Report - Assignment 4 : Adding a Splash of Color

Diksha Prakash Kathikeyan Sethuraman Arun Sinhmar

December 2019

Abstract

The purpose of this assignment is to demonstrate and explore some basic techniques in supervised learning and computer vision.

For the purpose of this project, we have implemented two methods, as under:

1. Method 1: Without using packages, satisfying the requirements of the assignment
2. Method 2: Using external packages. The reason for this is that Method 1 wasn't producing prefect results, despite tweaking the parameters.

Thus, this report answers all the questions with reference to both the methods individually.

1 Representing the Process

METHOD 1:

Our approach is to extract and the pixels values for each Red, Blue and Green from an image and store them in independent arrays.

As provided in the problem statement, each colored image; with R, G and B as the corresponding Red, Green and Blue pixel values; is converted into a grayscale image using the formula:

$$Grayscalevale(R, G, B) = 0.21R + 0.72G + 0.07B$$

With the above formula, the input array will be consisting of grayscale pixel value and the ultimate output has to be three independent arrays with red, blue and green pixel values independently. As we can see that capturing good quality of information isn't feasible as a single grayscale pixel value has to correspond to three colours (RGB) on a pixel by pixel basis.

Thus, 'filters' have been used in our implementation to capture the data around a given pixel. Filters help capture more data from the input space before mapping it to the output space. If filter of size 9*9 is taken, there exists a 9:1 input to output mapping i.e. Grey to 'any' colour mapping (I.e. either of Red, Green or

Blue). Since it is important to capture the right amount of information, neither too much nor too less, choosing the right filter size is also important.
After experimenting and tweaking the parameters around, we decided the filter size to be 9. Since data extraction from the corner cases is quite different than that from the non-corner cells, zero padding is done. Also, the formula to calculate the output size is as follows:

$$OutputSize = \frac{W - F + 2P}{S} + 1$$

Where, W : Input Size, F : Filter Size, P: Padding and S : Stride (Reference: http://cs231n.github.io/convolutional-networks/#layers)
For the purpose of this project, the stride has been taken as 1. Also, the pad width is calculated as P = ((F-1))/2 as it ensures that the input volume and output volume will have the same size spatially.

METHOD 2:

We decided to a represent each row of the gray input as a sub-array of nearest neighbors for each of the coordinate in the original data. We also tried to parameterize this Kernel-Width and that could be one of the tuning parameters in choosing the best model. For example, K=3 would represent each point (x, y) as a 3 X 3 matrix, where the (x ,y) is the center of the Matrix.
Accordingly, the input Gray matrix is padded with zeroes so that we get a K X K Sub-array for every point in the data. Pad-width for Zero padding = Floor(K/2)

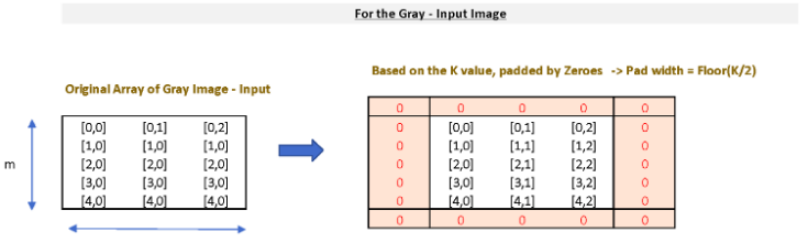

Figure 1: Illustration of Data Representation for neighbor span (Kernel Width) K=3 - PART 1

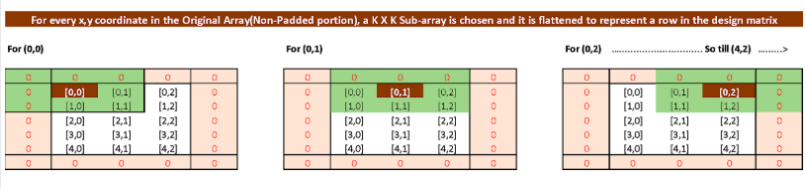

Figure 2: Illustration of Data Representation for neighbor span (Kernel Width) K=3 - PART 2

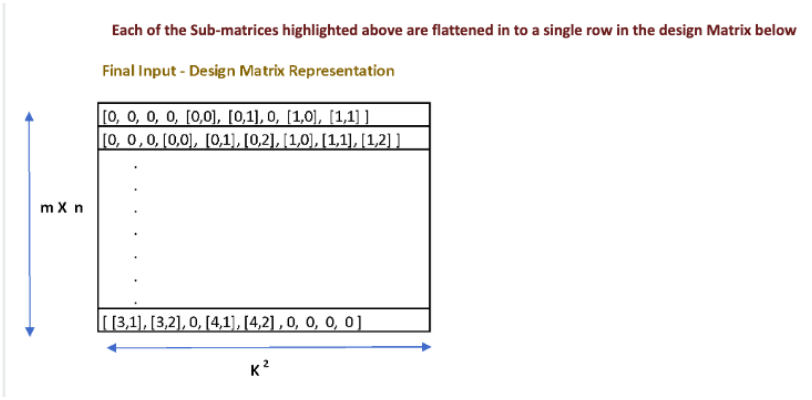

Figure 3: Illustration of Data Representation for neighbor span (Kernel Width) K=3 - PART 3

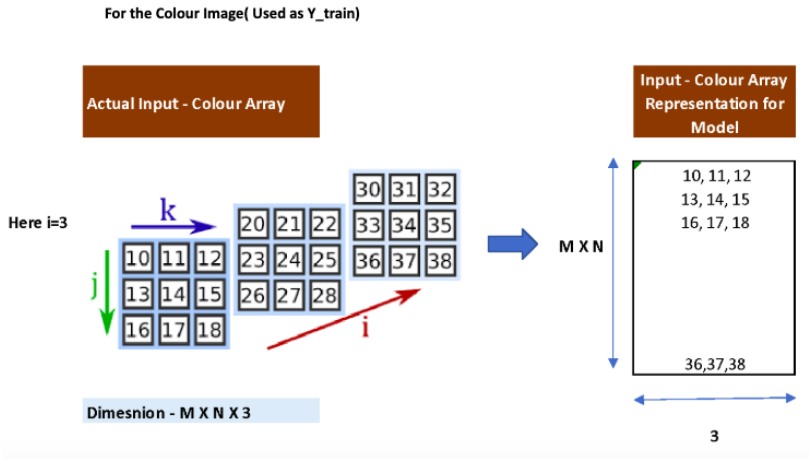

Figure 4: Illustration of Data Representation for neighbor span (Kernel Width) K=3 - PART 4

2 Data

METHOD 1:

For the purpose of this project, 11 random images were chosen from Google and Unsplash. The sizes and the layouts of the images taken for training data were different and resizing them has been handled by the code.
Overall, the following can be noted about the data and how they are processed and dealt with:

1. For the sake of computation efficiency, all the images of varied sizes are shrunk to a constant dimension of 100 * 100

2. The training data is an array of Grayscale values which is obtained by converting RGB image to grayscale using the formula: Grayscale vale (R, G, B) = 0.21R + 0.72G + 0.07B

3. Zero padding has been done to ensure uniformity and the pad width is calculated using the formula P = ((F-1))/2

4. Filter size of 9*9 has been chosen i.e. data from corresponding 9 cells has been taken into account too. Thus, there exists a 9:1 input to output mapping.

5. The images that we used for training as under. The reason for selection of these images was to train the network on varied colors using the combination of RGB variants.

Figure 5: Training Data

3 Evaluation of the Model

METHOD 1:

The model is evaluated by its extent of reciprocation of the RGB image from its grayscale image. Our model somewhat reciprocates the original image, although not with high accuracy. Although the colours may not be perfect, as seen in the output images, it is able to very well distinguish that there are different colours for each object (grass, train, trees, etc) present in the image.
The image that we used for testing purpose as under. The reason for selection of this images is to test the ability of the 'Image Colorizer' model to distinguish colors and objects well.

The evaluation metric used, to quantify how well our model is performing, is 'Mean Squared Error' (MSE). Also, perceptual error plays a vital role in this model as we are dealing with images. It is a natural human instinct to distinguish objects and the colors associated with them. For example, we know for sure that the leaves on the tree are green and not blue! If our model colors the

Figure 6: Test Image

Figure 7: Output of the Neural Network

leaves with blue, we know that the model isn't accurate.

METHOD 2:

We decided upon using an autoencoder architecture. Input dimensions are in the form of 32, 52 OR. . . . . K2 based on the K-value (nearby pixels) we choose to represent the gray data. It is projected on to the input layer to produce a 16-dimensional embedding after which it is fed to an Auto encoder with a bottleneck layer of 32 – dimensions.

```
def model_define(self):
  #code for model design
  model = Sequential()
  model.add(Dense(16, activation='relu', input_dim=self.neighbor_span*self.neighbor_span))
  model.add(Dense(32, activation='relu'))
  model.add(Dense(16, activation='relu'))
  model.add(Dense(8, activation='relu'))
  model.add(Dense(3, activation='sigmoid'))
  # model.summary()
```

The assumption behind using an auto-encoder is to learn Sparse embeddings the bottle neck layer which would filter out noise from the data and the Decoder component reduces 3- dimensions (r, g, b) of the center point in our matrix.
To avoid overfitting- Early stopping rounds – Very few Epochs were given, and early stopping based on the increase in Accuracy was done to prevent model from overfitting.
Dropouts – In case if the test data performance turns out low, we thought of trying out this approach. But since even with less epochs the test data performance was decent,

```
Inital dimesions of the color array given as input (174, 281, 3)
Dimesions of the train color array converted to (48894, 3)
Dimesions of the train gray array converted to (174, 281)

Inital dimesions of the uploaded test gray array (361, 641, 1)
Dimesions of the uploaded test gray array converted in to shape (361, 641)
The dimensions of the train design matrix of the gray input to the model is (48894, 9)
The dimensions of the train response matrix of the gray input to the model is (48894, 3)
Epoch 1/5
48894/48894 [==============================] - 62s 1ms/step - loss: 0.0259 - acc: 0.5609
Epoch 2/5
48894/48894 [==============================] - 62s 1ms/step - loss: 0.0204 - acc: 0.6036
Epoch 3/5
48894/48894 [==============================] - 62s 1ms/step - loss: 0.0194 - acc: 0.6122
Epoch 4/5
48894/48894 [==============================] - 61s 1ms/step - loss: 0.0188 - acc: 0.6159
Epoch 5/5
48894/48894 [==============================] - 61s 1ms/step - loss: 0.0182 - acc: 0.6234
After conversion, the shape is now (174, 281, 3)
The dimensions of the test design matrix of the gray input to the prediction is (231401, 9)
After conversion, the shape is now (361, 641, 3)
```

```
    model.compile(loss='mean_squared_error',
        optimizer='adam',
        metrics=['accuracy'])

    return model
```

"Mean-squared Error" was used as the loss function to train the model.
Optimizer – Adam optimizer was chosen. Based on the research online a lot of sources said Adam optimizer is well suited for these deep learning problems as opposed to a gradient descent which would require a lot of hyper parameter tuning that needs to be done to converge faster.

4 Training the Model

METHOD 1:

The work implements a fully connected feed forward neural network. After tweaking around with the parameters, we came to a conclusion to use 4 hidden layers with 10, 20, 20 and 10 neurons each as this combination is able to capture optimal information. Also, 'Backpropagation' with 'Mini-Batch Gradient Descent' has been used to reduce errors and optimize the network.
Initially, we implemented a 'Linear Regression' and a 'Logistic Regression' model for the image colorization. However, the quality of the color reciprocation wasn't good enough. Thus, we decided to build a neural network where every layer other than the last layer has 'Sigmoid Activation Function'. The last layer has 'Linear Activation Function'.
To determine convergence, epochs were set to 50, which proved to be more than sufficient to determine an optimal solution.
Along with the train data, a small fraction of data was used in the model as the validation. This data was not used in the training process of determining weights. After every epoch, error on the validation data was determined. If the error on the validation data did not improve over two consecutive epochs by

a tolerance factor (as mentioned previously), training would stop. This helped in stopping the model from over-fitting or learning the weights very specific to training data.
A small fraction of the training data fed to the model, is used for validation purpose. These images don't play a role in determining the weights. After every epoch, the error on validation set is determined and if this error doesn't improve over any two consecutive epochs by a factor of 10e-5, the training was stopped to prevent the model from over-fitting.

5 Accessing the Project

5.1 How good is your final program? How can you determine that?

METHOD 1:
The performance of our model is fair as it is able to distinguish objects and colors in the grayscale image while colorizing it i.e. it understands the object boundaries. It has the ability to associate colors unique to typical things like, blue color for sky, black color for roads, green color for trees.

METHOD 2:

1. Prediction Accuracy on test Data: We tried colorizing the test image (similar to the train) and below given are the results for various values of K-chosen to represent the input Gray image fed in to the model.

Figure 8: Test Predictions on the gray test Image Used: for K = 3

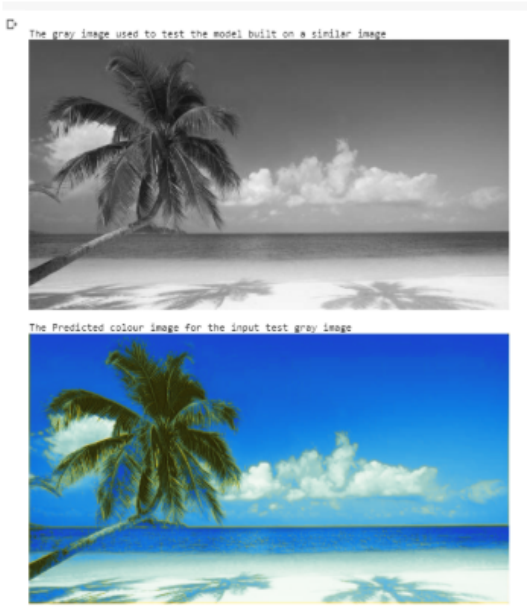

Figure 9: Test Predictions on the gray test Image Used: for K = 5

Figure 10: Test Predictions on the gray test Image Used: for K = 9

2. Generalization Capability: A good architecture should perform well for different set of images. So a variety of images needs to be used to test the robustness of the model. We tried choosing another image(given below) where the test image is sort of somewhat related but not as similar as the 1st image we had chosen. The model was producing decent results with K-value= 3. The results are as given below.

Figure 11: Train Image- Actual

Figure 12: Train Image- Fitted by the model

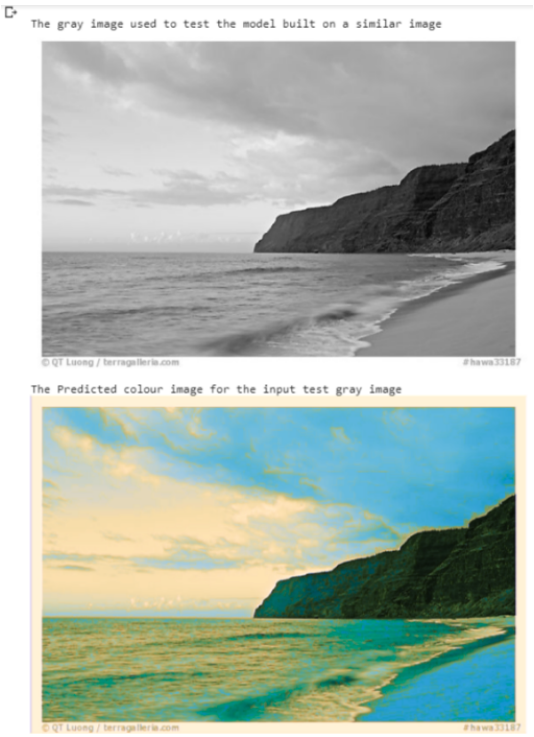

Figure 13: Prediction on Test Data

5.2 How did you validate it?

METHOD 1:

For this purpose, we used numerous images downloaded from Google Images and Unsplash. As we can see from the output images of the neural network, the network recognizes the boundaries in the image and colorizes using the basic sense of standard colors for typical objects.

However, sometimes the model doesn't color accurately due to the dominance of a particular color in the training data. For e.g. initially we used lots of images with green color in the background in our training data. As a result, our model was biased towards coloring objects in shades of green.

5.3 What is your program good at?

METHOD 1:

Our program is good at identifying object boundaries and assigning different colors to the images. As we can see, although not accurate, the model colors the grayscale test image in a similar shade as the original test image.

5.4 What could use improvement?

METHOD 1:

A wise selection of the training data which includes a uniform distribution of colors can improve the model a lot.

As we trained and tested our data, we understood that the training data plays an important role in the output of the model. Initially, we used lots of images with green color in the background hence, our model was biased towards coloring objects in shades of green. Next, due to dominant presence of red color, the model was biased to assigning red hues to objects. Thus, we carefully curated a training data-set to be versatile and exhaustive.

5.5 Do your program's mistakes make sense?

METHOD 1:

Yes, the model's consistent mistake to be biased towards a particular color makes sense as it has seen that color more often in the training data.

5.6 What happens if you try to colour images unlike anything the program was trained on?

METHOD 1:

If we attempt to color images that are fundamentally distinct from those that the model has been trained on, the model isn't able to reciprocate the colors well. This makes sense as the model depends a lot on the color distribution of the training images.

During the testing of the model, we had used Image 14 for testing. Since the model hadn't seen anything of this kind before, although it was able to distinguish the object boundaries, it failed miserably at the colouring part.

Figure 14: Test Image

Figure 15: Output of the Neural Network

Thus the whole idea of humongous datasets for extremely accurate model is quite convincing.

5.7 What kind of models and approaches, potential improvements and fixes, might you consider if you had more time and resources?

METHOD 1:
For the purpose of this assignment, we have implemented a very basic neural network with 3 hidden layers to produce a fairly accurate colorized image from the grayscale image. Provided we had more times and resources, we would like to consider the following:

1. Variation Auto-Encoders: Due to the innate ability of the AVEs to synthesize new data (here, new images) from the given input data (here, input training images), we can expect a better colorization of test images using VAEs.

2. Convolutional Neural Networks: This is an obvious choice as they are able to identify and learn color distributions more accurately than a basic feed-forward network.

3. Datasets like McGill, MIT CVCL, ILSVRC 2015 CLS-LOC, MIRFLICKR can be used to train our network on a variety of images and colours/hues. In this manner, we can expect the output to be more precise.

    (a) The MIT CVCL Urban and Natural Scene Categories dataset contains several thousand images partitioned into eight categories.

    (b) The McGill Calibrated Colour Image Database contains more than a thousand images of natural scenes organized by categories.

    (c) The ILSVRC 2015 CLS-LOC dataset is the dataset used for the ImageNet challenge in 2015

    (d) The MIRFLICKR dataset comprises 25000 Creative Commons images downloaded from the community photo sharing website Flickr