NEURAL STYLE TRANSFER

B.Tech Project-II Report

Submitted in partial fulfillment of the requirements for the award of the degree of

Bachelor of Technology

In

Computer Science & Engineering

Submitted By

Raghav Punjani (2K17/CO/249)

Rakshit (2K17/CO/259)

Ritik Handa (2K17/CO/271)

with the guidance of

Mrs. Abhilasha Sharma, Assistant Professor,

COMPUTER SCIENCE & ENGINEERING,

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

MAY 2021

CANDIDATE’S DECLARATION

We, Raghav Punjani (2K17/CO/249), Rakshit (2K17/CO/259) and Ritik Handa (2K17/CO/271), students pursuing Bachelors in Technology in Computer Engineering, hereby declare that the project dissertation titled “Neural Style Transfer” which we submit to the Department of Computer Engineering, Delhi Technological University, Delhi in partial fulfillment of the requirement for the award of the degree of Bachelor of Technology in Computer Engineering is original and not directly reproduced from any source without proper credit. This work has not erstwhile formed the foundation for the award of any Degree, Diploma Associateship, Fellowship, or other similar title or recognition.

[illegible]

Raghav Punjani (2K17/CO/249)

Rakshit

Rakshit (2K17/CO/259)

Ritik Handa

Ritik Handa (2K17/CO/271)

Certificate

This is to certify that Project Report entitled Neural Style Transfer submitted by *Raghav Punjani (2K17/CO/249)*,*Rakshit (2k17/CO/259)* and *RitikHanda (2K17/CO/271)* for partial fulfillment of the requirement for the award of degree Bachelors of Technology in Computer Engineering is a record of the candidate work carried out under my supervision.

Abhilasha Sharma

Mrs Abhilasha Sharma, Assistant Professor,
Project Guide,
Department of Computer Science and Engineering, Delhi Technological University

ACKNOWLEDGEMENT

I would like to express my gratitude and appreciation to all those who gave me the support to complete this project. A special thanks to our mentor and project guide, **Mrs. Abhilasha Sharma**, Assistant Professor, Delhi Technological University.

Raghav

(Raghav Punjani)

Rakshit

(Rakshit)

Ritik Handa

(Ritik Handa)

Abstract

This paper uses machine learning programs to transfer everyday images to art style images by using a style image to get the transferred style and a content image we want to transfer the style onto. This project can show how machine learning techniques do well in the use of images and how artificial intelligence can achieve great success in art and creativity. Current style transfer techniques such as Prisma emphasized more on the color and layouts of style image. Hence, the synchronized image might lose color, and the content image's outline gets distorted. This project explores a process that can preserve content images' elements and apply artistic styles from style images. The neural style algorithm is very successful in discovering the style of famous pieces of art. There have been attempts to cover this style in images on various levels of success.

Contents

1. Insight

   1.1 Introduction

   1.2 Convolutional Neural Networks(CNN)

   1.3 Problem Statement

2. Applications

   2.1 User-assisted Creation Tool s

   2.2 NST on Audio

   2.3 Augmenting Medical Data

3. Methodology

4. Implementation and Results

   4.1 Implementation

   4.2 Results

5. Future Challenges

   5.1 Evaluation Methodology

   5.2 Interpretable Neural Style Transfer

   5.3 Three-way Trade-off in Neural Style Transfer

6. Conclusion

7. References

List of Tables

Table 1: Iteration – current loss value table
Table 2: Iteration – improvement percentage table

List of Figures

Fig 1: An example of Neural Style Transfer
Fig 2: Typical Structure of Convolutional Neural Network
Fig 3: Audio Style Transfer Framework
Fig 4: CNN layer and Style Matrix
Fig 5: Current loss value vs iteration graph
Fig 6: Improvement Percentage vs iteration graph

1. Insight

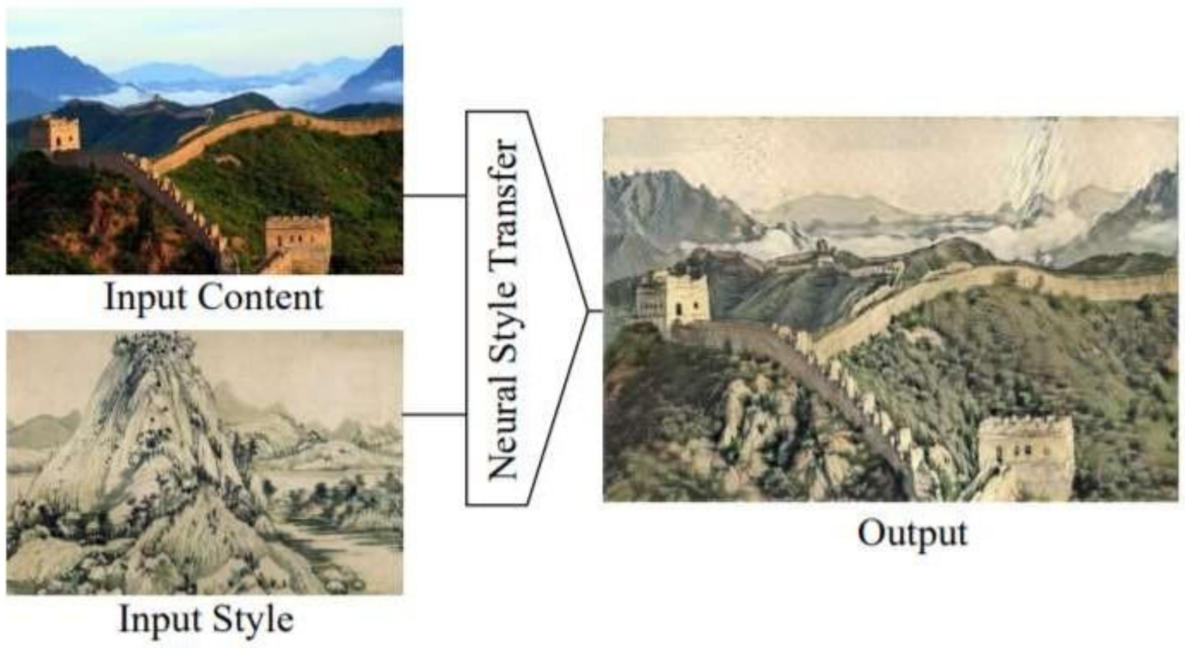
1.1 Introduction

Painting is a popular art form. For more than Thousand years, people are fascinated by the art of painting with the appearance of many attractive paintings, e.g. Van Gogh's "Starry Night" Previously, re-painting in a different style required a highly trained artist and a lot of time[1].

Since the '90s, the art theories behind the appealing artworks have been attracting the attention of computer science researchers along with the artists. There are a plethora of studies and techniques analyzing how to convert the images into synthetic artworks on their own[1]. Among these studies, the advances in the non-photorealistic rendering are influencing, but now it has become established in the community of computer graphics.

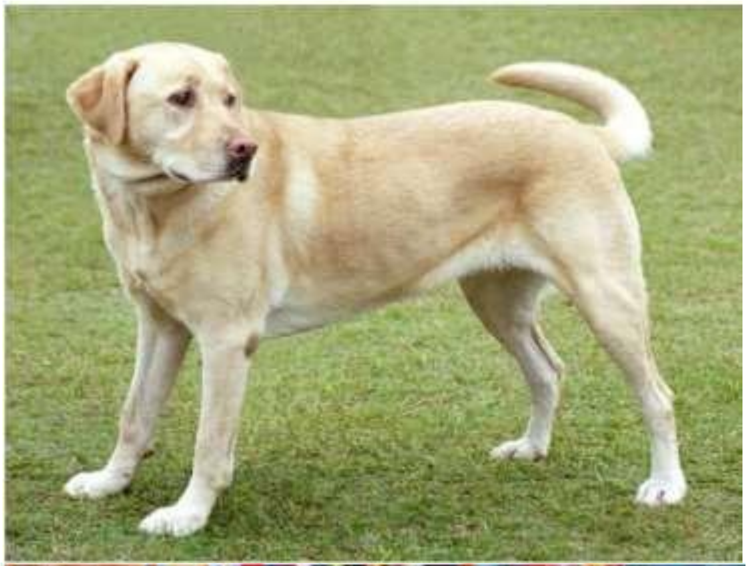
However, most of these NPR style algorithms are designed for different art styles and cannot be easily transferred to other styles. Style transfer is often studied as a common problem of the integration of textures in the computer vision community, which is to extract and transfer text from source to target[1]. Hertzmann et al. re-suggest the framework for making a universal style transfer by reading the same changes from the example provided by a pair of static images and styles called image analogies[1]. However, the common shortcoming of these methods is that they often fail to capture the image structures effectively and use only low-level image features.

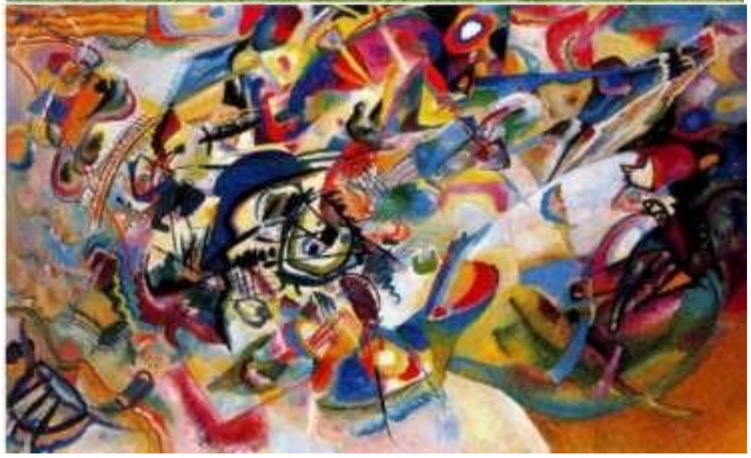
Figure 1- An example of Neural Style Transfer.



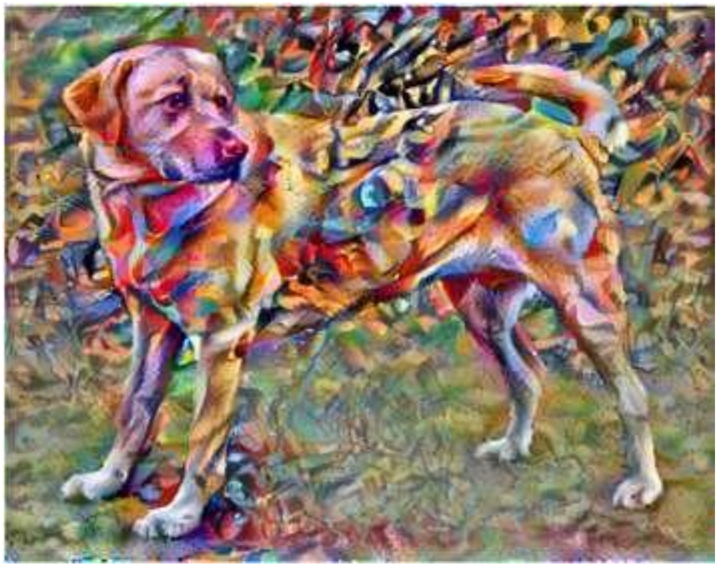
Neural-style transfer is a fine-tuning technique that takes a content image which is like a base image that will act as the basic structure of the new generated image and a style reference image which is basically the structure that we will be applying on the base image and finally we will merge the two images together. Output image will look like a content image that is painted in style reference.

For example, let's take an image of this dog and Wassily Kandinsky's Composition 7:





Now if Kandinsky decided to paint the picture of this Dog exclusively with this style. The image would look something like this.



1.2 Convolutional Neural Networks(CNN)

CNN is a category of deep neural networks, often used to analyze visual images. It is a part of artificial neural networks that has a commanding method in computer vision tasks since the astounding results that were shared on the object recognition competition known as the ImageNet Large Scale Visual Recognition Competition (ILSVRC) in 2012. Expert-level performances in various fields have been achieved by CNN.

The image transformations are possible thanks to advances in computing processing power that allowed the usage of more complex neural networks.

This is the typical structure of a Convolutional Neural Network:

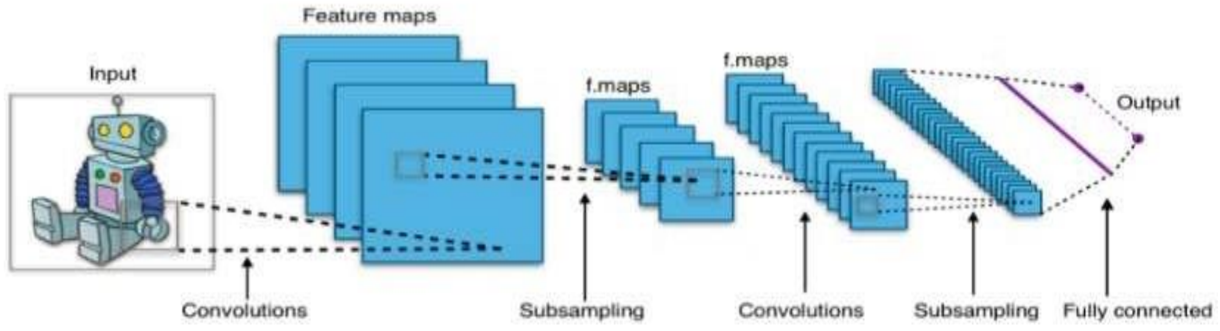
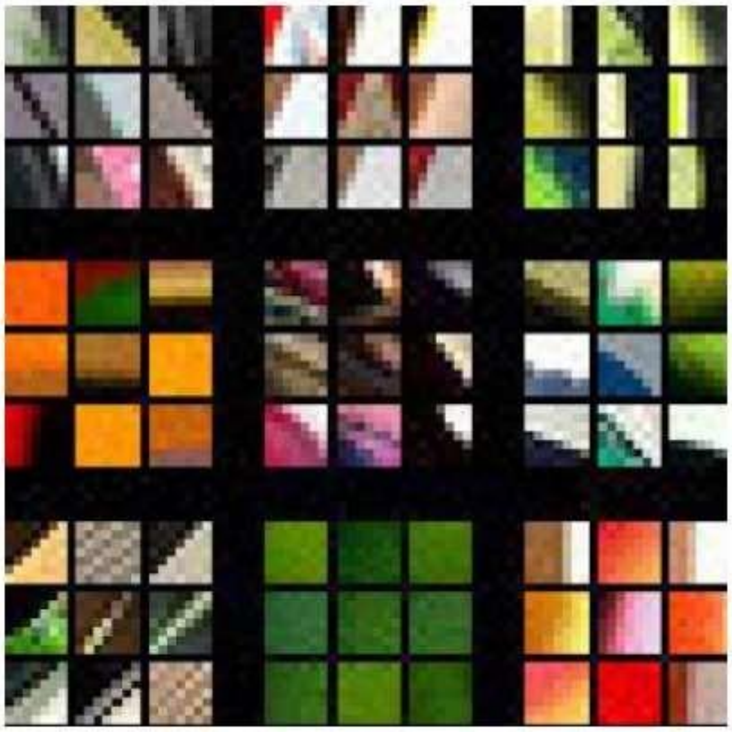
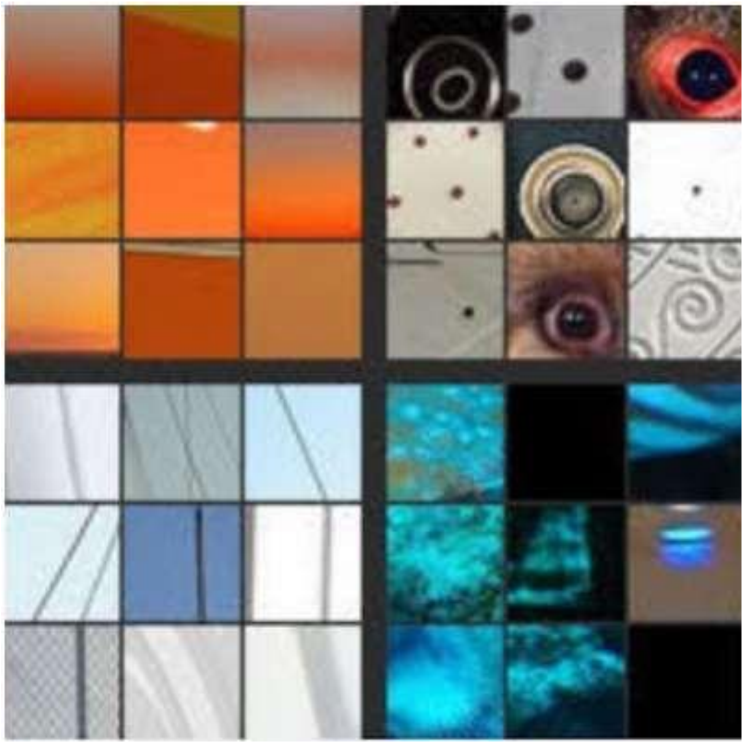


Figure 2.

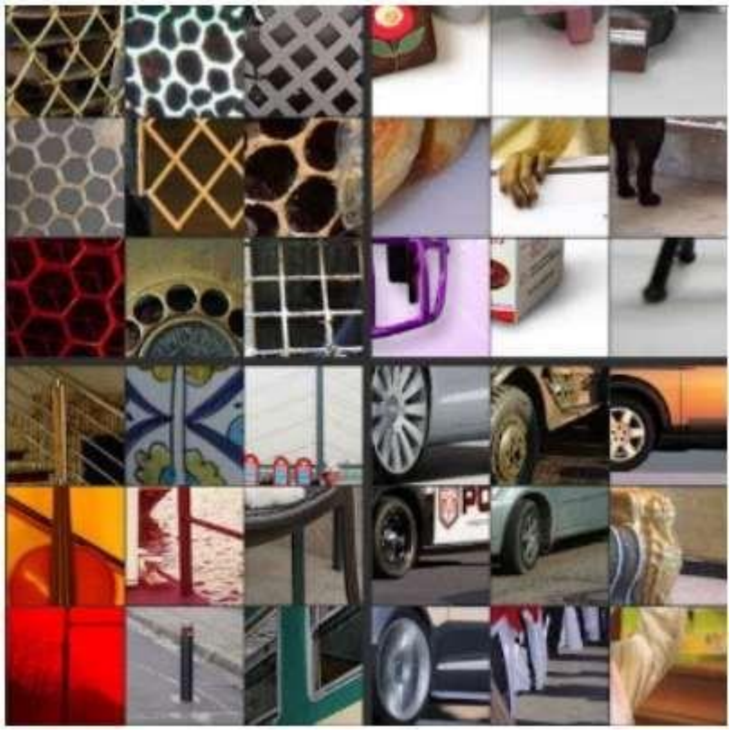
The first layers detect the most basic features of the image like edges.



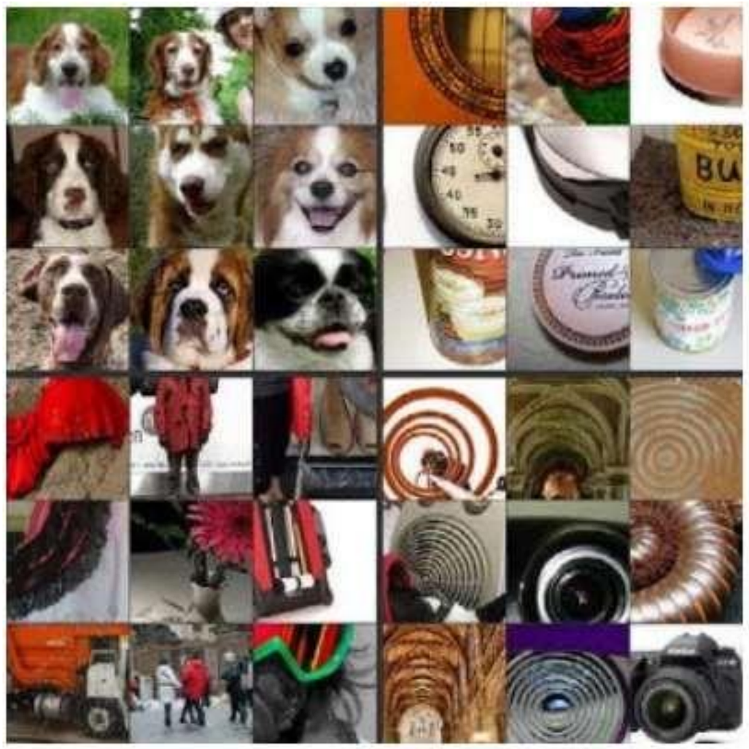
The next layers combine the information of the previous layer to detect more complex features like textures.



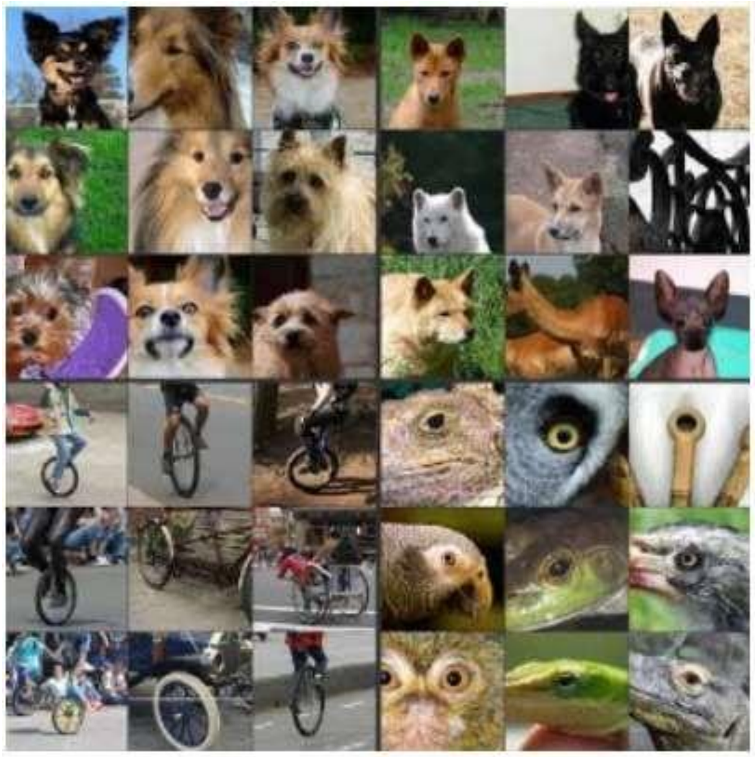
Following layers, continue to use the previous information to detect features like repetitive patterns.



The latest network layers are able to detect complex features like object parts.



The final layers are capable of classifying complete objects present in the image.



The possibility of detecting complex image features is the key enabler to perform complex transformations to those features, but still perceiving the same content in the image.

1.3 Problem Statement

Here we have to present an installation program based on the Deep Neural Network that creates high-quality art images for comprehension. The system has to use neural presentations for the separation and re-integration of style and content of arbitrary images, giving us an algorithm based upon a neural network for the creation of artistic images. Moreover, due to the striking similarities between the improved performance of neural networks and biological perspectives, our work should provide a way to convey the perception of art by humans and their way of thinking about it.

When learning about object recognition, we have to make sure that the network becomes invariant to all image variation that preserves object identity. Representations that factorize the variation in the content of an image and the variation in its appearance would be efficient for this task. Thus, our ability to abstract content from style and, therefore, our ability to create and enjoy art might be primarily a preeminent signature of the powerful inference capabilities of our visual system.

2. Applications

As a result of the visible visual effects, NST research has led to many successful industrial applications and began to bring commercial benefits. In this section, we summarize these programs and present some of the alternatives.

2.1 User-assisted Creation Tools

NST can run as user-assisted creative tools. Although there are no popular programs that use the NST process in innovative tools, they are believed to have fair use in the future.
As a creative tool for designers and artists, the NST can make it much easier for the artist to create a work of art of a particular style, especially when it comes to computer- generated construction of art. Moreover, with NST algorithms, it is a small product made in the manner of fashion designers too. Stylish CAD drawings for various road builders, which will be more expensive if you create them by hand.

2.2 NST on Audio

Neural style sound transmission has applications in the music industry. Music made using AI is very popular these days. This algorithm can be used to produce new music by lovers and industry experts. New songs can be produced by recording sounds like content and music tone as a style. This can be made into an Android app (like Prisma *), where artists can record their sound and apply any style of music to them.

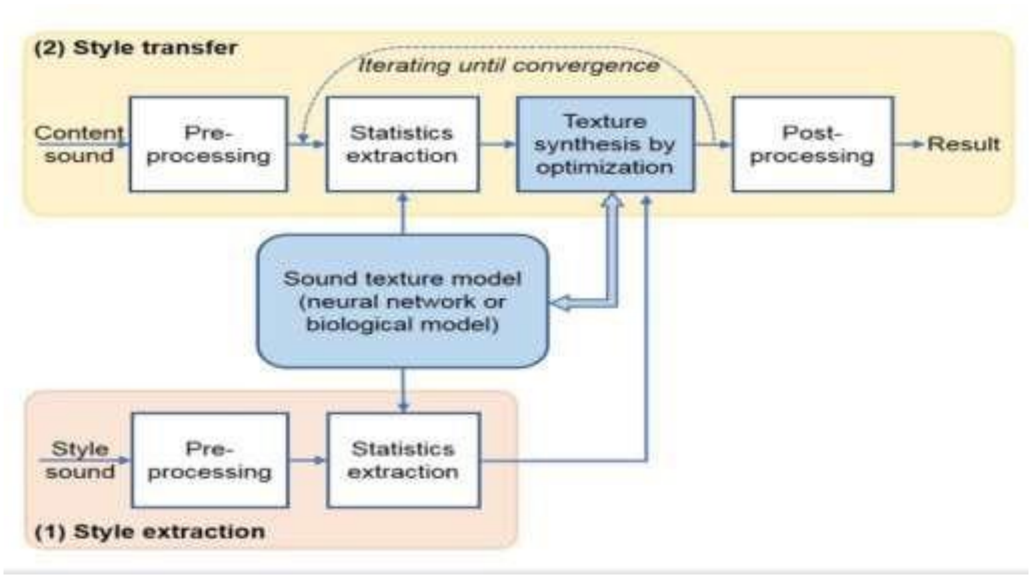


Figure 3: Audio Style Transfer Framework

2.3 Augmenting Medical Data

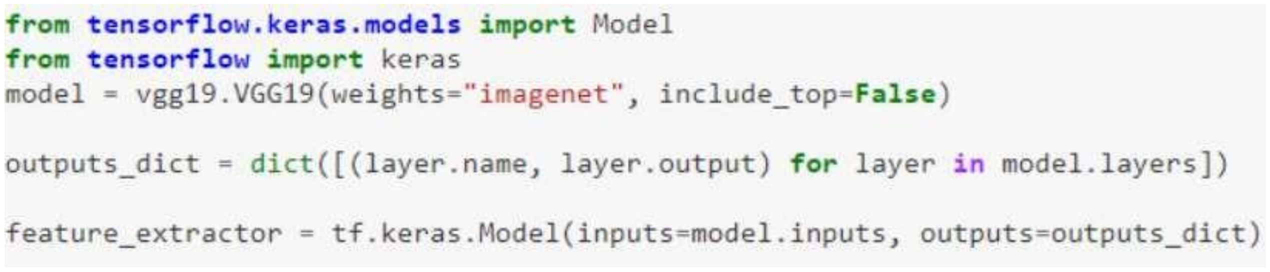
TensorFlow is a powerful tool to develop any machine learning pipeline. This model explores the concept of using neural style transfer to augment medical images. Medical images are rarely ever processed in the same way. Different institutions, different machines, different technicians, and many other factors lead to variation within medical scans. These variations can impact how well an ML model can learn from given data. If images processed by institution #1 work better with the model than images processed by institution #2, we would want our institution #2 images to look more like institution #1 images.

On a very high level, neural style transfer is used to compose an image in the style of another using deep learning. It has been observed that neural style transfer does indeed improve scores for images processed by lower-scoring methods.

3. Methodology

We will use pre trained networks of convolutional networks that is a category of deep neural networks, often used to analyze visual images.

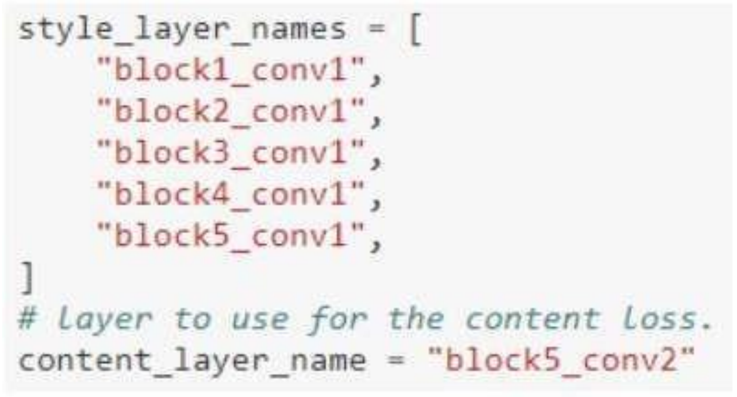
To minimize the work, we must use libraries like Keras and imply it on our problems. It is used for transfer learning and we can load the model as shown below.



```
from tensorflow.keras.models import Model
from tensorflow import keras
model = vgg19.VGG19(weights="imagenet", include_top=False)

outputs_dict = dict([(layer.name, layer.output) for layer in model.layers])

feature_extractor = tf.keras.Model(inputs=model.inputs, outputs=outputs_dict)
```

Initially we import the libraries. Then the vgg19 model is downloaded where include_top=False shows that we do not want the last layer of softmax, which is the output layer used to divide 1000 classes in this competition.

Finally a dictionary is created which stores key as the name of layer and value as the layer outputs. We then defined our model by including it as an indication of input and VGG results as a dictionary for each layer. Next, we will describe the layers from which we will extract content elements and styles.



```
style_layer_names = [
    "block1_conv1",
    "block2_conv1",
    "block3_conv1",
    "block4_conv1",
    "block5_conv1",
]
# layer to use for the content loss.
content_layer_name = "block5_conv2"
```

We already have a dictionary where we can lay out these layers and extract the results.

Loss Functions

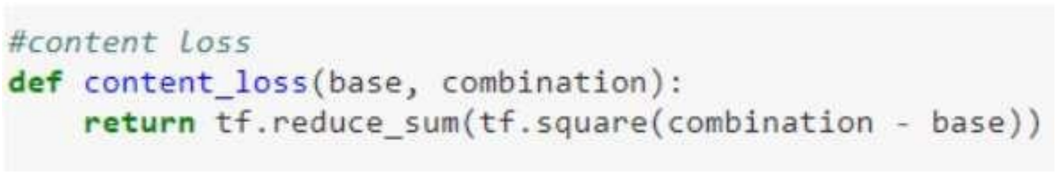
Finding the image you want will define the loss function that will help us to get the desired result. per pixel losses concept will be applied here. Per Pixel Loss is a measuring unit used to understand the difference between images at pixel level. It compares pixel output values with input values. Sometimes the loss of pixels has its problems in terms of representing all logical features. This is where the permanent loss comes into play[17]. following are the two main loss terms-

1.Content Loss

2.Style Loss

Content Loss

It ensures that the content we are looking for in our result is well received. It has been shown that each pixel value is being focused at lower levels while CNN captures information about content at high levels of the network.

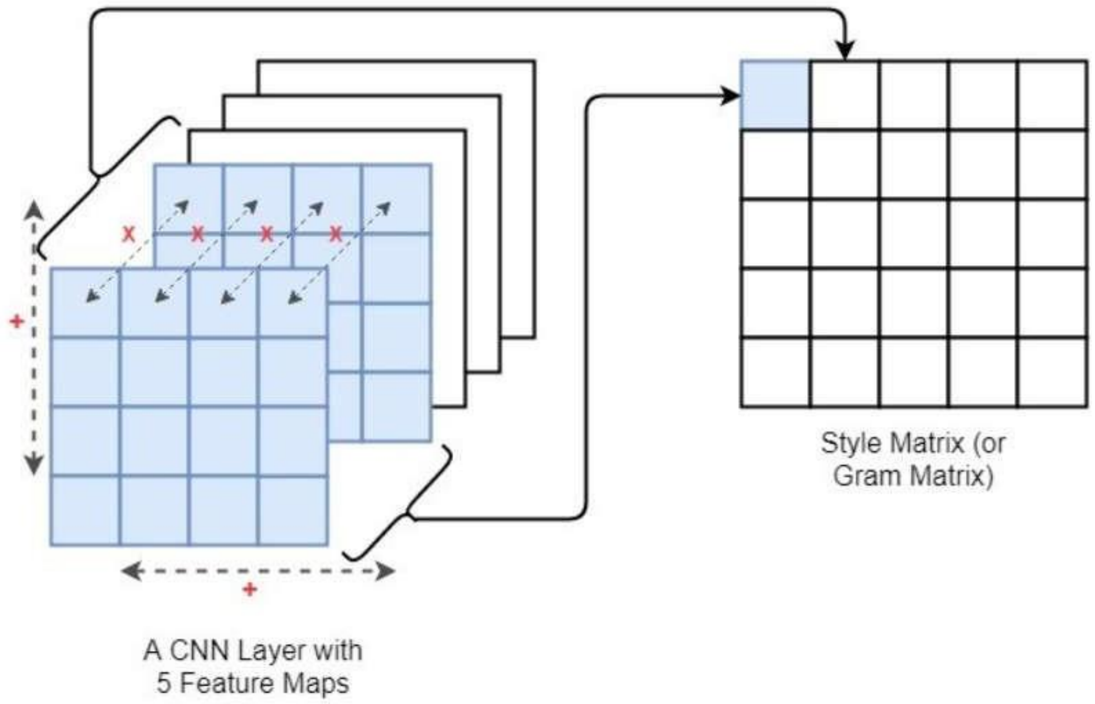


```python
#content loss
def content_loss(base, combination):
    return tf.reduce_sum(tf.square(combination - base))
```

Here the features of the content are basic while the features of the output image that are produced are composite. Here reduce_sum includes the sum of objects in all the specified parameters, in this case, the corresponding pixel difference between the input and the generated image[17].

Style Loss

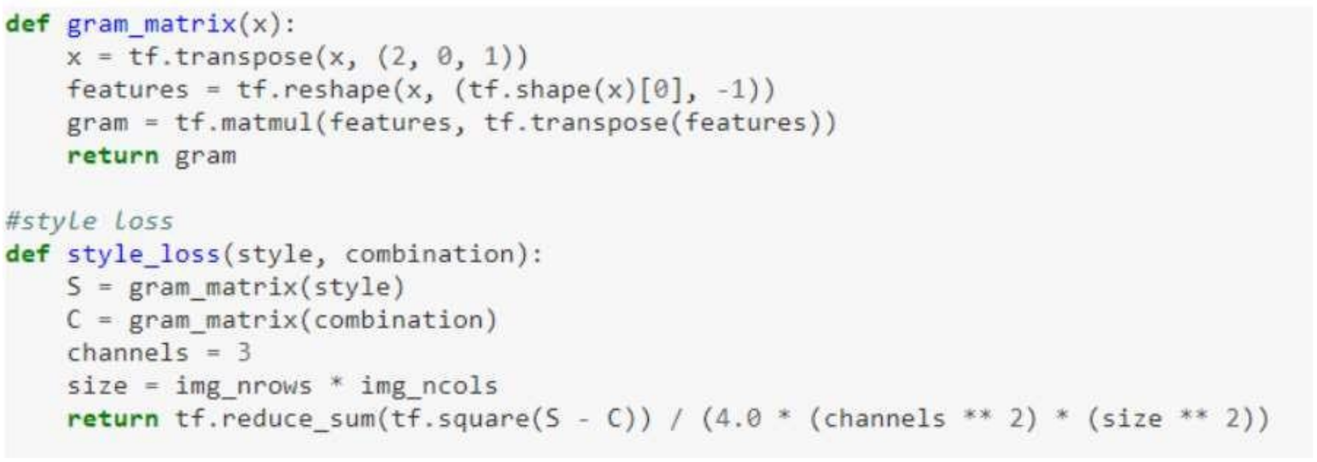
Many layers are involved in calculating the loss function for style which makes it difficult to define it as compared to content. Style details are measured as the number of intersections available between feature maps for each layer. Here we apply the GM to compute the loss of style.



+
+
Style Matrix (or Gram Matrix)
A CNN Layer with 5 Feature Maps

Figure 4

So now we have to do the style matrices and the result matrices using GM and calculate the difference between them. Gij is the product of the ith and jth layer map and is summarized in height and width as shown above[17].



```
def gram_matrix(x):
    x = tf.transpose(x, (2, 0, 1))
    features = tf.reshape(x, (tf.shape(x)[0], -1))
    gram = tf.matmul(features, tf.transpose(features))
    return gram

#style loss
def style_loss(style, combination):
    S = gram_matrix(style)
    C = gram_matrix(combination)
    channels = 3
    size = img_nrows * img_ncols
    return tf.reduce_sum(tf.square(S - C)) / (4.0 * (channels ** 2) * (size ** 2))
```

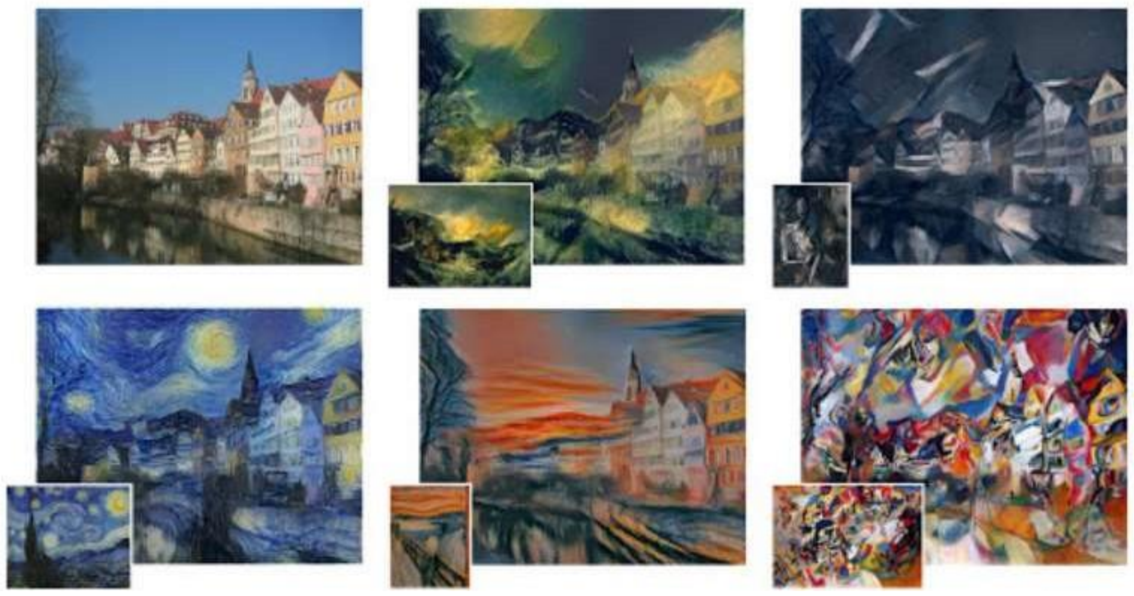
Now both the loss functions are computed. We will have to compute a weighted summation of both the style losses and computed content to calculate the final loss.The final loss is defined as,

$$L = \alpha L_{content} + \beta L_{style} ,$$

where α and β are user-defined hyperparameters[15]. We can control the percentage of content and style image being incorporated into the desired image by carrying these hyperparameters[15].
The code above is the final combination of losses by crossing the layers and taking a weighted summary from the last second line to calculate the final loss. Finally, we will have to define an optimizer (Adam or SGD) that will optimize network losses.
Finally we will have a working Neural Style Transfer System and we will be able to get the desired artistic image using content image and the style image.

Output

The output images will be as shown below:



4. Implementation and Results

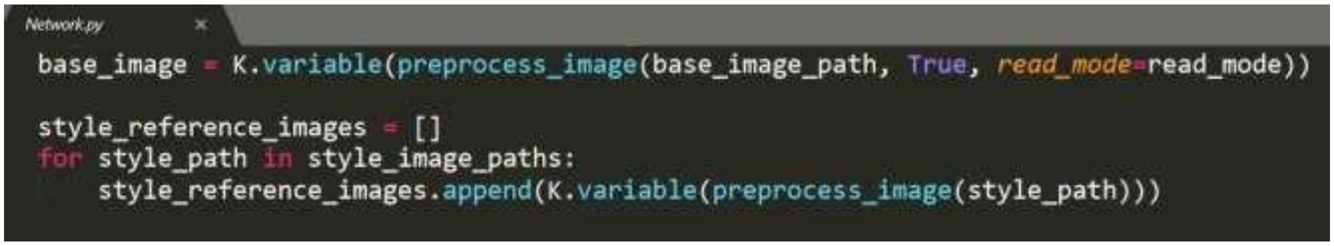
4.1 Implementation

Neural Style Transfer

Network.py is used for the Neural Style Transfer. It takes the content image and style image as input and merge the style into the content image to form the result image.

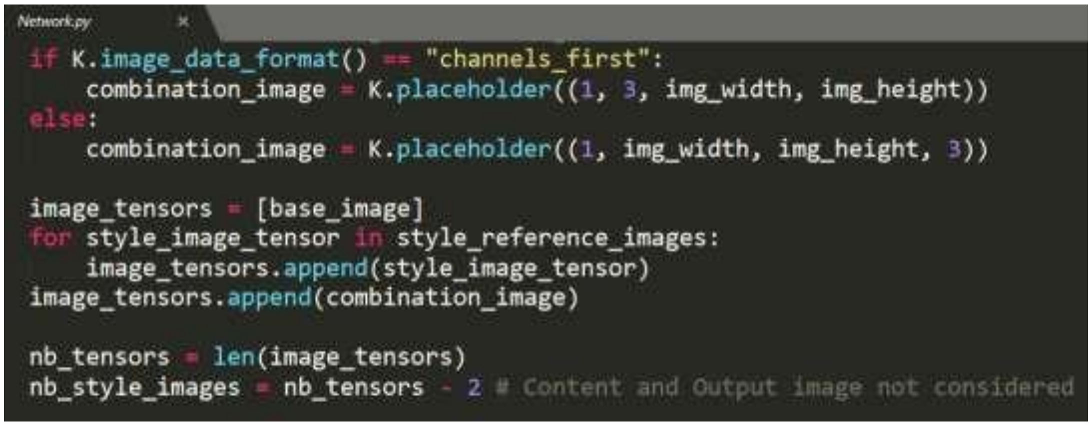
python network.py/inetwork.py "/path/to/content image" "path/to/style image" "result prefix or /path/to/result prefix"

- First get a Tensor representation of our images.



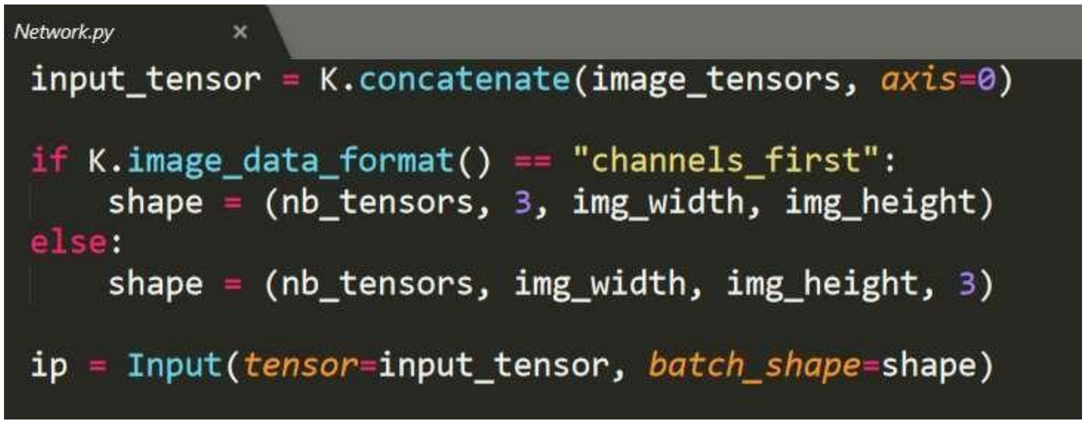
```
base_image = K.variable(preprocess_image(base_image_path, True, read_mode=read_mode))

style_reference_images = []
for style_path in style_image_paths:
    style_reference_images.append(K.variable(preprocess_image(style_path)))
```

- The following code contains our generated image.



```
if K.image_data_format() == "channels_first":
    combination_image = K.placeholder((1, 3, img_width, img_height))
else:
    combination_image = K.placeholder((1, img_width, img_height, 3))

image_tensors = [base_image]
for style_image_tensor in style_reference_images:
    image_tensors.append(style_image_tensor)
image_tensors.append(combination_image)

nb_tensors = len(image_tensors)
nb_style_images = nb_tensors - 2 # Content and Output image not considered
```

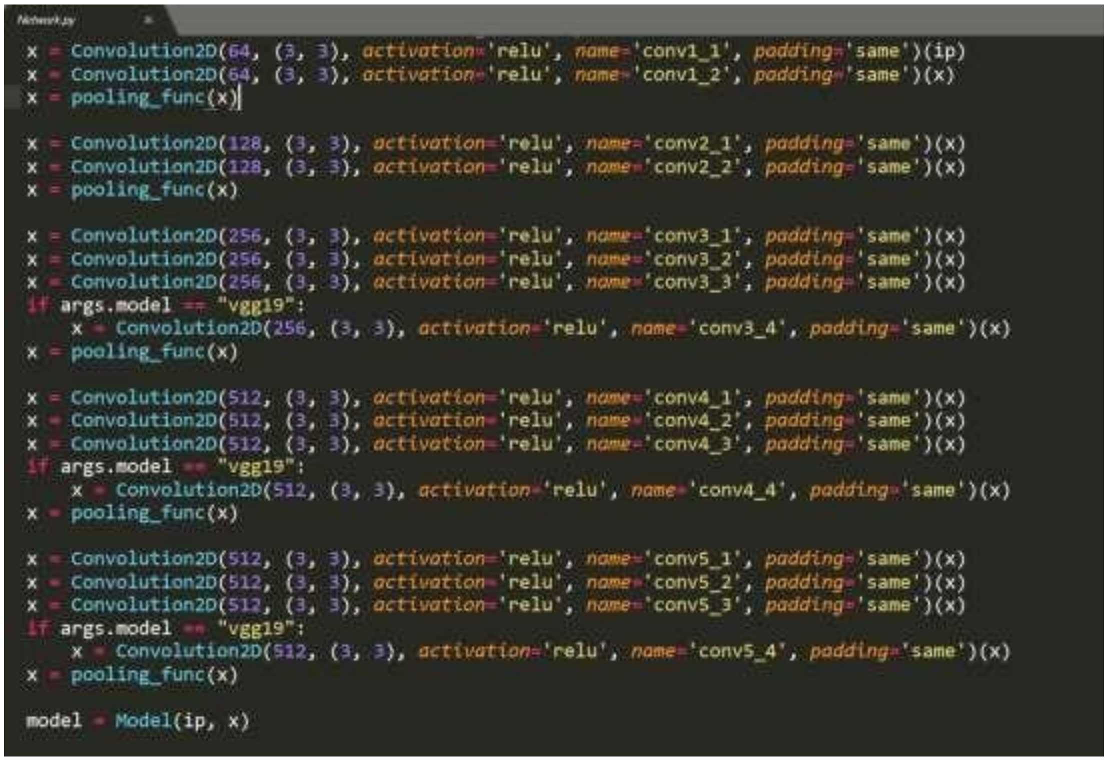
- Now combine various images into a single Keras Tensor



Network.py

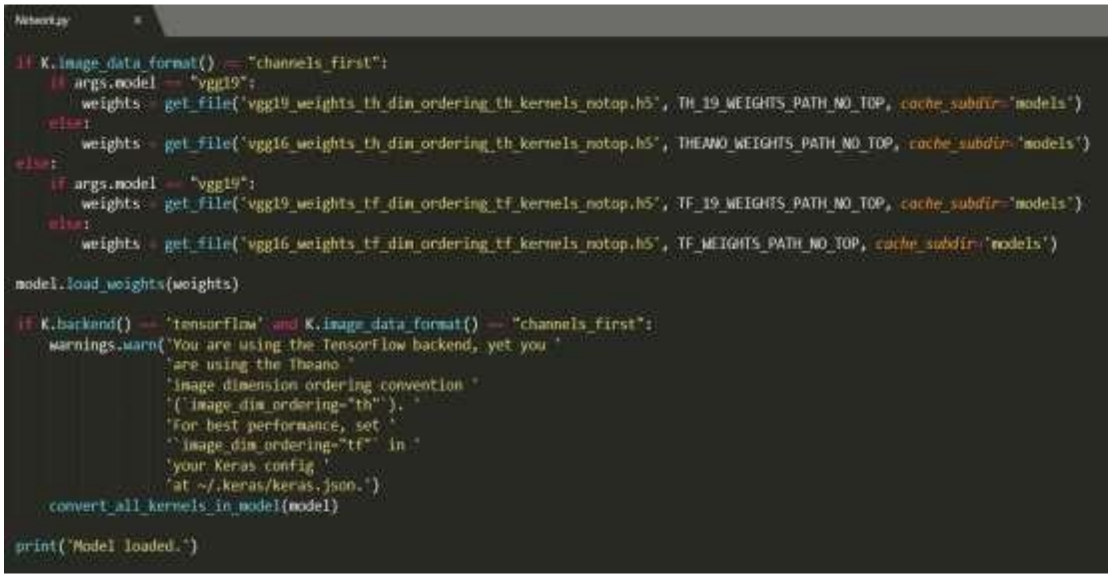
```python
input_tensor = K.concatenate(image_tensors, axis=0)

if K.image_data_format() == "channels_first":
    shape = (nb_tensors, 3, img_width, img_height)
else:
    shape = (nb_tensors, img_width, img_height, 3)

ip = Input(tensor=input_tensor, batch_shape=shape)
```

- VGG network with our 3 images as input



Network.py

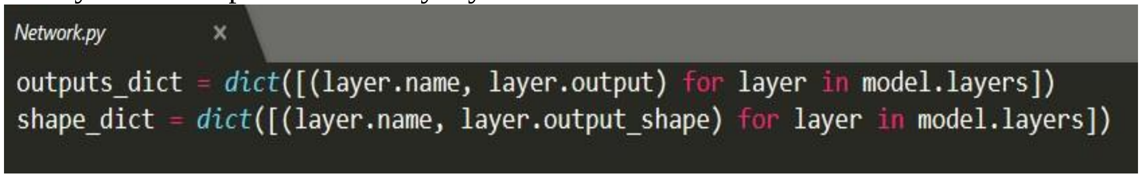
```python
x = Convolution2D(64, (3, 3), activation='relu', name='conv1_1', padding='same')(ip)
x = Convolution2D(64, (3, 3), activation='relu', name='conv1_2', padding='same')(x)
x = pooling_func(x)

x = Convolution2D(128, (3, 3), activation='relu', name='conv2_1', padding='same')(x)
x = Convolution2D(128, (3, 3), activation='relu', name='conv2_2', padding='same')(x)
x = pooling_func(x)

x = Convolution2D(256, (3, 3), activation='relu', name='conv3_1', padding='same')(x)
x = Convolution2D(256, (3, 3), activation='relu', name='conv3_2', padding='same')(x)
x = Convolution2D(256, (3, 3), activation='relu', name='conv3_3', padding='same')(x)
if args.model == "vgg19":
    x = Convolution2D(256, (3, 3), activation='relu', name='conv3_4', padding='same')(x)
x = pooling_func(x)

x = Convolution2D(512, (3, 3), activation='relu', name='conv4_1', padding='same')(x)
x = Convolution2D(512, (3, 3), activation='relu', name='conv4_2', padding='same')(x)
x = Convolution2D(512, (3, 3), activation='relu', name='conv4_3', padding='same')(x)
if args.model == "vgg19":
    x = Convolution2D(512, (3, 3), activation='relu', name='conv4_4', padding='same')(x)
x = pooling_func(x)

x = Convolution2D(512, (3, 3), activation='relu', name='conv5_1', padding='same')(x)
x = Convolution2D(512, (3, 3), activation='relu', name='conv5_2', padding='same')(x)
x = Convolution2D(512, (3, 3), activation='relu', name='conv5_3', padding='same')(x)
if args.model == "vgg19":
    x = Convolution2D(512, (3, 3), activation='relu', name='conv5_4', padding='same')(x)
x = pooling_func(x)

model = Model(ip, x)
```



Network.py

```python
if K.image_data_format() == "channels_first":
    if args.model == "vgg19":
        weights = get_file('vgg19_weights_th_dim_ordering_th_kernels_notop.h5', TH_19_WEIGHTS_PATH_NO_TOP, cache_subdir='models')
    else:
        weights = get_file('vgg16_weights_th_dim_ordering_th_kernels_notop.h5', THEANO_WEIGHTS_PATH_NO_TOP, cache_subdir='models')
else:
    if args.model == "vgg19":
        weights = get_file('vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5', TF_19_WEIGHTS_PATH_NO_TOP, cache_subdir='models')
    else:
        weights = get_file('vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5', TF_WEIGHTS_PATH_NO_TOP, cache_subdir='models')

model.load_weights(weights)

if K.backend() == 'tensorflow' and K.image_data_format() == "channels_first":
    warnings.warn('You are using the TensorFlow backend, yet you '
                  'are using the Theano '
                  'image dimension ordering convention '
                  '(`image_dim_ordering="th"`). '
                  'For best performance, set '
                  '`image_dim_ordering="tf"` in '
                  'your Keras config '
                  'at ~/.keras/keras.json.')
    convert_all_kernels_in_model(model)

print('Model loaded.')
```

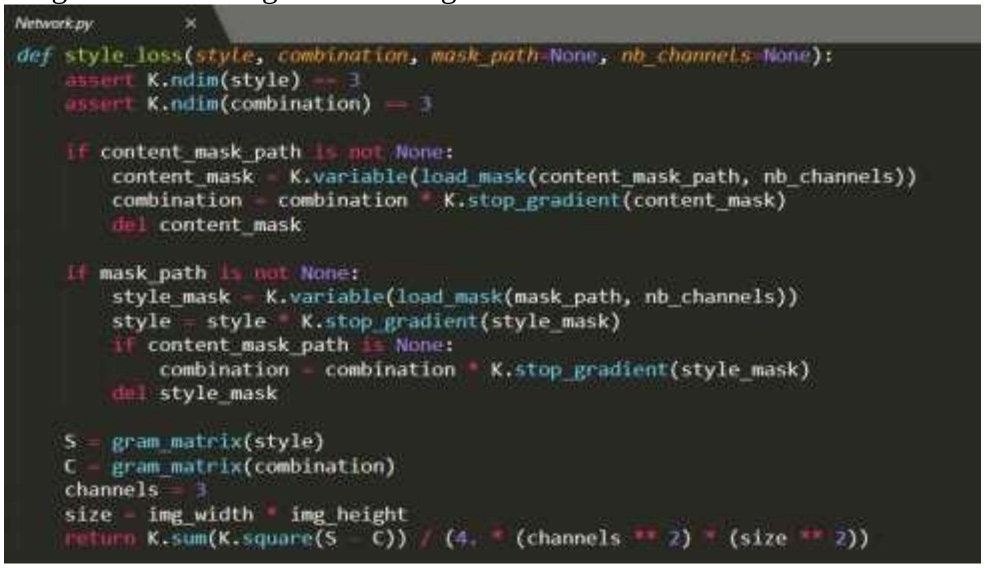
- The symbolic outputs of each key layers.



Network.py

```python
outputs_dict = dict([(layer.name, layer.output) for layer in model.layers])
shape_dict = dict([(layer.name, layer.output_shape) for layer in model.layers])
```

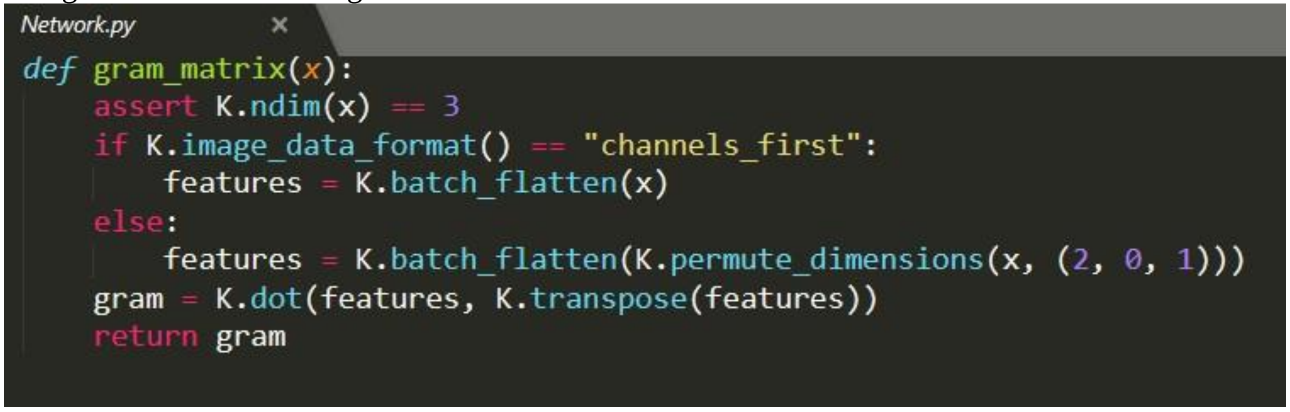
- The “style loss” is designed to maintain the style of the reference image in the generated image. It is based on the gram matrices(which capture style) of feature maps from the style reference image and from the generated image.



Network.py

```python
def style_loss(style, combination, mask_path=None, nb_channels=None):
    assert K.ndim(style) == 3
    assert K.ndim(combination) == 3

    if content_mask_path is not None:
        content_mask = K.variable(load_mask(content_mask_path, nb_channels))
        combination = combination * K.stop_gradient(content_mask)
        del content_mask

    if mask_path is not None:
        style_mask = K.variable(load_mask(mask_path, nb_channels))
        style = style * K.stop_gradient(style_mask)
        if content_mask_path is None:
            combination = combination * K.stop_gradient(style_mask)
        del style_mask

    S = gram_matrix(style)
    C = gram_matrix(combination)
    channels = 3
    size = img_width * img_height
    return K.sum(K.square(S - C)) / (4. * (channels ** 2) * (size ** 2))
```

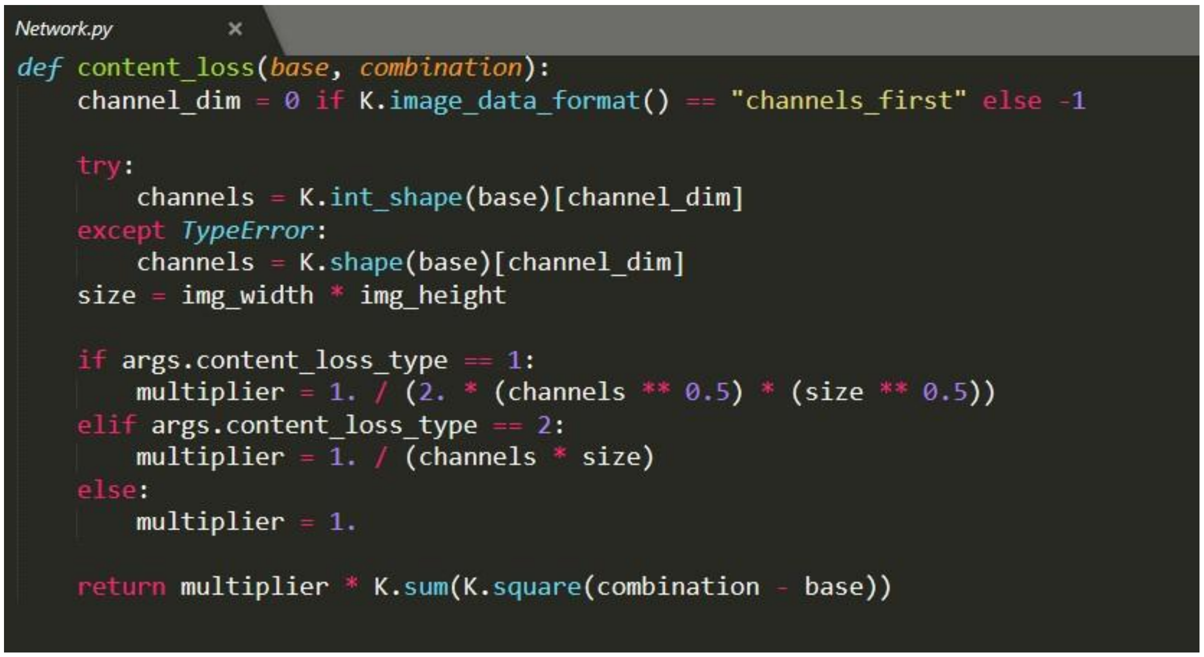
- The gram matrix of an image tensor



Network.py

```python
def gram_matrix(x):
    assert K.ndim(x) == 3
    if K.image_data_format() == "channels_first":
        features = K.batch_flatten(x)
    else:
        features = K.batch_flatten(K.permute_dimensions(x, (2, 0, 1)))
    gram = K.dot(features, K.transpose(features))
    return gram
```

- An auxiliary loss function designed to maintain the “content” of the base image in the generated image.

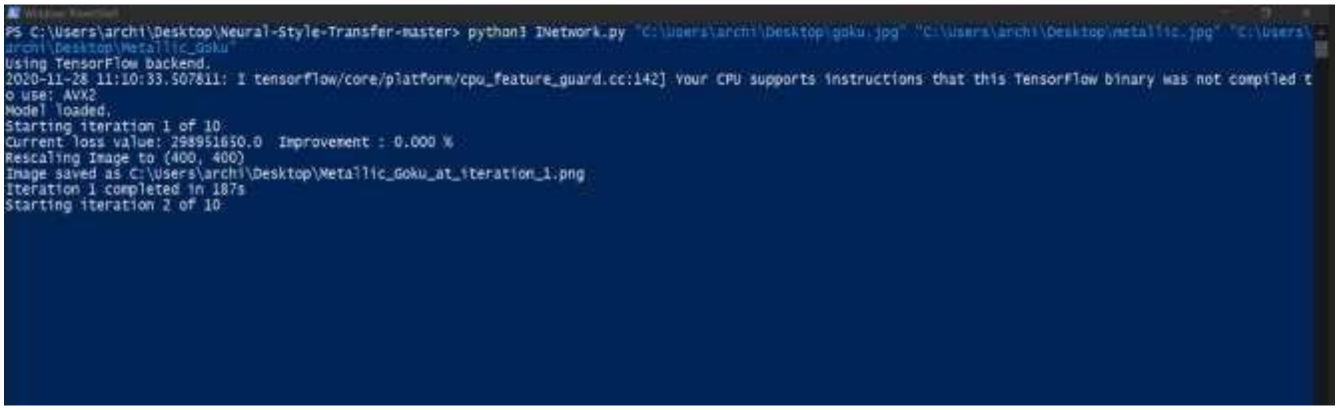


```
Network.py
def content_loss(base, combination):
    channel_dim = 0 if K.image_data_format() == "channels_first" else -1

    try:
        channels = K.int_shape(base)[channel_dim]
    except TypeError:
        channels = K.shape(base)[channel_dim]
    size = img_width * img_height

    if args.content_loss_type == 1:
        multiplier = 1. / (2. * (channels ** 0.5) * (size ** 0.5))
    elif args.content_loss_type == 2:
        multiplier = 1. / (channels * size)
    else:
        multiplier = 1.

    return multiplier * K.sum(K.square(combination - base))
```

4.2 Result

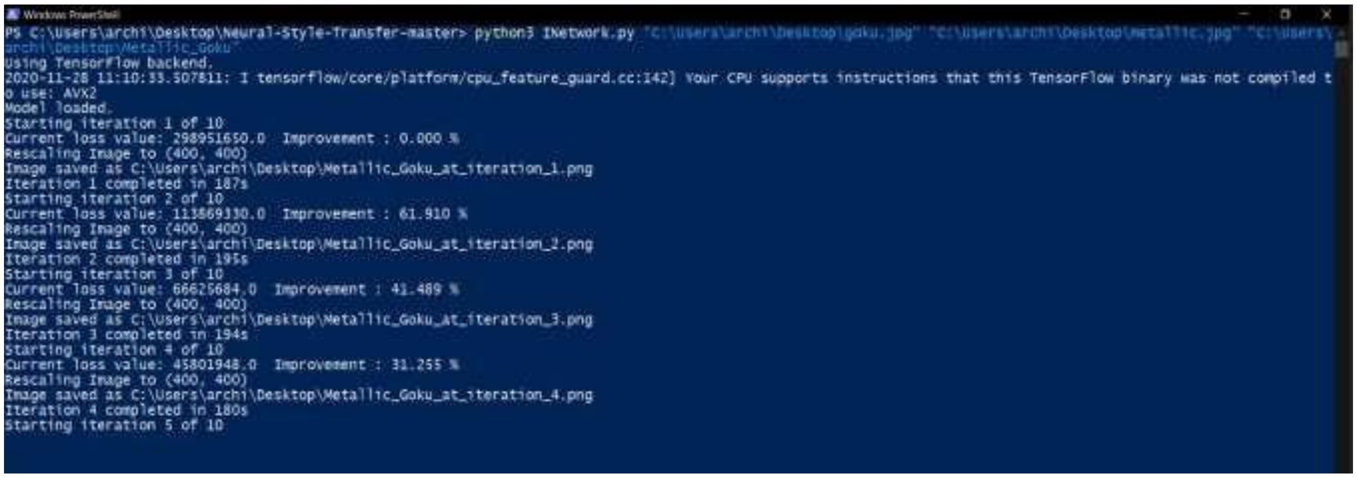
In summary, we found that the convolutional neural network has performed very well in this task due to its excellent performance in analyzing visual images. We detected decent images included using the CNN model with tuned parameters.

Program during 2nd iteration out of 10.



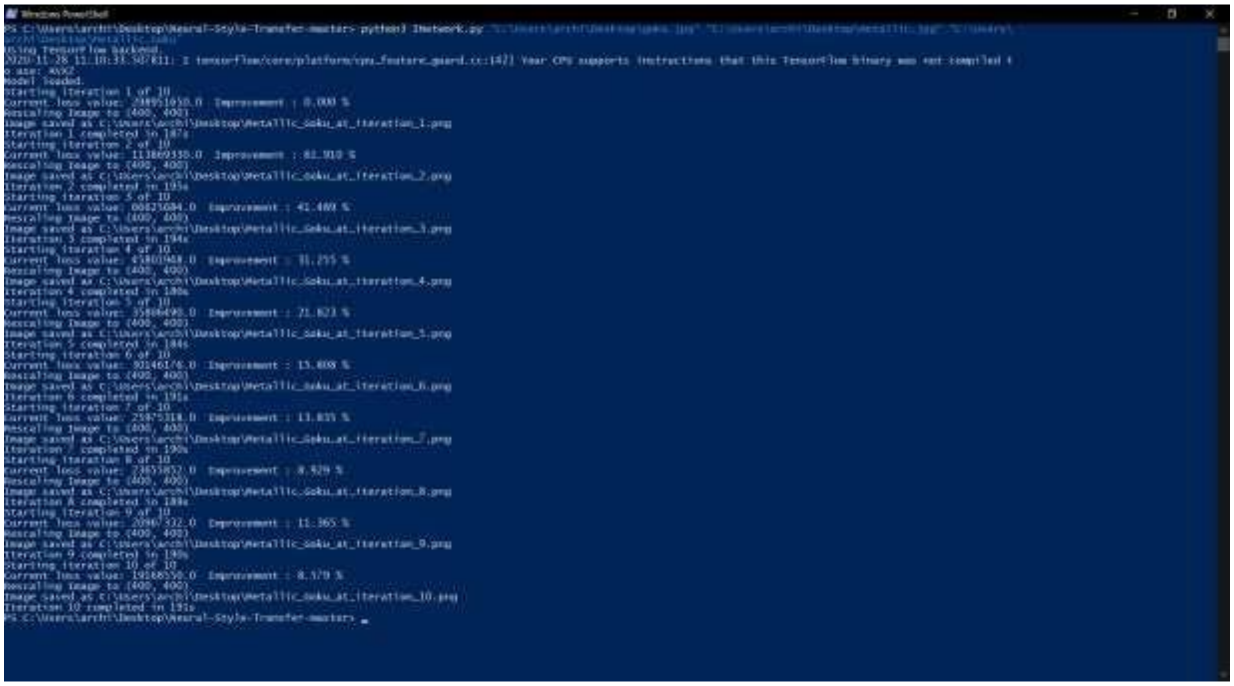
```
PS C:\Users\archi\Desktop\Neural-Style-Transfer-master> python3 INetwork.py "C:\Users\archi\Desktop\goku.jpg" "C:\Users\archi\Desktop\metallic.jpg" "C:\Users\
archi\Desktop\Metallic_Goku"
Using TensorFlow backend.
2020-11-28 11:10:33.507811: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU supports instructions that this TensorFlow binary was not compiled t
o use: AVX2
Model loaded.
Starting iteration 1 of 10
Current loss value: 298951650.0  Improvement : 0.000 %
Rescaling Image to (400, 400)
Image saved as C:\Users\archi\Desktop\Metallic_Goku_at_iteration_1.png
Iteration 1 completed in 187s
Starting iteration 2 of 10
```

Program during 5th iteration out of 10



```
PS C:\Users\archi\Desktop\Neural-Style-Transfer-master> python3 INetwork.py "C:\users\archi\Desktop\goku.jpg" "C:\Users\archi\Desktop\metallic.jpg" "C:\users\archi\Desktop\Metallic_Goku"
Using TensorFlow backend.
2020-11-28 11:10:33.507811: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU supports instructions that this TensorFlow binary was not compiled t
o use: AVX2
Model loaded.
Starting iteration 1 of 10
Current loss value: 298951650.0  Improvement : 0.000 %
Rescaling Image to (400, 400)
Image saved as C:\Users\archi\Desktop\Metallic_Goku_at_iteration_1.png
Iteration 1 completed in 187s
Starting iteration 2 of 10
Current loss value: 113869330.0  Improvement : 61.910 %
Rescaling Image to (400, 400)
Image saved as C:\Users\archi\Desktop\Metallic_Goku_at_iteration_2.png
Iteration 2 completed in 195s
Starting iteration 3 of 10
Current loss value: 66625684.0  Improvement : 41.489 %
Rescaling Image to (400, 400)
Image saved as C:\Users\archi\Desktop\Metallic_Goku_at_iteration_3.png
Iteration 3 completed in 194s
Starting iteration 4 of 10
Current loss value: 45801948.0  Improvement : 31.255 %
Rescaling Image to (400, 400)
Image saved as C:\Users\archi\Desktop\Metallic_Goku_at_iteration_4.png
Iteration 4 completed in 180s
Starting iteration 5 of 10
```

Program after Completing all the 10 iterations.

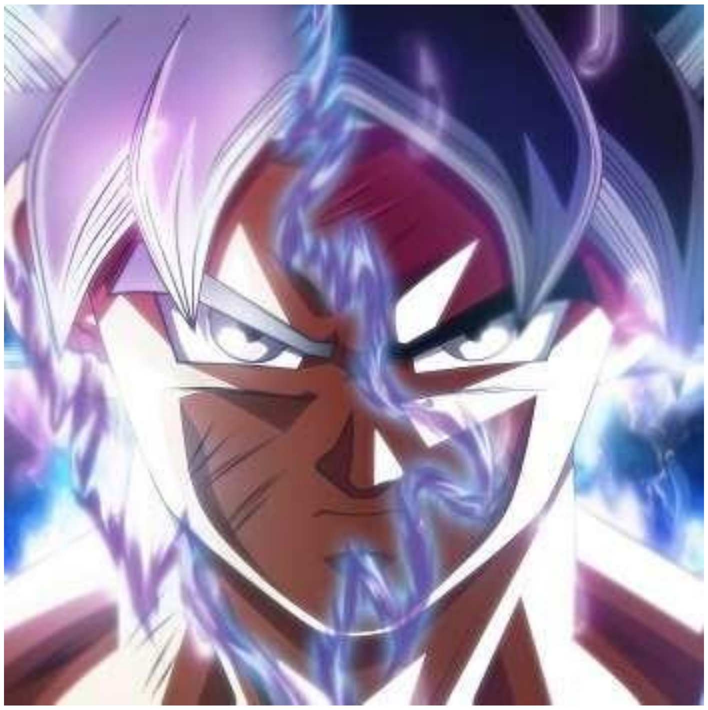


```
PS C:\Users\archi\Desktop\Neural-Style-Transfer-master> python3 INetwork.py "C:\users\archi\Desktop\goku.jpg" "C:\Users\archi\Desktop\metallic.jpg" "C:\users\
archi\Desktop\Metallic_Goku"
Using TensorFlow backend.
2020-11-28 11:10:33.507811: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU supports instructions that this TensorFlow binary was not compiled t
o use: AVX2
Model loaded.
Starting iteration 1 of 10
Current loss value: 298951650.0  Improvement : 0.000 %
Rescaling Image to (400, 400)
Image saved as C:\Users\archi\Desktop\Metallic_Goku_at_iteration_1.png
Iteration 1 completed in 187s
Starting iteration 2 of 10
Current loss value: 113869330.0  Improvement : 61.910 %
Rescaling Image to (400, 400)
Image saved as C:\Users\archi\Desktop\Metallic_Goku_at_iteration_2.png
Iteration 2 completed in 195s
Starting iteration 3 of 10
Current loss value: 66625684.0  Improvement : 41.489 %
Rescaling Image to (400, 400)
Image saved as C:\Users\archi\Desktop\Metallic_Goku_at_iteration_3.png
Iteration 3 completed in 194s
Starting iteration 4 of 10
Current loss value: 45801948.0  Improvement : 31.255 %
Rescaling Image to (400, 400)
Image saved as C:\Users\archi\Desktop\Metallic_Goku_at_iteration_4.png
Iteration 4 completed in 180s
Starting iteration 5 of 10
Current loss value: 35806498.0  Improvement : 21.823 %
Rescaling Image to (400, 400)
Image saved as C:\Users\archi\Desktop\Metallic_Goku_at_iteration_5.png
Iteration 5 completed in 184s
Starting iteration 6 of 10
Current loss value: 30146176.0  Improvement : 15.808 %
Rescaling Image to (400, 400)
Image saved as C:\Users\archi\Desktop\Metallic_Goku_at_iteration_6.png
Iteration 6 completed in 191s
Starting iteration 7 of 10
Current loss value: 25975318.0  Improvement : 13.835 %
Rescaling Image to (400, 400)
Image saved as C:\Users\archi\Desktop\Metallic_Goku_at_iteration_7.png
Iteration 7 completed in 190s
Starting iteration 8 of 10
Current loss value: 23655852.0  Improvement : 8.929 %
Rescaling Image to (400, 400)
Image saved as C:\Users\archi\Desktop\Metallic_Goku_at_iteration_8.png
Iteration 8 completed in 188s
Starting iteration 9 of 10
Current loss value: 20967332.0  Improvement : 11.365 %
Rescaling Image to (400, 400)
Image saved as C:\Users\archi\Desktop\Metallic_Goku_at_iteration_9.png
Iteration 9 completed in 190s
Starting iteration 10 of 10
Current loss value: 19186550.0  Improvement : 8.379 %
Rescaling Image to (400, 400)
Image saved as C:\Users\archi\Desktop\Metallic_Goku_at_iteration_10.png
Iteration 10 completed in 191s
PS C:\Users\archi\Desktop\Neural-Style-Transfer-master> _
```

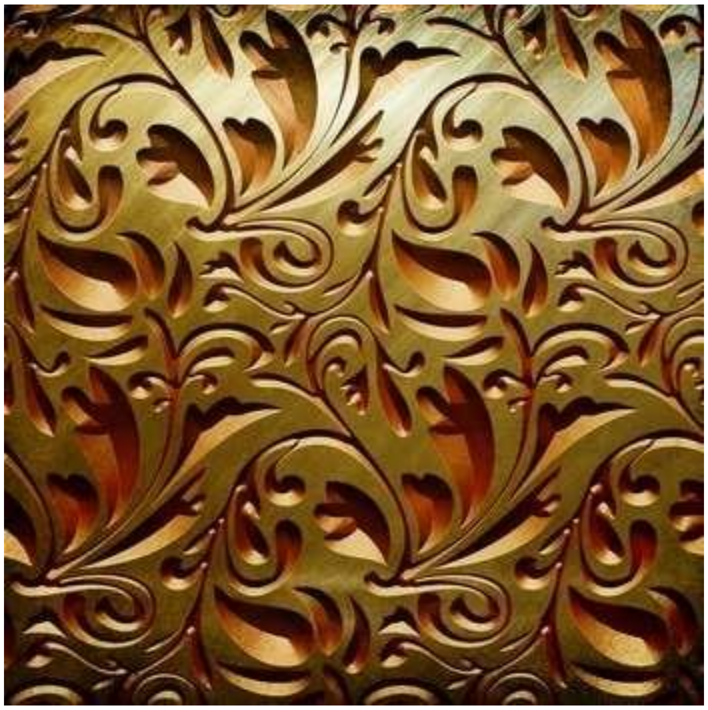
We can see that almost each iteration took an equal amount of time around 180 to 190 seconds. The loss value decreases with each iteration. Improvement percentage also keeps on decreasing with each iteration.

Some Result Images after each iteration:

Content Image



Style Image



Iteration 1

Iteration 2

Iteration 3

Iteration 4

Iteration 5

Iteration 6

Iteration 7

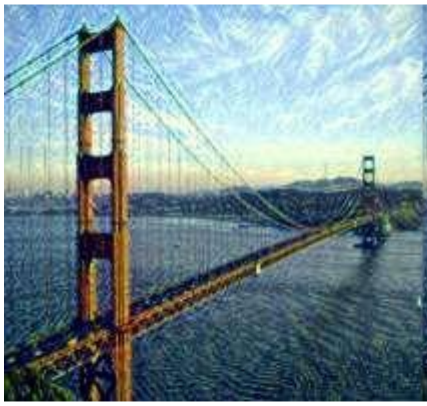
Iteration 8

Iteration 9 **Iteration 10**

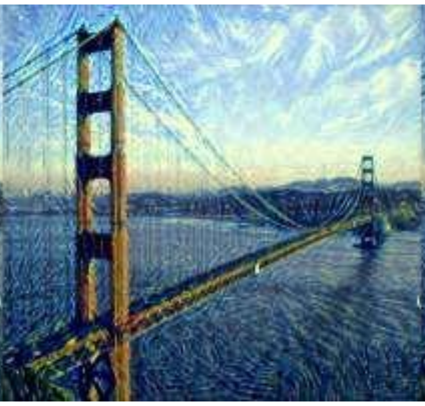
<u>Content Image</u>

<u>Style Image</u>

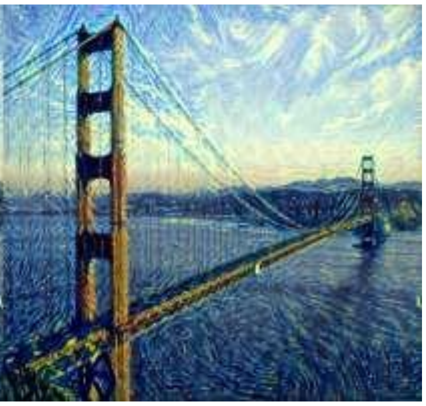
Iteration 1



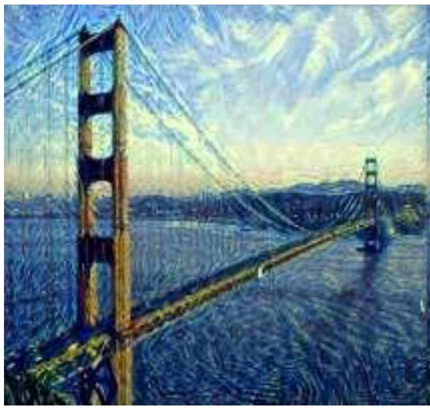
Iteration 2



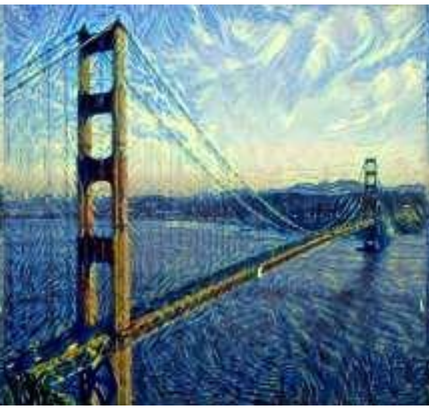
Iteration 3



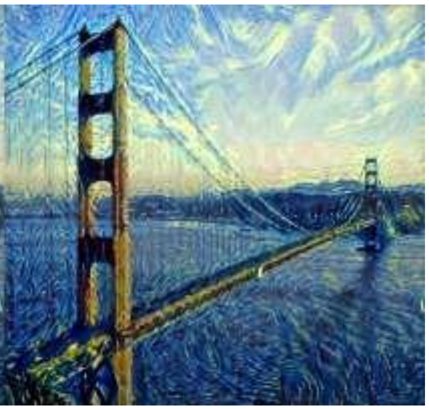
Iteration 4



Iteration 5

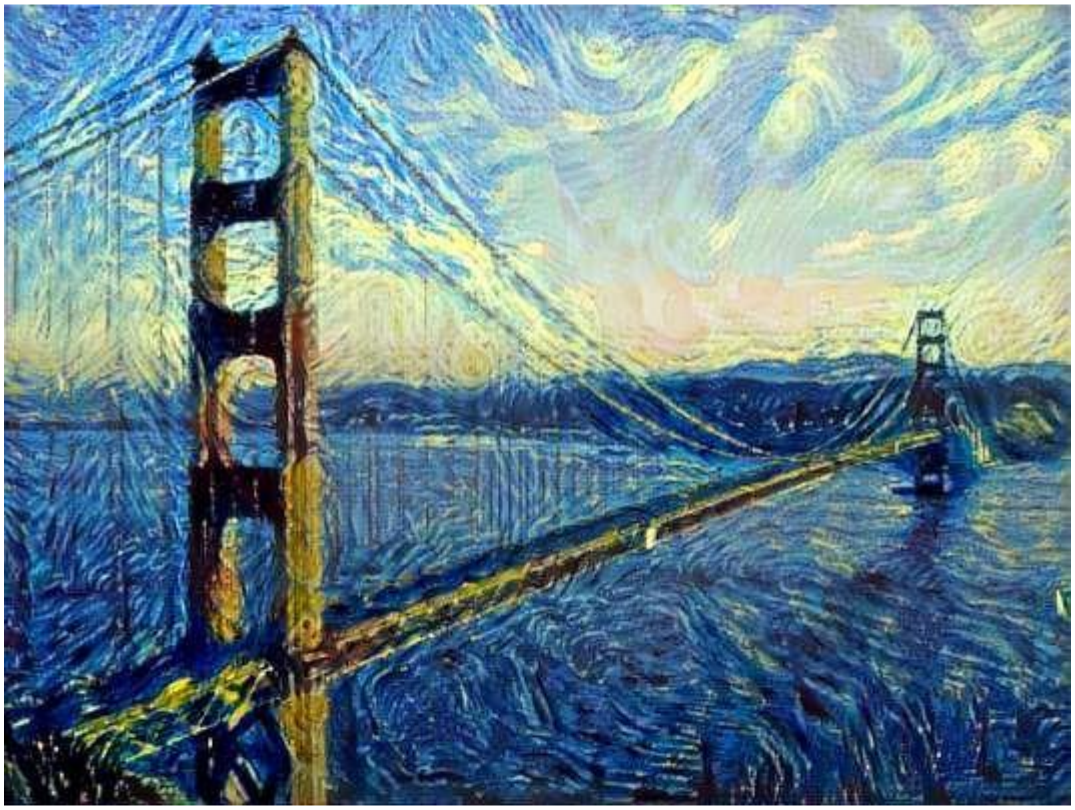


Iteration 6



Iteration 7 **Iteration 8** **Iteration 9**

Iteration 10



From the above results, we can clearly see that with each iteration, the image is getting better and smoother. From this we can say that the loss value is decreasing with each iteration. We can also see that, for the first few iterations, the change in the generated image seems to be drastic and after that the change becomes very gradual. For the ease of the reader, we have displayed the total current loss value and the improvement percentage of optimizing the desired result image for some particular input images in the form of tables and bar graphs as shown below. In each table we will show the respected values corresponding to the first 10 iterations.

Table-1- iteration – current loss value table

ITERATION	CURRENT LOSS VALUE
Itn-1	298951650.0
Itn-2	113869330.0
Itn-3	66625684.0
Itn-4	45801948.0
Itn-5	35806490.0
Itn-6	30146176.0
Itn-7	25975218.0
Itn-8	23655852.0
Itn-9	20967332.0
Itn-10	19168550.0

From the above Table-1, we can see that the loss value is decreasing with each iteration and we have also shown this decrease in a bar graph(GRAPH-1) shown below.

Graph-1

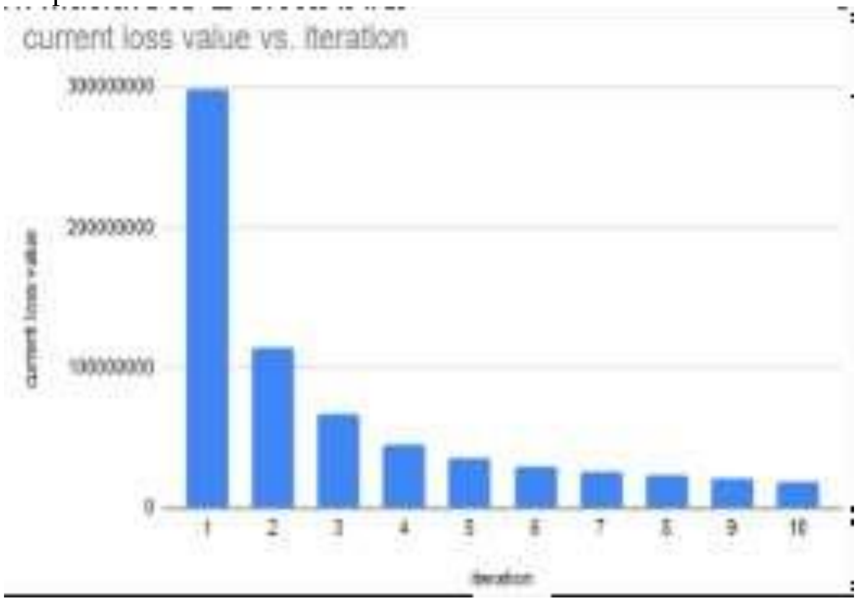


Figure 5. Current loss value vs iteration graph

We can see that initially the drop in loss value is very huge but it becomes gradual after a few iterations matching with the trend of generated image results that we have shown above.

TABLE 2. Iteration - Improvement percentage

ITERATION	IMPROVEMENT %AGE
Itn-1	0.000%
Itn-2	61.910%
Itn-3	41.489%
Itn-4	31.255%
Itn-5	21.823%
Itn-6	15.808%
Itn-7	13.835%
Itn-8	8.929%
Itn-9	11.365%
Itn-10	8.579%

From the above Table-2, similar to Table-1, we can see that the improvement percentage is decreasing with each iteration and we have also shown this decrease in a bar graph(GRAPH-2) shown below.

Gtaph=2

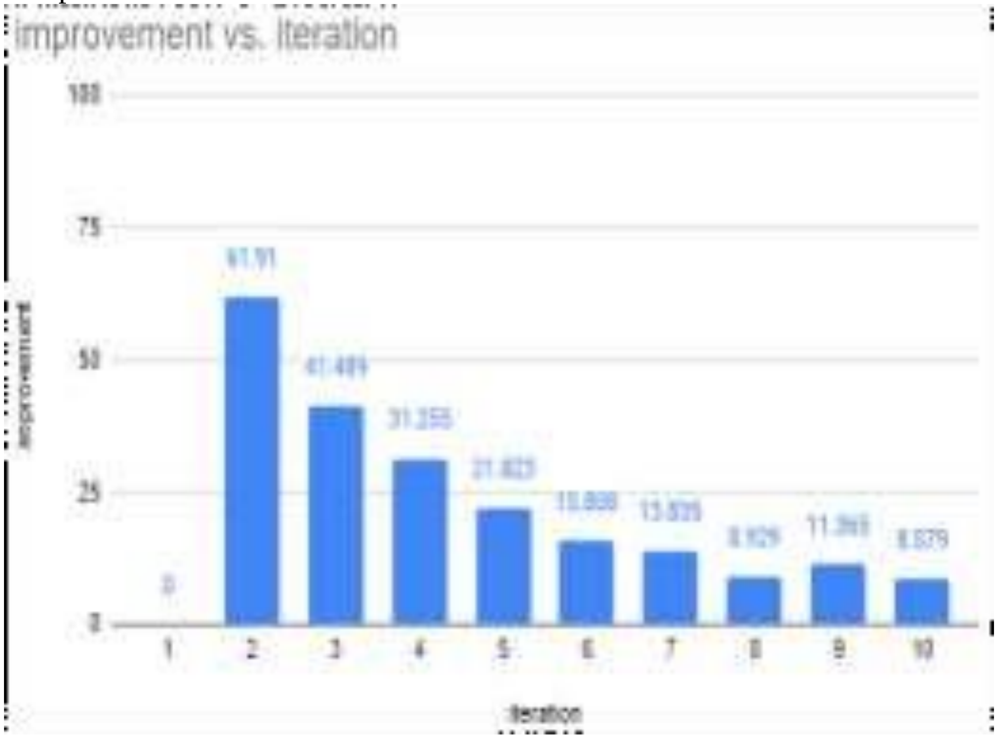


Figure 6. Improvement percentage vs Iteration graph

We can see that initially the drop in improvement percentage is very huge similar to loss value but it becomes gradual after a few iterations matching with the trend of generated image results that we have shown above

After testing different images and analyzing from what we have seen, we recommend choosing style images with simple outlines and large blocks of colorful blushes to convey artistic style and avoid using paintings with multiple small color blocks.

In summary, our implementation of the CNN model can effectively detect the transfer of art style. The innovation of the project is that we are exploring how we can use style transfer when a content image emphasizes a well-executed image.

5. Conclusion

In this project, we implemented Neural Style Transfer that allows us to merge two images to create a new artistic image. Initially we learned about NST and its architecture. After that, we described the details of the neural style transmission network and TensorFlow along with loss functions, VGG Network. We also talked about the two main losses which helped us in getting the desired results; loss of content and loss of style in detail, and they see how they come together to explain the final loss. Eventually we were able to use our NST model and saw the art made by this model.

Due to the technical challenges and the increasing demand of industry, Neural Style Transfer has become an interesting field to do research. A large amount of research has been done on NST. It is a very rapid growing technique and still there is a lot of work that can be done on it.

NST as a field of research is still immature despite the remarkable progress that has been made in these few past years. Right now, we have to make the current algorithms of NST more efficient so that we can incorporate multiple styles more accurately. For this we must improve the quality of the generated image in a large dataset of input content and style images and we must avoid bad results. We have seen that NST works with different efficiency with different artistic styles. Like it generates good results when working with arbitrary irregular styles but cannot generate good results with regular kinds of styles. Distorted results are produced because of image reconstruction by CNN. Previous papers have used natural environmental images as content images to show their results but these techniques do not usually work with abstract images because pre- trained classifiers are not able to extract proper features of these types of images. An additional practice of the NST is not only to imitate man-made art by NST techniques but rather to create a new form of art created by AI under the guidance of the basic principles of art[1].

6. References

1. Yongcheng Jing, Yezhou Yang, Zunlei Feng, Jingwen Ye, Yizhou Yu & Mingli Song, Neural Style Transfer: A Review. In IEEE Transactions on Visualization and Computer Graphics, vol. 26, no. 11, pp. 3365-3385, 1 Nov. 2020, doi: 10.1109/TVCG.2019.2921336.
2. Yanghao Li., Naiyan Wang, Jiaying Liu & Xiaodi Hou, Demystifying neural style transfer. arXiv preprint arXiv:1701.01036, 2017.
3. Agrim Gupta, Justin Johnson, Alexandre Alahi & Li Fei-Fei, Characterizing and improving stability in neural style transfer. In Proceedings of the IEEE International Conference on Computer Vision (pp. 4067-4076), 2017. Gantugs Atarsaikhan, Brian Kenji Iwana, Atsushi Narusawa, Keiji Yanai & Seiichi Uchida, Neural font style transfer. In 2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR) (Vol. 5, pp. 51-56). IEEE, November, 2017.
4. Leon A. Gatys, Matthias Bethge, Aaron Hertzmann & Eli Shechtman, Preserving color in neural

artistic style transfer. arXiv preprint arXiv:1606.05897, 2016.

5. Prateek Verma & Julius O.Smith, Neural style transfer for audio spectograms. In 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA. arXiv preprint arXiv:1801.01589, 2018.
6. Daniel Holden, Ikhsanul, Ikuo Kusajima & Taku Komur, Fast neural style transfer for motion data. IEEE computer graphics and applications, 37(4), 42-49, 2017.
7. Xun Huang & Serge Belongie, Arbitrary style transfer in real-time with adaptive instance normalization. In Proceedings of the IEEE International Conference on Computer Vision (pp. 1501-1510). 2017.
8. Roman Novak & Yaroslav Nikulin, Improving the neural algorithm of artistic style. arXiv preprint arXiv:1605.04603, 2016.
9. Rujie Yin, Content aware neural style transfer. arXiv preprint arXiv:1601.04568, 2016.
10. Karen Simonyan & Andrew Zisserman, Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014.
11. Elvin Mirzazada, Neural Style Transfer with Deep VGG Model. May 14, 2020. Available: https://medium.com/@mirzezadeh.elvin/neural-style-transfer-with-deep-vgg-model
12. ImageNet, Available: http://www.image-net.org.
13. VGG19 convolutional neural networks. Available: https://www.mathworks.com/help/deeplearning/ref/vgg19
14. Thushan Ganegedara, Intuitive Guide to Neural Style Transfer. Jan 9, 2019. Available: https://towardsdatascience.com/light-on-math-machine-learning-intuitive-guide-to-neural-style-transfer-ef88e46697ee3
15. Vikas Solegaonkar, Convolutional Neural Network. Feb 18, 2019. Available: https://towardsdatascience.com/convolutional-neural-networks-e5a6745b2810
16. Introduction And Implementation to Neural Style Transfer - Deep Learning. Oct 22, 2020. Available: https://www.analyticsvidhya.com/blog/2020/10/introduction-and-implementation-to-neural-style-transfer-deep-learning/

DECLARATION

We hereby certify that the work which is presented in the Major Project-II entitled Neural Style Transfer in fulfilment of the requirement for the award of the Degree of Bachelor of Technology in Computer Engineering and submitted to the Department of Computer Engineering, Delhi Technological University, Delhi is an authentic record of my/our own, carried out during a period from January to May 2021, under the supervision of Abhilasha Sharma.

The matter presented in this report has not been submitted by us/me for the award of any other degree of this or any other Institute/University. The work has been published/accepted/communicated in SCI/ SCI expanded/SSCI/Scopus indexed journal OR peer reviewed Scopus indexed conference with the following details:

Title of the Paper: Keras Implementation of Neural Style Transfer
Author names (in sequence as per research paper): Abhilasha Sharma; Ritik Handa; Rakshit; Raghav Punjani
Name of Conference/Journal: International Conference on Machine Learning Big Data Management Cloud And Computing (ICMBDC)
Conference Dates with venue (if applicable): 1st May. 2021, New Delhi, India
Have you registered for the conference (Yes/No)?: Yes
Status of paper (Accepted/Published/Communicated):Presented
Date of paper communication: 28 April 202!
Date of paper acceptance: 30 April 2021
Date of conference: 1 May 2021

2K17/CO/271, Ritik Handa [illegible]

2K17/CO/259, Rakshit Rakshit

2K17/CO/249, Raghav Punjani [illegible]

SUPERVISOR CERTIFICATE

To the best of my knowledge, the above work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere. I further certify that the publication and indexing information given by the students is correct.

Place: Delhi

Abhilasha Sharma

Supervisor name and Signature:

Abhilasha Sharma

Date: 5/26/2021