CS 6220 Big Data Sys & Analytics Project Report
Colorization and Style Transfer of Images

Group 15 : Kalyani Prasanna Jagdale, Karthik Nama Anil, Prajwal Mavinkere Revanna, Sakshi Mittal, Vandana Ramesh

1. Introduction

Early forms of media and visual arts were Black and White. Gradually over time, as technology improved, it transitioned to color. Some current forms of visual art, such as fine art photography and art films still use Black and White (widely known as Monochrome) images. Many portrait photographers use monochrome because people perceive more depth and details in a monochrome photo over a color photo. Enthusiasts tried to find old black and white photos and motion pictures and used novel image processing techniques to create the colored version of the images and videos. With the introduction of Neural Network models, the calculation involved became more accurate and simplified. A model is trained to take a Black and White version of a colored image and to generate the colorized image as the result. The testing involves passing an old Black and White image through the trained model to create the colored version of it. We will work on colorizing images as a part of our first model.

Painting has been one of the most important ways of artistic expression for the longest time. Right from Michelangelo of the 15th century to Banksy of current art space, artists have used painting to capture the viewer's attention. Presently, people are finding ways to introduce the styles and artistic impressions of such noted works into normal photos captured by them to evoke more emotion and context from the photos. The initial trial of this implementation involved incorporating basic image processing techniques such as filters and morphological transforms. But since it was very complex to create a mathematical construct to capture the style or effect of the noted works, Neural Network models are being used to learn it. The models were given a stock photo and a photo of the artwork (style) and trained to have the style of the artwork imprinted on the stock photo. We will work on performing style transfer on images as part of our second model.

Thus on the whole, we aim to take a black and white image, generate a colorized image of the same using one neural network, and then use another neural network to stylize the colored image with the style of a famous artwork. The implementation also involves applying the same to GIFs and videos by transforming individual frames and reconstructing the video/GIF using the transformed frames ignoring the glitches caused by not taking inter-frame relationships into consideration.

2. Motivation & Objectives

Humans have grasped the dexterity to make rare visual experiences by bringing together a variety of styles in images. Visual perception is a key area where neural networks have been widely used to create high-level artistic impressions. We developed a system using neural networks to perform colorization as well as style transfer of images. We think it would be interesting to see how a machine can not only create but also perceive and apply changes to images.

Image colorization is a technique to add colors to an image that were originally taken in black and white. The algorithm uses deep learning techniques (such as convolutional neural networks) to analyze the colors across a set of color images, and their black and white versions. The algorithm uses multiple feed-forward passes to take in a grayscale image and produce vibrant realistic colorizations. The objective is to produce colorized images so realistic that for a viewer it is difficult to spot the original image amongst the ground truth image and the image produced by our model. This whole process of colorization is done without any direct human assistance.

Neural style transfer is a machine learning algorithm belonging to the image-to-image translation techniques. It helps us merge the contents of one image with the style of another image to provide a third image which is a blend of the first two. The algorithm combines the two images based on Gram index matching of pre-trained deep features.

3. Related Work

Existing implementations of colorization are based on a color transfer process [1] which use one image's color characteristics from another in any three dimension color space directly. The same idea is also proposed in [2] using a feed forward Artificial Neural Network(ANN). In the recent years, Convolutional Neural Networks (CNNs) [3] gained popularity for learning the generic feature representation and using it independently processing and manipulating the content and the style of natural images. [4] presents a review of Neural Style Transfer (NST), a process of using CNNs to generate an image in different styles.

One of the deep learning approaches in [5] suggests combining low-level cues from user edits with high-level semantic information for user-guided image colorization. [6] address the problem of hallucinating a plausible color version of the photo by evaluating the images, generated using a fully automated approach, using a colorization Turing test. The classification task method [6] has performed 32% better than previous methods when tested by user surveys. We also studied video colourisation and [7] utilizes the concept of conceptual awareness in colorization with the use of LSTM CNNs, which can be explained as colouring objects in the current frames using the knowledge on the colour of same objects in the previous frames.

4. System Architecture

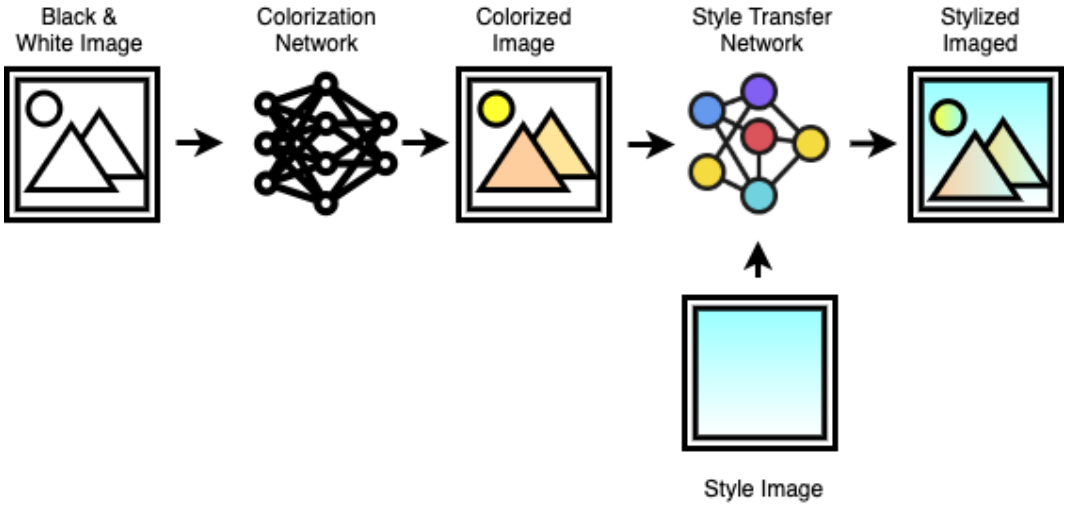


Figure 1: System architecture

We take a grayscale image as our input and pass it to the colorization model to obtain the colorized image output. Then we provide the colorized image along with a style image as an input to the Style transfer model. It generates a colourised style transferred image for a given black and white image input.

5. Colorization

5.1 Dataset

We used the following datasets for our colorization model :

A. CIFAR-10: Contains 60K 32x32 color images in 10 classes with 50K training images and 10k test images. It has 5 training batches and 1 test batch overall.

B. Places365: This contains more than 10M images comprising 400+ unique scene categories. It contains 5K to 30K training images per class.

C. CelebA: This is a large scale face attributes dataset containing over 200K celebrity images, with 40 attribute annotations.

5.2 Colorization Network

Colorization algorithms differ in ways in which they retrieve and process the data for modeling the correspondences between grayscale and color images. The input gray scale images are first mapped to reference colour images (which is also present in the dataset) in case of non-parametric colourisation methods. Parametric methods look at colorization as an optimization problem, prediction problem or classification problem and train on large image datasets to map each pixel of a gray scale image to an individual color or a color distribution. We

use a parametric colorization deep learning model trained on a large dataset with improvements in the loss function used and color prediction.

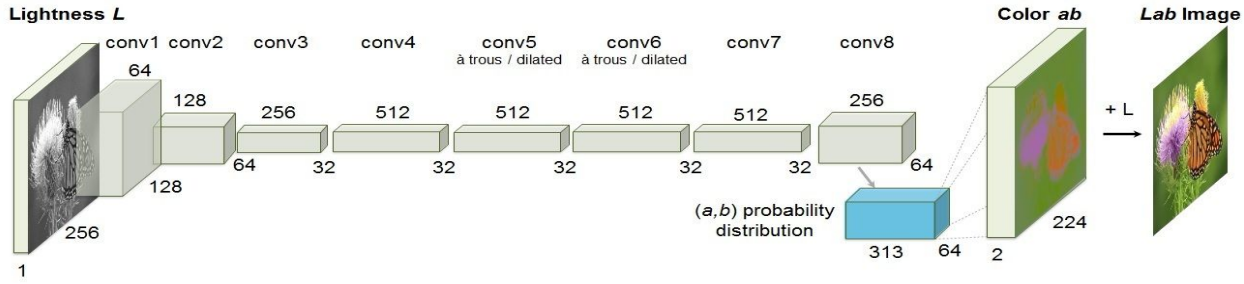


Figure 2: Colorization network

We use a pre-trained model developed by Richard Zhang from UC Berkeley in 2016 in this project. The pre-trained model is a variation of AlexNet that was trained on ImageNet with more than 1 million images. In this model, we train a CNN to map a grayscale input to a distribution over quantized color value output. Each convolution layer is a block of 2 or 3 repeated convolution and ReLU layers, followed by a BatchNorm layer. The network does not have any pooling layers. Spatial downsampling or upsampling between convolution blocks to achieve changes in resolution. The Color ab layer is used for inferring pixel colors from the predicted color distribution. We unfreeze convolution layer 7 and layer 8 and train the model on a subset of images from CIFAR 10 and Places365. We first convert the color image from these datasets to black and white, resize them and pass it to the colorization to obtain the colorized output. We retrained the model for 15 epochs on Google Colab and took us about 6 hours.

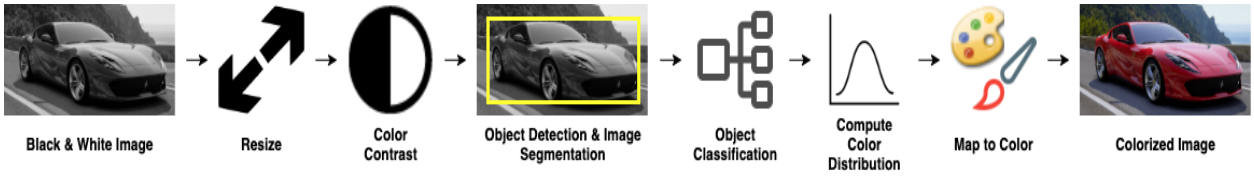


Figure 3: Transformations in the colorization network

The transformations within the colorization network is shown in the image above. The input black and white image is resized to 256 x 256. The black and white contrast is increased so that the object boundaries are more profound. Clear boundaries are useful for the next step of object detection and image segmentation.Once the objects are detected, they are classified to determine its color. A pixel wise Gaussian color distribution is computed in the color ab layer. Finally the pixels are mapped to a particular color in the color ab layer based on the distribution.

5.3 Colorization Loss Functions

There are many loss functions in play in the colorization network but we highlight two most important functions in this section.

5.3.1 Cross Entropy Loss for Object Classification

$$L(\hat{Z}, Z) = -\frac{1}{HW} \sum_{h,w} v(Z_{h,w}) \sum_{q} Z_{h,w,q} \log(\hat{Z}_{h,w,q})$$

Rarity weighting — Target distribution — Predicted distribution

Object classification is a multi-class classification problem and cross entropy is a good loss function since it minimizes the distance between the target and predicted distribution. Here, we try to minimise the cross entropy loss associated with the object class prediction. We use rarity weighting to account for unequal distribution of types of objects in images.

5.3.2 Color Class Rebalancing

$$v(Z_{h,w}) = w_{q^*}, \text{ where } q^* = \arg\max_q Z_{h,w,q}$$

$$w \propto \left((1-\lambda)\tilde{p} + \frac{\lambda}{Q}\right)^{-1}, \quad \mathbb{E}[w] = \sum_q \tilde{p}_q w_q = 1$$

reweighting — empirical distribution — combine with uniform

We perform color class rebalancing to ensure rare colors are correctly applied to objects in the image in the color ab layer. The image in Figure 4shows a snapshot of the internal computations of the color ab layer. We see how the ab layer determines the best color assignment to the input grayscale images.

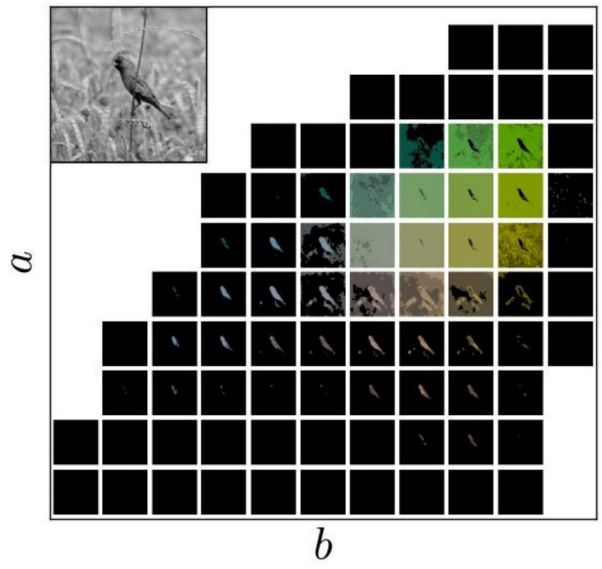


Figure 4: Best color assignment in Color ab layer in colorization network

5.4 Colorization Pipeline

We select a subset of images from CIFAR10 and Places365 dataset for model training. These color images are converted to grayscale and passed to the Colorization network. The colorization network returns the colored image as output. We further perform a few different evaluations on these images discussed in further sections.

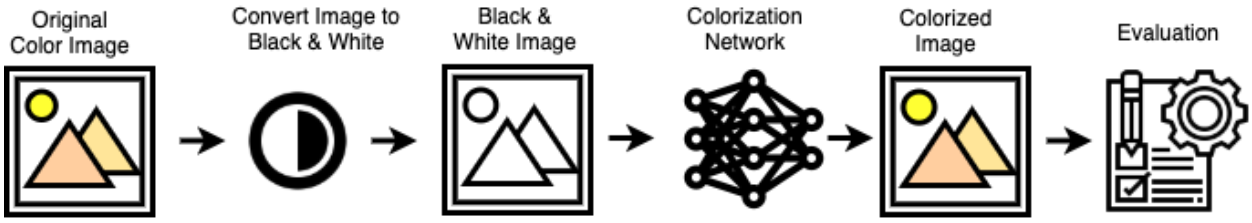


Figure 5: Colorization pipeline

5.5 Colorization Code Snippet

```
In [6]: def colorize_image(image_filename=None, cv2_frame=None):
            """
            Where all the magic happens.  Colorizes the image provided. Can colorize either
            a filename OR a cv2 frame (read from a web cam most likely)
            :param image_filename: (str) full filename to colorize
            :param cv2_frame: (cv2 frame)
            :return: Tuple[cv2 frame, cv2 frame] both non-colorized and colorized images in cv2 format as a tuple
            """
            # load the input image from disk, scale the pixel intensities to the range [0, 1], and then convert the image from the BGR
        to Lab color space
            image = cv2.imread(image_filename) if image_filename else cv2_frame
            scaled = image.astype("float32") / 255.0
            lab = cv2.cvtColor(scaled, cv2.COLOR_BGR2LAB)

            # resize the Lab image to 224x224 (the dimensions the colorization network accepts), split channels, extract the 'L' chann
        el, and then perform mean centering
            resized = cv2.resize(lab, (224, 224))
            L = cv2.split(resized)[0]
            L -= 50

            # pass the L channel through the network which will *predict* the 'a' and 'b' channel values
            print("[INFO] colorizing image...")
            net.setInput(cv2.dnn.blobFromImage(L))
            ab = net.forward()[0, :, :, :].transpose((1, 2, 0))

            # resize the predicted 'ab' volume to the same dimensions as our input image
            ab = cv2.resize(ab, (image.shape[1], image.shape[0]))

            # grab the 'L' channel from the *original* input image (not the resized one) and concatenate the original 'L' channel with
        the predicted 'ab' channels
            L = cv2.split(lab)[0]
            colorized = np.concatenate((L[:, :, np.newaxis], ab), axis=2)

            # convert the output image from the Lab color space to RGB, then clip any values that fall outside the range [0, 1]
            colorized = cv2.cvtColor(colorized, cv2.COLOR_LAB2BGR)
            colorized = np.clip(colorized, 0, 1)

            # the current colorized image is represented as a floating point data type in the range [0, 1] -- let's convert to an unsi
        gned 8-bit integer representation in the range [0, 255]
            colorized = (255 * colorized).astype("uint8")
            return image, colorized


        def convert_to_grayscale(frame):
            gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)  # Convert webcam frame to grayscale
            gray_3_channels = np.zeros_like(frame)  # Convert grayscale frame (single channel) to 3 channels
            gray_3_channels[:, :, 0] = gray
            gray_3_channels[:, :, 1] = gray
            gray_3_channels[:, :, 2] = gray
            return gray_3_channels
```

Figure 6: Colorization code snippet

This code snippet showcases the main function responsible for colorizing the image. The image is read from the given file path and converted to grayscale if it is not already in grayscale. The image is resized and fed to the network for colorization. Finally we obtained the colorized image output.

5.6 Colorization Evaluation

We explore two variants of the same evaluation method in this section. First we convert the generated color image and the original color image into tensors. Each pixel of both the images are represented as a RGB value between 0 - 255 in each color dimension in their respective tensors. We subtract one tensor from the other pixel by pixel. In the first variation, we clamp the pixel wise tensor difference between 0 and 1. In the second variation we compute pixel wise absolute difference.

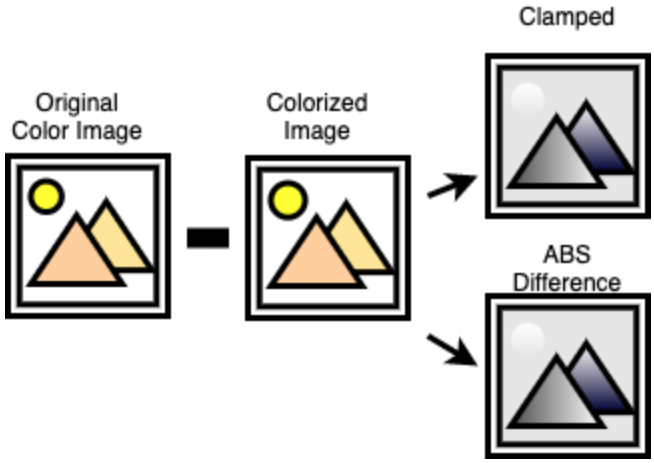


Figure 7: Colorization evaluation

(a) Original color image (b) Black and white image (c) Generated color image

Figure 8: Einstein statue at Tech Green, Georgia Tech

Figure 8 is a photo taken near Tech Green in Georgia Tech. We converted that to black and white to use as our test image. We can see that the algorithm is able to colorize the trees, sky and grass accurately. The red color of the Clough Commons building is shown as a gray in the colorized output as the model is unable to pick up these colors.

On close observation in Figure 10, we see few patches corresponding to the red hues of the Clough Commons Building showing up in the output images. The patches in the output images are brighter in the regions where the color difference between the generated color image and original color image is higher.

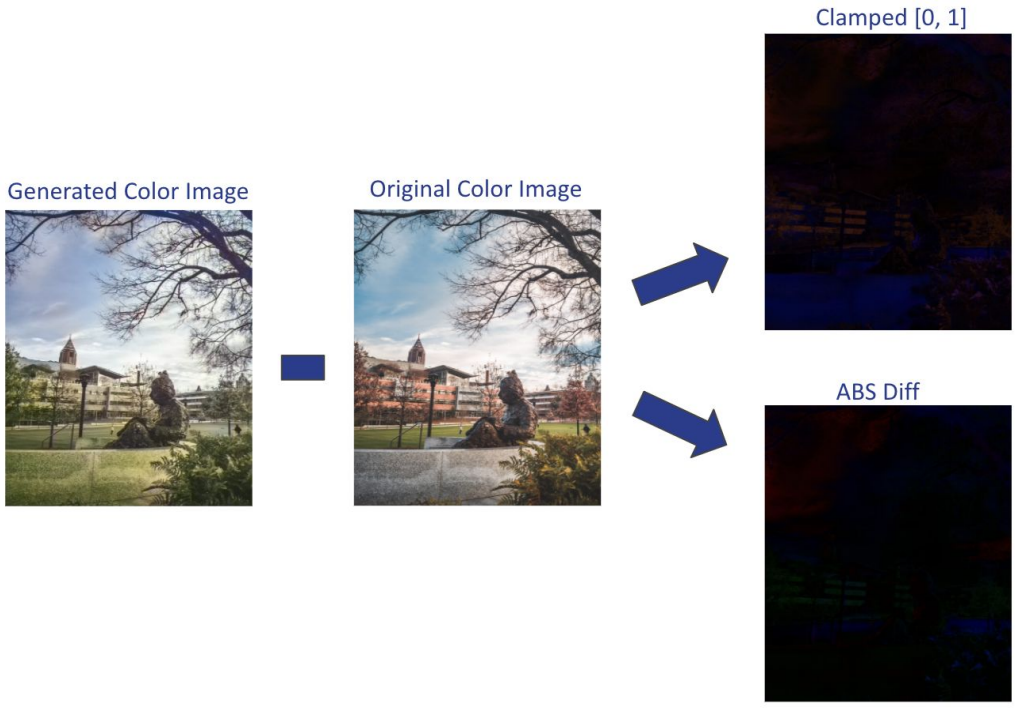
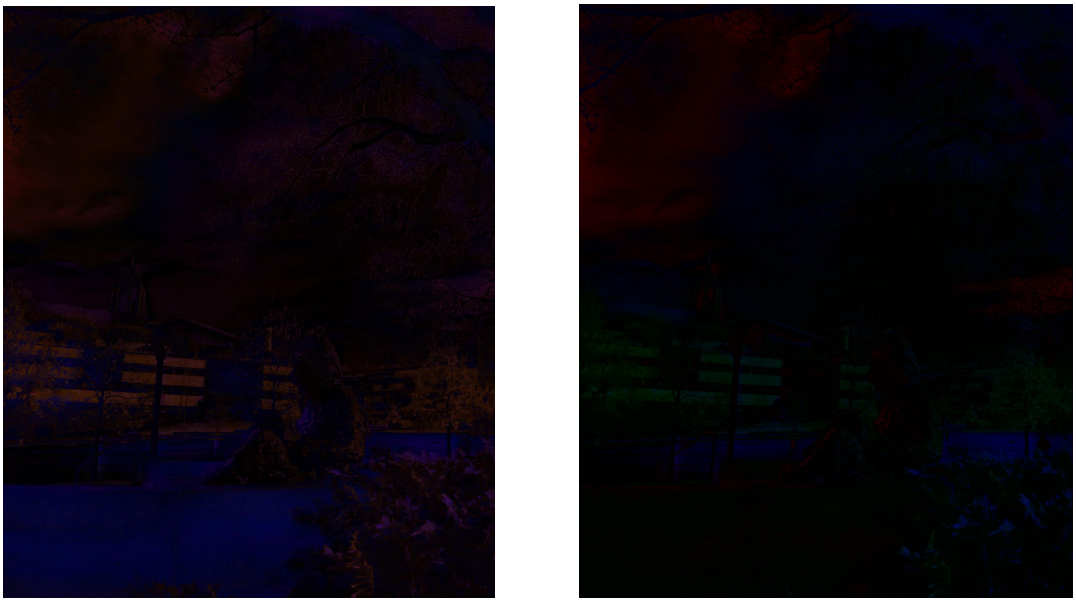


Figure 9: Einstein statue at Tech Green, Georgia Tech colorization evaluation



(a) Clamped (b) ABS difference

Figure 10: Einstein statue at Tech Green, Georgia Tech colorization evaluation enlarged output

Similarly, we applied the colorization model on this image of a lion in Figure 11. We notice the colorized output assigns the lion's body a more earthy brown while compared to the orangish brown observed in the original color. We also notice some red hues near the mouth and nose region. In these images we notice patches where we noticed variation in the colors between the original color and generated color image

(a) Original color image (b) Black and white image (c) Generated color image

(d) Clamped (e) ABS difference

Figure 11: Colorization of lion's face

The colorization was applied on a car in the next example as shown in Figure 12, but the output might not match the original color image. We obtain different colored cars in each execution of this input image. The colorization network is able to identify that there is a car in the image but

can not correctly determine its color since there are no clues or signals corresponding to the color of the car in the black and white image. Thus it assigns a valid car color as seen in the output.

(a) Original color image

(b) Black and white image

(c) Generated color image

(e) ABS difference

(f) Clamped

Figure 12: Colorization of a sports car

As expected, we see in Figure 12(e), Figure 12(f), bright patches in the locations where the car is shown in the image since there is a large color variation between the original color image and generated color image.

5.7 Colorization Examples

5.7.1 Challenging Colorization

(a) Black and white image (b) Colorized image

Figure 13: Challenging colorization example - city skyline

In Figure 13, the model could not identify the color of the buildings because there are too many objects giving it a tightly clustered look. The colorization model finds it difficult to identify each object separately and happens to assume this to be one large object. When the model cannot identify the color, it colors the object yellow by default.

5.7.2 Moderately Challenging Colorization

(a) Black and white image (b) Colorized image

Figure 14: Moderately challenging colorization example - mountain range

Figure 14 is a case of medium colorization as the big objects such as the mountain and the trees are rightly colored but the smaller objects such as the sheep grazing at the bottom of the image aren't. You can see shades of yellow around them because they aren't identified correctly. Another thing to notice is that the trees happened to be much darker and grainy compared to the actual color of the trees.

5.7.3 Best Colorization

(a) Black and white Image (b) Colorized image

Figure 15: Moderately challenging colorization example - mountain road

Figure 15 one the best images produced by the model. We can see that the colorized image is rightly illuminated giving it a very natural look. Not just the rocks and the trees, but the river and the snow filled pathway is also rightly colored.

5.7.4 Colorization of Human Faces

(a) Black and White Image (b) Colorized Image

Figure 16: Human face colorization - Marilyn Monroe

We improved the training of our model by including images of human faces from the CelebA dataset which contains thousands of celebrity faces. In the image of Marilyn Monroe in Figure 16 we can observe that our model does perform well on human faces as well.

5.7.5 Bad Colorization

(a) Black and white image (b) Colorized image

Figure 17: Bad colorization example

Figure 17 above is an example of bad colorization and for which cases our colorization algorithm does not work very well. Here, only the sky and the trees are colorized correctly. Our algorithm can match or identify most of the generic objects in the images. The sky and trees are examples of such generic objects and are therefore colorized accurately. The other objects are not colorized properly and this is due to the following reasons -

1. The colorization algorithm works on object identification. There are too many objects in this image and all are very closely cluttered. The colorization algorithm therefore finds it difficult to identify them separately.
2. The Coca Cola symbol in this image is a very specific type of object that needs to be trained to get the correct results. As an improvement, we should include logos and symbols dataset as well during the training.

As a result, the colorization algorithm adds a yellow color to the unidentified object.
If training is done on a larger dataset consisting of a high number of diverse objects, we can produce better results.

5.8 Colorization Turing Test

The Colorization algorithm was evaluated using a Colorization Turing Test, asking the participants to select between a colorized and the ground truth image. We conducted the 'Colorization Turing Test' using a survey link in which we added a set of images - original and the synthesized colors of an image. Participants were asked to identify the original image in each pair. The results of the test indicated that we were able to fool the participants for 66 percent of the total instances which is significantly higher and more than we expected.
These were the results of our turing test:

1a Colorized 1b Original

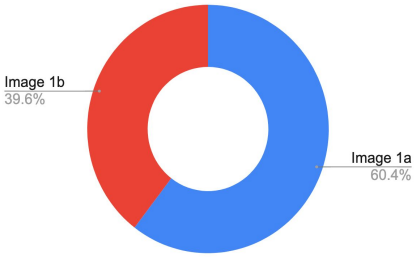


Figure 18: Colorization Turing test image pair 1

For image pair 1, the original image is the one on the right, but the majority of the participants picked the colorized as the original image. This is due to the correct colorization of the objects based on their mapping to the real world.

2a Colorized 2b Original

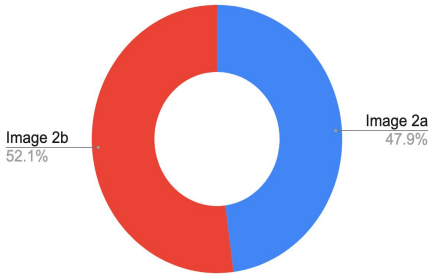


Figure 19: Colorization Turing test image pair 2

For image pair 2, the results were pretty divided as the difference between the two images is comparatively very low.

3a Colorized 3b Original

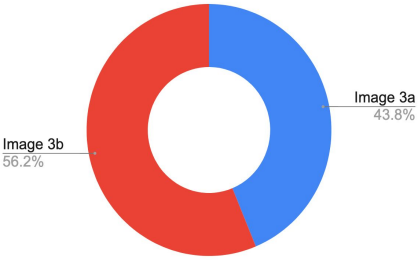


Figure 20: Colorization Turing test image pair 3

For image pair 3, Many of the participants were able to pick the original image. This could be because of the blue hues on the bridge which is not accurate.

4a Colorized 4b Original

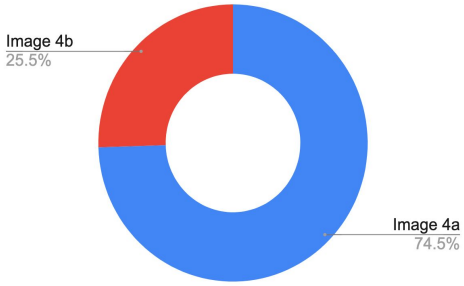


Figure 21: Colorization Turing test image pair 4

Image pair 4 was pretty tricky. About 75% of the participants picked the colorized image as the original image. The color of the grass and the fur of the dog were colorized pretty dull giving it a more natural look.

5a Original 5b Colorized

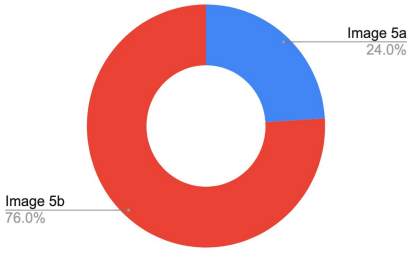
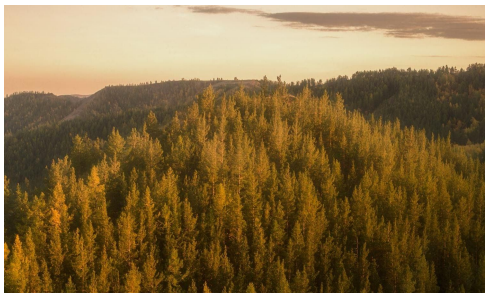


Figure 22: Colorization Turing test image pair 5

Image pair 5 was easy to identify as the original image had pink hues in the sky which was not picked by our model as it is not very generic and difficult to achieve. Parts of the boat are turned gray in the colorized image as these are small objects not correctly identified by the model.



6a Original

6b Colorized

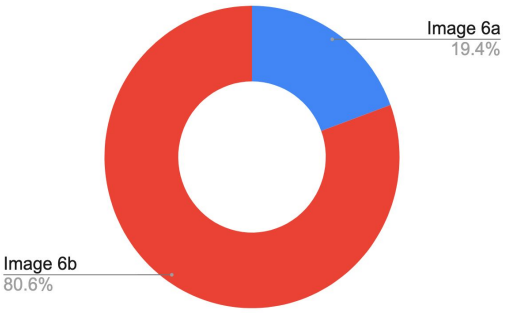


Figure 23: Colorization Turing test image pair 6

Image pair 6 was tricky to participants as the original one was captured around sunset during fall, but the colorized image produced by our model made the sky and trees generic colors of blue and green.The Places365 dataset that we used has trees that are mostly in green and hence our colorization model did not show trees to be reddish brown color.

From the observations made so far we believe that our algorithm performs pretty well, but is not perfect and it requires more training. If the model is trained on a larger dataset consisting of a high number of diverse objects, we could produce better results.

5.9 Colorization Comparison With Other Colorization Models

We compared our colorization model with SIGGRAPH 17[10] and DeOldify[11] and discuss the result in this section.

SIGGRAPH 17 used two subnetworks. The Local hints network is responsible for incorporating user preferences provided through the UI and predict the color distribution. The Global hints network is responsible for overall colorization and it uses 1 x 1 convolutional layer. Each box represents a convolution layer, with vertical dimension indicating feature map spatial resolution, and horizontal dimension indicating number of channels. Changes in resolution are achieved

through subsampling and upsampling operations.

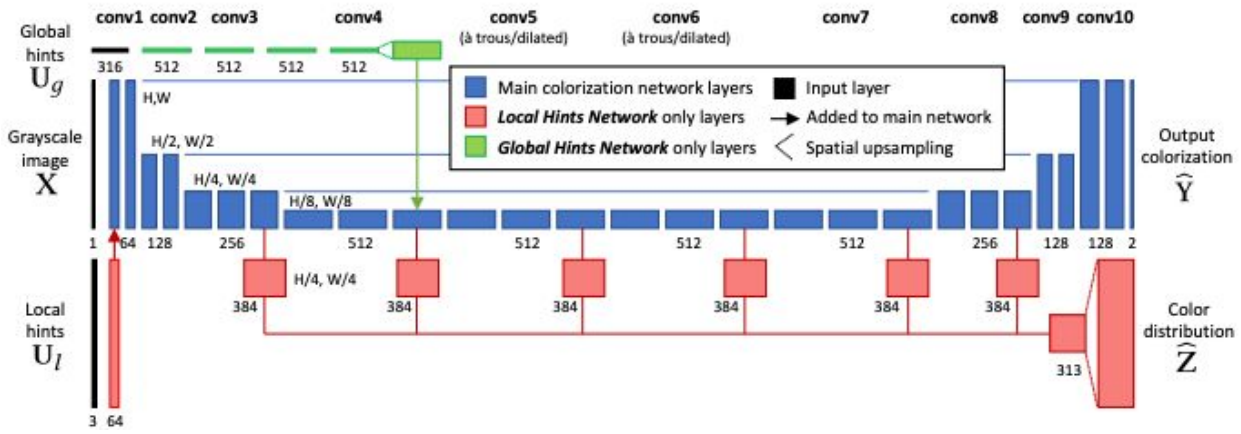


Figure 24: SIGGRAPH17 network architecture

DeOldify is a GAN model with a generator and comparator. It uses NoGAN, which provides the benefits of GAN training while spending minimal time doing direct GAN training. Instead, most of the training time is spent pretraining the generator and critic separately which are faster and more reliable conventional methods.

(a) Black and white (b) Our model (c) SIGGRAPH17 (d) DeOldify

Figure 25: Colorization comparison 1

In these images in Figure 25, our model does not recognize the red color of coca-cola and does not correctly distinguish all the objects and instead adds a yellow color. The SIGGRAPH17, also does not fetch the colors for the objects in the image while DeOldify is able to correctly perform object detection and colorize them based on it.

(a) Black and white (b) Our model

(c) SIGGRAPH17 (d) DeOldify

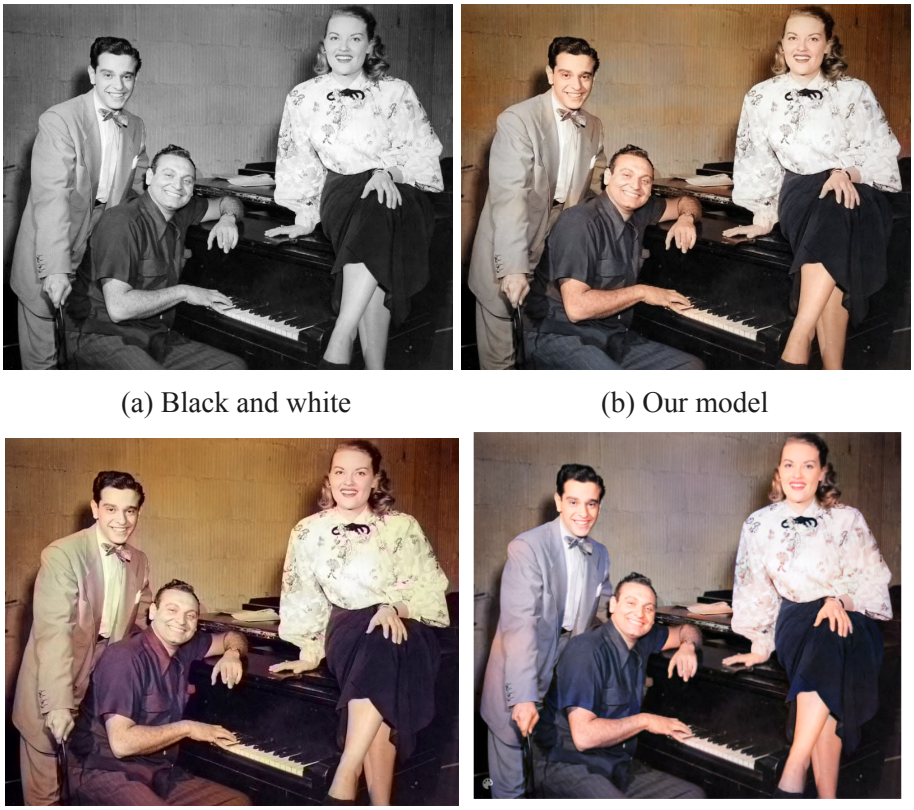
Figure 26: Colorization comparison 2

In the image in Figure 26 (a), our model performed better than the other two (Figure 26(b),(c), and (d)) and it colors the flowers in bright yellow color as expected. The SIGGRAPH17 is able to color the image but the colors added are not bright and have more green color. DeOldify does not perform well on this image and adds white color to the flowers at some places.

(a) Black and white (b) DeOldify (c) SIGGRAPH17 (d) Our Model

Figure 27: Colorization comparison 3

In Figure 27, although all three models were able to color the celebrity face, there are some variations in each. Our model, colorized the celebrity's face as well as the background, but there is more illumination on the face. For SIGGRAPH17, the image is beautifully colored giving it a very natural look. The results of DeOldify show the face containing more white balance and red tones. Also the background is not colored and the only main object is detected and colored.

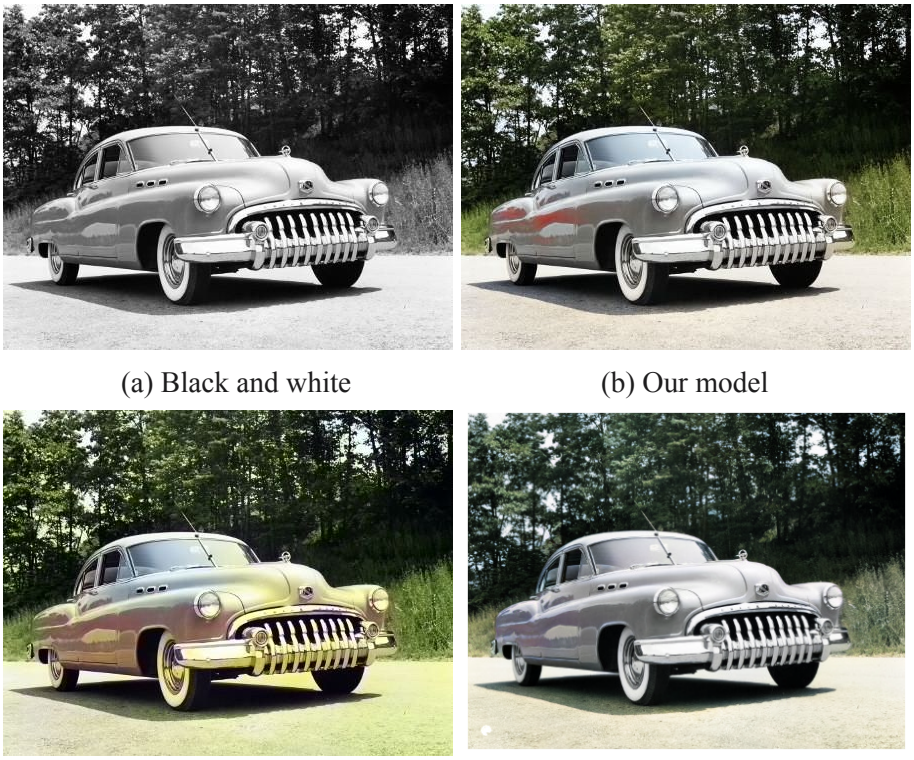


(a) Black and white (b) Our model

(c) SIGGRAPH17 (d) DeOldify

Figure 28: Colorization comparison 4

For this image, the colorized images generated from all three models differ significantly. Our model, colorizes all three humans and the background more accurately and gives a more natural appeal. SIGGRAPH17 colorized the image but added a yellow tint on the image, which is not desired. In DeOldify, the images are colorized but there is red tint at some places like on the suit of the man standing on the left. Thus it looks less natural.



(a) Black and white (b) Our model

(c) SIGGRAPH17 (d) DeOldify

Figure 29: Colorization comparison 5

The car in the original image had blue color, but none of the models colored the car blue. The reason is the car can have any multiple colors. Majority of the cars in the dataset have metallic color. Our model picked up the background colors properly but did not add any colors to the car, and instead added a red tint on the sides of the car. For SIGGRAPH 17, the background color was colorized best, but it added a very unnatural yellow color on the car. DeOldify, neither picked up the background color nor it colorized the car correctly. But the clarity and sharpness of the car object is best here.

All models perform differently for different types of images. Our model works best for nature images and is also good for human faces. We can update our dataset and perform more training on different images to produce better results.

6. Style Transfer

6.1 Style Transfer Neural Network

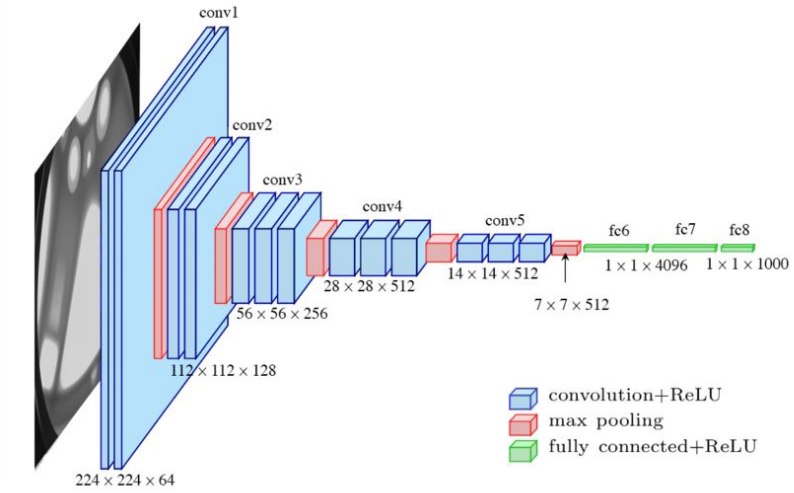


Figure 30: Style transfer architecture

For Style Transfer, we use a VGG-16 network composed of 16 layers and proposed by Visual Geometric Group from Oxford University. In figure 30, all blue rectangles represent the convolutional layers along with the non-linear activation function called rectified linear unit (ReLU). There are 13 convolutional layers and 5 max-pooling layers along with 3 fully connected layers. The network accepts color images as input of size 224 x 224 and these images pass through a stack of convolution layers where every convolution filter has a receptive field of 3 X 3 and stride of 1. Having a kernel of 3 x 3 leads to a much better feature extraction. Max pooling is performed over a max-pool window size of 2 x 2 with stride equal to 2. The first 2 fully connected layers have 4096 channels each and the third layer which is also an output layer has 1000 channels, one for each category of image in the dataset. In this architecture, we start with a very low channel size of 64 and then gradually increase by a factor of 2 after each max-pooling layer, until it reaches 512.

In style transfer, two images are provided as input to this neural network i.e. a content image and the style image. The motive is to generate a mixed image that has contours of the content image and texture, color pattern of the style image. This is done by optimizing several loss functions. The loss function for the content image minimizes the difference of the features activated for the content image corresponding to the mixed image at one or more layers. This preserves the contour of the content image to the resultant mixed image. Whereas the loss function for the style image minimizes the difference between the so-called Gram-matrices between style image and the mixed image. This is done at one or more layers. The usage of the Gram matrix is to identify which features are activated simultaneously at a given layer.

6.2 Style Transfer Code Snippet

The snippet in Figure 31 shows the main function used for style transfer. In this function, we configured the number of epochs or steps, style weight and content weight. The number of epochs was configured to 250. Also, for style transfer more weight is given to the style than content since we have more style compared to the content of the original image. Thus, we have configured a very high value for style weight and small value of 1 to content weight. In the screenshot, the code for the content loss and style loss is provided.

```
def run_style_transfer(cnn, normalization_mean, normalization_std,
                       content_img, style_img, input_img, num_steps=250,
                       style_weight=1000000, content_weight=1):

    """Run the style transfer."""
    print('Building the style transfer model..')
    model, style_losses, content_losses = get_style_model_and_losses(cnn,
        normalization_mean, normalization_std, style_img, content_img)
    optimizer = get_input_optimizer(input_img)
    print('Optimizing..')
    run = [0]
    while run[0] <= num_steps:

        def closure():
            # correct the values of updated input image
            input_img.data.clamp_(0, 1)

            optimizer.zero_grad()
            model(input_img)
            style_score = 0
            content_score = 0

            for sl in style_losses:
                style_score += sl.loss
            for cl in content_losses:
                content_score += cl.loss

            style_score *= style_weight
            content_score *= content_weight

            loss = style_score + content_score
            loss.backward()

            run[0] += 1
            if run[0] % 50 == 0:
                print("run {}:".format(run))
                print('Style Loss : {:4f} Content Loss: {:4f}'.format(
                    style_score.item(), content_score.item()))
                print()

            return style_score + content_score

        optimizer.step(closure)

    # a last correction...
    input_img.data.clamp_(0, 1)
    return input_img
```

```
# losses
content_losses = []
style_losses = []

# assuming that cnn is a nn.Sequential, so we make a new nn.Sequential
# to put in modules that are supposed to be activated sequentially
model = nn.Sequential(normalization)

try:
    i = 0  # increment every time we see a conv
    for layer in cnn.children():
        if isinstance(layer, nn.Conv2d):
            i += 1
            name = 'conv_{}'.format(i)
        elif isinstance(layer, nn.ReLU):
            name = 'relu_{}'.format(i)
            # The in-place version doesn't play very nicely with the ContentLoss
            # and StyleLoss we insert below. So we replace with out-of-place
            # ones here.
            layer = nn.ReLU(inplace=False)
        elif isinstance(layer, nn.MaxPool2d):
            name = 'pool_{}'.format(i)
        elif isinstance(layer, nn.BatchNorm2d):
            name = 'bn_{}'.format(i)
        else:
            raise RuntimeError('Unrecognized layer: {}'.format(layer.__class__.__name__))

        model.add_module(name, layer)

        if name in content_layers:
            # add content loss:
            target = model(content_img).detach()
            content_loss = ContentLoss(target)
            model.add_module("content_loss_{}".format(i), content_loss)
            content_losses.append(content_loss)

        if name in style_layers:
            # add style loss:
            target_feature = model(style_img).detach()
            style_loss = StyleLoss(target_feature)
            model.add_module("style_loss_{}".format(i), style_loss)
            style_losses.append(style_loss)
except Exception as e:
    print(e)

# now we trim off the layers after the last content and style losses
for i in range(len(model) - 1, -1, -1):
    if isinstance(model[i], ContentLoss) or isinstance(model[i], StyleLoss):
        break

model = model[:(i + 1)]

return model, style_losses, content_losses
```

Figure 31: Style transfer code snippet

6.3 Style Transfer Examples

Here are a few examples of style transfer we achieved using our model. We ran the model till 900 epochs.

In the first example, the left image shows the original image of Brooklyn Bridge at New York City, the center image shows the style being applied and the right image is the stylized image we achieved after running the model. The style transfer did a good job getting both the colors and the texture of the style.

(a) Content image (b) Style image (c) Stylized image

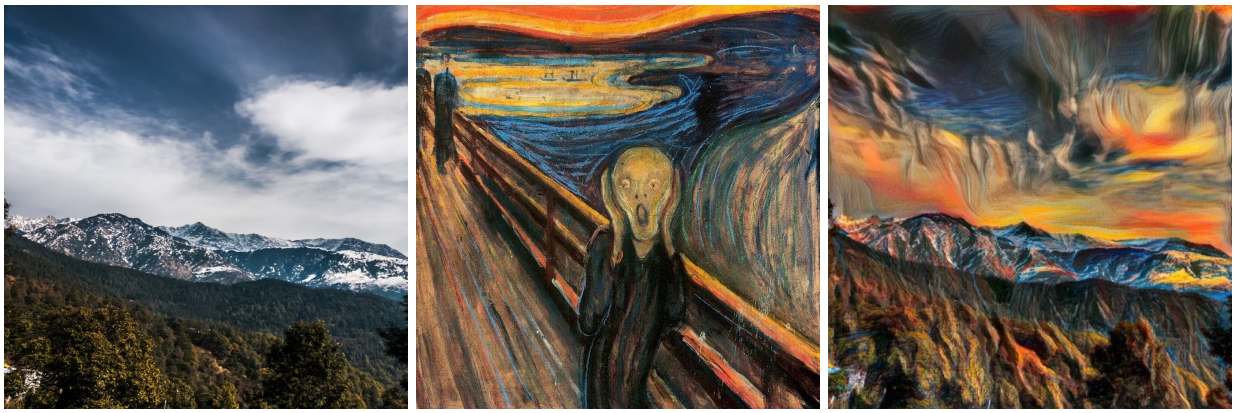
Figure 32: Style transfer example - New York bridge

Here's another example with the image from Roswell Mill in Georgia and the style of a colorful painting.

(a) Content image (b) Style image (c) Stylized image

Figure 33: Style transfer example - Roswell Mill, Georgia

Another example with a capture of Dhauladhar mountains in India with Da Vinci's Scream being used as the artwork.



(a) Content image (b) Style image (c) Stylized image

Figure 34: Style transfer example - Dhauladhar mountains, India

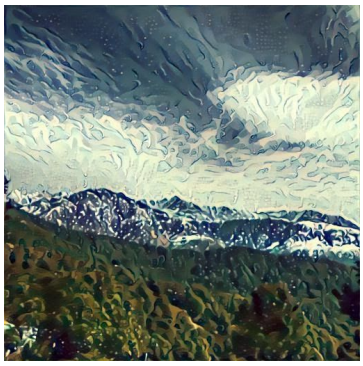
6.4 Style Transfer Analysis

6.4.1 Hyperparameter Optimization - Number of Epochs

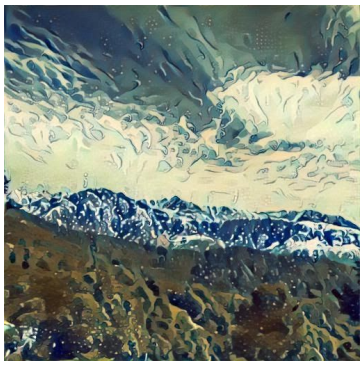
We ran our model for different epochs to see the extent of style transfer achieved. We ran the model for 100, 300, 500, 700 and 900 epochs. The original image is the Dhauladhar mountain image and we used the very famous waves artwork “The Great Wave off Kanagawa” as the style.

(a) Original image

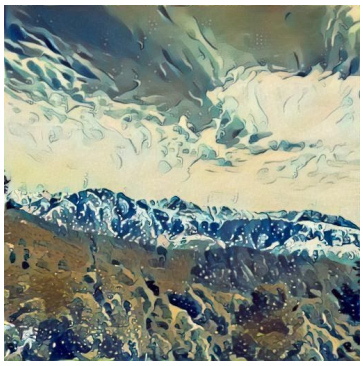
(b) Style image



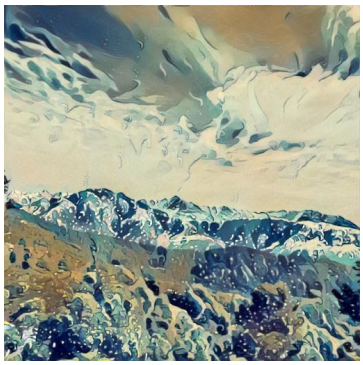
(c) 100 epochs



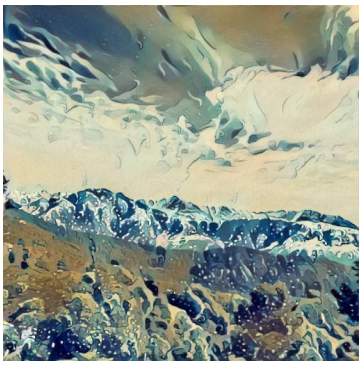
(d) 300 epochs



(e) 500 epochs



(f) 700 Epochs



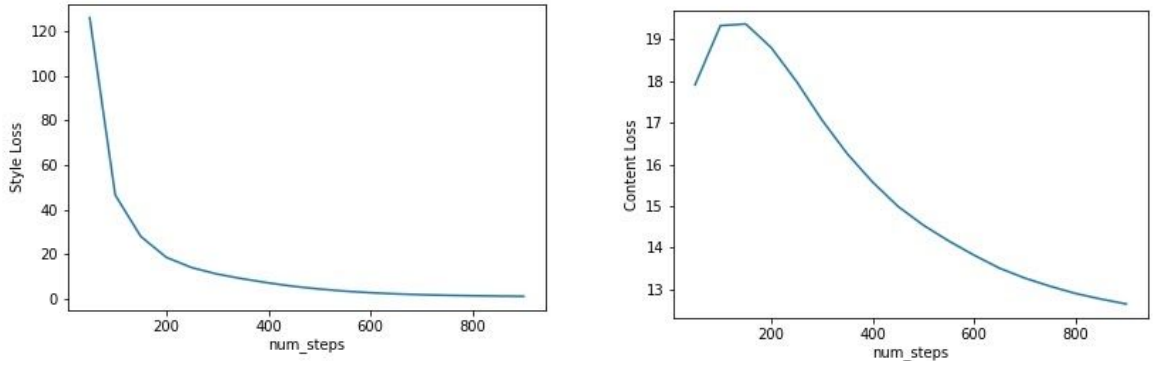
(g) 900 Epochs

Figure 35: Style transfer hyperparameter optimization - number of epochs

We observed that at 100 epochs the stylized image had got some features of the style image and some color as well. But it still looked more like the original image itself. At 300 epochs, we observed a better effect, but still retained more of the original image. From epoch 500 onwards, the style became predominant and the texture and color were transferred smoothly. At 700 and 900 epochs, the transfer looked complete with good transfer of color and texture without affecting the key parts of the original image.

6.4.2 Style and Content Losses

We observed the style and content loss across epochs till 900 epochs for the mountain image and the waves style. As expected we saw a steep decrease in style loss, since we are transferring the original image to look more like the style image. The drop was drastically high at the start and after around 600 epochs it started leveling out. The content loss rose at first and then dropped almost linearly after that.

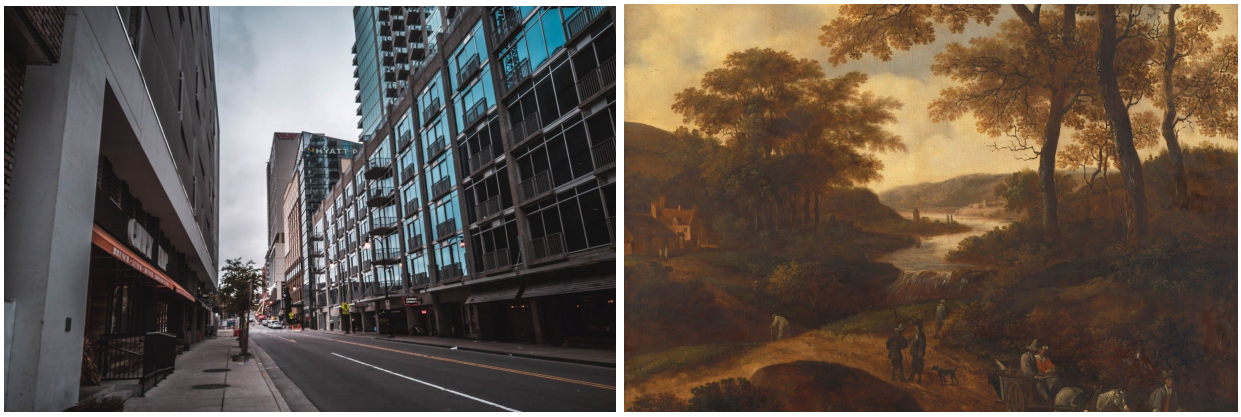


(a) Style loss vs number of epochs (b) Content loss vs number of epochs

Figure 36: Style loss and content loss

6.5 Style Transfer Comparison with other Models

To compare different models we ran the style transfer on 4 different models. We chose Alexnet, VGG-11, VGG-16 and VGG_19. We ran the style transfer on an image of the Nashville Downtown on all 4 models for 900 epochs.



(a) Original Image (b) Style Image

Figure 37: Style transfer model comparison inputs

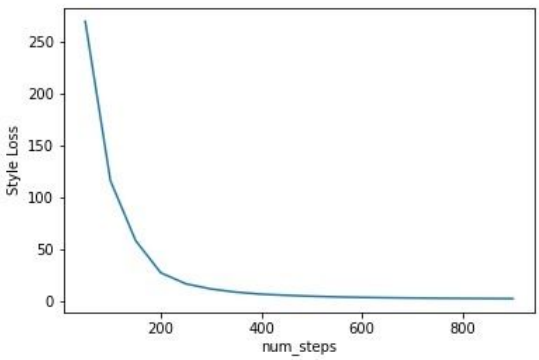
As expected, Alexnet being the shallowest model, resulted in a very blurry style transfer. Even VGG-11 resulted in a slightly blurry image, but better than that achieved by Alexnet. VGG-16 and VGG-19 both gave very good results, with the latter giving better details. This may be due to the fact that VGG-19 is deeper compared to VGG-16.

(a) AlexNet (b)VGG11

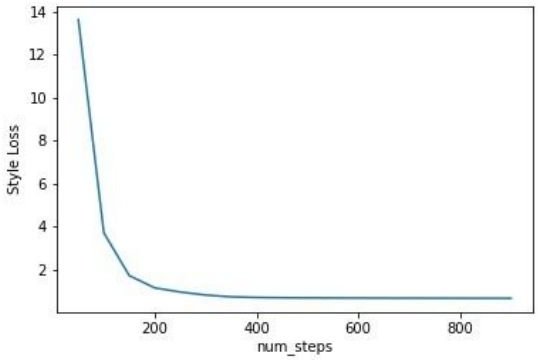
(c) VGG16 (d) VGG19

Figure 38: Style transfer model comparison outputs

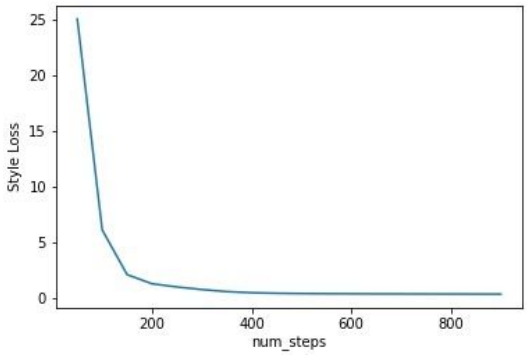
Here is the style loss comparison for the 4 models.



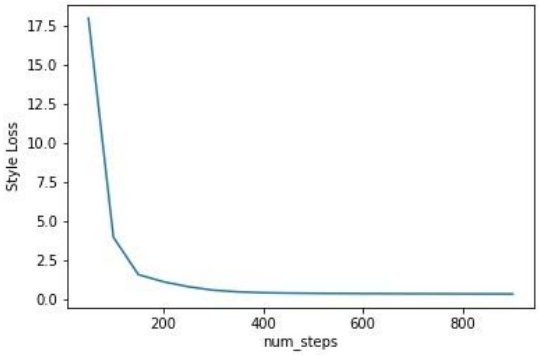
(a) AlexNet



(b)VGG11



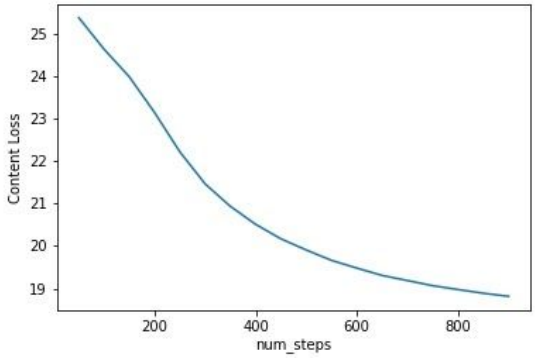
(c) VGG16



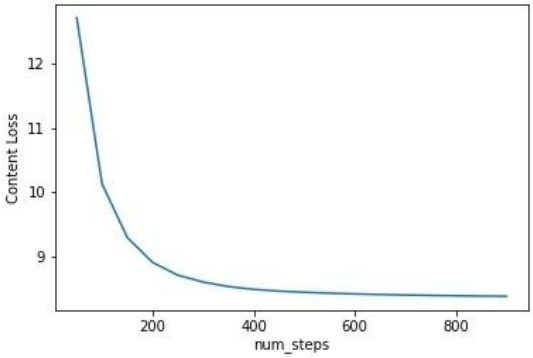
(d) VGG19

Figure 39: Style transfer model comparison - style loss vs number of epochs

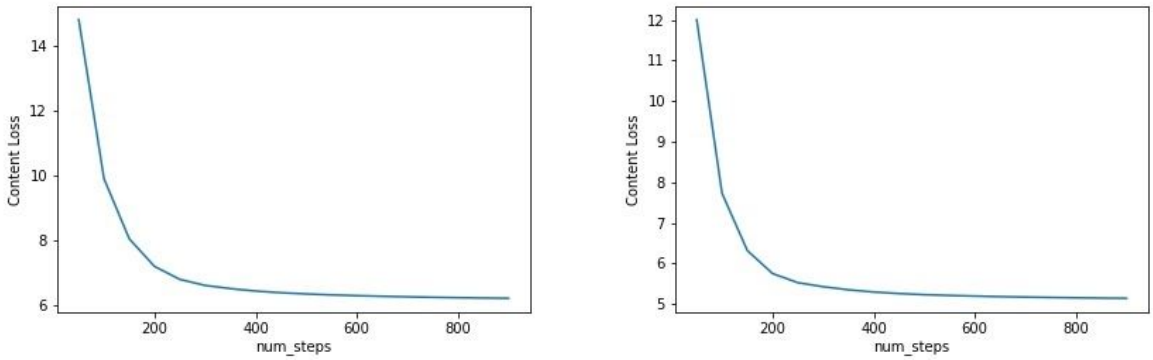
Here is the content loss comparison of the 4 models.



(a) AlexNet



(b)VGG11



(c) VGG16 (d) VGG19

Figure 40: Style transfer model comparison - content loss vs number of epochs

Comparing the 4 models, VGG16 and VGG19 gave the best results. Good content and style losses were better as well. We chose VGG16 as it was less resource hungry than VGG19 and the detail improvement we observed in VGG19 was not substantial enough to choose the deeper model.

7. User Interface

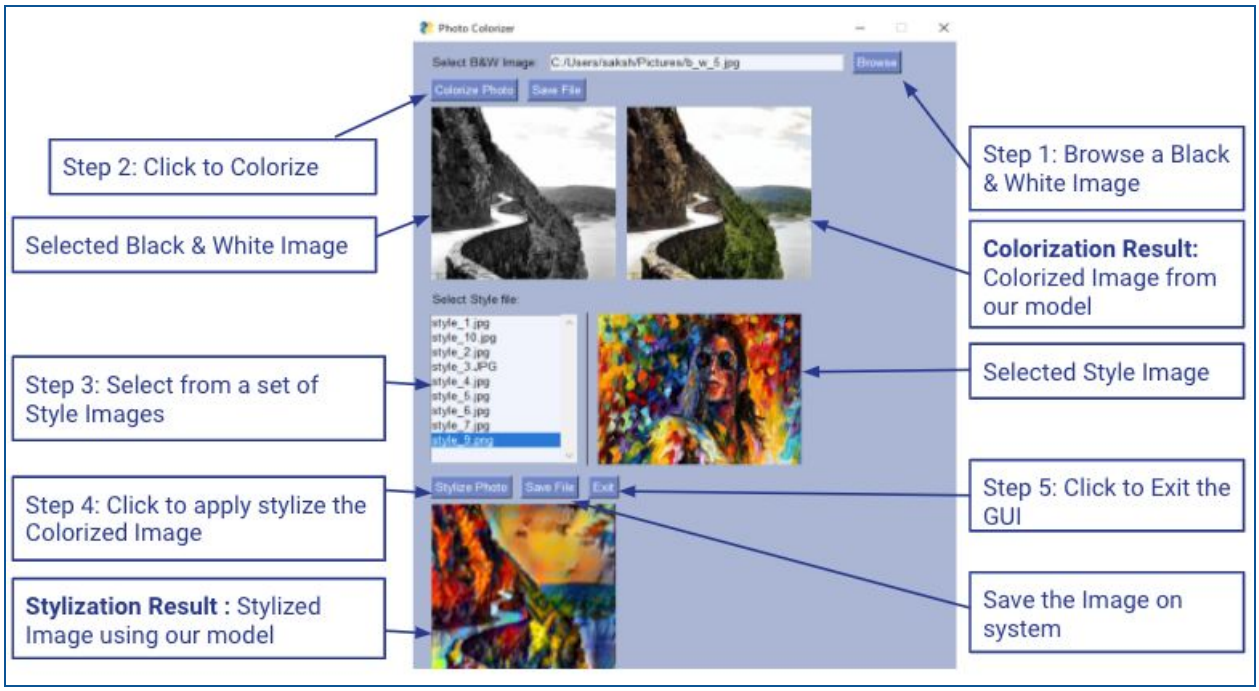


Figure 41: GUI screenshot

The image indicates a screenshot of the GUI that we developed for the application using PySimpleGUI. The image shows a stepwise guide to use the application.

```
In [23]: # ---------------------------------- The GUI ----------------------------------

folder = "C:\MS\BigData\Final_Project\Style_Images"

sg.theme('LightBlue2')

# First the window layout...2 columns
images_col = [[sg.Text('Select B&W Image:'), sg.In(enable_events=True, key='-IN FILE-'), sg.FileBrowse()],
              [sg.Button('Colorize Photo', key='-PHOTO-'), sg.Button('Save File', key='-SAVE-')],
              [sg.Image(filename='', key='-IN-', size=(5,5)), sg.Image(filename='', key='-OUT-', size=(5,5))]]

style_col = [[sg.Button('Stylize Photo', key='-STYLEPHOTO-'), sg.Button('Save File', key='-SAVE STYLE-'), sg.Button('Exit')],
              [sg.Image(filename='', key='-STYLE OUT-', size=(20,20))]]

image_list_column = [
    [sg.Text('Select Style file:')],
       [
        sg.Listbox(
            values=[], enable_events=True, size=(20, 10), key="-FILE LIST-"
        ), sg.VSeperator(), sg.Text("Choose a style image from list on left:", key='-TEXT-'), sg.Image(key="-IMAGE-")
    ]
]


# ----- Full layout -----
layout = [[sg.Column(images_col, key='-COLORIZE-')], [sg.Column(image_list_column, key='-IMAGE LIST-', visible=False)],
          [sg.Column(style_col, key='-STYLE-', visible=False)]]

# ----- Make the window -----
window = sg.Window('Photo Colorizer', layout, grab_anywhere=True, location=(0,0), size=(600,800), keep_on_top=True).Finalize()
```

Figure 42: GUI code snippet

Few key components of the code are shown in the code snippet above.

8. Stretch Goals: Colorization and Style Transfer on Videos/GIFs.

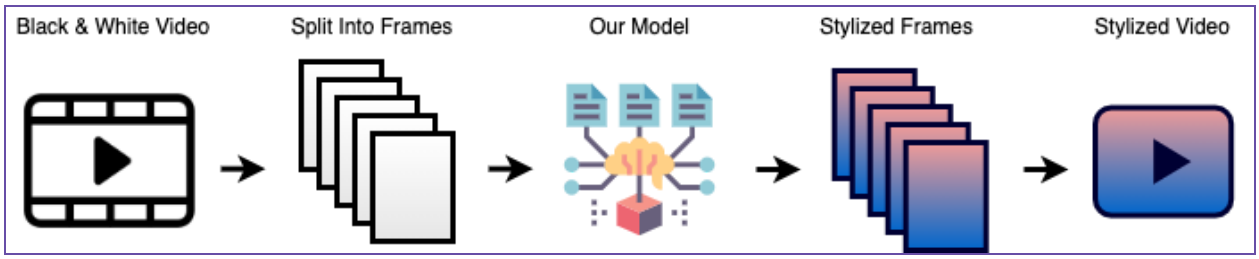


Figure 43: Colorization and style transfer on videos and GIFs

We've extended the project to videos and GIFs. First, we break down the black and white videos into individual frames, colorize the frames and obtain the colorized video. The colorized frames are also stylized by the style transfer model. The stylized frames are then recombined to obtain the stylized videos. The main challenge with applying stylization on videos is that we lost the inter-frame data when stylization was applied to consecutive frames which resulted in the flickering effect as seen in the gifs below. Subtle transitions and changes in successive frames were lost during the stylization process. Furthermore, successive frames sometimes get different colors for the same components depending on the style transfer algorithm. This extended section of our project has a lot of scope for improvement and we plan to work on it in future. Following are the code snippets and examples for this experiment:

This part of the code loops through the frames in the video that were generated using cv2. VideoCapture() and while we have the frames in the folder, the style-transfer method is applied

on each individual frame. The stylized frames are stored in a folder and then are later combined as a video or gif using cv2.VideoWriter().

```
# converts video to image frames
import cv2
vidcap = cv2.VideoCapture('/content/drive/My Drive/BigDataProject/lion.mp4')
success,image = vidcap.read()
count = 1
while success:
   # save frame as JPG file
  cv2.imwrite("/content/drive/My Drive/BigDataProject/frames/image_frame%d.jpg" % count, image)
  success, image = vidcap.read()
  count+=1
```

(a) Code snippet to split video into frames

```
from torchvision.utils import save_image
import torch.nn.functional as nnf

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
imsize = 512

loader = transforms.Compose([
    transforms.Resize([imsize, imsize]),  # scale imported image
    transforms.ToTensor()])  # transform it into a torch tensor


def image_loader(image_name):
    image = Image.open(image_name).convert('RGB')
    # fake batch dimension required to fit network's input dimensions
    image = loader(image).unsqueeze(0)
    # print(image.size())
    return image.to(device, torch.float)

style_img = image_loader("/content/drive/My Drive/BigDataProject/style_image.jpg")

count = 0
while True:
  cnn = models.vgg19(pretrained=True).features.to(device).eval()
  content_layers_default = ['conv_4']
  style_layers_default = ['conv_1', 'conv_2', 'conv_3', 'conv_4', 'conv_5']
  normalization_mean = torch.tensor([0.485, 0.456, 0.406]).to(device)
  normalization_std = torch.tensor([0.229, 0.224, 0.225]).to(device)
  content_image = image_loader('/content/drive/MyDrive/BigDataProject/fox_frames/frame_%d.jpeg' % count)
  input_img = content_image.clone()
  image = run_style_transfer(cnn, normalization_mean, normalization_std,
                      content_image, style_img, input_img, num_steps=200,
                      style_weight=1000000, content_weight=1)

  save_image(image,"/content/drive/My Drive/BigDataProject/fox_try2/fox_style_%d.jpg" % count)     # save frame as JPEG file
  count += 1
  print("On image %d" % count)


Building the style transfer model..
Optimizing..
```

(b) Code snippet to perform style transform each frame

```
import cv2
import numpy as np
from PIL import Image

# choose codec according to format needed
fourcc = cv2.VideoWriter_fourcc(*'mp4v')
video=cv2.VideoWriter('stylized_fox.avi', fourcc, 24,(1024,1024))

for j in range(0,353):
    img = cv2.imread('/content/drive/My Drive/BigDataProject/fox_try2/fox_style_'+str(j)+'.jpg')
    frame = cv2.resize(img, (1024,1024))
    video.write(frame)

cv2.destroyAllWindows()
video.release()
```

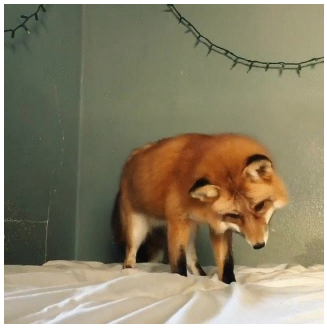
(c) Code snippet to merge the stylized frames into a video

Figure 44: Code snippet of colorization and style transfer on videos and GIFs

8.1 Style Transfer On GIFs

Example 1:

Original Video: https://bit.ly/original_fox_video
Stylized Video: https://bit.ly/stylized_fox_video



(a) Colored Frame

(b) Style Image

(c) Stylized Frame

Figure 45: Style transfer on fox video

We also attempted to perform stylization on a short video. This was done by extracting individual frames and applying style transfer on each frame. Since interframe dependency causes us to lose metadata, a smooth transition between the frames is lost. If we observe two consecutive frames of the stylized video, we would notice that there is a difference in texture and color between the frames, because they are being processed as frames and not the entire video. The video has 352 frames in total, with 21 frames per second speed. It took about an hr time to run the model for this 14 second video using Google Colab with GPU enabled. Total 200 epochs were run for each frame with style weight being 1M and content weight being 10. The frame splitting and recombination on an average took 1.1 and 1.3 seconds per frame. And the average

per frame processing time is 10.79 seconds, a major chunk of which is consumed by stylization(9.4 seconds).

9. Overall Results

9.1 Colorization + Style Transfer Examples

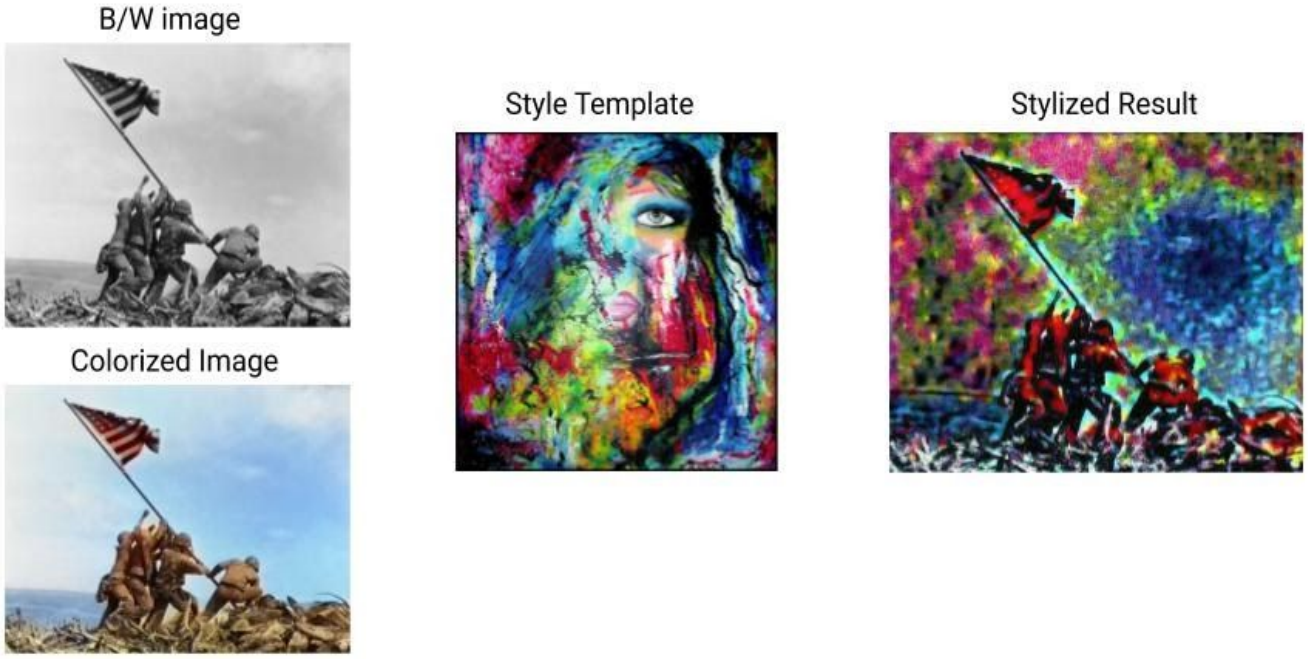


Figure 46: Colorization and style transfer on WW2 image

The black and white image present here is from WW2, dating back to the year 1945 and there is no actual colorized version of this image available. On colorization, we obtained the image as shown below. And after applying the artwork here as the style template, we obtained this stylized result.

Figure 47: Colorization and style transfer on Grand Central Station, New York image

Similarly, this image of the Grand central station has no colorized version. We apply colorization and style transfer on this image to obtain the above results.

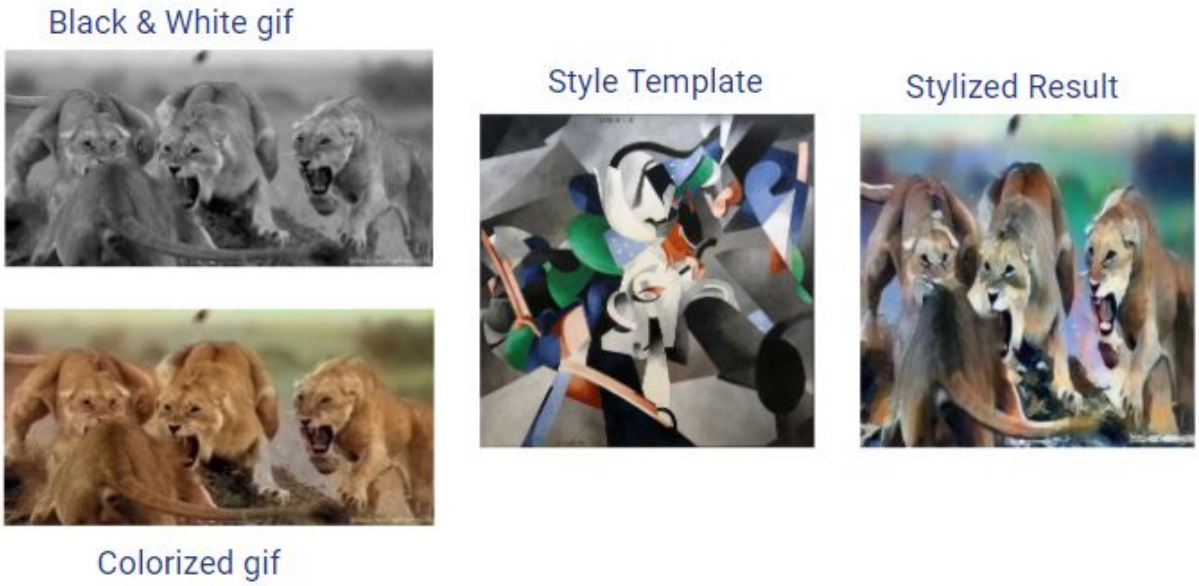


Figure 48: Colorization and style transfer on lion attack GIF

Original Video: https://bit.ly/lion_blackwhite
Colorized Video: https://bit.ly/lion_colorized
Stylized Video: https://bit.ly/lion_stylized

In this example as shown in Figure 48, we have taken a black and white GIF of three lion cubs attacking another lion cub and first colorized it frame by frame. Later we have applied stylization frame by frame and combined them to obtain the stylized GIF output.

10. Application: Digital Art

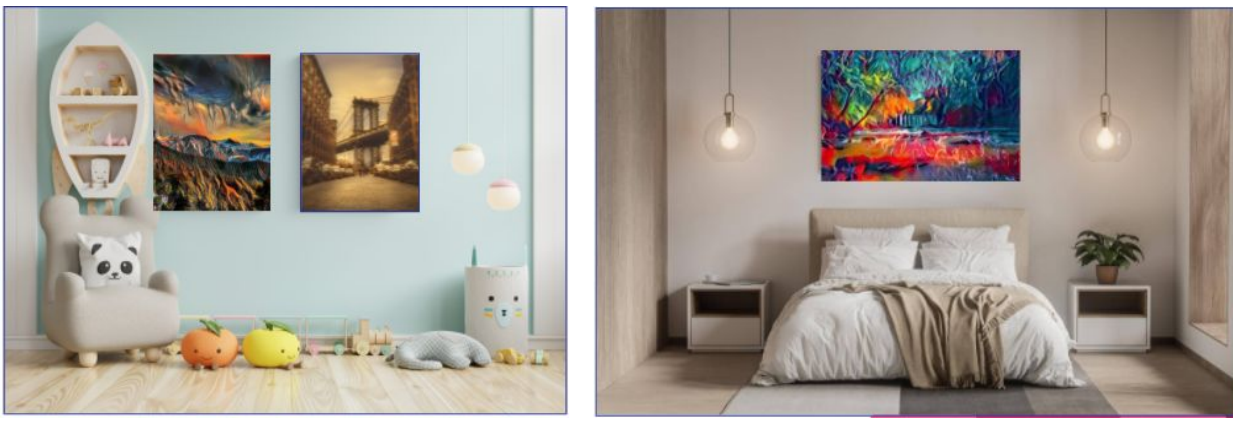


Figure 49: Digital art using colorization and style transfer

There is growing interest in style tools that enhance digital art. We can create art pieces using this project to decorate our home or office spaces as shown in Figure 49.

11. Areas of Improvement

We identified a few areas of improvement for our project. Colorization can match most generic objects. If trained on a larger dataset consisting of a high number of objects, we could produce better results. Adding inter frame correlation to improve style transfer applied on videos is important as without inter-frame correlation, there were several unwanted artifacts. Multiple forms of evaluation and performing benchmarking on the results obtained is another area to venture into. The user could have the ability to dynamically set the relative weight of the style and content.

12. Conclusion and Future Work

The current architecture for colorization not just works on color, but is also useful to detect objects, classify and segment them. For example, a tree is correctly recognized, thus colored brown and green in the results. Future work could involve modifying the models to run on lower compute devices like mobiles and tablets. Colorization could also be improved by user interactive approaches which is not part of our current methods. Colorization and the style transfer networks can be used for image augmentation tasks. Style can be transferred to semantic segmentations in the content image without including the color.

13. References

[1] Xiao, Xuezhong & Ma, Lizhuang. (2006). Color transfer in correlated color space. Proceedings - VRCIA 2006: ACM International Conference on Virtual Reality Continuum and its Applications. 305-309. 10.1145/1128923.1128974.

[2] Khalil, Yazen & Muhammad Ali, Peshawa. (2013). “A Proposed Method for Colorizing Grayscale Images”.

[3] L. A. Gatys, A. S. Ecker and M. Bethge, "Image Style Transfer Using Convolutional Neural Networks," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, 2016, pp. 2414-2423, doi: 10.1109/CVPR.2016.265.

[4] Y. Jing, Y. Yang, Z. Feng, J. Ye, Y. Yu and M. Song, "Neural Style Transfer: A Review," in IEEE Transactions on Visualization and Computer Graphics, doi: 10.1109/TVCG.2019.2921336.

[5] Zhang, R., Zhu, J., Isola, P., Geng, X., Lin, A.S., Yu, T., & Efros, A.A. (2017). Real-time user-guided image colorization with learned deep priors. *ArXiv, abs/1705.02999*.

[6] Zhang, R., Isola, P., & Efros, A.A. (2016). Colorful Image Colorization. *ECCV*.

[7] B. S. Thomas, R. Dogra, B. Dixit and A. Raut, "Automatic Image and Video Colourisation using Deep Learning," *2018 International Conference on Smart City and Emerging Technology (ICSCET)*, Mumbai, 2018, pp. 1-4, doi: 10.1109/ICSCET.2018.8537308.

[8] Jing, Y., Yang, Y., Feng, Z., Ye, J., & Song, M. (2019). Neural Style Transfer: A Review. IEEE transactions on visualization and computer graphics.

[9] Haochen Li, “A Literature Review of Neural Style Transfer”, pp. 4321-4329.

[10] Zhang, Richard, et al. "Real-time user-guided image colorization with learned deep priors." arXiv preprint arXiv:1705.02999 (2017).

[11] Antic, Jason. “Jantic/DeOldify.” GitHub, 2018, github.com/jantic/DeOldify