Part_II.Deep Colorization with CNNs

Shivam Basia - (7838-4123)
Aakansh Togani - (6005-9201)
Pulin Soni - (7303-3793)

Implementation flowchart:

How to run the code

- Make sure the ‘face_images’ file is in the same directory as the main.ipynb file.
- Run the main.ipynb file with kernel ‘NGC-PyTorch 1.9’
- The code will check for availability of GPU and select the appropriate device for computation.
- Code works for both devices namely CPU and GPU.

Explanation of the code :

Functions:

def load_data () : We use the glob library to extract all image paths in a list variable. We further split the training and testing data. Establish different paths for training and testing.

def train_test_split () : In this function we have divided the data into appropriate train and test datasets. As requested the testing dataset consists of 10% of the total number of images in the dataset provided and training dataset consists of the remaining 90%. We have also created the directories that are required to store the datasets in an organised manner.

def build_dataset () : This is the function where we augment the dataset for better results. At first we transform all training images with respect to size, horizontal flips and crop all randomized. For each activation function we append the training images selected plus their transform into one variable. Using the Inbuilt DataLoader function we shuffle our dataset and create batches. This function then returns the augmented batch defined dataset for training as well as for testing.

execute_colorizer_sigmoid (): It executes the colorizer model with the activation function ‘sigmoid’. We create an object of class Colorizer_Manager and train this model with the augmented dataset made available. We also test this model using the function from the class Colorizer_Manager. The regressor model is called to be trained and tested here.

execute_colorizer_tanh (): It executes the colorizer model with activation function ‘tanh’. We create an object of class Colorizer_Manager and train this model with the augmented dataset made available. We also test this model using the function from the class Colorizer_Manager. The regressor model is called to be trained and tested here.

train_regressor (): This is where the Hyperparameters have been defined. Hyperparameters such as epochs, learning rate, weight decay, Number of Input channels, Number of hidden layers and the number of output. Furthermore we initialize a model of regressor class with all mentioned hyperparameters. We initialize the loss function and the optimizer. We start the training with respect to initialize batches. We use appropriate functions to get mean values for a and b inputs. We run this until we reach a defined epoch value. Later this model is saved in repositories for future usage.

test_regressor (): Similarly for the test we initialize all the above mentioned hyperparameters. We then load the previously saved model. We calculate the MSE for the test dataset. We store the predicted value of Mean a and Mean b for respected images.

get_device (): This function is created to check the presence of cuda while running our project. We assign the number of workers if cuda is present and return the status of cuda, the number of workers and the device as the output.

get_ab_mean (): This function is used to calculate the mean from both the a (red to green axis) & b (blue to yellow axis) channels.

plot_loss_epoch (): This function plots a graph keeping the X axis as the Epoch number plotted against the Loss(y-axis) during that epoch.

Classes :

AugmentImageDataset:

The images in the dataset have been passed to this class. Here the images are converted into image 'Lab' format from 'rgb' format. For each image its chrominance values 'a' and 'b' and luminance value 'l' have been extracted.

Regressor:

This class implements 'neural network'. The constructor initializes the defined hyperparameters for the object model. Here we initialize the steps for the neural network. We start by convolving the image with a convolution window of size 4 (kernal_size = 4), followed by batch normalization and passing the inputs through the activation function specified. Then these outputs act as inputs for the next layer.
At last we use the linear function to convert the 512 generated output through the convolutional neural network to desired Output dimension here 2 namely a_mean and b_mean.
The regressor class is responsible for downsampling the image

Colorizer

This class is similar to the regressor class, the regressor class neural network outputs 512 at its final step. Using sequential keyword, we continue the made neural network in the colorizer and keep on decreasing the output until initial size is reached.
The colorizer is responsible for upsampling the image.

Manage_colorize

This class is used to define the function train() which is used to load the training dataset and hyperparameters such as the epoch, learning rate, weight decay, hidden channels are declared.
An object model is defined of the colorizer class and an optimizer is also used.
The loss at every epoch during training is posted.
The function test() is also defined in this class where the saved model is given the
Test dataset to provide us with the losses. The hyperparameters are also declared.

Extra Credit - tanh as activation function

We used 'Sigmoid' at the final layer of the colorizer for normal credit and 'tanh' at the final layer of the colorizer for extra credit. The outputs have been stored at output_sigmoid and output_tanh respectively.

Diagram for CNN similar to what has been used for Colorization:

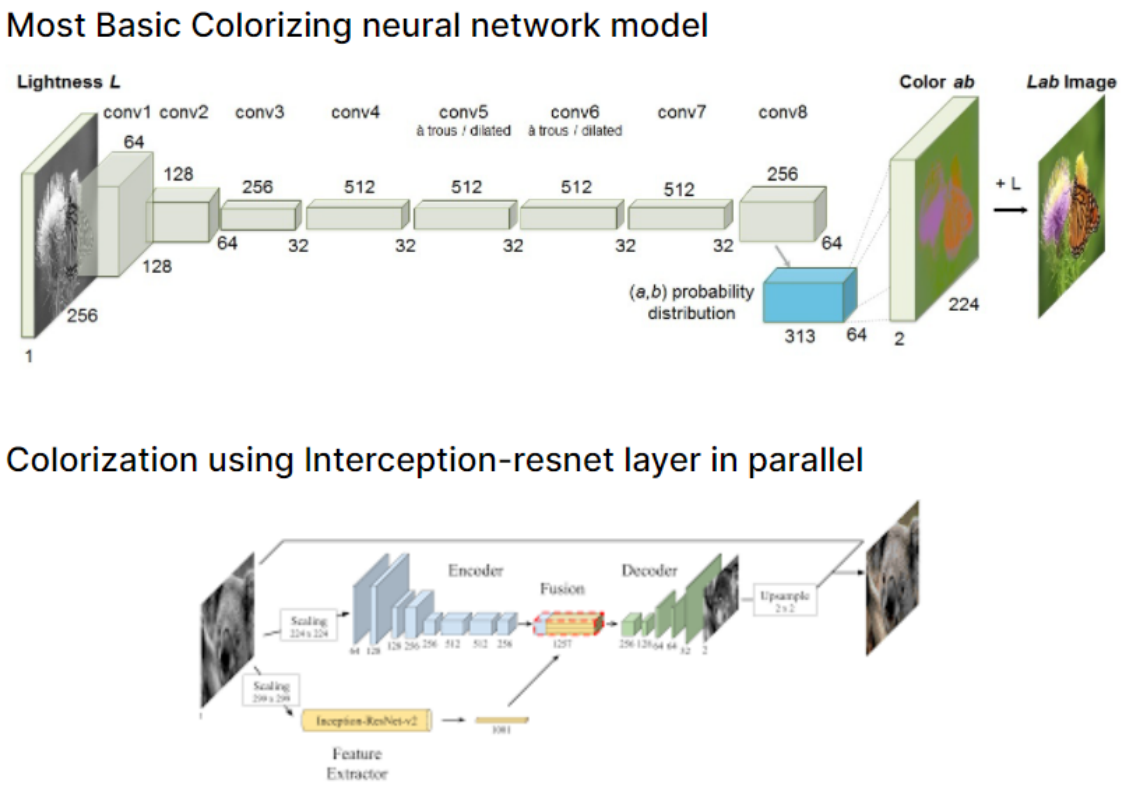


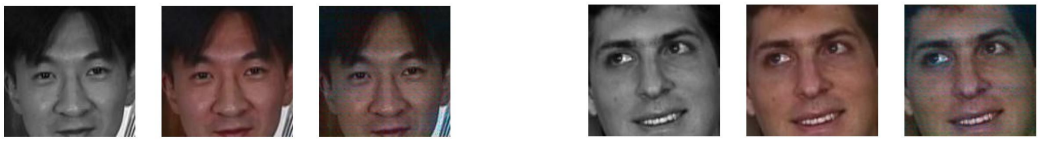
Figure 1: Neural Network Model

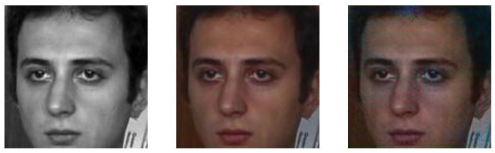
Evaluation Results:

Images Generated Using CPU (100 Epochs)(Optimal Hyperparameters used)

Sigmoid: (Grayscale : Original : Generated)

Tanh: (Grayscale : Original : Generated)





Plots

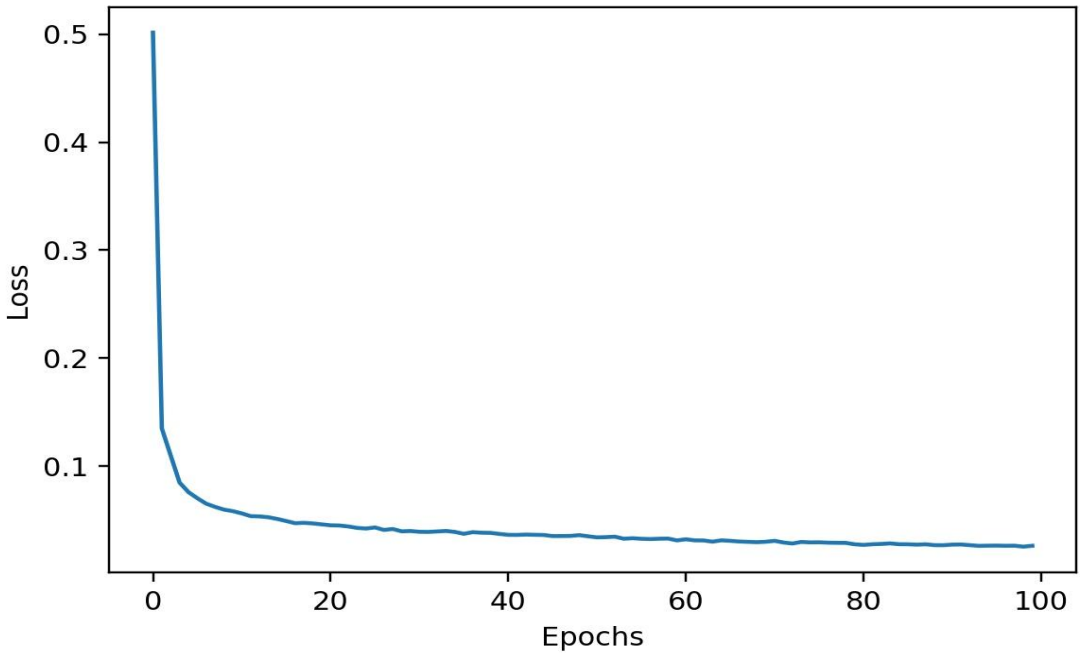


Figure 2: Colorizer loss plot (Sigmoid)

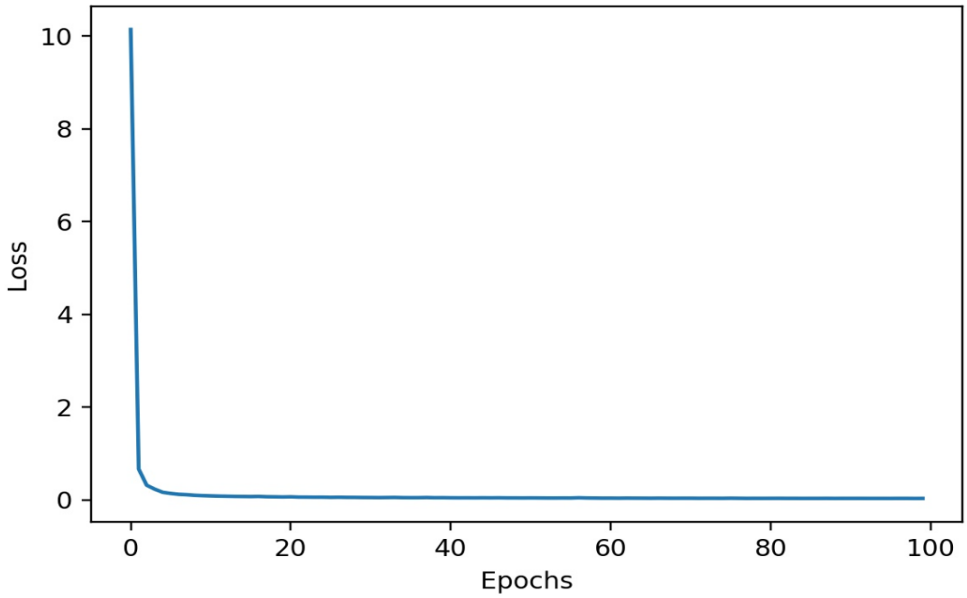
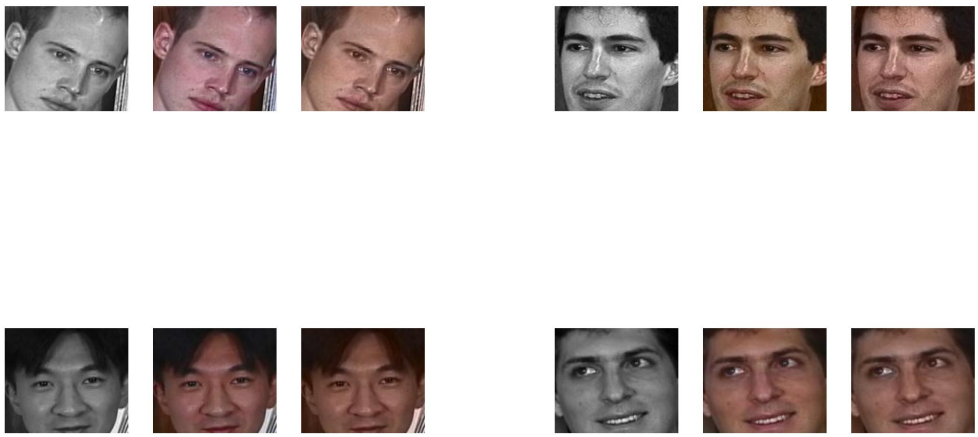
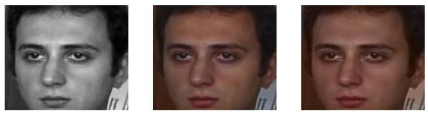


Figure 3: Colorizer loss plot (Tanh)

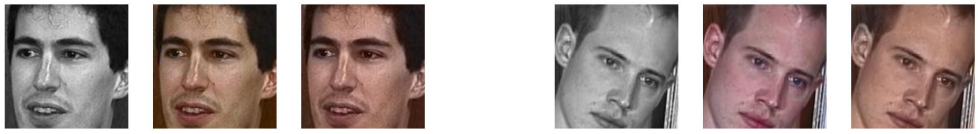
Images Generated Using Cuda. (400 Epochs)(Optimal Hyperparameters used)

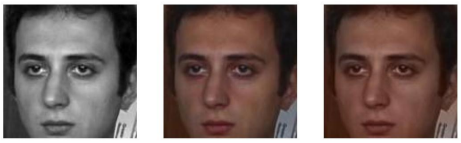
Sigmoid: (Grayscale : Original : Generated)





Tanh: (Grayscale : Original : Generated)





Plots

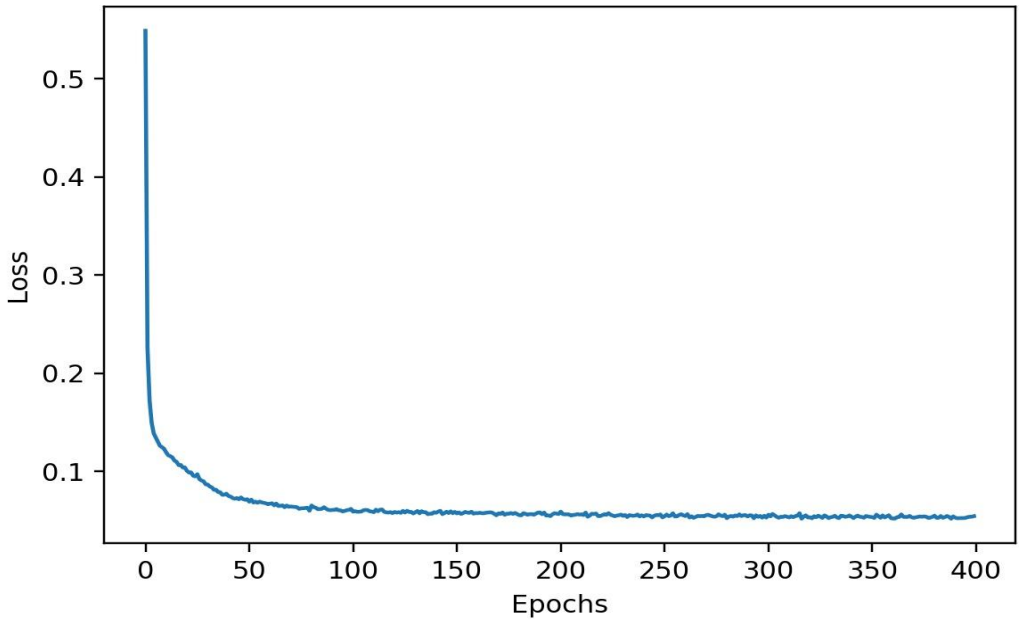


Figure 4 : Colorizer Sigmoid Epochs:Loss(Cuda)

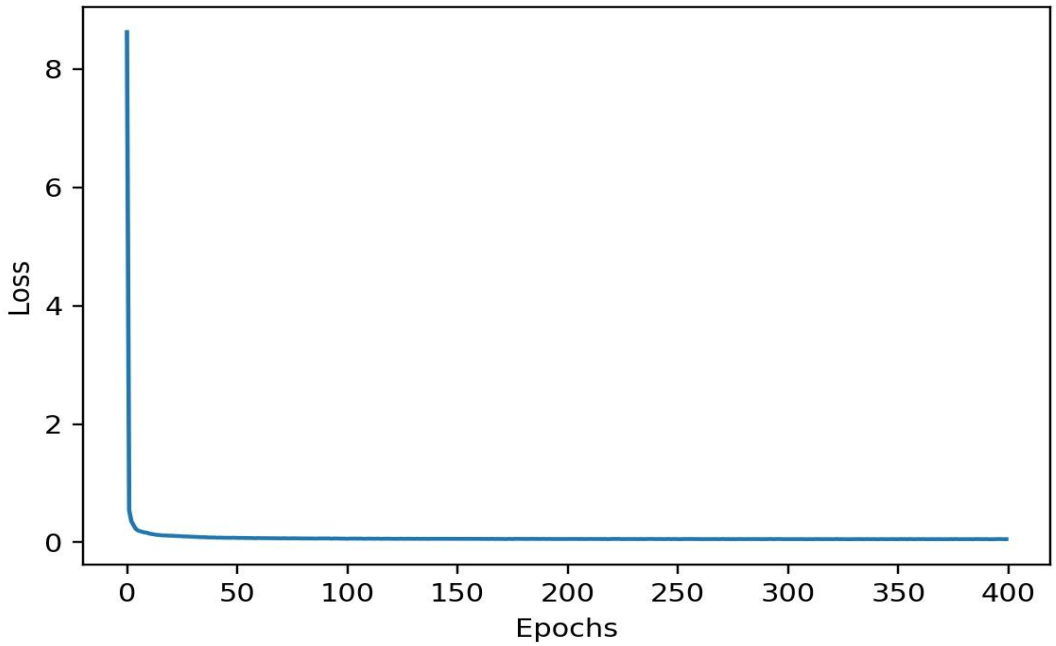


Figure 5: Colorizer Tanh Epochs:Loss(Cuda)

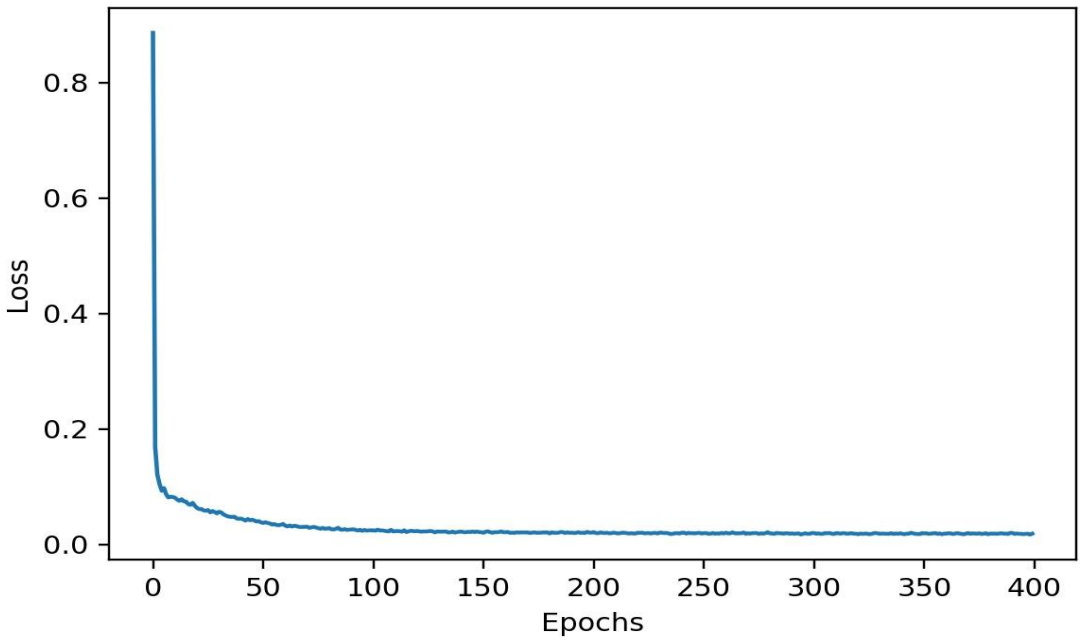


Figure 6: Regressor Epochs:Loss (Cuda)

Extra Credit-Changing no. of feature maps

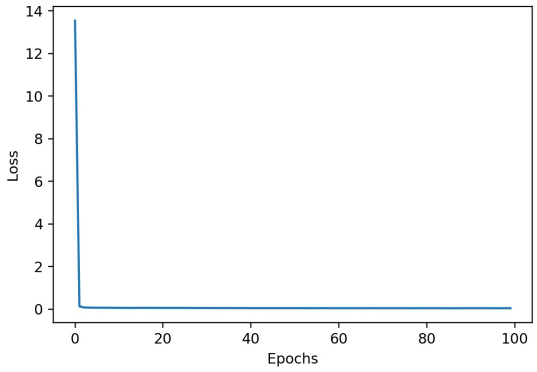
Layer	Type	Depth	Kernel	Stride	Padding
1	ConvoTranspose2D	32	4*4	2	1
2	BatchNorm	32	-	-	-
3	ConvoTranspose2D	64	4*4	2	1
4	BatchNorm	64	-	-	-
5	ConvoTranspose2D	128	4*4	2	1
6	BatchNorm	128	-	-	-
7	ConvoTranspose2D	256	4*4	2	1
8	BatchNorm	256	-	-	-
9	ConvoTranspose2D	256	4*4	2	1
10	BatchNorm	256	-	-	-
11	ConvoTranspose2D	512	4*4	2	1
12	BatchNorm	512	-	-	-

Architecture 1

Hyperparameters	
Hyperparameter Case 1	Hyperparameter Case 2
epochs = 100	epochs = 400
lr = 0.005	lr = 0.0001
weight_decay = 1e-7	weight_decay = 1e-5

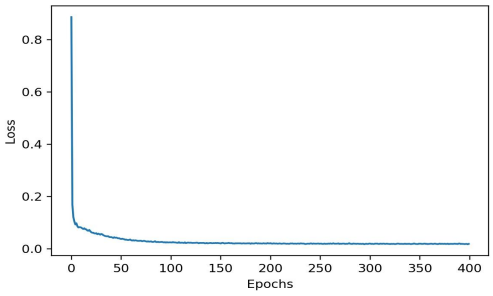
Hyperparameters for Architecture 1

MSE for Architecture 1(with two hyperparameter cases):




```
..Regressor testing started..
MSE: 0.00015243241108315057
Image_num || Mean a || Mean b
```

Architecture 1 Case 1




```
..Regressor testing started..
MSE: 0.00011346710870216874
Image_num || Mean a || Mean b
```

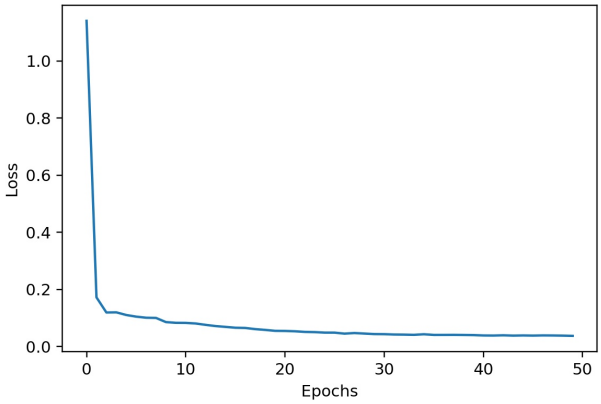
Architecture 1 Case 2

Layer	Type	Depth	Kernel	Stride	Padding
1	ConvoTranspose2D	8	4*4	2	1
2	BatchNorm	8	-	-	-
3	ConvoTranspose2D	16	4*4	2	1
4	BatchNorm	16	-	-	-
5	ConvoTranspose2D	32	4*4	2	1
6	BatchNorm	32	-	-	-
7	ConvoTranspose2D	64	4*4	2	1
8	BatchNorm	64	-	-	-
9	ConvoTranspose2D	128	4*4	2	1
10	BatchNorm	128	-	-	-
11	ConvoTranspose2D	256	4*4	2	1
12	BatchNorm	256	-	-	-

Architecture 2

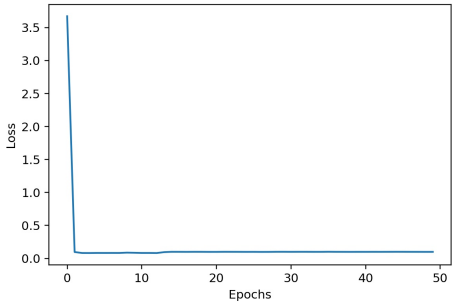
Hyperparameters	
Hyperparameter Case 1	Hyperparameter Case 2
epochs =50	epochs =50
lr = 0.005	lr = 0.005
weight_decay = 1e-7	weight_decay = 1e-7

Hyperparameters for Architecture 2




```
..Regressor testing started..
MSE: 0.0001554764270245438
Image_num || Mean a || Mean b
```

Architecture 2 Case 1




```
..Regressor testing started..
MSE: 0.00021356041729123527
Image_num || Mean a || Mean b
```

Architecture 2 Case 2

Therefore,the optimal hyperparameter and architecture was of Architecture 1 Case 2

Extra Credit-Colorization using Classification vs Regression

Colorization in Regression is mostly done and optimized by euclidean distance loss function thus,it generally remains unaffected by unsaturated pixel bias however,it is the same reason that the generated colorization could not upscale to a very higher value compared to its neighbouring pixels. However,in case of Classification since the predicted distribution is mapped to the point estimates it has a good chance of upscaling to a very higher or lower value compared to its neighbouring pixels which can help in predicting images with very high contrast however for the same reason there occurs a chance of bias created by more number of unsaturated pixels.

Thus,Classification can produce better results than regression in the following scenarios:

(i)If the classifier is trained on a very large dataset allowing it to produce better data augmentation results.
(ii)Weight class rebalancing is used to eliminate the bias generated by more unsaturated pixels.

So,in all other conditions than the above,regression will perform better than classification.