Mean Chrominance of an image and Image Colorizer Using Deep Learning

Chintan K. Acharya

CAP 5404 - UNIVERSITY OF FLORIDA

October 31, 2022

Introduction:

Image colorizing is the process of predicting RGB values for grayscale images. This project uses Deep learning to achieve image colorization. This project also includes calculating the mean chrominance values of an image. Chrominance refers to the colorimetric variation between a specific color in a television image and a reference color with a comparable brightness. The images that were used in the project are standard 3-channel RGB Images. The images are first converted into LAB from RGB format. In the LAB format is L channel denotes the lightness of the image, the A channel denotes the red/green color values and the B channel denotes yellow/blue color values. Hence L channel gives the luminance and the A, and B channel gives the chrominance of the image. The mean chrominance values are found by taking the mean values of every pixel in the A, and B channels. In this project, a deep learning model has been trained to predict the mean chrominance value of the image. For the image colorizer model, the project uses the same LAB format images but in this case, the model developed predicts the value of each pixel of the A and B channels. The project used the Georgia Tech faces dataset for both tasks. Furthermore, this project goes on to use transfer learning for achieving the image colorization goal on a different dataset. Transfer Learning is the process of storing knowledge gained while solving one problem and applying it to solve a different but related problem. It uses the same process as the standard image colorization but uses a different dataset. This part of the project uses the NCD dataset. The report states each aspect of the project in detail.

Dataset:

The project used 2 datasets:

1. **Georgia Tech Faces Dataset:** This dataset contains 750 images of 50 people taken in 2 or 3 sessions. The images used in this project are 128x128 pixel images each. Each image is a 3-channel standard RGB Image.
2. **NCD Dataset:** This dataset is an image colorization dataset where images are true to their colors. It contains around 720+ images of different vegetables and fruits taken from the internet that contain their natural colors. These images are 3-channel standard RGB images. But the size of these images is not constant. This project converts these images to 128x128 images and then performs transfer learning using these images.

Image Augmentation:

Image augmentation is a process of creating images for the training of a model using different pre-processing techniques and a combination of these techniques. For this project, the datasets consist of approximately 700+ images each. The dataset is first split into train and test sets with 90% as the train set and 10% as a test set. Then image augmentation is applied to the train set. The techniques used in these projects for image augmentation are:

1. **Vertical Shift:** In this method, the pixels of the images are vertically shifted by a random factor without changing the dimension of the image. This produces an image that is shifted upwards or downwards.
2. **Horizontal Shift:** Similar to vertical shift, in this method the pixels of the image is shifted horizontally without changing the dimensions of the image.
3. **Crop:** In this method, the original image is cropped and enhanced to have the same dimension. The output of this method is basically a part of the original image and has the same dimension.
4. **Scale:** In this image augmentation method, the value of each pixel in the image is scaled by a random factor. In this project, a random value between the range [0.6, 1] is selected to scale each pixel.

For the project, each image in the trainset is subject to each of these methods of image augmentation three times and each time the factor by which the images are augmented is randomly selected. Hence, the output of image augmentation is 12 images per image in the dataset. So, if the train set contains 90% of 750 images i.e. 675 images, after image augmentation it will contain 8775 images.

Tech Stack:

1. **Software:**
   The project uses Google's Tensorflow to design and train the models. The models are trained and tested on Jupyter Notebooks. It uses the Python OpenCV library to handle images and image augmentation.

2. **Hardware:**
   The platform used for training the models is Google Colab. Google Colab uses GPUs for implementing TensorFlow operation as a hardware accelerator. It can be selected from the "Manage Resources -> Change RunTime" on the Colab.

Pre-processing:

In this step, all the values in the input and output are scaled to be between 0 to 1 since our models use the Relu Activation function. All the values are between 0 to 255 since these are the pixel values. For the mean chrominance model, it is the mean of the values that are between 0 and 255 so that will be also between 0 and 255. So simply dividing by 255 will result in a value between 0 to 1. The Conv2D layers take only 3D inputs i.e and the inputs are reshaped to be 3 dimensional i.e. 128x128 is reshaped to be 1x128x128 and similarly, the output is also reshaped to be compatible with the model.

For the models with the Tanh activation function in the last layer of the model, the output is scaled to be between -1 and 1. The A and B channels have values between 0 and 255. Hence, 128 is subtracted from each value and divided by 127 to have value between -1 and 1.

Implementation:

1. **Mean Chrominance Model:**
   The project uses Convolutional Neural Networks to predict the mean chrominance values of an image. First, all the images of the Faces Dataset are subjected to image augmentation and split into train and test sets. Hence, the train set contains 8775 images, and the test set contains 975 images. The images are first converted to pixels using the OpenCV library. The default format of the CV2 library is the BGR format. These BGR images are then converted from BGR to LAB format. The L channel is of shape 128x128 and is taken as input for the model. The mean value of all pixels in the A and B channels are calculated and taken as the output. So the input shape of the model is 128x128 and the output shape is 2. The model used 2D Convolution layers starting with an input size of 128x128 and filters size such that the output is half of the previous layer i.e. 64 x 64. Each 2D Convolutional layer uses ReLU as the activation function. The final layer i.e.output layer is a Dense layer since the output is one dimensional. The model architecture is as follows:

Layer (type)	Output Shape	Param #
conv2d_7 (Conv2D)	(None, 1, 64, 64)	32832
conv2d_8 (Conv2D)	(None, 1, 32, 32)	8224
conv2d_9 (Conv2D)	(None, 1, 16, 16)	2064
conv2d_10 (Conv2D)	(None, 1, 8, 8)	520
conv2d_11 (Conv2D)	(None, 1, 4, 4)	132
conv2d_12 (Conv2D)	(None, 1, 2, 2)	34
conv2d_13 (Conv2D)	(None, 1, 1, 2)	18
dense_1 (Dense)	(None, 1, 1, 2)	6

Total params: 43,830
Trainable params: 43,830
Non-trainable params: 0

FIG 1: Chrominance Model Summary

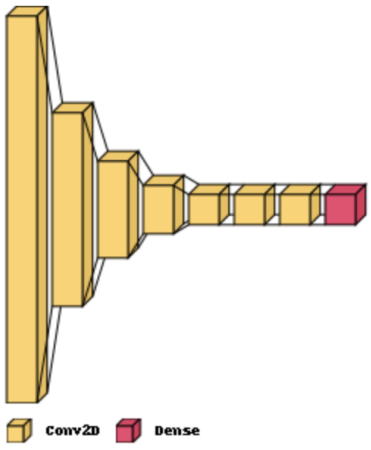


FIG 2: Chrominance Model Architecture

The Model uses the Adam Optimizer and Mean Squared error as the loss functions. It uses a callback of decreasing the learning rate when there's a plateau in validation loss to train the model optimally. The model was trained for 50 epochs with 512 batch size.

2. **Image Colorizer**

For the image colorization model, like the mean chrominance model, the images are converted into pixel values using the OpenCV python library. The images are split into train and test set in ratio 0.9 and 0.1. The dataset used is Faces Dataset. The train set is then subjected to image augmentation. Hence, the image augmentation results in 8775 training images and 75 test images. Now, each image is in BGR format as it is the default format of the CV2 library. The images are converted into LAB format and now the L channel was taken as the input of the model and the A, and B channels are the output of the model. The input shape now is 1x128x128 and the output shape is 2x128x128.

The first layer is a Transpose 2D convolutional layer with a filter such that the output is double the size i.e. 1x256x256 and then the next 5 layers are Convolution 2d Layer with a filter such that the output is half the size i.e. 1x128x128 then 1x64x64 and so on till 1x8x8. After that, the next layer is the 2D Transpose Convolutional layer with a filter that outputs 2x8x8 output and after that 5 similar layers that double the input in other 2 dimensions i.e. 2x16x16 then 2x32x32, and so on till 2x256x256. And the last layer is again a Spatial convolutional layer the inputs are 2x256x256 and the outputs are 2x128x128 i.e the shape of our output. Each layer uses ‘ReLU’ as the activation function. After every Spatial Convolutional or Spatial Transpose Convolutional layer, the model is subject to Batch Normalization.

In order to normalize the inputs of the layers by re-centering and re-scaling, **batch normalization** (also known as the batch norm) is a technique used to make the training of artificial neural networks quicker and more reliable. Because we normalize the inputs for each layer during training using the mean and variance of the values in the current mini-batch, it is known as "batch" normalization.

The model uses Mean Squared Error as the loss function and Adam’s optimizer. It was trained for 30 epochs with a batch size are 25.

A similar model was trained on the same dataset. The only difference was the last 2 layers of the model had the Tanh activation function. Hence, during the pre-processing the output values (A channel and B channel) were scaled to be between -1 to 1 rather than the usual 0 to 1. Hence, the output is pre-processed as mentioned above in pre-processing step. This model was trained for 50 epochs with batch size of 25. The difference between the output has been shown in the Results section and the evaluation section.

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_7 (Conv2D)           (None, 1, 128, 128)       65664

 batch_normalization_14 (Bat  (None, 1, 128, 128)      512
 chNormalization)

 conv2d_transpose_7 (Conv2DT  (None, 1, 256, 256)      33024
 ranspose)

 batch_normalization_15 (Bat  (None, 1, 256, 256)      1024
 chNormalization)

 conv2d_8 (Conv2D)           (None, 1, 128, 128)       131200

 batch_normalization_16 (Bat  (None, 1, 128, 128)      512
 chNormalization)

 conv2d_9 (Conv2D)           (None, 1, 64, 64)         32832

 batch_normalization_17 (Bat  (None, 1, 64, 64)        256
 chNormalization)

 conv2d_10 (Conv2D)          (None, 1, 32, 32)         8224

 batch_normalization_18 (Bat  (None, 1, 32, 32)        128
 chNormalization)

 conv2d_11 (Conv2D)          (None, 1, 16, 16)         2064

 batch_normalization_19 (Bat  (None, 1, 16, 16)        64
 chNormalization)

 conv2d_12 (Conv2D)          (None, 1, 8, 8)           520

batch_normalization_20 (Bat  (None, 1, 8, 8)          32
chNormalization)

conv2d_transpose_8 (Conv2DT  (None, 2, 8, 8)          264
ranspose)

batch_normalization_21 (Bat  (None, 2, 8, 8)          32
chNormalization)

conv2d_transpose_9 (Conv2DT  (None, 2, 16, 16)        528
ranspose)

batch_normalization_22 (Bat  (None, 2, 16, 16)        64
chNormalization)

conv2d_transpose_10 (Conv2D  (None, 2, 32, 32)        2080
Transpose)

batch_normalization_23 (Bat  (None, 2, 32, 32)        128
chNormalization)

conv2d_transpose_11 (Conv2D  (None, 2, 64, 64)        8256
Transpose)

batch_normalization_24 (Bat  (None, 2, 64, 64)        256
chNormalization)

conv2d_transpose_12 (Conv2D  (None, 2, 128, 128)      32896
Transpose)

batch_normalization_25 (Bat  (None, 2, 128, 128)      512
chNormalization)

conv2d_transpose_13 (Conv2D  (None, 2, 256, 256)      131328
Transpose)

batch_normalization_26 (Bat  (None, 2, 256, 256)      1024
chNormalization)

conv2d_13 (Conv2D)          (None, 2, 128, 128)       131200

 batch_normalization_27 (Bat  (None, 2, 128, 128)      512
 chNormalization)

=================================================================
Total params: 585,136
Trainable params: 582,608
Non-trainable params: 2,528
_________________________________________________________________
```

FIG 3: Image Colorizer Model Summary

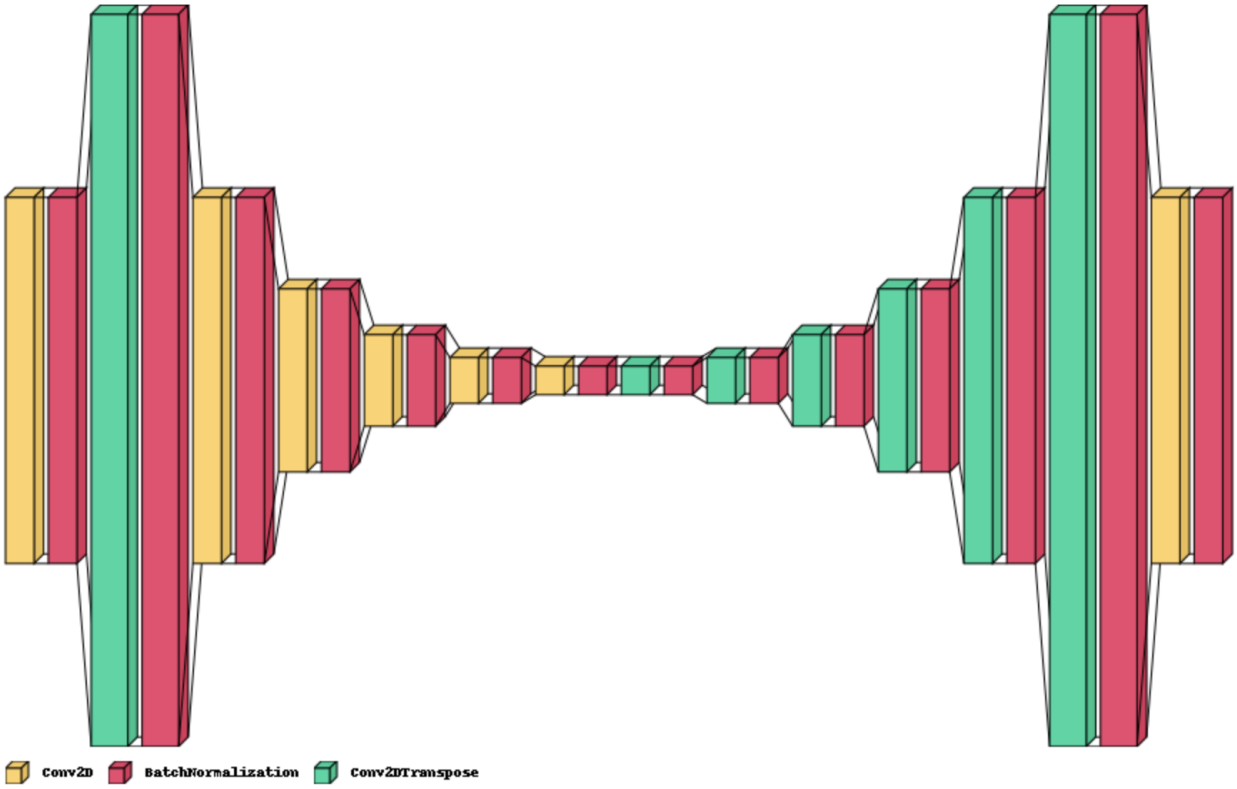


FIG 4: Image Colorizer Model Architecture

3. **Transfer Learning:**
   Transfer learning (TL) is a process that focuses on preserving information obtained while solving one problem and transferring it to another similar problem. For example, as humans, a person learns to code in one language say python. If then the person wants to learn C++ coding we can use the conceptual knowledge we learned in python like if_else, etc. to learn C++ we have to learn the syntax and so on. In this project, a similar concept is used. The previously trained image colorizer models trained on the faces dataset are trained to learn the natural colors of the objects using the NCD Dataset. The dataset is first to split into train tests in the same ratio and subjected to the same image augmentation. The loss function and optimizer are the same. The images in the dataset are not of constant sizes but during conversion from image to pixel values they are resized to 128x128x3. The first model was pre-trained for 30 epochs on this with a small batch size of 5. The Tanh pre-trained model was trained on this dataset for 25 epochs with a batch size of 5.

4. **GPU Computing:**
   For this project, the GPU was used to pre-process the dataset and train the model. The images were augmented and converted to pixel values. After converting images to input

and output the NumPy arrays were converted to Tensor and pushed to GPU for any further computations. Hence, the pre-processing was done on the GPU that Google Colab provides. After specifying that the TensorFlow should use GPU for all its operations the models were trained and hence the models were trained on GPU. Necessary changes in the code were done to run on GPU-enabled systems.

Results:

1. **Mean Chrominance Model:**
   For the mean Chrominance Model, after training a validation loss of 0.000182 was achieved.

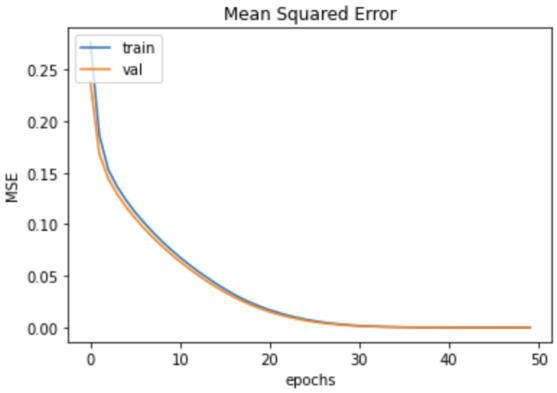


FIG 5: Model Loss while training

```
1/1 [==============================] - 0s 211ms/step
136.30896  ---  141.38735800981522
138.71051  ---  138.89636367559433
1/1 [==============================] - 0s 17ms/step
140.15686  ---  141.21843427419662
138.09473  ---  138.83258789777756
1/1 [==============================] - 0s 15ms/step
140.6933  ---  140.46058773994446
136.65057  ---  138.5464784502983
1/1 [==============================] - 0s 21ms/step
140.27051  ---  142.76061952114105
138.36157  ---  139.41482305526733
1/1 [==============================] - 0s 18ms/step
139.85138  ---  137.8762400150299
138.86694  ---  137.5707972049713
1/1 [==============================] - 0s 17ms/step
143.90881  ---  140.34701943397522
141.67438  ---  138.5036015510559
1/1 [==============================] - 0s 16ms/step
139.01947  ---  139.33505773544312
135.00116  ---  138.12155485153198
1/1 [==============================] - 0s 17ms/step
140.02875  ---  141.37150526046753
143.6737  ---  138.89039039611816
1/1 [==============================] - 0s 17ms/step
141.30145  ---  142.30979651212692
143.72296  ---  139.24462258815765
1/1 [==============================] - 0s 25ms/step
140.37384  ---  141.04736745357513
144.38916  ---  138.76800656318665
```

FIG 6: Prediction of 10 test images using our model in below format
a_real — a_pred
b_real — b_pred

2. **Colorizer Model (ReLU):**
   For the Colorizer model with ReLU,after training for 30 epochs model achieved a validation loss of 6.972 x 10^{-5}.

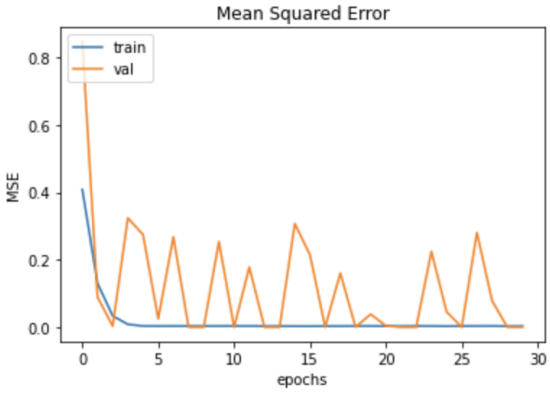


FIG 7: Model Loss while training

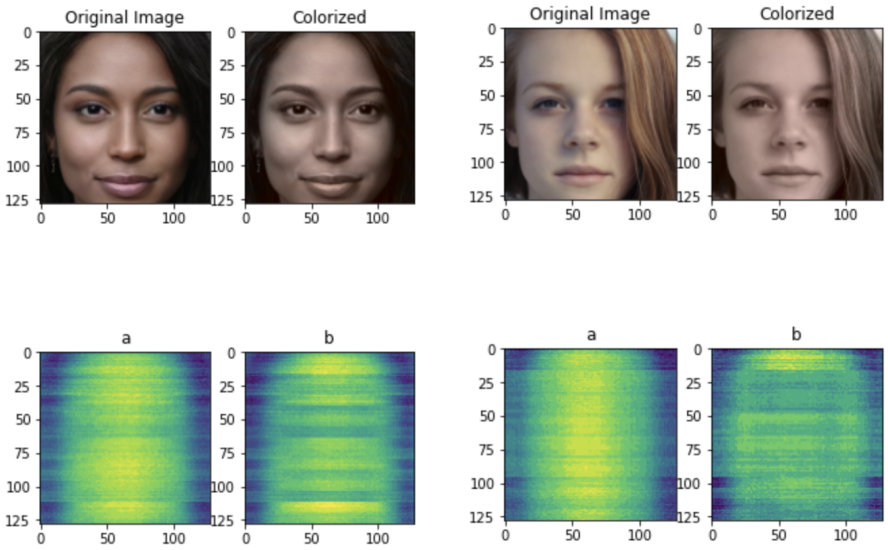


FIG 8: Model Prediction on real Images (Not test images)

As it can be seen in the prediction, the model has learned the chrominance of the image to Good extent and the prediction is close to the original image.

3. **Colorizer Model (TanH):**
   For the Colorizer model with ReLU,after training for 50 epochs model achieved a validation loss of 0.00847.

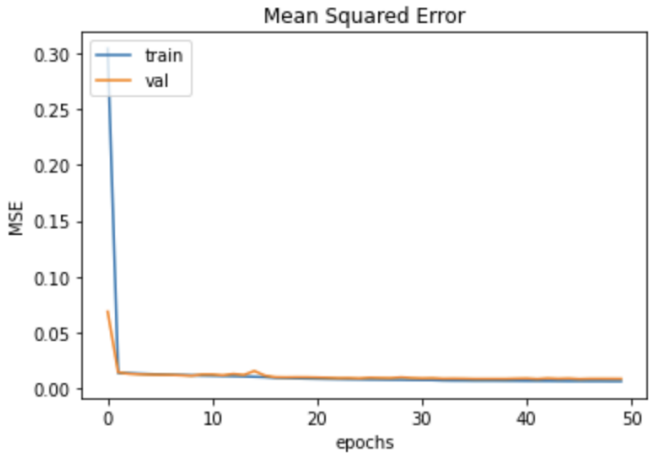


FIG 9: Model Loss while training

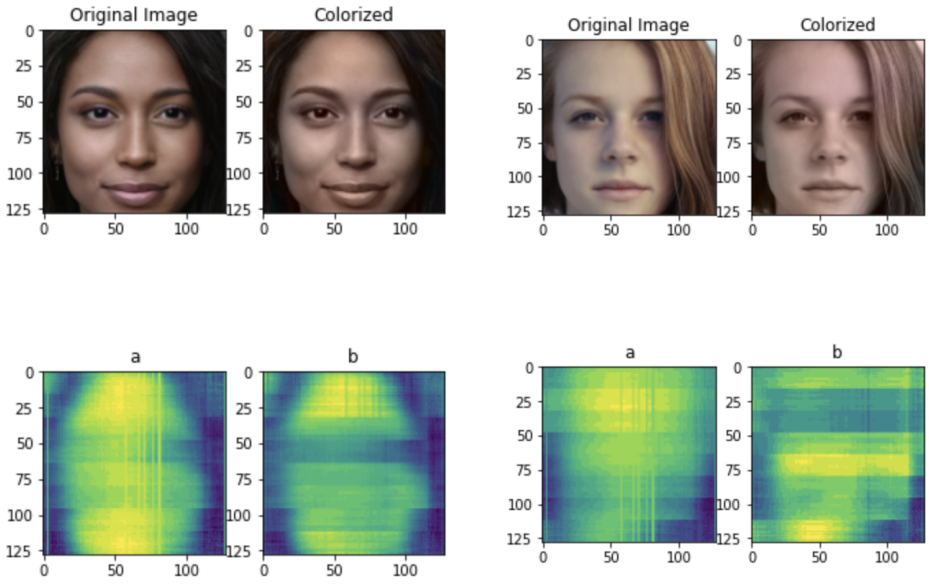


FIG 10: Model Prediction on real Images (Not test images)

As shown in the prediction, the Tanh model learns the face structure and the chrominance according to the structure better than the ReLU model. The A channel and the B channel plotted below the images show when the chrominance value is high and where it is low showing the intensity of the corresponding colors at that pixel.

4. **Transfer Learning Colorizer Model(ReLU):**
   After Transfer Learning on the pre-trained ReLU model, the model achieved a validation loss of 0.0032 on the NCD Dataset.

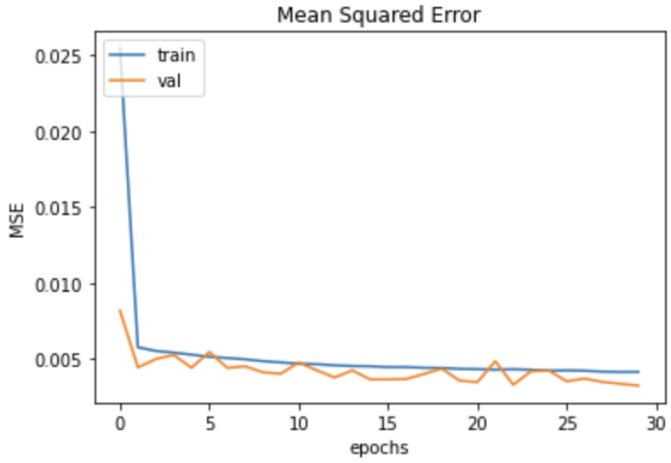
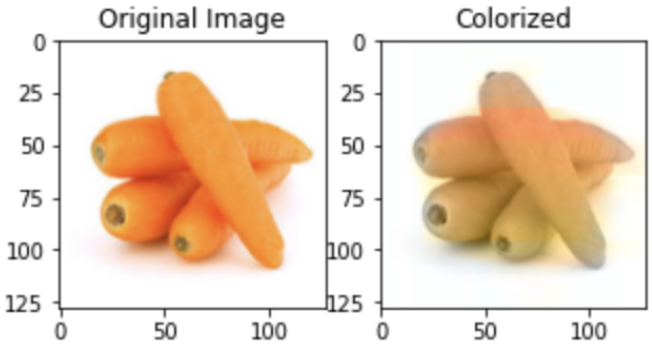


FIG 11: Model Loss while training



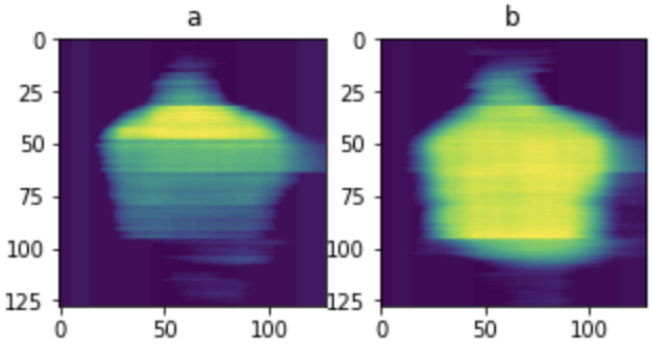


FIG 12: Model prediction on test images.

5. **Transfer Learning Colorizer Model(Tanh):**
   After transfer learning , the pre-trained TanH model achieved a validation loss of 0.028.

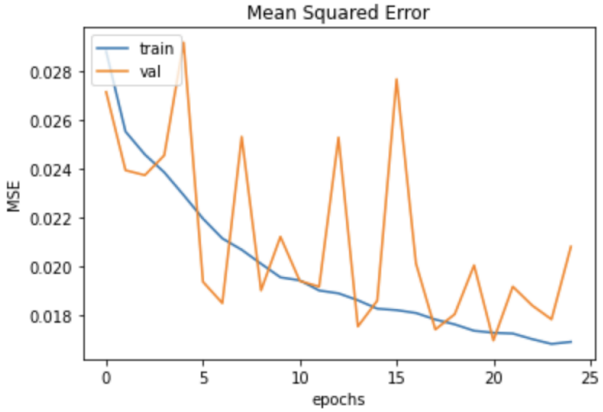


FIG 13: Model loss while training

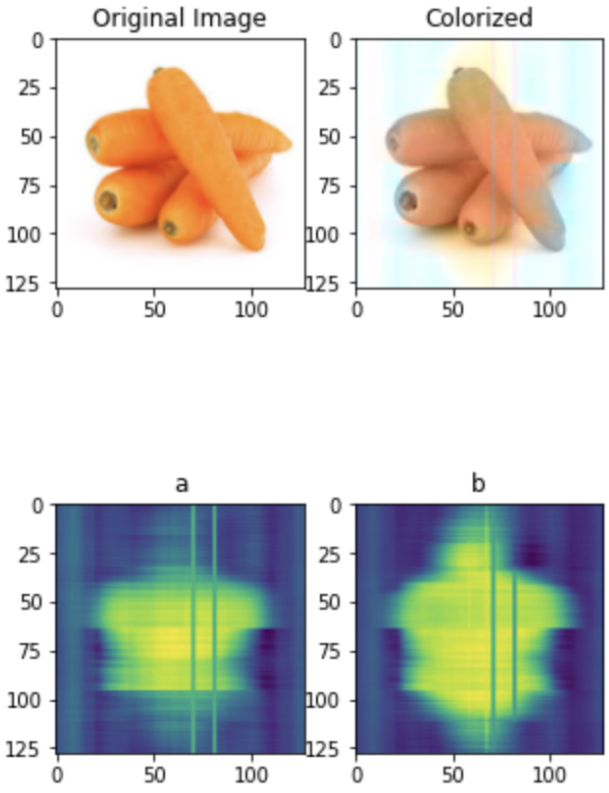


FIG 14: Model Prediction on a test image

As per the model training results, TanH model performs the best and does a really good job in predicting image chrominance based on the structure of the image. As we can see in the later results transfer learning model was able learn the natural color of a object in the image.

Evaluation and Analysis:

Qualitative Evaluation: Model evaluation by comparing prediction on images.

1. **Model (ReLU)**

FIG 15: Model Predictions on Faces and NCD Datasets

2. Model (TanH)

FIG 16: Model Predictions on Faces and NCD Datasets

3. Transfer Learning Model (ReLU)

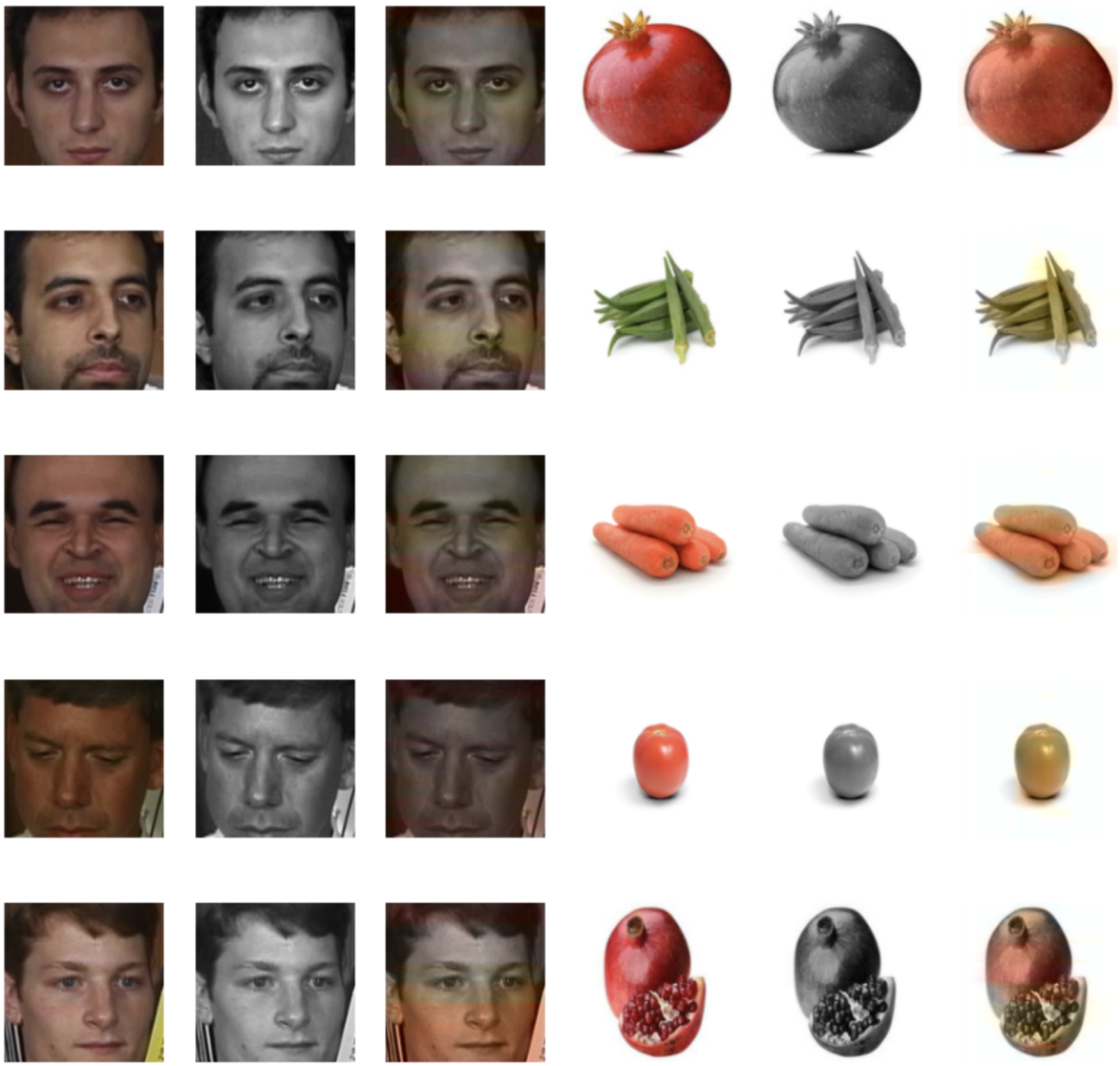


FIG 17: Model Predictions on Faces and NCD Datasets

4. Transfer Learning Model (TanH)

FIG 18: Model Predictions on Faces and NCD Datasets

After Evaluation, the TanH model proves to be the best in coloring the face images. It predicts nearly same as the real image. The ReLU model on the other hand performs the best in predicting the natural color of the images. It accurately predicts the color of the carrots or oranges or other natural objects. It also well to a great extent in colorizing the face images.

Quantitative Evaluation:

The metrics used for qualitative evaluations are:

1. **Peak Signal-To-Noise Ratio (PSNR):**

   The peak signal-to-noise ratio between two pictures, measured in decibels, is computed by the PSNR block. This ratio is used to compare the original and compressed images' quality. The quality of the compressed or rebuilt image improves with increasing PSNR. Image compression quality is compared using the peak signal-to-noise ratio (PSNR) and mean-square error (MSE). The PSNR gives a measure of the peak error, whereas the MSE represents the cumulative squared error between the original and compressed picture. The error is inversely correlated with the MSE value.

$$MSE = \frac{\sum_{M,N} [I_1(m,n) - I_2(m,n)]^2}{M * N}$$

$$PSNR = 10 \log_{10}\left(\frac{R^2}{MSE}\right)$$

   Here, R is the maximum value of the pixel. If its a 8-bit unsigned integer, R is 255.

2. **Structural Similarity Index Measure (SSIM):**

   A method for forecasting the perceived quality of digital television and film images, as well as other types of digital images and videos, is the structural similarity index measure (SSIM). SSIM is a tool for calculating how similar two photos are to one another. The SSIM is a perception-based model that takes into account how structural information is seen to have changed while also adding significant perceptual phenomena like brightness and contrast masking terms. MSE will calculate the mean square error between each pixels for the two images we are comparing. Whereas SSIM will do the opposite and look for similarities within pixels; i.e. if the pixels in the two images line up and or have similar pixel density values.

$$\mathrm{SSIM}(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

- μ_x the pixel sample mean of x;
- μ_y the pixel sample mean of y;
- σ_x^2 the variance of x;
- σ_y^2 the variance of y;
- σ_{xy} the covariance of x and y;
- $c_1 = (k_1 L)^2$, $c_2 = (k_2 L)^2$ two variables to stabilize the division with weak denominator;
- L the dynamic range of the pixel-values (typically this is $2^{\#bits\ per\ pixel} - 1$);
- $k_1 = 0.01$ and $k_2 = 0.03$ by default.

	Georgia Tech Faces Dataset		NCD Dataset	
Models	**PSNR**	**SSIM**	**PSNR**	**SSIM**
Colorizer Model (ReLU)	30.055	0.973	17.26	0.768
Colorizer Model (TanH)	33.878	0.977	17.488	0.831
Transfer Learning Model (ReLU)	24.099	0.921	20.9	0.847
Transfer Learning Model (TanH)	21.277	0.836	19.392	0.789

TABLE 1: PSNR and SSIM values of predictions

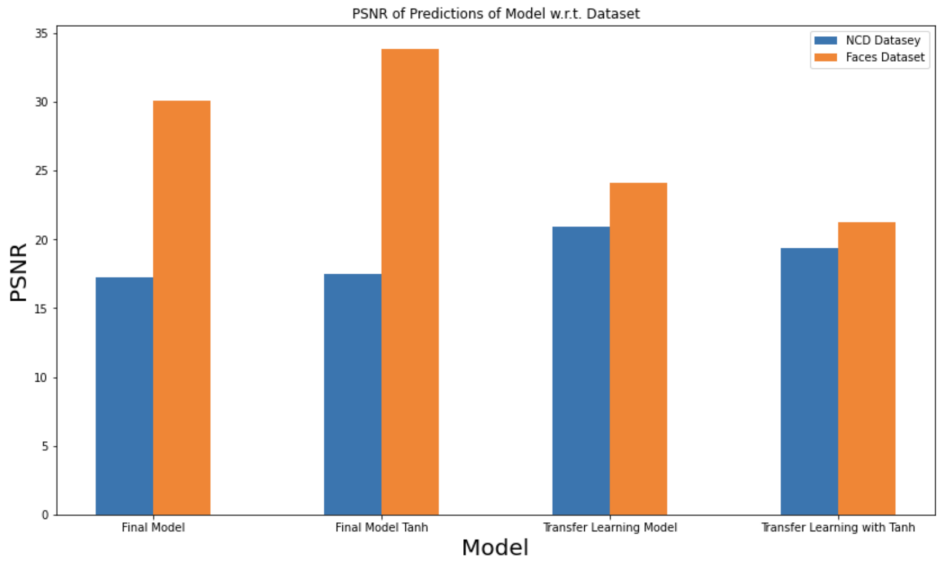


FIG 19: PSNR Visualization

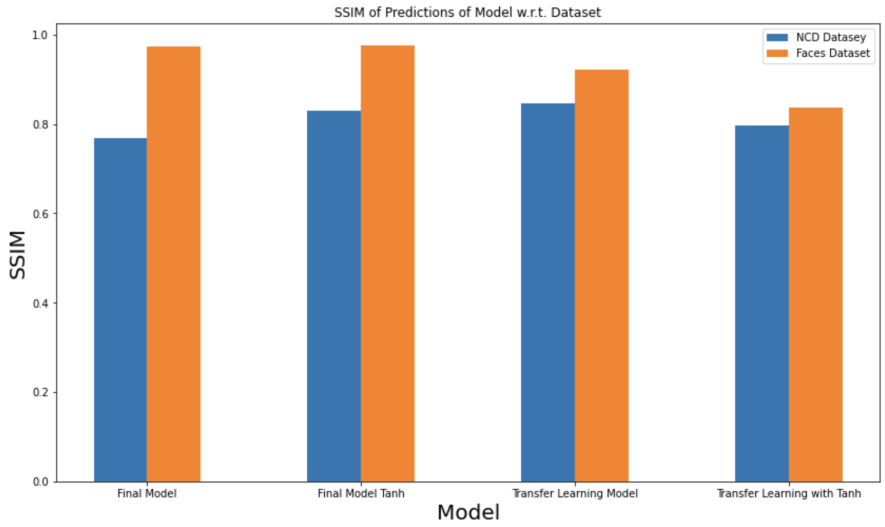


FIG 20: SSIM Visualization

After quantitative analysis, it was observed that on the Faces dataset we get the highest PSNR with the Colorizer model and the highest SSIM. Hence, similar to qualitative evaluation, this model performs the best in coloring face images. And transfer learning ReLU model performs the best in coloring natural datasets as it has the highest PSNR and SSIM.

Instructions to run the code:

1. Start Jupyter Server.
2. Make sure these Libraries are installed:
    a. tensortlow-gpu / tensorflow
    b. opencv-python
    c. numpy
    d. Pandas
    e. Matplotlib
3. Open the Jupyter Notebook that has to be run.
4. Run each cell in the order it is displayed.
5. Wait for the training to complete.
6. For colorizing an image run this in the terminal

```
python colorize.py [model path] [image path][modelno]
```

   Here, the model Number is
   0 for ReLU model
   1 for Tanh colorizer model
   2 for Tanh Transfer Learning model
7. The result will be saved as “Result.jpg”

Conclusion:

In conclusion, this project was able to achieve the goal of creating an image colorizer model. The project has successfully, applied transfer learning to the model trained on Georgia Tech faces dataset to achieve natural colorization of images using the NCD Dataset. The colorizer model with the activation function as TanH in the last two spatial convolutional layers performed the best in colorizer human faces. The model was able to achieve near-perfect similar colorization as the original image. On the contrary, after transfer learning the pre-trained colorizer model with the ReLU function as an activation function for each spatial convolutional layer perform the best in colorizing the natural images. This has been aptly concluded from the quantitative and qualitative evaluation of the models.
The problem with image colorization on any image, in general, is that for a black-and-white image, without any context just based on the luminance value of a pixel, predicting the chrominance of the pixel is challenging because in a black-and-white image of more than two colors may have the same luminance. So, achieving the goal of creating a general image colorizer is a challenging task on the contrary predicting the luminance value of similar images is possible since they have a specific pattern in their chrominance value. This has been displayed in the project.
The project also displayed the concept of transfer learning very concisely and clearly with accurate evaluation techniques.

References:

Complete image augmentation in opencv | towards data science. (n.d.). Retrieved November 1, 2022, from https://towardsdatascience.com/complete-image-augmentation-in-opencv-31a6b02694f5