Colorizing grayscale images

Nolan Bock, Sai Nikhil Thirandas, Naveen Verma
bock.n@northeastern.edu, thirandas.s@northeastern.edu, verma.na@northeastern.edu

Objectives and significance

The goal of the project will be to implement a series of Neural Networks that probabilistically colorize grayscale images and compare their performance to each other. The first steps will be to implement Convolutional Neural Networks, some of which have already been implemented in academia and some that we will implement for the first time, to establish a baseline performance and accuracy. We'll first establish a baseline with the Convolutional Neural Network model outlined in Zhang et al [1]. From there, we'll change the layers in that Convolutional Neural Network to analyze how each component contributes to accurate colorization. We highlight differences between Zhang et al and our approach in the background and proposed approach sections.

We will compare our model with the baseline model and certain pretrained models that are considered state of the art in image colorization like those provided in Zhang et al [1]. A stretch goal our team has is to implement a Fusion-Encoder-Decoder model [2] and analyze its performance vs. all other Convolutional Neural Networks that we implement.

Our motivation for the project is to explore Convolutional Neural Networks and their ability to place colors in certain regions. By the end of the project, we hope to both understand the problem space (colorization) and the general architecture of the CNN model. We will also analyze and plot how the model's accuracy changes with respect to the component/layer of a CNN model and also with respect to certain hyperparameters and various loss functions.

Background

Our project will deal with Convolutional Neural Networks that map grayscale input to a distribution over quantized color value outputs [1]. The function of our trained CNN will be to output 2 channels of colors 'a' and 'b', that can be concatenated with 'L' channels to get the colored image, but not necessarily the correct colorization. These colors are the ones nearest to the actual image when the model is well trained. For example, apples are likely red or green, but distinguishing between one versus the other in a specific image will be a probability question - which color is more probable for the apple given the other features in the image. A very improbable color for an apple would be black. Given that, our trained CNN will have inherent inaccuracy since our evaluation strategy will be to compare true pixel color to estimated pixel color. Zhang et al proposed an evaluation strategy by using Amazon Mechanical Turk to pay individuals to pick the "real" image out of one colorized and one with true color, but that is out of the scope of this project.

The stretch goal for our project will be to implement a Fusion-Encoder-Decoder model which uses a deep CNN fused with high level features extracted from the pretrained Inception-ResNet-v2 model [2]. The model for our Fusion-Encoder-Decoder will be discussed further in our proposed approach section. Broadly, it will supplement the existing CNN to do the same colorization that we expect from the CNN, but with additional feature extraction and layers.

Since there exists a paper which we will initially recreate, we will now highlight the differences between our implementation and that from Zhang et al. Specifically, our first model is inspired by the Zhang et al model but with different architecture. We will use the total error

over the entire dataset as a parameter to judge the performance of the model - Zhang et al used Amazon Mechanical Turk evaluation to determine efficacy. We will also change the layers in the Convolutional Neural Network to analyze how each component contributes to accurate colorization and pick the one with least error to compare with the state of the art. One particular interest to our team is the effect of using a long array of thick layers in cnn and also dilated convolution (atrous convolution) on the accuracy that is shown from Zhang et al [1] - we'll analyze how the layers and their degree of transitions while downsampling (striding) or upsampling affects the performance. We will also experiment with different loss functions, which will be discussed further in the next section.

Proposed approach

Data

For our project, we will use the CIFAR-10 dataset [3], Places dataset [4] and ImageNet dataset [5]. All these datasets are also embedded in the torchvision package of PyTorch Library [6]. The CIFAR-10 dataset contains 60,000 32x32 size color images in 10 classes, with 6,000 images per class. The following is an example of one class (airplane) in the CIFAR-10 dataset (as well as 10 random images from that class):

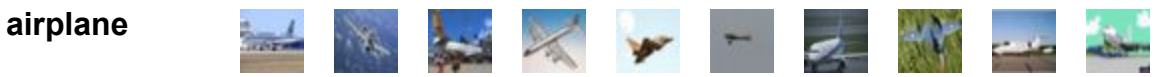

Fig. 1. Class and image examples from the CIFAR-10 dataset [3]. Each class has 10 examples separated into 10 classes.

As you can see, the classes come prelabeled, which our training will ignore. The classes are mutually exclusive, which will not be relevant to our training, but may provide relevant insights about what is easiest to recolor once our model is trained. The Places dataset contains more than 10 million images comprising 400+ unique scene categories. The dataset features 5,000 to 30,000 training images per class, consistent with the real-world frequencies of occurrence. Using Convolutional Neural Networks (CNN), the Places dataset allows learning of deep scene features for various scene recognition tasks. For our purposes, the Places dataset will add a significant amount of training data for our recolorization.

Additionally, ImageNet is an image database organized according to the WordNet hierarchy (currently only the nouns), in which each node of the hierarchy is depicted by hundreds and thousands of images. The WordNet dataset will provide us with the option to analyze efficacy on images that would typically not be in the Places dataset. For example, Places is likely to have a scene that includes a fish whereas WordNet does have that. The torchvision package of PyTorch consists of popular datasets, model architectures, and common image transformations for computer vision. Finally, the ImageNet dataset was used in the original implementation of Zhang et al and we will obtain that from the torchvision package of Pytorch.

Proposed method and implementation

The first several iterations of our implementation will use and build upon the CNN outlined in Zhang et al. The specific architecture of Zhang et al's implementation is shown below and further explored throughout this section.

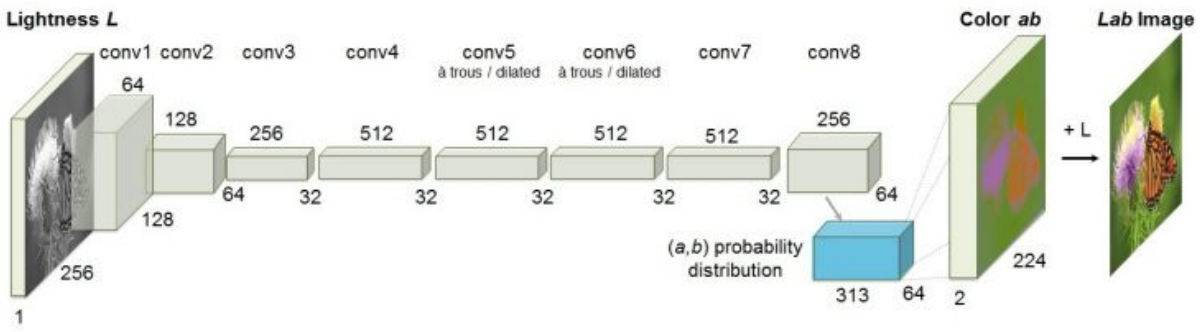

Fig. 2. The network architecture used by Zhang et al. Each convolutional layer refers to a block of 2 or 3 repeated convolutions and ReLU layers followed by a BatchNorm layer. There are no pool layers [7].

Broadly, the CNN architecture above will map grayscale input to color as shown in the image above. Zhang et al uses two channels: the L channel, which has light information but no color information, and the ab channel, which has color information.The a channel shows you the values on the red to green axis, while the b channel shows the values on the blue to yellow axis. Perhaps a reasonable assumption would be to define a loss function that measures loss by the square difference between the true and estimated *ab* pixel values for each pixel in an image, but this loss function will find no difference between the following two images: one which misclassified color in the foreground and one that misclassified color in the background on an unimportant object.

Instead, Zhang et al formulated the problem as a problem of multinomial classification. Given that, the *ab* space was translated into a discrete space. Figure 3 shows the translation from continuous colors in *ab* space (a regression problem) and discrete colors in *ab* space (multinomial classification).

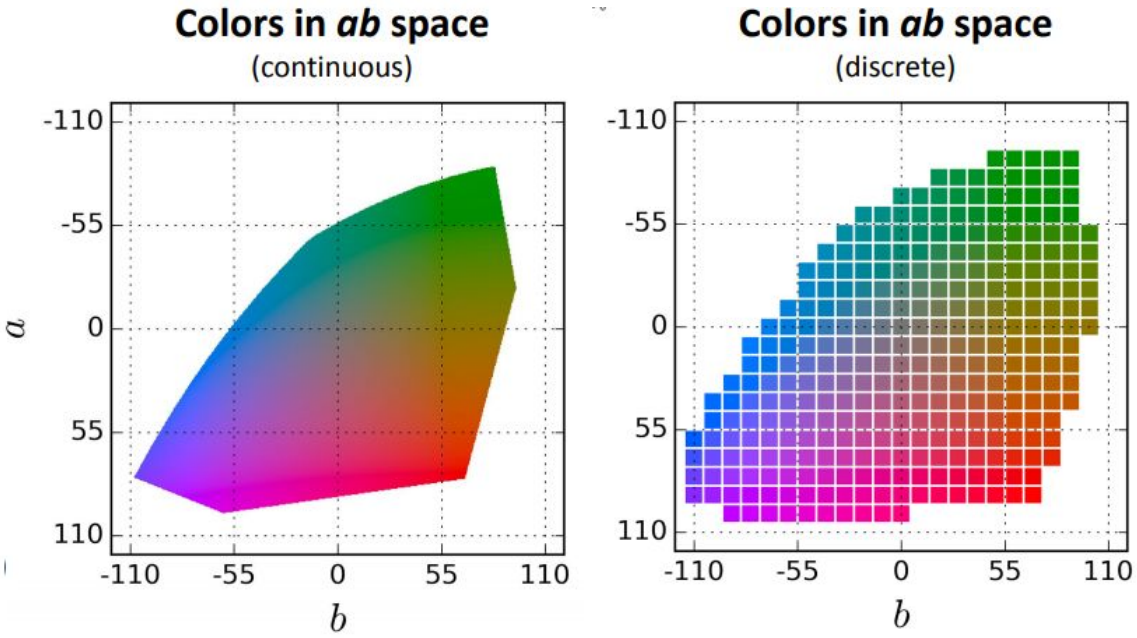

Fig. 3. Quantized *ab* color spaces. The discrete space was used for the CNN trained in Zhang et al [7].

The colors in *ab* space were broken into bins of size 10, which means there are a total of 313 *ab* pairs. An example classification is shown in Figure 4, which highlights that the foreground object

(the bird) is blue, red, or purple and the background object is green, yellow, or brown. The heavier the coloring, the more likely that color is the true color according to our model.

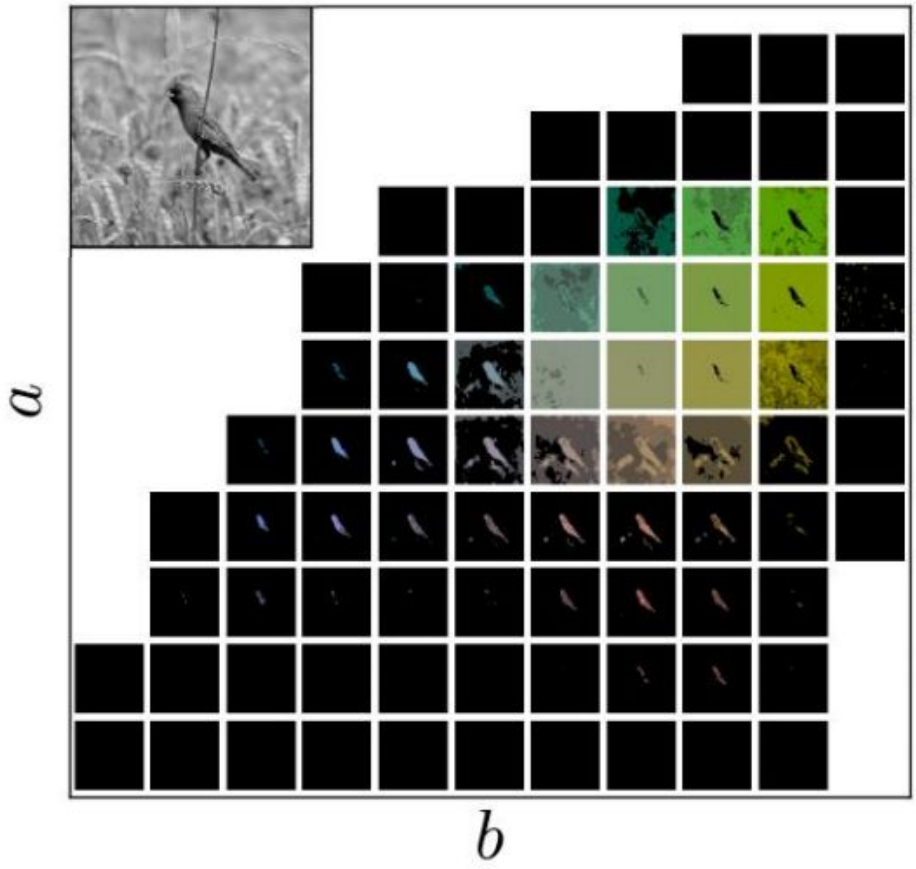

Fig. 4. Bird classification with foreground and background differential classification [7].

Here we can see that the bird image is classified by the discrete *ab* color space. Zhang et al also considered that the vast majority of pixels in their training set were focused around (0,0) of the *ab* space (dark red), which created images that were desaturated. Thus, a bias towards desaturated images was created because the CNN predicted unnaturally bland images. To account for this, Zhang et al added a class rebalancing term to resample the rarest colors in a given dataset. Thus, their loss function becomes a cross entropy loss function with class rebalancing, which will have the final form as shown in Figure 5.

$$\mathrm{L}(\widehat{\mathbf{Z}}, \mathbf{Z}) = -\frac{1}{HW} \sum_{h,w} v(\mathbf{Z}_{h,w}) \sum_{q} \mathbf{Z}_{h,w,q} \log(\widehat{\mathbf{Z}}_{h,w,q})$$

Fig. 5. Class rebalancing in the loss function to encourage learning rare colors. The variables h,w are the image dimensions and q is the number of quantized *ab* values (313 in our case) [7].

The loss function uses several terms that we need to define further. $v(\dot{\bullet})$ is a weighting term that can be used to rebalance the loss based on color-class rarity. The $\hat{Z}$ term is the predicted

probability distribution for some image and Z is the true probability distribution. The H and W terms are the dimensions of the image.

The final layer of Zhang et al's CNN is to convert from the 313 by 64 probability distribution to a 224 by 2 layer that represents the *ab* channel of the image. As a result of this loss function favoring rare colors, the images produced will be more colorful. Our implementation will focus on implementing the simple sse loss function as well as both cross entropy loss function and a weighted cross entropy loss as described above. Our fallback will be a simple cross entropy loss function to train the model described by Zhang et al [1].

The final goal of our project will be to implement a Fusion Encoder Decoder model, which has an architecture that is summarized below both in the figure and paragraph below.

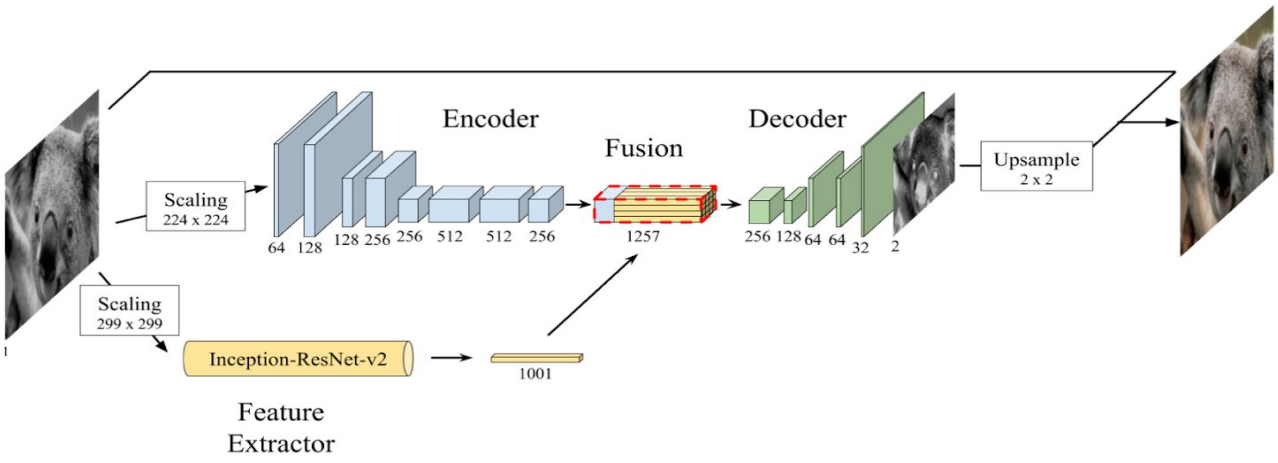

Fig. 6. Fusion Encoder Decoder model.

Instead of training a feature extraction branch from scratch, we make use of an Inception-ResNet- v2 network and retrieve an embedding of the gray-scale image from the last layer of the decoder. As summarized by Baldassarree et al, "the network is logically divided into four main components. The encoding and the feature extraction components obtain mid and high-level features, respectively, which are then merged in the fusion layer. Finally, the decoder uses these features to estimate the output" [2]. The encoder uses 8 convolutional layers to output a 512 depth feature initial representation. Feature extraction scales the image and stacks it upon itself separated by the three channels (l, a, and b from the l and *ab* channels). Fusion performs several additional convolutional layers that eventually perform 256 convolutional kernels of size 1 x 1 that are used to generate a H/8 x W/8 x 256 volume. Finally, the decoder takes the H/8 x W/8 x 256 dimension input, upsamples using the nearest neighbor approach, and outputs height and width at twice the height of the input [2]. A full architectural overview and the associated code is provided in Baldassarre et al, which we will reference throughout our implementation.

Evaluation strategy

We will first evaluate success by measuring the sum of the entire test set's misclassified pixels magnitude: the amount one predicted pixel varies from the truth. We know that we will predict more colorful images on purpose, so we will expect a relatively large error in our classification. For example, consider Figure 7, which shows a predicted image and the ground truth of that image. We may consider that image very close to the truth if a human was

classifying success and failure since the main issue is the background and tie color (not very important) but our evaluator will classify this image as fairly incorrect since the entire background will be incorrect by a large margin. One potential fix to our solution will be to evaluate using absolute pixel error and randomly sample to pictures to find a rough percentage of how many colorizations could fool a human - this will only be used for analysis since it is not a programmatic solution. Another possible solution that we will explore will be to use an image segmentation api to find the sub-objects in an image and classify success as a function of correct or incorrect pixel colors. In this process, we evaluate an object's "correctness" by checking if the majority of pixels are close to correct.

Fig. 7. Inherent ambiguity in the model evaluation.

One area of concern that we have is the amount of data we will need to train and the relative lack of compute power we have available locally. Specifically, the Places dataset is extremely large. If training the dataset takes too long then, we will randomly select a subset of that dataset or any dataset and train on that subset. Additionally, we plan to use the Discovery cluster and save the weights periodically so we can have long running jobs (jobs that run longer than a day on saved weights).

Expected outcomes

We expect that the baseline CNN from Zhang et al will perform the best of the CNN's that we train. However, we hope that tweaking the convolutional layers and the degradation in performance will highlight the most important and impact layers in their CNN. Similarly, we hope that our implementation of the Fusion-Encoder-Decoder will match the performance of the Zhang et al CNN, but we have no benchmark to set our initial goal, so we hope to measure the efficacy of a Fusion-Encoder-Decoder on images of this type. Our initial goal will be to recreate the Zhang et al findings without the class rebalancing. Next, we will implement the same with class rebalancing and experiment with the convolutional layers (e.g. dilation, BatchNorm, degree of transition of layers, and size of image) to find the model that performs the best. Our fallback options will be to implement the Zhang et al paper without class rebalancing and additional convolution layers.

Individual tasks

Sai Nikhil will work on preprocessing the image data, which will include removing labels and randomizing selection, and tuning the objective functions in our CNNs. Nolan will work on downloading the correct data, there are some locations with ab pre-processed that he will explore, and he will be calculating lab to rgb and rgb to lab channels in Python - this will be a somewhat complicated custom function that we will need to write, previously it was implemented in a package that is now end of life. Additionally, Nolan will explore the use of image segmentation apis to score our colorized images. Naveen will define the initial layers to the convolutional neural network and the tweaks to the neural net that we will apply to the CNN. Additionally, he will define the initial requirements and be in charge of the initial implementation for our Fusion-Encoder-Decoder model.

As discussed above, our work differs from Zhang et al [1] in that we will change the hyperparameters and loss function, but more specifically we will: change the objective function, change the convolutional layers, change the datasets used, and explore the possibility of using a Fusion-Encoder-Decoder model versus a Convolutional Neural Network. We will then compare and contrast the approaches and highlight any differences in implementation.

References

[1] Richard Zhang, Phillip Isola, Alexei A. Efros. Colorful Image Colorization (https://arxiv.org/pdf/1603.08511.pdf).

[2] Federico Baldassarre, Diego Gonzalez Morin, Lucas Rodes-Guirao. Deep Koalarization: Image Colorization using CNNs and Inception-Resnet-v2. (https://arxiv.org/pdf/1712.03400.pdf).

[3] Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images. CIFAR-10 Dataset (https://www.cs.toronto.edu/~kriz/cifar.html).

[4] Bolei Zhou, Agata Lapedriza, Aditya Khosla, Antonio Torralba, Aude Oliva. Places Dataset (http://places2.csail.mit.edu/index.html).

[5] Stanford Vision Lab, Stanford University, Princeton University. ImageNet Dataset (http://image-net.org/index).

[6] PyTorch torchvision package. (https://pytorch.org/vision/0.8/datasets.html#imagenet).

[7] Lecture slides from Richard Zhang presenting Colorful Image Colorization. (http://videolectures.net/site/normal_dl/tag=1078750/eccv2016_zhang_image_colorization_01.pdf)