I am a painter myself, using GAN

This project is about imitating the artwork of Claude Monet using Generative Adversarial Networks. Before starting any model implementation we would first like to analyze the dataset.

Data Exploration

We have two folders in our datasets. One contains the images, and the other folder contains the paintings of Claude Monet.

To explore the images and paintings in the data we displayed 5 images from each and here are the paintings.

Here are the images

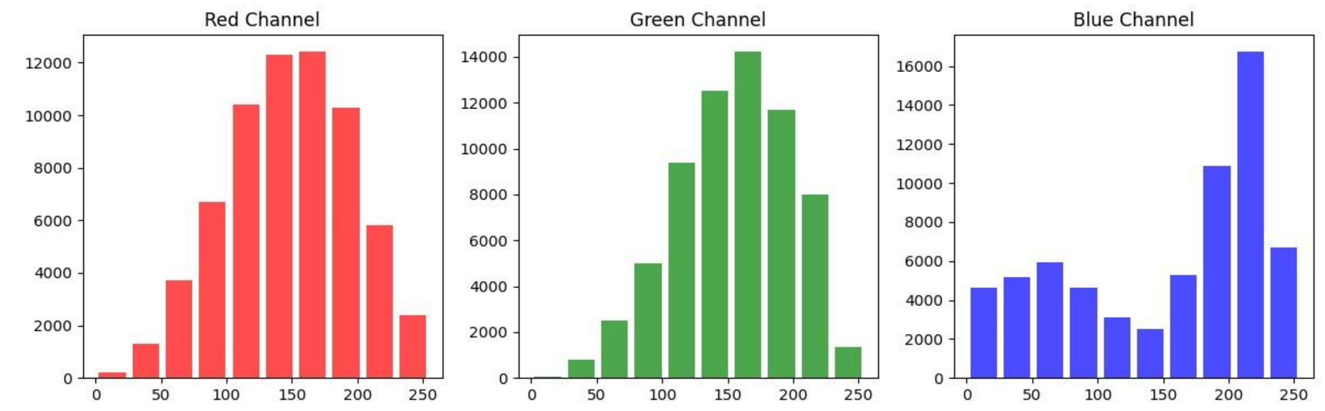
Train/Test Data Distribution

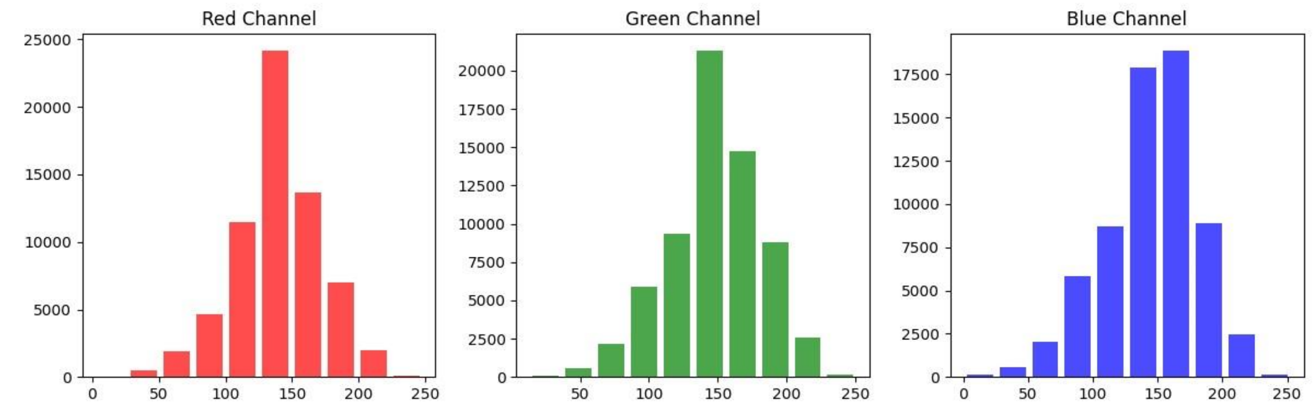
There are total of 7038 images in the data and 300 paintings s in the data.

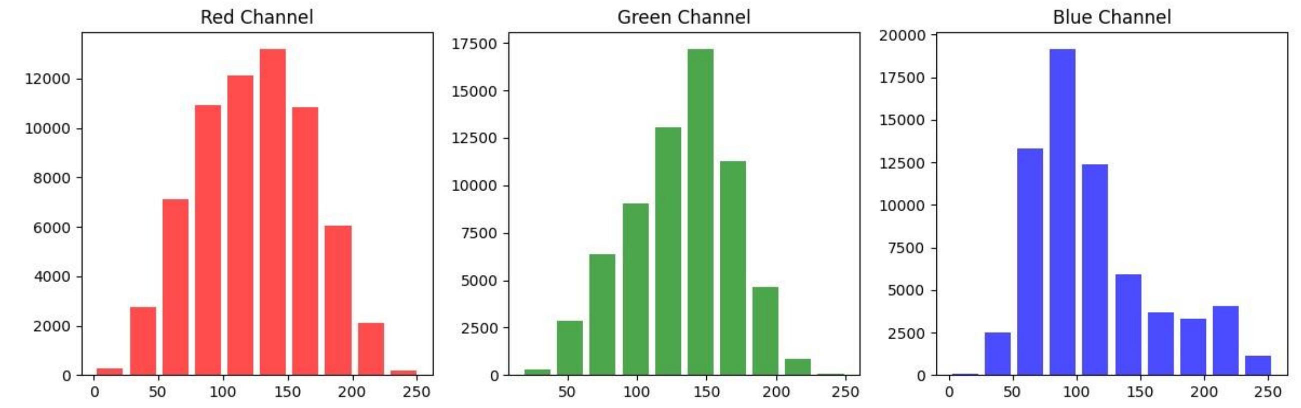
Exploratory Data Analysis

Now we will draw some histograms to analyze color distribution of test and training dataset

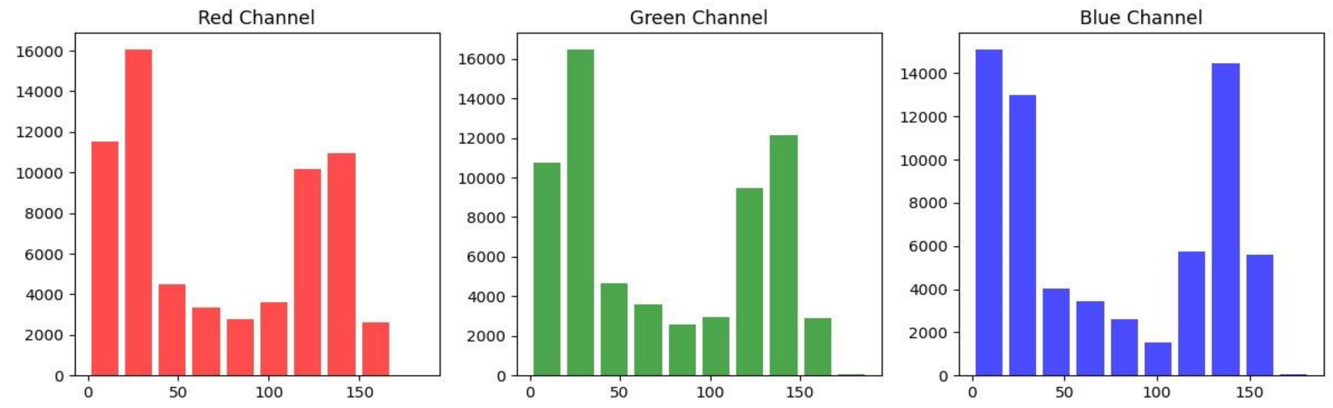
Color distribution for paintings

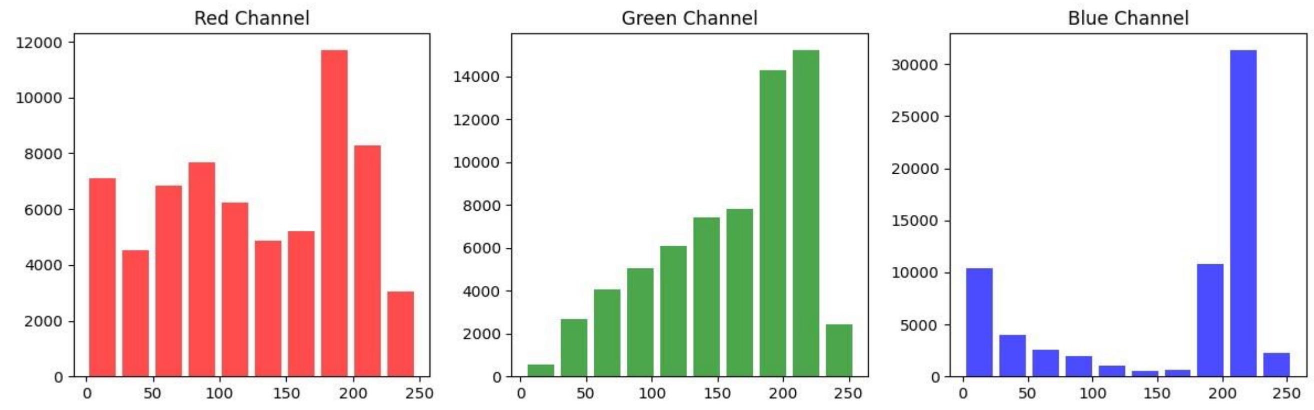


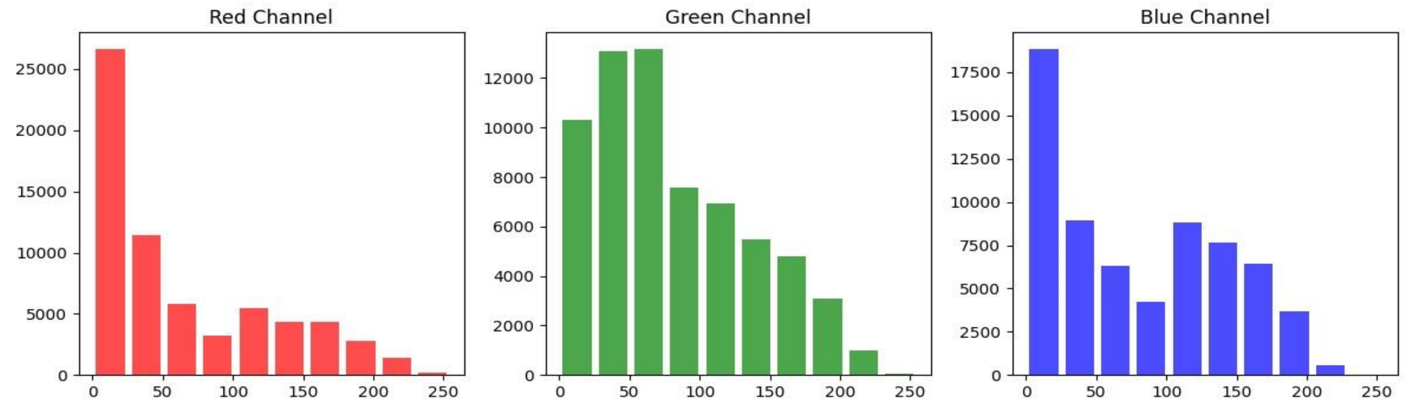




Color distribution for pictures:







As we can see that in case of painting, the color distribution looks like normal distribution in most of the cases and the color values mostly lie between 50 and 200. While in the case of the paintings the distribution is not normal and most of the values lies towards extreme i.e. less than 50 or greater than 150.

Generative Adversarial Networks

Generative Adversarial Networks (GANs) are state of the art machine learning models. GANs consist of two neural networks, the Generator and the Discriminator used in adversarial training process. Here is a short description about them.

Generator:

The generator model Transforms random noise into synthetic data. The purpose of this model is to generate data(images) like real data.

Discriminator:

This model tries to discriminate real data from fake data. It tries to classify which images are real and which are generated by generator model.

Adversarial Training:

This training process works in iteration where generator model generated the fake data, and the discriminator model tries to discriminate the real data from fake data. Generator data uses the feedback from the discriminator and tries to improve the quality of fake data, so the discriminator is not able to detect the difference between real and fake data.

My GAN Architecture:

In this architecture I built the custom generator and discriminator networks.

The generator comprises an encoder-decoder structure, each consisting of multiple convolutional (encoder) or transpose convolutional (decoder) layers.

The discriminator network consists of multiple convolution layers, and it works as a classifier to classify images as real or fake.

Custom Generator:

Here are the details of the encoder and decoder of generator network.

Encoder:

The encoder takes as input an initial RGB image (3 channels) and progressively reduces its spatial dimensions while increasing the number of channels. The first convolutional layer with a kernel size of 3, stride of 1, and padding of 1 initiates the encoding process, extracting low-level features. Subsequent convolutional layers increase the complexity of features, capturing more abstract representations.

All the layers in the encoder are 2D convolution layers. Here are the details of all the layers in the encoder

	Input Size	Out size	Kernel size	Stride	Padding	Activation
Layer1	3	32	3	1	1	relu
Layer2	32	64	3	1	1	relu
Layer3	64	128	3	1	1	relu
Layer4	128	64	3	1	1	relu

Decoder:

The decoder, symmetric to the encoder, takes the encoded feature representation and reconstructs it into a full-sized RGB image. Convolutional transpose layers are used to upsample the feature maps, gradually restoring spatial dimensions. ReLU activation functions after each transpose convolutional layer ensure non-linearity in the decoding process. The final layer in the decoder produces the synthesized RGB image.

	Input Size	Out size	Kernel size	Stride	Padding	Activation
Layer1	64	128	3	1	1	relu
Layer2	128	64	3	1	1	relu
Layer3	64	32	3	1	1	relu
Layer4	32	3	3	1	1	relu

Custom Discriminator:

The CustomDiscriminator functions as the discriminative part of the GAN, responsible for distinguishing between real and generated images. Its architecture consists of a series of convolutional layers, terminating in a final layer with a kernel size of 1, effectively producing a single-channel output.

The discriminator's initial convolutional layers operate similarly to those in the generator's encoder, capturing hierarchical features in the input image. ReLU activation functions are applied after each convolutional layer to introduce non-linearity. The final convolutional layer with a kernel size of 1 condenses the feature representation into a single-channel output. The discriminator is trained to produce high values for real images and low values for generated images.

	Input Size	Out size	Kernel size	Stride	Padding	Activation
Layer1	3	32	3	1	1	relu
Layer2	32	64	3	1	1	relu
Layer3	64	128	3	1	1	relu
Layer4	128	64	3	1	1	relu
Layer5	64	3	1	1	1	-

Loss functions:

There were two kinds of loss function used in GAN and different learning rates are also used. Here are the details about loss function.

Binary Cross Entropy with Logits

Binary Cross Entropy with Logits (BCEWithLogitsLoss) is a commonly employed loss function in traditional GAN architectures. This loss function is designed to measure the dissimilarity between the predicted logits and the true labels. Specifically, for the GAN model, it quantifies the difference between the discriminator's predictions for real and generated samples and their respective actual labels.

It is suitable for binary classification problems, such as the task of discriminating between real and fake images.

Wasserstein GAN (WGAN) Loss

We also tried Wasserstein GAN (WGAN) Loss in our training process. The decision to use the WGAN loss is motivated by its benefits in terms of training stability and its impact on the quality of generated images.

The WGAN loss is defined as the difference between the mean logits of fake and real samples. This loss function encourages the generator to produce images that are not only realistic but also in a distribution like the real data.

Hyper-Parameter Tunning

We used different parameters to train our model, here are the details of the hyper-parameters.

Loss function	**Learning rate**
BCEwithlogits	0.0001
BCEwithlogits	0.001
Wasserstein GAN (WGAN) Loss	0.0001
Wasserstein GAN (WGAN) Loss	0.001

Based on the performance we selected BCEwithlogits with a learning rate of 0.0001.

Conclusion:

In summary, using Generative Adversarial Networks (GANs) to recreate paintings from pictures is a cool mix of art and tech. GANs are good at copying an artist's style, making pictures that look like the real deal. While it's not perfect, GANs let artists try out new ideas and styles. We need to think about things like who owns the art and what's real. GANs keep getting better because we keep working on them. They're helping art and technology come together in a new way.