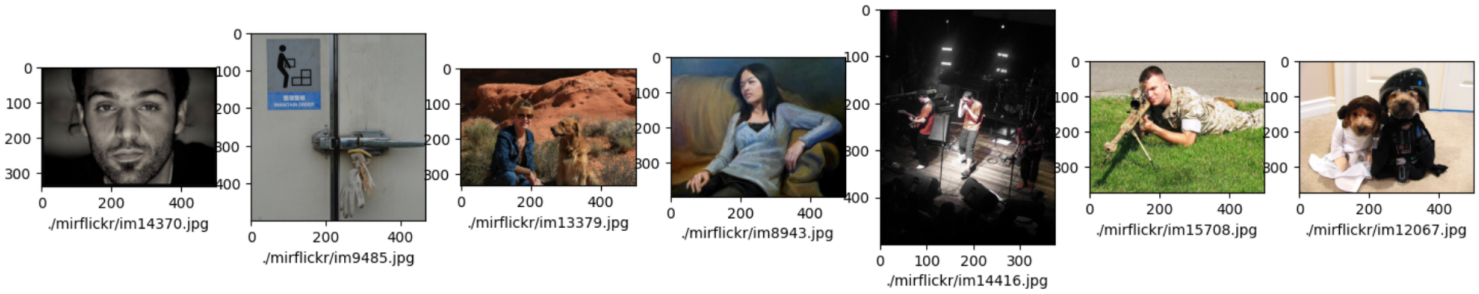

Figure 2

The second data source we have is a subset of the ImageNet dataset which was downloaded from *kaggle*. The original ImageNet dataset contains well over a million images, due to this very large volume, and our lack of computational power, we will not be able to train on all of these images. Instead we took a subset of the image net dataset, which is 70,000 images, totaling 7GB. The images are split into 45,000 and 25,000 for training and validation respectively. Figure 3 shows some sample images from this dataset.

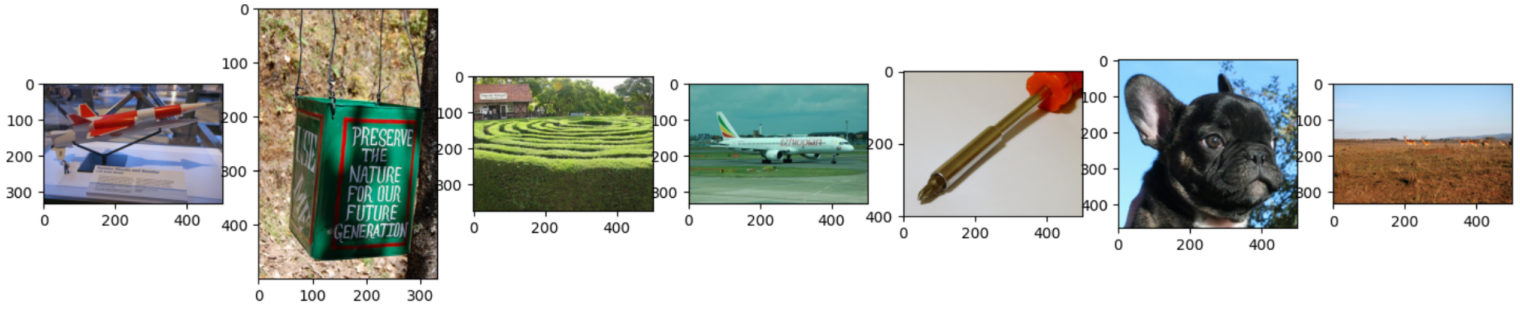

Figure 3

As we mentioned in the introduction, we need to convert these images into the LAB color space before feeding the L channel into our network. Once the network produces an output (2-channel A/B layer), we will compare that output to the original A&B channels of the image. Additionally, we will need to resize the images into 256x256 before feeding them to our network. To illustrate the color conversion and different channels of an LAB image refer to figure 4 below.

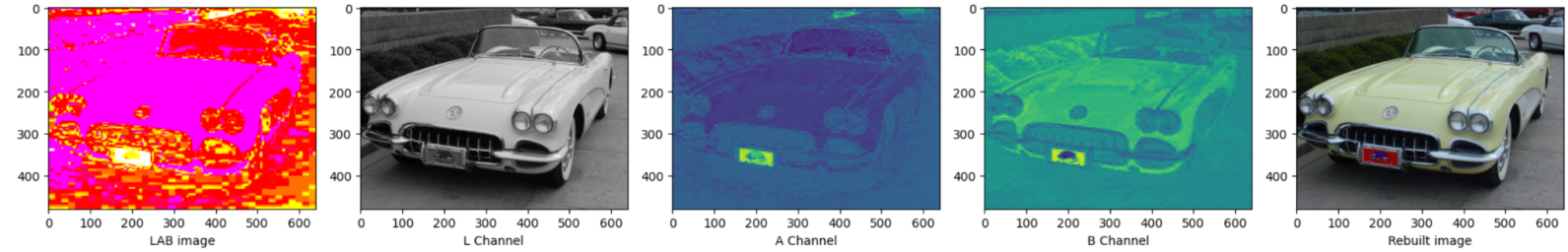

Figure 4

To summarize, our dataset is a combination of two different datasets, totalling 95000 images, which are split into 70,000 train images and 25,000 validation images. The only transformations done on the data before feeding our network are simple resizing and color space conversion from RGB to LAB.

Methodology

a) Data Pipeline

Our data pipeline for this project is fairly simple. Images are converted into the LAB space from the RGB space. This is done using the color library from sci-kit learn. These images are then resized into 256x256 so that we can feed them into our network. For our application it does not make sense to do any color transformations, this is because we are trying to predict the color itself! Random adjustments in brightness, saturation, and contrast would simply confuse the model. In addition, applying rotations or other transformations that adjust pixels would simply cause problems for our model. One transformation that we may consider in the future is random cropping, because random cropping does not effect individual pixels, rather it selects a subset from the image without altering any of them. Figure 5 shows our code snippet for resizing images. Figure 6 shows the entire transformation pipeline of an image from RGB to LAB.

```
'''
This function takes in an image and resizes it to 256 by 256
'''
def resize_img(self, img):
    return (img.resize((self.height, self.width)))
```

Figure 5

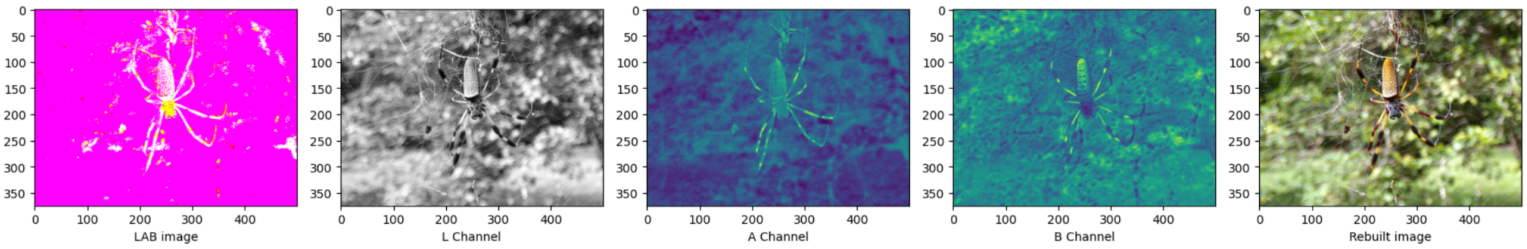

Figure 6

b) Model Architecture

In this project, we borrowed some ideas from the model eccv16 introduced by the article written by Rached. Zhao. Since we used the CIELAB color scale which has two color channels A and B each of them has a range from -128 to 127 and one lightness channel L which is from 0 to 100, since the lightness is directly given by the original input.

Therefore, our model only needs to learn how to predict the proper paired A,B value. There are actually (256*256) / 2 number of combinations of colors. Since we decide to resize all the input data to size 256 * 256, our model first take the convolution on our input images with the kernel size 3*3, down to 32*32*512, every time it has been applied convolution layer, it will be also applied a batchnorm transformation to increase the computation. After convolution, dilation convolution will be applied to the output of the convolution, which will offer a wider field of view, so it can get more info without increasing the size of the kernel. Finally using ConTransposed to map our data to 64*64*313, then use upsample by scale of 4 to get our final result which is 256*256*2 tensors contain predicted A, B values.

The CUnet is a model by Na Wang, Guo-Dong Chen, and Ying Tian. The CUnet is based on the Unet model, which utilizes skip connections over a series of convolutions/downsampling and deconvolutions/upsampling. The skip connections (which copy the output of a downsampling step and concatenate it to an upsampling input) allow low-level features to be accessible during the deconvolution/upsampling steps. The basic Unet uses 4 downsamples and 4 corresponding upsamples, with convolutions in between.

The CUnet replaces the lowest downsample/upsample steps with extended convolution or dilated convolution. This allows the model to use features from other parts of the image without losing spatial resolution as a traditional max pool + convolution does. Using a dilated 3x3 convolution instead of a 5x5 convolution also saves processing time and may prevent overfitting.

The input should be a 256x256 image (or any multiple of 8) with channel, the lightness, and the output will be a 256*256*2 tensors contain predicted A, B values, the same as the eccv16 network.

c) Post Processing

Since the model could give out the predictions of A, B values in only 256x256 resolution, it needs to resize the images back to the original size through bilinear interpolation. (This could be done with another neural network, but that is not our current approach). Then, this two channel tensor will be combined with the pre-crop lightness tensors, then convert the CIELAB color scale back to RGB scale through built-in functions for display or saving.

Challenges & Solutions

Challenge: Computing power, training time. Even though we didn't use very deep networks, ImageNet is a large dataset and epochs may take a long to go through. We are also starting with a completely untrained network with no transfer learning, which also increases training time.

Solution: We have access to a fairly advanced graphics card which should help cut down on computing time. By adding functionality for saving and loading the model, we can also train over many different sessions. (Keeping access to prior models can also help prevent overtraining/overfitting.)

Challenge: Implementing some advanced mechanisms for better colorization for example color hinting.

Solution: Read the papers on those mechanisms and try to implement those techniques.

Challenge: Matching sizes of input and output.

Solutions: Currently, we are using a simple bilinear interpolation. However, this results in low-resolution color channels (the A and B in LAB). Another solution would be to implement another super-resolution network, or use a pipeline of a low-resolution recolorization first and further higher-resolution ones with color hints based on the first result.

Challenge: Images becoming desaturated using Mean Squared Error.

Solution: The eccv16 paper groups colors into “buckets” and treats it as a categorization problem, so “conservative” gray colors contribute to loss just as much as saturated, incorrect colors. Furthermore, it may be possible to use a differentiable LAB - to HSL converter inside the loss function, and have the difference in saturation contribute separately to the loss as well.

Challenge: ImageNet contains some images which aren’t helpful for colourization, such as completely black-and-white photos.

Solution: Include a filter in the automated pipeline to remove black and white photos from training.

Updated Timeline

- Get full pipeline working (with model saving, post processing, etc.) and have training-ready code by March 23rd
- Have a recolourizer that produces better-than-random output by March 28th
- Finish a round of model/pipeline changes by March 29th
- Continue updating the model as needed.
- Final iteration of model by March 31st, and train until complete.
- Presentation ready by April 3rd.