Coloring Black and white images using Deep Neural Networks

Sylvia Boddu Jayaraman Sridharan Rahul Gomathi Sankarakrishnan

Luddy School Of Informatics, Computing and Engineering
Indiana University Bloomington
sboddu@iu.edu, jsridhar@iu.edu, rgomathi@iu.edu

***Abstract*— There are a lot of black and white images from various sources, from history, old personal photographs, etc. Coloring these images is a pain staking task that must be done by an expert, and it takes so much time and energy. Another problem is the large degrees of freedom that are available while assigning the color to the image. We try to automate this task by training various deep learning models on different datasets to read black and white images and generate the colored version of the input. We have compared the results of the different neural networks we trained.**

Keywords—CNN, GAN, coloring, black and white.

I. Introduction

There is a shift in the trend to color the black and white images to color, from adobe photoshop and other manual tools to the Generative adversarial networks (GAN) and Convolution neural networks (CNN). In our project we have implemented 3 different models simple CNN, Autoencoder CNN and GAN. In this paper we explain detailed implementations of each of the model. Our goal is to take black and white images as input and output a colored image of it.

II. Simple cnn

A. Color Spaces

Color spaces are usually three-dimensional spaces that represent all possible colors that can be represented. The most common color space that is used is RGB color space. In RGB space each image is represented by 3 channels: Red, Green, and Blue. Each channel contains the corresponding color information. In this color space, we can represent all the combinations of the colors: red, green, and blue.

CIELAB or LAB is one other color space. In this color space, the colors are represented in 3 channels: L*, a*, b*. The L* channel represents the Lightness, a* channel represents the red and green colors, b* channel represents the yellow and blue channels. Unlike the RGB color space, the values of L* channel ranges from 0 to 100. The values of a* and b* channels range from -128 to +128.The major difference between the two channels is that the color information is represented in only 2 channels in LAB space rather than 3 channels in the RGB space. So, we are choosing to represent images in LAB space, as it would be little easier for the model to learn 2 channels representing color rather than 3 channels.

Another reason to go for LAB space is that model can focus only on predicting the colors rather than the structure or details of the image. Since all the structure will be preserved in the L space, it is enough to just predict the a* and b* channels.

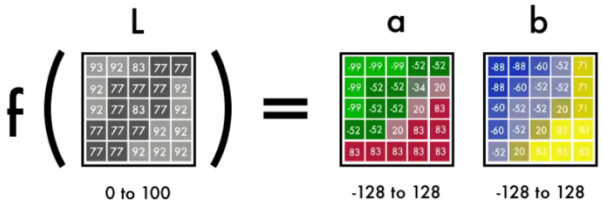

Figure1: LAB Layer

B. Methodology

This is a very basic model to do the coloring, the architecture is very simple as shown in the below diagram. The black and white images are in gray scales. They range from 0-225. To choose the accurate color we convert the input layer of gray scale to two layers of colored layers we use convolution layers.

Each filter can add or remove some information, the network can combine these filters or select filters to form a new image. CNNs adjust the filters to obtain better results. lot of filters are stacked to form 2 layers ‘a’ and ‘b’. Predicted values and the real values that fall under the same interval are mapped which are ranging from -1 to 1. TanH function is used for mapping. From the Gray scale input layers the outputs are mapped to these Lab color layers and they are normalized, after that the errors are

calculated the network updated, the same process is repeated till the error value reduces to the min.

C. Architecture

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, None, None, 8)	80
conv2d_1 (Conv2D)	(None, None, None, 8)	584
conv2d_2 (Conv2D)	(None, None, None, 16)	1168
conv2d_3 (Conv2D)	(None, None, None, 16)	2320
conv2d_4 (Conv2D)	(None, None, None, 32)	4640
conv2d_5 (Conv2D)	(None, None, None, 32)	9248
up_sampling2d (UpSampling2D)	(None, None, None, 32)	0
conv2d_6 (Conv2D)	(None, None, None, 32)	9248
up_sampling2d_1 (UpSampling2D)	(None, None, None, 32)	0
conv2d_7 (Conv2D)	(None, None, None, 16)	4624
up_sampling2d_2 (UpSampling2D)	(None, None, None, 16)	0
conv2d_8 (Conv2D)	(None, None, None, 2)	290

Total params: 32,202
Trainable params: 32,202
Non-trainable params: 0

Figure 2: Simple CNN architecture.

The above architecture is yielding good results for small datasets, but for larger datasets its not performing well. The results will be discussed in the below section.

III. CNN WITH AUTOENCODER

A. Introduction

The second method of image colorization is mainly centered on an autoencoder system. While the model primarily constitutes an encoder and a decoder, it also comprises of an immensely powerful image classification network 'InceptionResNetv2' used for transfer learning. The Inception model has been trained on over a million images from the ImageNet database and helps in classifying a particular image into one of 1000 categories. The combined model, as such comprises of the Inception network, the encoder model, their merged model, and a decoder model.

B. Architecture

The encoder model starts with an input layer of (,256,256,1) and further contains eight two-dimensional convolution layers. Each layer has a 3-by-3 kernel, with strides to reduce dimensions in alternative layers. A 'category' model takes in an input of size (,1000), after which it has a repeat vector and a reshape layer with arguments (1024) and (32,32,1000) respectively.

A merge model is then constructed, which receives its input as combined outputs from the two previous models. It is followed by a convolution layer with a (1x1) kernel. The next model is that of the decoder, which comprises of six convolution layers with (3x3) kernels along with three up-sampling layers. Every padding argument used in these models is 'same', while every convolution layer except the final layer uses 'relu' as its activation function. The final layer utilizes 'tanh' as its activation function. The final model combines the inputs of the encoder and categorizes models with the outputs of the decoder.

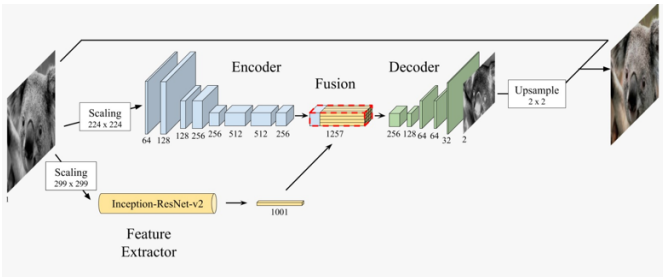

Figure 3: Encoder and decoders architecture.

C. Methodology

The encoder model takes in this input and extracts a few prominent features through its convolution layers. Simultaneously, the inception model is used to classify the underlying image as one of 1000 categories after converting the image into the required shape of (299, 299, 3) and then preprocessing this input to the scale of [-1,1]. The predicted feature vector is then sent into a 'categorize' model which repeats this input (32x32) times and reshapes it into a format having the same dimensions of features as the encoder model.

The resultant outputs from the encoder and categorize models are (32,32,256) and (32,32,1000) respectively. These are then combined in the merge model with the help of a concatenate layer, after which a convolution layer is used to convert it back into the shape of the encoder output. The output from this model is consequently sent into a decoder model, which up-samples the received input into producing the predicted 'a' and 'b' values, which contain information on the colors of the image. The combined model takes as its input the shape of the encoder input (256,256,1)

An 'Image Data Generator' is used to provide numerous iterations of the same image. Each generated batch is then used to obtain the categorical features predicted by the inception model. After converting each batch into the 'lab' format, the obtained feature vectors are merged with 'l' value of the image to be provided as its input. The output is the combination of 'a' and 'b' values of the image. The respective inputs and outputs are then yielded to train the combined model.

To test the model, test images are converted from 'RGB' to 'lab', after which their 'l' values and predicted feature vectors are passed to the model to predict their 'a' and 'b' values. The collection of 'l', 'a' and 'b' values are converted into 'RGB' to obtain the predicted color image.

IV. GAN

Generative Adversarial Neural Network is a type of Image generation model. In GANs, one model, generator tries to generate the image while another model, discriminator tries to find out if the image is generated by the generator or not. The goal of the generator here is to fool the discriminator. To achieve that goal, the generator has to generate images that are closer to the ground truth. Every time, the discriminator correctly identifies that the generator has created the image, the generator updates its weights to try to generate a better image to fool the discriminator.

A. Generator:

Our generator architecture is based on the pix2pix model and "U-net" architecture. It is an encoder- decoder structure with skip connections connecting the similar shaped layers in the encoder to the respective similar shaped ones in the decoder. Each skip connection concatenates all the channels from encoder side with all the channels in the decoder side. The encoder side consists of series of layers that progressively down sample the input and the decoder is created in such a way that it progressively up samples the input. This generator model has a symmetric architecture with 7 encoding blocks and 7 decoding blocks. Each encoding block consists of a Convolution layer with 4x4 filters and stride 2, BatchNormalization layer followed by a LeakyReLU layer. The architecture of the generator is shown in *Figure 4*.

Similarly, each decoder block consists of a Conv2DTranspose layer with 4x4 filters and stride 2, BatchNormalization layer followed by a LeakyReLU layer. This is followed by a final Conv2DTranspose layer which upscales the image to 256x256 with 2 channels. One channel representing a* and another representing b* in the LAB color space. The architecture of the decoder is shown in *Figure 5*.

Figure 4: Generator architecture.

B. Discriminator:

The Discriminator is also a Convolutional Neural Network. The architecture of the discriminator is like that of the encoder part of the network. It consists of series of blocks each containing a Conv2D layer, BatchNormalization, LeakyReLU activation followed by a Dropout layer.

Each block downscales the image and the image size is reduced from 256 to 8 gradually. Then it is flattened and is followed by a final Dense layer with sigmoid activation function. The purpose of the Discriminator is to read in the ground truth and the generated images and try to predict which is real and which is fake. The Generator's goal is to fool the discriminator. By doing so, the generator tries to produce images that are more like the ground truth.

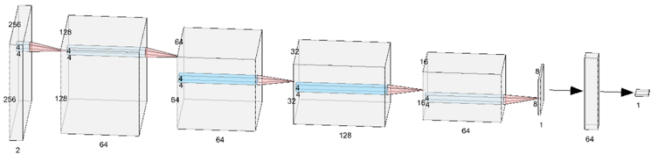

Figure 5: Discriminator architecture.

C. TRAINING METHODOLOGY:

We used He Normal Weight Initializer to initialize all the weights in our network.

Loss function: It has two components; the first component is the cross-entropy loss to train the generator and discriminator, the second part of the loss is to ensure that the generator generates images that are close to the ground truth. The hyperparameter λ was set to 100. This hyperparameter is used to control how much of the reconstruction loss to be added to the total loss.

$$\min_{\theta_G} J^{(G)}(\theta_D, \theta_G) = \min_{\theta_G} -\mathbb{E}_z \left[\log(D(G(\mathbf{0}_z|x)))\right] + \lambda \|G(\mathbf{0}_z|x) - y\|_1$$

Label Smoothing: To prevent the discriminator from predicting very high confident outputs, we can replace the hard target 0 and 1 with soft targets 0.1 and 0.9 respectively. Label smoothing usually works a great regularizer while training CNNs.

Batch Normalization: The goal of the GAN training is to ensure that the generator learns the distribution of the source images. BatchNormalization keeps the distribution similar between each layer of the network, helping the model learn faster.

No Pooling layers: To downsize images, instead of using the pooling layers like how it is usually done, strided convolutions were used. This forces the network to learn to downsize on its own and using this method has shown improvements in its performance.

Reduced Momentum in the Optimizer: We have used Adam optimizer to train both the discriminator and the generator. Recent research shows that having a β_1 value of 0.9 results in more instability and oscillations while training. So, we reduced the momentum to a value of 0.5.

LeakyReLU: Recent research also suggested the use of LeakyReLU over the standard ReLU as it helps the model to perform a little better.

V. DATASETS

The Unsplash dataset was used for training and testing the autoencoder model. It comprises of over 20,000 images gathered from across the world. The dataset used was the lite version of the original dataset which contains over 250,000 images. The images themselves are in the shape of 256x256.

After seeing the results of the autoencoder model, which was trained on the very diverse Unsplash dataset, we wanted to start our experiments with GANs using a simpler, less diverse dataset.

LHQ (Landscapes High-Quality) dataset consists of high-resolution images that are restricted to the nature domain. The dataset comprises of 96,000 HD landscape images that were resized to 256x256.

VI. RESULTS

Below are the results generated by Simple CNN, first column is the grey scale of the image second column is the groudtruth, third column is the obtained result from simple CNN model. We can see that the below results don't show good results with all the test data. The coloring is partial towards the trained learnings.

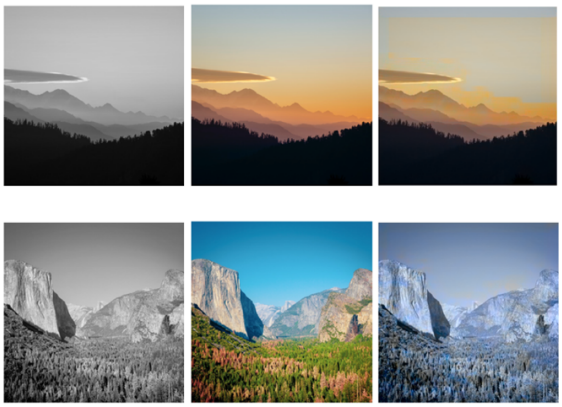

Figure 6: Results from simple CNN

Next are the results generated by CNN with Autoencoder, first column is the grey scale of the image second column is the groudtruth, third column is the obtained result from CNN with Autoencoder model.

Figure 7: Results from CNN with Autoencoder.

Below are the results generated by GAN, first column is the grey scale of the image second column is the groudtruth, third column is the obtained result from Gan model.

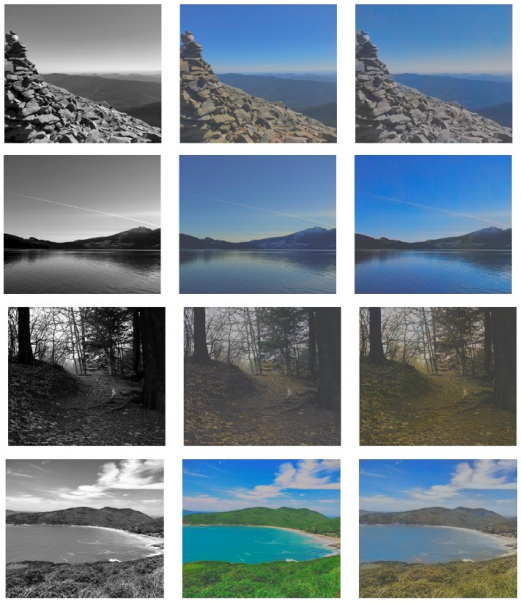

Figure 8: Good results from GAN

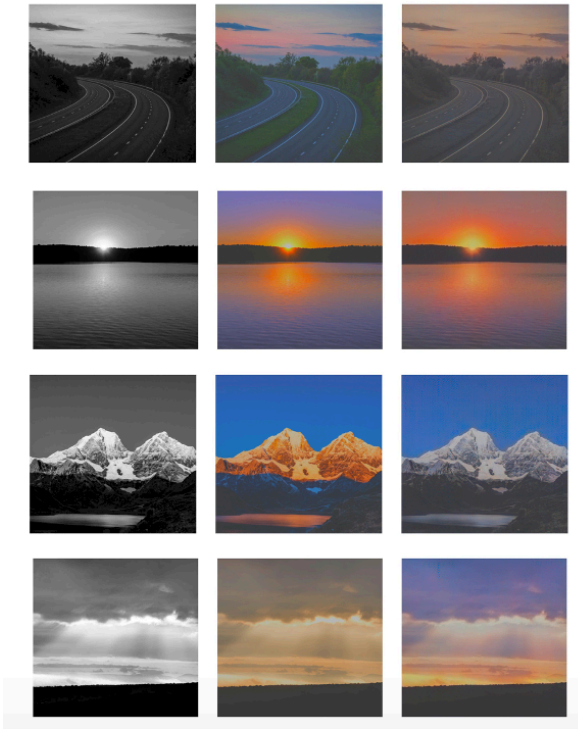

Figure 9: Good results from GAN.

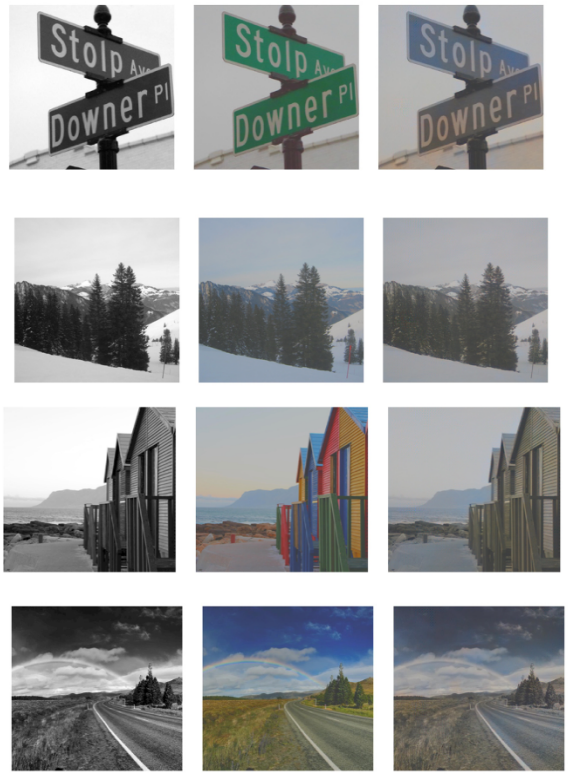

Figure 10: Bad Results from GAN model

VII. Conclusion

After training simple CNN to perform image colorization, we were able to see that it was performing good with a small dataset. When we tried a larger dataset, it couldn't color the images.

With the encoder-decoder architecture, the model was unable to identify and color the different objects in the image. The pixels within the same object had different colors. The model was not able to identify the boundaries between different objects and because of that the images had patches of colors

The results from GANs were much better than the other methods. GAN was able to identify and properly color each image. We can see that the model doesn't like to explore a lot of colors, it sticks to the basic ones. If it figures out that wood is of brown color, all instances of wood are generated with the same color. For the same reason, this model cannot generate images with vibrant, bright colors.

The future work could be done on finding ways to let the model generate with a wider and brighter color palette. Choosing a bigger dataset with a diverse sample would be another thing to do. There are many bias issues with image colorization. Depending on the data the model was trained on, it could color a person of darker skin tone with lighter skin tone. Tackling these kinds of biases while generating images would be an interesting problem to solve.

Our work is not just limited to image colorization. The same model and architecture can be applied to various image to image translation tasks such as converting edges to photos, sketches to photos, day light image to night, and image inpainting, where are there missing pixels in the image.

VIII. Reference

[1] Szegedy, Christian, et al. "Inception-v4, inception-resnet and the impact of residual connections on learning." Thirty-first AAAI conference on artificial intelligence. 2017.

[2] Baldassarre, Federico, Diego González Morín, and Lucas Rodés-Guirao. "Deep koalarization: Image colorization using cnns and inception-resnet-v2." arXiv preprint arXiv:1712.03400 (2017).

[3] Isola, Phillip, et al. "Image-to-image translation with conditional adversarial networks." Proceedings of the IEEE conference on computer vision and pattern recognition. 2017.

[4] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (2016)

[5] Springenberg, Jost Tobias, et al. "Striving for simplicity: The all convolutional net." arXiv preprint arXiv:1412.6806 (2014).

[6] Nazeri, Kamyar, Eric Ng, and Mehran Ebrahimi. "Image colorization using generative adversarial networks." International conference on articulated motion and deformable objects. Springer, Cham, 2018.

[7] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.

[8] An, Jiancheng, Koffi Gagnon Kpeyiton, and Qingnan Shi. "Grayscale images colorization with convolutional neural networks." Soft Computing 24.7 (2020)

[9] Hwang, Jeff, and You Zhou. "Image colorization with deep convolutional neural networks." Stanford University, Tech. Rep.. 2016.