

Kathmandu University
Department of Computer Science and Engineering
Dhulikhel, Kavre



A Mini Project Report
on
“AUTOMATIC COLORIZATION OF GRAYSCALE IMAGE”
COMP 484

(For partial fulfillment of 4th Year/ 1st Semester in Computer Engineering)

Submitted by:

Nischal Bhandari (Roll No. 10)

Babin Khatri (Roll No. 26)

Amar Kumar Mandal (Roll No. 58)

Neha Verma (Roll No. 59)

Aarush Timalisina (Roll No. 61)

Submitted to:

Dr. Bal Krishna Bal

Department of Computer Science and Engineering

Submission Date:

12th September, 2022

Abstract

Automatic grayscale colorization is a process in which black and white images are converted into colorful images. Grayscale represents those images whose each pixel value is a sample of a single channel. Grayscale images are not informational as well as they do not have great aesthetic presence. Color images represent more information in terms of composition of images as well as they are aesthetically more pleasing. Therefore this project is targeted to develop a model that can learn to regenerate color images from b/w images. Our neural network finds characteristics that link grayscale images with their colored versions. Thus, this architecture is based on autoencoding that consists of two networks: encoder and decoder.

In order to colorize the image the encoder through a series of CNN and downsampling, learns a reduced dimensional representation of the input data while decoder through the use of CNN and upsampling, attempts to regenerate the data from these representations.

Keywords: *Autoencoder, Downsampling, Convolutional Neural Network, Grayscale, Upsampling*

Table Of Contents

Abstract	i
Acronyms / Abbreviations	iii
Chapter 1: Introduction	1
1.1 Background	1
1.2 Objectives	2
1.3 Motivation and Significance	2
Chapter 2: Related Works	3
Chapter 3 : Design and Implementation	4
3.1 System Requirement Specification	4
3.1.1 Software Specification	4
3.1.2 Hardware Specification	4
3.2 Flowchart	5
3.3.1 Dataset	6
3.3.2 Data Preprocessing	6
3.3.3 Feature Extraction	6
3.3.4 Defining the Model	7
3.3.5 Testing and Evaluating the Model	10
3.3.6 Improving the Accuracy	10
Chapter 4 : Discussion on the Achievement	12
Chapter 5 : Conclusion	14
5.1 Limitations	14
5.2 Future Enhancements	14
References	15
Appendices	16

Acronyms / Abbreviations

CNN: Convolutional Neural Network

RGB: Red, Green and Blue

PCA: Principal Component Analysis

KNN: K-Nearest Neighbor Algorithm

MAP: Maximum a Posteriori

ML: Machine Learning

UHD: Ultra High Definition

BSDS: Berkeley Segmentation Dataset

Chapter 1: Introduction

1.1 Background

Image colorization is the process of adding colors to a grayscale picture using as a source a colored image with similar content to make it more aesthetically appealing and perceptually meaningful. It is used to increase the visual appeal of images such as old black and white photos, classic movies, and scientific visualizations. Colorization techniques are widely used in astronomy, MRI scans, and black-and-white image restoration. Many institutions use image colorization services for assigning colors to grayscale historic images as well as for colorization purposes in the documentation image. These are recognized as sophisticated tasks that often require prior knowledge of image content and manual adjustments to achieve artifact-free quality. Automatic image colorization often involves the use of a class of CNNs called autoencoders. These neural networks are able to distill the salient features of an image, and then regenerate the image based on these learned features.

Colorizing grayscale images involves assigning three-dimensional (RGB) values to each pixel value in the target grayscale image, whose elements are characterized by their luminance intensity. The colorization problem is challenging since, given a grayscale image, there is no unique correct colorization in the absence of any information from the user. Even though many implementations require the user to specify initial colors in certain regions of the picture, our project focuses on automatic image colorization without any additional input from the user.

In this project, an attempt has been made to come up with methods to colorize images without human assistance. We describe our first attempts at colorizing a gray image using a similar colored picture. We begin by implementing a simplistic method that transfers colors between pixels based on the similarity in their luminosity. We improve our approach by considering a larger feature space that takes into account the provenance of each pixel and strives for spatial consistency. Then we apply PCA and KNN in the feature space, followed

by a confidence voting that labels our gray pixels and determines from which colored pixels we have to transfer the chromatic channels.

1.2 Objectives

The main objectives are:

1. To colorize a black and white image making it more visually appealing.
2. To save time and effort while coloring grayscale images.
3. To learn about Convolutional Neural Network

1.3 Motivation and Significance

There can be many reasons for users having to colorize a black and white or gray scale image. It may be old photos that may need a realistic view or a gimmick that the user wants to see coloured picture of. Almost all of the old cameras would capture black and white images which would be needed to colorize. Unfortunately, image colorization using different softwares requires a large amount of human effort, time and skill. People generally use Photoshop to add color to b/w photos. It is such an extensive process and every detail is considered to colorize an image. It may take upto a month to colorize a single image. So, there is a need for automatic colorization of grayscale images.

With automatic colorization, it would help to hallucinate what an input gray scale image would look like when colorized. Not only does it make an image look and feel better than grayscale, we could also grab the realistic view of the photo. Automatic Colorization can save our time and effort by colorizing a grayscale image automatically within a few seconds instead of manually photoshopping the image for hours or weeks.

Chapter 2: Related Works

The idea of using machine learning to colorize images first became popular in the early 2000s. Welsh et al. began with an algorithm which transferred colors simply based on the value and standard deviation of luminance at each pixel [1]. Although the algorithm runs quickly, the results we obtained by implementing their method are far from optimal in that each prediction is purely local and that the feature space they are searching over has only two dimensions. Soon after, Levin et al. took colorization in a completely different direction with their reformulation as a global optimization problem over the image [2].

Unlike Welsh's technique, which accepted a color image as training input, Levin's optimization algorithm accepted hand-drawn "scribbles" of color from the user which are then used as linear constraints for the optimization of a quadratic cost function. This algorithm sought to label pixels the same color if they had similar luminance values, which is one of the first examples of automatic colorization algorithms explicitly incorporating spatial coherence as part of their classification.

Irony et al. then improved the technique by incorporating a two-phase voting procedure [3]. In the first step, pixels are independently colored based on their individual texture features. In the second, each pixel's color is replaced by a weighted majority vote over its neighboring pixels. The most confidently labeled pixels are then fed into Levin's original algorithm as "micro-scribbles." Noda et al. formulated the problem as Bayesian inference using a Markov random field model of an image [4]. This formulation naturally includes spatial coherency considerations while computing MAP estimates of the colors.

The method implemented in this project is an extension of the example based colorization method where a large image dataset is provided to the algorithm and the model transfers colors by considering the observed patterns in the provided dataset [5].

Chapter 3 : Design and Implementation

3.1 System Requirement Specification

3.1.1 Software Specification

The following tools have been used in the course of this project

- Programming Language: Python
- Libraries: Numpy, Tensorflow, Keras, Scikit-image, Matplotlib
- Tools: Jupyter Notebook, VS Code

3.1.2 Hardware Specification

To train our models, we needed a device with medium RAM and CPU. We trained our model on Intel i5 with 8 GB Ram. However, with GPU we can accelerate the training process. To generate results, any modern device with web browser support can be used.

3.2 Flowchart

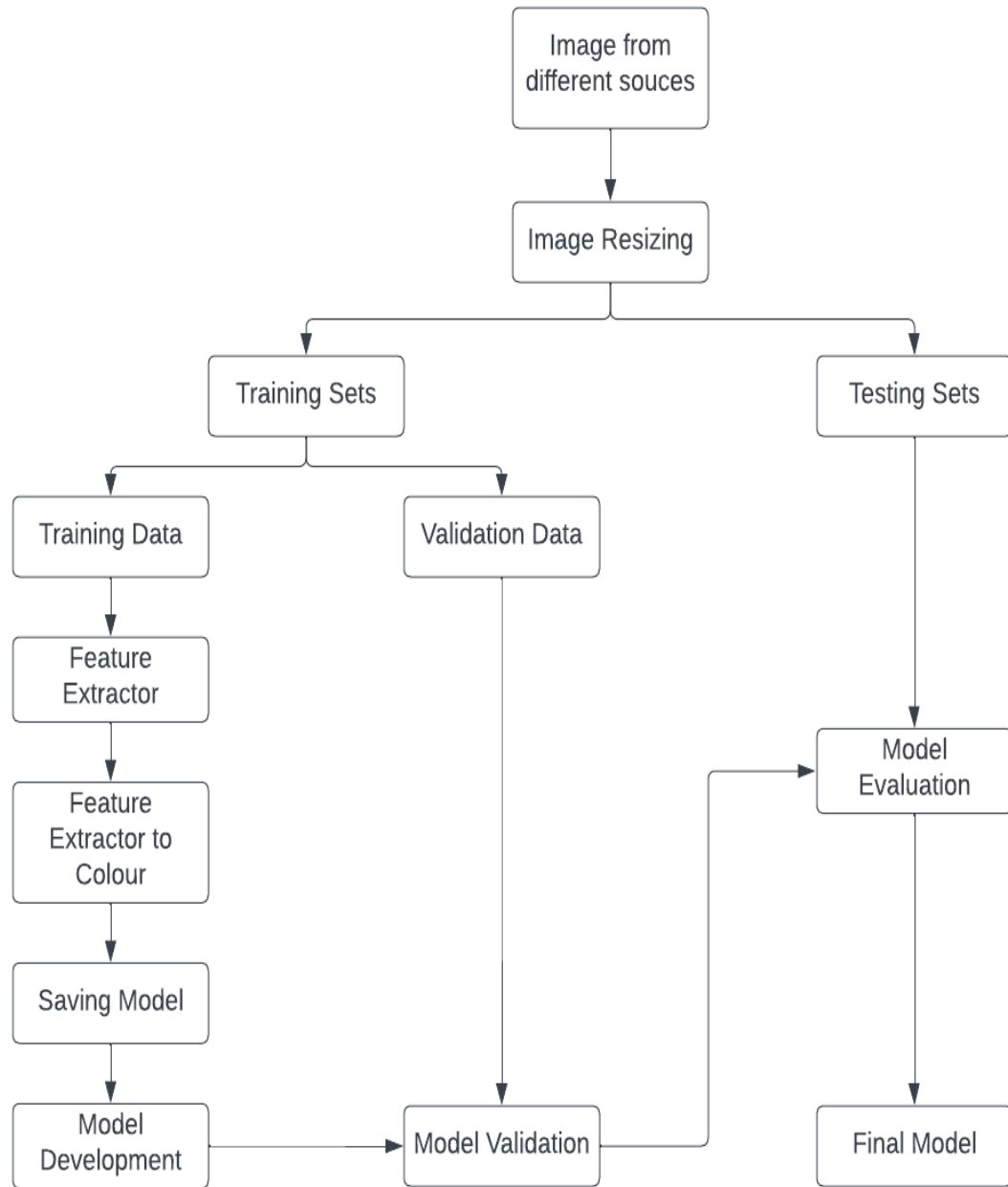


Fig: Flowchart of Design and Implementation

3.3 Model Implementation

3.3.1 Dataset

Colorization of image is conceivable just for similar images on which the dataset has been prepared. Subsequently a dataset which can cover wide varieties of images is viewed as a decent dataset. For this project we have used Berkeley Segmentation Dataset BSDS 300 (<https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/>) and also collected some similar images manually. Our datasets consist of training and validating examples that consist of color images of a variety of objects.

3.3.2 Data Preprocessing

The images from the dataset are the input to our model. First of all we get the landscape image, resize them and then append them in the array. We begin by converting the given image into a n-dimensional numpy array where each pixel point on the image is matched to its corresponding RGB value. Finally, the images are normalized in the range of [0-1] using min-max scaling and finally appended into an empty list.

3.3.3 Feature Extraction

Our neural network finds characteristics that link grayscale images with their colored versions. It does so by scanning each image from top left to bottom right and trying to predict which color each pixel should be. It works in a trial and error manner. It first makes a random prediction for each pixel. Based on error for each pixel, it works backward through the network to improve the feature extraction.

It first looks for some simple patterns: a diagonal line, all black pixels etc. Then it looks for the same exact pattern in each square and removes the pixels that don't match. We can get images from the mini filters. If we scan the images again, we'd see the small patterns that we had already detected and after we combine all the new pixels we can detect more complex

patterns like half circle, small dot, line. We repeatedly extract the same pattern from images to get better filtered images.

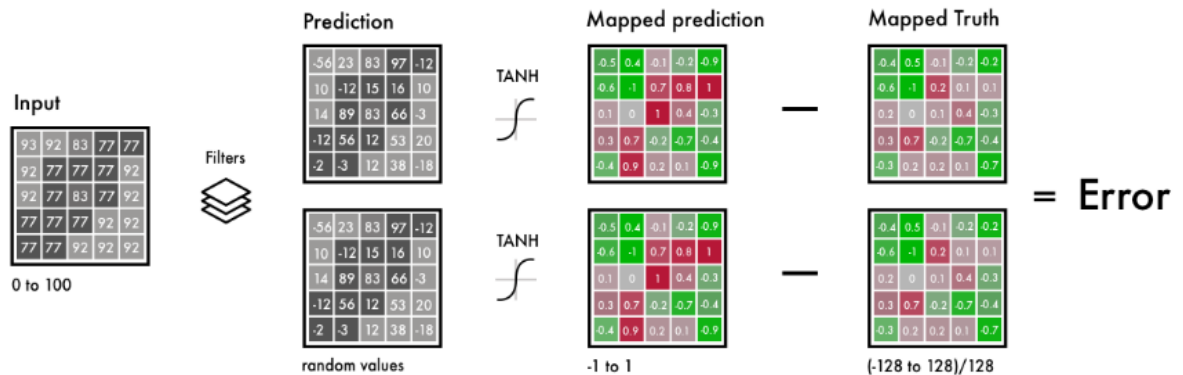


3.3.4 Defining the Model

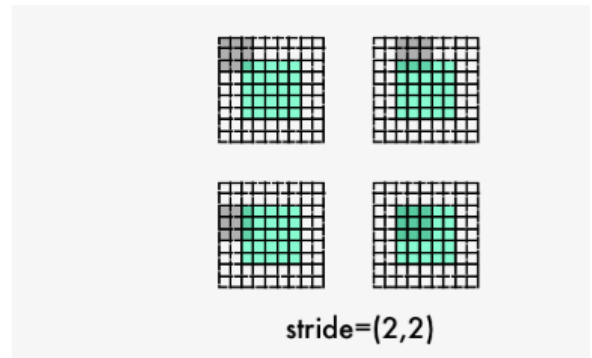
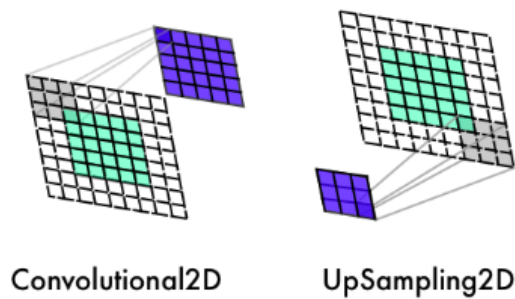
After the image is converted into its equivalent numerical value, we then set up the Sequential model (i.e model that feeds our inputs sequentially from Input to Output).

We have a grayscale layer for input and we want to predict the other two color layers. To turn a single layer into two layers, we have used convolutional filters. They can highlight or remove something to extract information out of the picture. Each filter determines what we see in the picture. For the convolutional neural network, each filter is automatically adjusted to help with the intended outcome.

In the next steps, we design our Convolution Neural Network (CNN) model with a multi-layers hidden unit. In the hidden layers, the summed activation of the node is then transferred to the activation function (RELU and Tanh) which then outputs the results.



Here, input is a grid consisting black and white image that outputs two grids with color values. Between the input and output values, we create filters to link them together called CNN. Here, we map the predicted values and real values within the same interval $[-1,1]$. This way we compare the values. To map the predicted values we use a Tanh activation function that will return the values between $[-1$ to $1]$.



The pixel location is important for coloring networks as the image size or ratio stays the same throughout the network. Here, we have used a stride of 2 to decrease the width and height by half. Stride specifies how much we move the convolution filter at each step. This increases the information density and also does not distort the image. Likewise, it is important to upscale the layer and maintain the image ratio. This is done by adding white padding like visualization above otherwise each convolutional layer cuts the images. It's done with the `*padding='same'*` parameter.

```

model = Sequential()

#Input Layer
model.add(Conv2D(64, (3, 3), input_shape=(256, 256, 1), activation='relu', padding='same'))

#Hidden Layers
model.add(Conv2D(64, (3, 3), activation='relu', padding='same', strides=2))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same', strides=2))
model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(256, (3, 3), activation='relu', padding='same', strides=2))
model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(UpSampling2D((2, 2)))

```

Here in training the dataset, the for loop counts all file names in the directory. Then it iterates through the image directory, combines the images into an array of pixels and combines them into a giant vector.

```

for imagename in os.listdir(train_dataset):
    X.append(img_to_array(load_img(train_dataset + "/" + imagename, target_size=(256, 256))))

```

In ImageDataGenerator, we adjust the setting for our image generator. This way, one image will never be the same, thus improving the learning. The shear_range tilts the image to the left or right

```

# Image transformer
datagen = ImageDataGenerator(
    shear_range=0.2,
    zoom_range=0.2,
    rotation_range=20,
    horizontal_flip=True)

```

We use the images from our folder, Xtrain, generating images based on the settings below. Then we extract the black and white layer for the X_batch and the two colors for the two color layers.

```
# Generate training data
batch_size = 10
def image_a_b_gen(batch_size):
    for batch in datagen.flow(Xtrain, batch_size=batch_size):
        lab_batch = rgb2lab(batch)
        X_batch = lab_batch[:, :, :, 0]
        Y_batch = lab_batch[:, :, :, 1:] / 128
        yield (X_batch.reshape(X_batch.shape+(1,)), Y_batch)
```

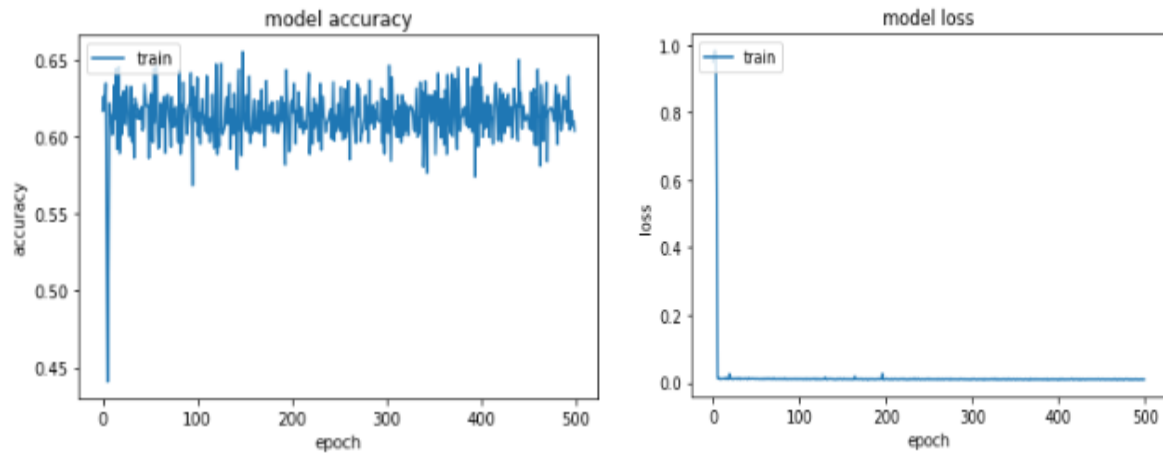
3.3.5 Testing and Evaluating the Model

We have used 95% of the dataset to train the model and remaining 5% to test it. The model seemed to fit well as it gives an accuracy score of about 0.6040 with loss of 0.0095 at 500th epoch.

```
Epoch 498/500
30/30 [=====] - 174s 6s/step - loss: 0.0091 - accuracy: 0.6069
Epoch 499/500
30/30 [=====] - 175s 6s/step - loss: 0.0101 - accuracy: 0.6104
Epoch 500/500
30/30 [=====] - 177s 6s/step - loss: 0.0095 - accuracy: 0.6040
```

3.3.6 Improving the Accuracy

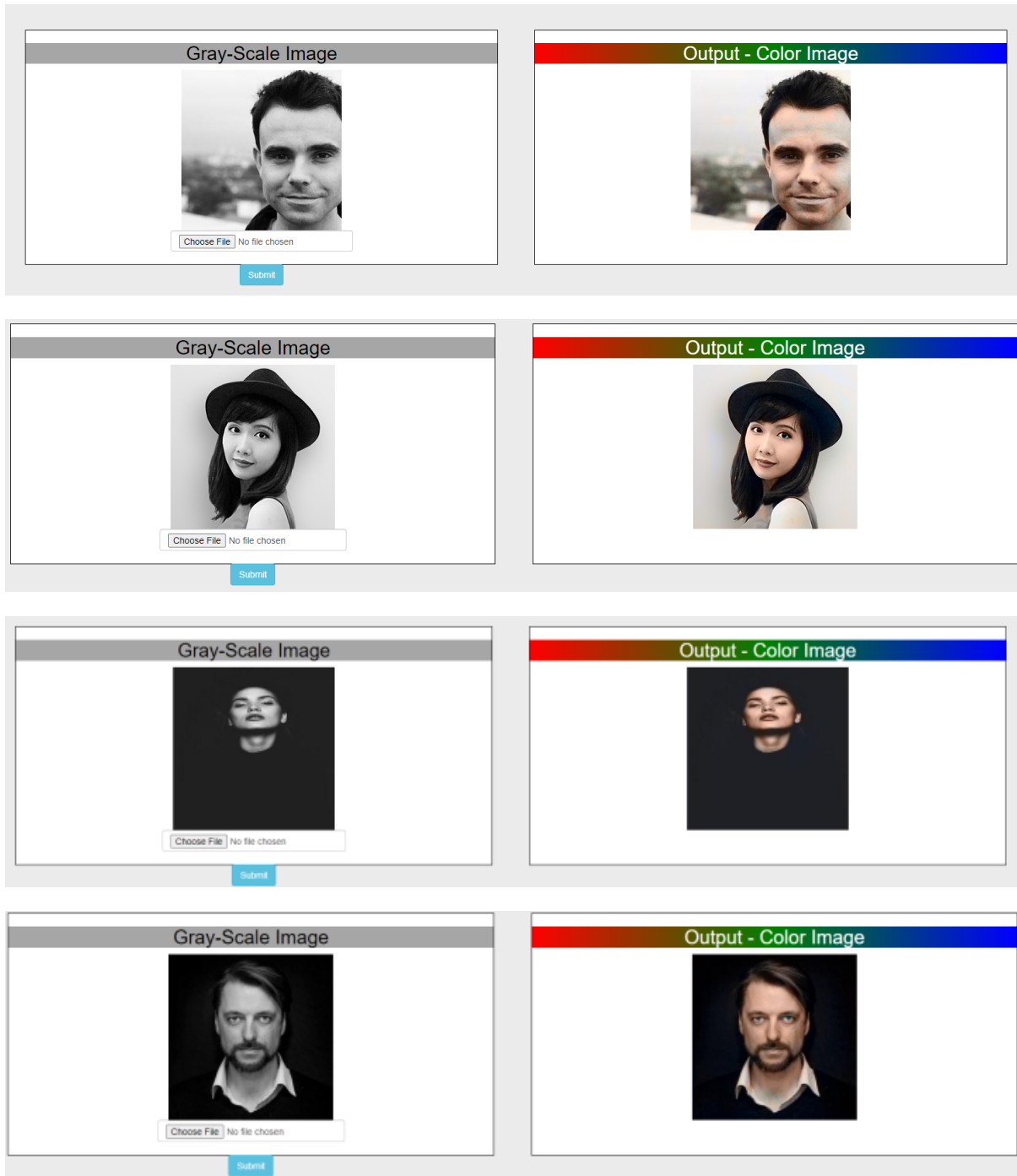
At this stage, the output given by the model is not very accurate, we can further increase the accuracy of our model by increasing the size of training data. However, a more diverse dataset can make the picture brownish. On the other hand, more similar images can get a decent result but the tradeoff is the network becomes worse at generalizing. Hence, we must select a uniform distribution of diverse images with similarity to improve the accuracy of our model.









Finally, We have created a web application interface to deploy our model. We have used HTML, CSS and Flask framework to design the web application that takes grayscale images and converts them to colored images.

Chapter 4 : Discussion on the Achievement

The results obtained from our model are shown below. The left column includes the input images and the right column includes the automatically colored images using CNN.



<p>Gray-Scale Image</p>  <p><input type="button" value="Choose File"/> No file chosen</p> <p><input type="button" value="Submit"/></p>	<p>Output - Color Image</p> 
<p>Gray-Scale Image</p>  <p><input type="button" value="Choose File"/> No file chosen</p> <p><input type="button" value="Submit"/></p>	<p>Output - Color Image</p> 
<p>Gray-Scale Image</p>  <p><input type="button" value="Choose File"/> No file chosen</p> <p><input type="button" value="Submit"/></p>	<p>Output - Color Image</p> 

Chapter 5 : Conclusion

We have discussed the Image colorization technique that involves the use of a special class of convolutional neural networks (CNN) called autoencoders. These neural networks are able to distill the salient features of an image, and then regenerate the image based on these learned features. Image colorization often requires a large amount of human effort, time and skill but this type of deep learning architecture has made this task quite easy.

5.1 Limitations

- More diverse dataset makes the pictures brownish. So, it becomes worse at generalizing.
- Cannot detect all shades of colors that result in pictures being less appealing.
- Size of each image has to be exact and proportional throughout the network otherwise it cause losing half a pixel
- Takes much longer to process the model during training.

5.2 Future Enhancements

- Better UI can be developed
- Model to be trained to detect different colors and patterns to make the image more fascinating.

References

- 1) T. Welsh, M. Ashikhmin, and K. Mueller, “Transferring color to greyscale images,” *ACM Transactions on Graphics (TOG)*, vol. 21, no. 3, pp. 277–280, 2002.
- 2) A. Levin, D. Lischinski, and Y. Weiss, “Colorization using optimization,” in *ACM Transactions on Graphics (TOG)*, vol. 23, pp. 689–694, ACM, 2004.
- 3) R. Irony, D. Cohen-Or, and D. Lischinski, “Colorization by example,” in *Eurographics Symp. on Rendering*, vol. 2, Citeseer, 2005.
- 4) H. Noda, H. Korekuni, N. Takao, and M. Niimi, “Bayesian colorization using mrf color image modeling,” in *Advances in Multimedia Information Processing-PCM 2005*, pp. 889–899, Springer, 2005.
- 5) G. Charpiat, I. Bezrukov, Y. Altun, M. Hofmann, and B. SCH, “Machine learning methods for automatic image colorization,” *Computational photography: methods and applications*. CRC Press, Boca Raton, pp. 395–418, 2010.
- 6) *Automaticcolorizationofgrayscaleimages - Stanford University*. (n.d.). Retrieved August 24, 2022, from <http://cs229.stanford.edu/proj2013/KabirzadehSousaBlaes-AutomaticColorizationOfGrayscaleImages.pdf>
- 7) *Colorize pictures: Turn black and white photos to color with ai*. Hotpot Design. (n.d.). Retrieved August 24, 2022, from <https://hotpot.ai/colorize-picture>
- 8) The berkeley segmentation dataset and benchmark. (n.d.). Retrieved August 24, 2022, from <https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/>

Appendices

Work Week	1	2	3	4	5	6	7
Research							
Data Preprocessing							
Parameter Tuning							
Training and Testing							
Documentation							

Fig: Gantt Chart

Screenshots

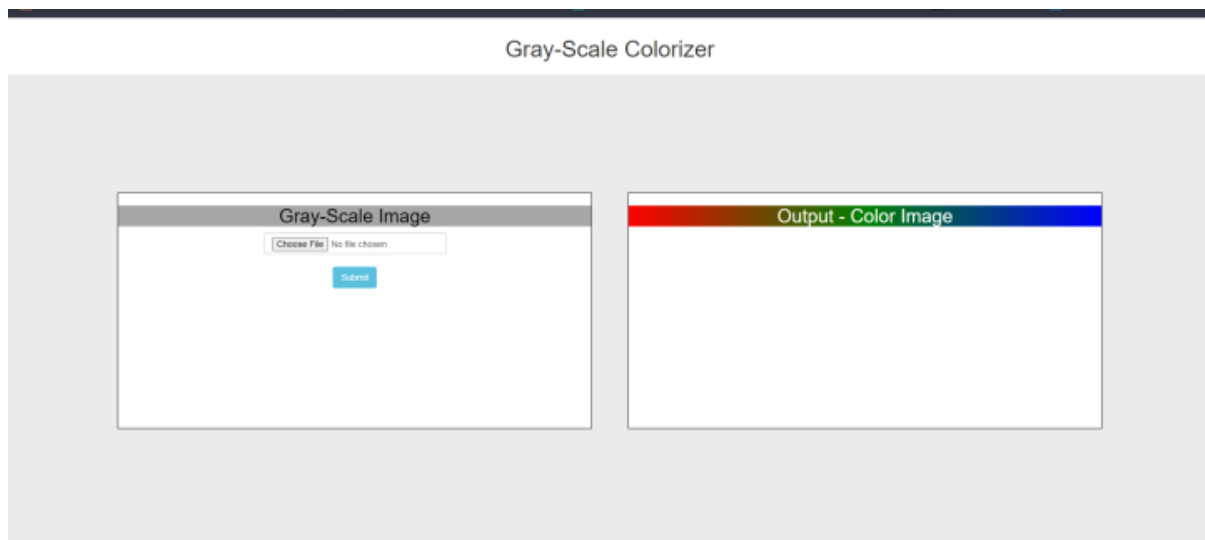


Fig: UI of Gray-Scale Colorizer

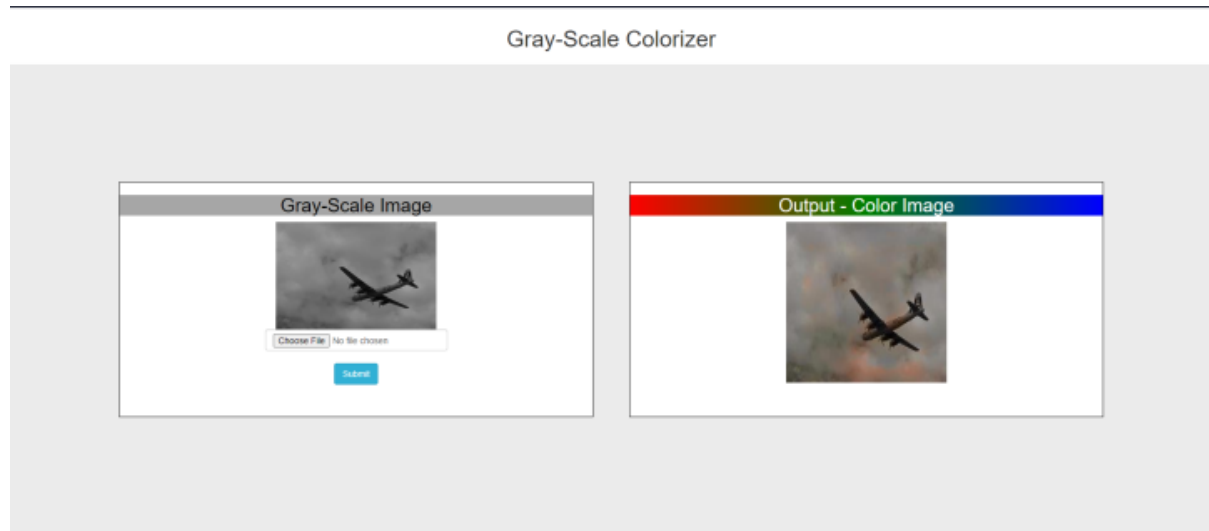


Fig: Automatically Colorization of the input image

Code Repository

[Automatic Colorization of Grayscale Image](#)

Video Demo Link

[Gray Scale Colorizer Demo](#)