MACHINE LEARNING PROJECT

PROJECT TITLE :

NEURAL STYLE TRANSFER

TEAM MEMBERS :

S No	Roll No	Name
1	2019103518	M Fayeka Masoodha
2	2019103584	Pooja S
3	2019103601	Gunnala Hema
4	2019103530	Jeyasri Meenakshi

1.TITLE OF THE PROJECT : NEURAL STYLE TRANSFER

2.PROBLEM STATEMENT :

Neural Style Transfer is a technique that allows us to generate an image with the same "content" as a base image, but with the "style" of our chosen picture.

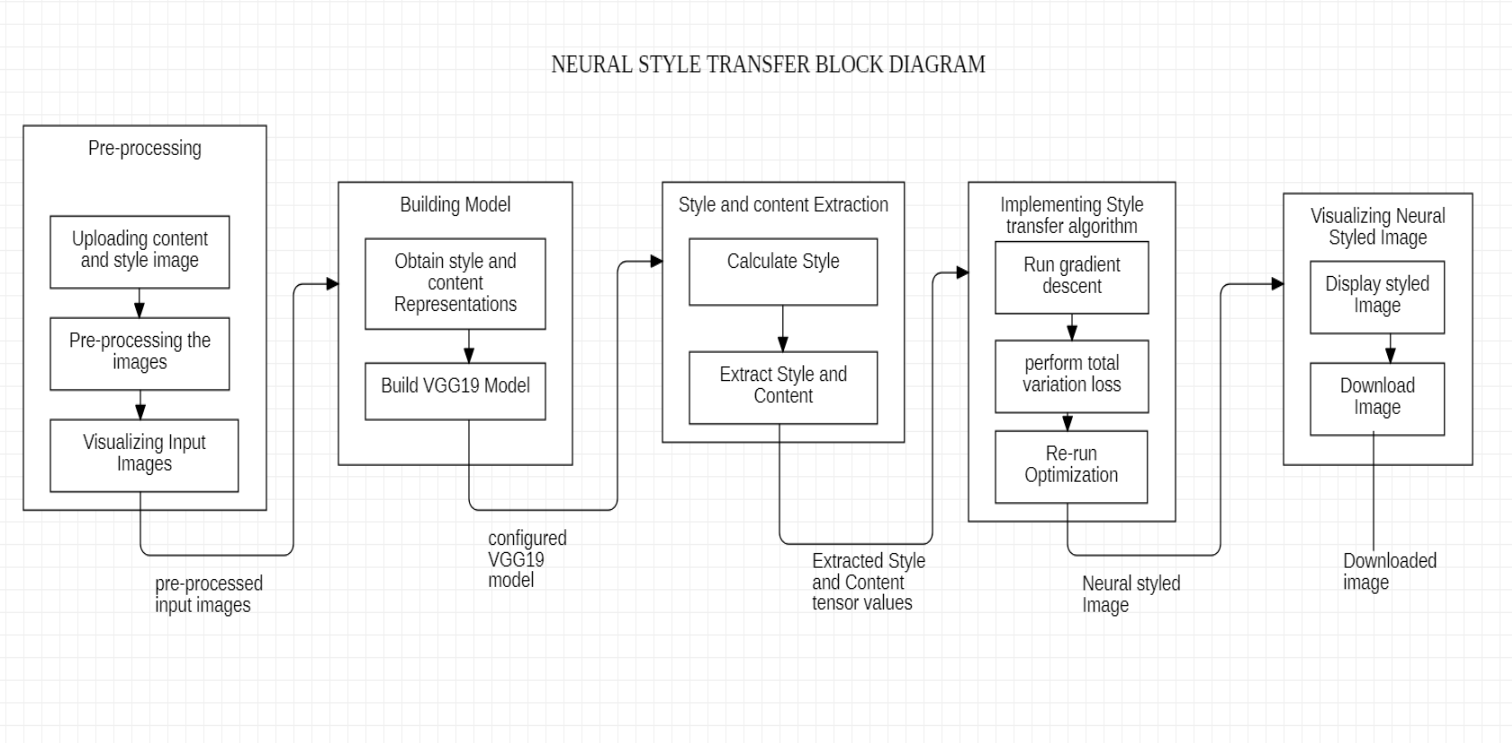
Style transfer is extensively used in photo and video editing software.These deep learning approaches and professional style transfer models can easily be applied to devices like mobile phones and give users a real-time ability to style images and videos. Style transfer also provides new techniques that can change the way we look and deal with art.It makes high-rated and over-priced artistic work reproducible for office and home decor, or for advertisements. Transfer models may help us commercialize art.There are also many cloud-powered video game streams that use image style transfer.

3.OBJECTIVE:

Neural style transfer (NST) refers to a class of software algorithms that manipulate digital images, or videos, in order to adopt the appearance or visual style of another image. NST algorithms are characterized by their use of deep neural networks for the sake of image transformation. Common uses for NST are the creation of artificial artwork from photographs, for example by transferring the appearance of famous paintings to user-supplied photographs. Several notable mobile apps use NST techniques for this purpose, including DeepArt and Prisma. This method has been used by artists and designers around the globe to develop new artwork based on existent style.

Given a training pair of images–a photo and an artwork depicting that photo–a transformation could be learned and then applied to create new artwork from a new photo, by analogy.

4.OVERALL ARCHITECTURE:



5.MODULES

I. PRE-PROCESSING

a. Uploading content and style image

- Importing and configuring the required modules like tensorflow, numpy, matplotlib etc.
- Obtain the content Image and style image.

b. Pre-process the images

- Pre-processing involves detecting the image extension and converting it into a Tensor of required type.

c. Visualizing input images

- Displaying the input images.

Input : Style and content Images
Output : Pre-processed Style and content Images.

II. BUILDING MODEL

a. Obtain style and content representations

- Loading the pre-trained arbitrary image stylization model from tensorflow hub.
- Intermediate layers of the model are used to get the content and style representations of the image.
- Here, the VGG19 network architecture which is a pretrained image classification network is used.

b. Build the VGG19 model

- Load the VGG19 model and test run it on our image to ensure it's used correctly.
- Choose the intermediate layers from the network to represent the style and content of the image and provide it as an input to the VGG19 model.
- The VGG19 model then returns a list of intermediate layer outputs.

Input : Pre-processed Style and content Images
Output : configured VGG19 model

III. STYLE AND CONTENT EXTRACTION

a. Calculate Style

- The content of an image is represented by the values of the intermediate feature maps.
- The style of an image can be described by the means and correlations across the different feature maps.
- Calculate the Gram matrix for a particular layer using the given formula.

$$G^l_{cd} = \frac{\sum_{ij} F^l_{ijc}(x) F^l_{ijd}(x)}{IJ}$$

b. Extract style and content

- Build a model that returns the style and content tensors.
- When called on an image, this model returns the gram matrix (style) of the style_layers and content of the content_layers.

Input : Configured VGG19 model
Output : Extracted style and content tensors.

IV. IMPLEMENT STYLE TRANSFER ALGORITHM

a. Run Gradient Descent

- With the above mentioned style and content extractor, implementing the style transfer algorithm.
- Calculating the mean square error for the image's output relative to each target, then taking the weighted sum of these losses.
- Create an optimizer. Optimization is then performed using the weighted combination of the two losses . The image is then updated and tested.

b. Perform total variation loss

- One downside to this basic implementation is that it produces a lot of high frequency artifacts. We can decrease these by using an explicit regularization term on the high frequency components of the image.This is total variation loss.

c. Re-run the optimization

- Choosing a weight for total variation loss
- Including it in the train step function.
- Reinitializing the optimization variable and running the optimization.

Input : style and content tensors
Output : neural styled image

V. VISUALIZING NEURAL STYLED IMAGE

a.Display styled image:

- The generated neural style clear output image can be displayed to the user.

b.Download image:

- The final image will be saved in .png file format and given a name.
- Can be downloaded in the user's system.

Input : neural styled image
Output : Downloaded neural styled image

6. IMPLEMENTATION WITH SNAPSHOTS:

I. PRE-PROCESSING

a. Uploading content and style image

```
[2]  import PIL.Image
     import time
     import functools
```

```
[3]  def tensor_to_image(tensor):
       tensor = tensor*255
       tensor = np.array(tensor, dtype=np.uint8)
       if np.ndim(tensor)>3:
         assert tensor.shape[0] == 1
         tensor = tensor[0]
       return PIL.Image.fromarray(tensor)
```

```
[4]  content_path = tf.keras.utils.get_file('YellowLabradorLooking_new.jpg', 'https://storage.googleapis.com/download.tensorflow.org/example_images/YellowL
     style_path = tf.keras.utils.get_file('kandinsky5.jpg','https://storage.googleapis.com/download.tensorflow.org/example_images/Vassily_Kandinsky%2C_191

     Downloading data from https://storage.googleapis.com/download.tensorflow.org/example_images/YellowLabradorLooking_new.jpg
     90112/83281 [================================] - 0s 0us/step
     98304/83281 [================================] - 0s 0us/step
     Downloading data from https://storage.googleapis.com/download.tensorflow.org/example_images/Vassily_Kandinsky%2C_1913_-_Composition_7.jpg
     196608/195196 [==============================] - 0s 0us/step
     204800/195196 [==============================] - 0s 0us/step
```

b. Pre-process the images

```
PRE-PROCESSING INPUT IMAGES
```

```
def load_img(path_to_img):
  max_dim = 512
  #reads file
  img = tf.io.read_file(path_to_img)
  print(img)
  img = tf.image.decode_image(img, channels=3)
  img = tf.image.convert_image_dtype(img, tf.float32)

  shape = tf.cast(tf.shape(img)[:-1], tf.float32)
  long_dim = max(shape)
  scale = max_dim / long_dim

  new_shape = tf.cast(shape * scale, tf.int32)

  img = tf.image.resize(img, new_shape)
  img = img[tf.newaxis, :]
  return img
```

```
[6]  def imshow(image, title=None):
       if len(image.shape) > 3:
```

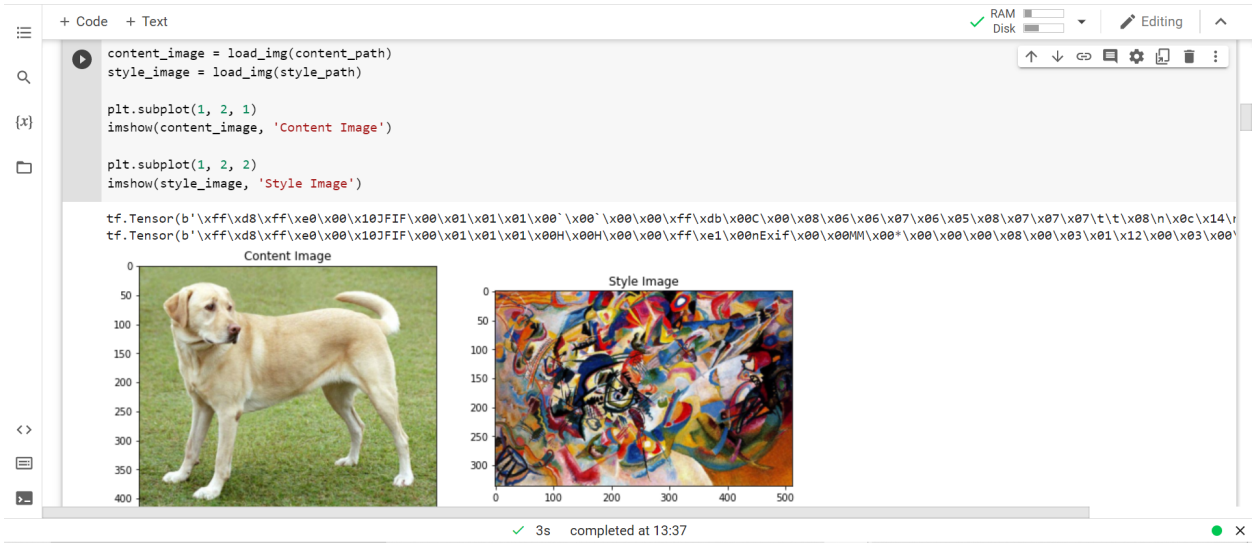
3s completed at 13:37

```
[5]  #reads file
     img = tf.io.read_file(path_to_img)
     print(img)
     img = tf.image.decode_image(img, channels=3)
     img = tf.image.convert_image_dtype(img, tf.float32)

     shape = tf.cast(tf.shape(img)[:-1], tf.float32)
     long_dim = max(shape)
     scale = max_dim / long_dim

     new_shape = tf.cast(shape * scale, tf.int32)

     img = tf.image.resize(img, new_shape)
     img = img[tf.newaxis, :]
     return img
```

```
def imshow(image, title=None):
  if len(image.shape) > 3:
    image = tf.squeeze(image, axis=0)

  plt.imshow(image)
  if title:
    plt.title(title)
```

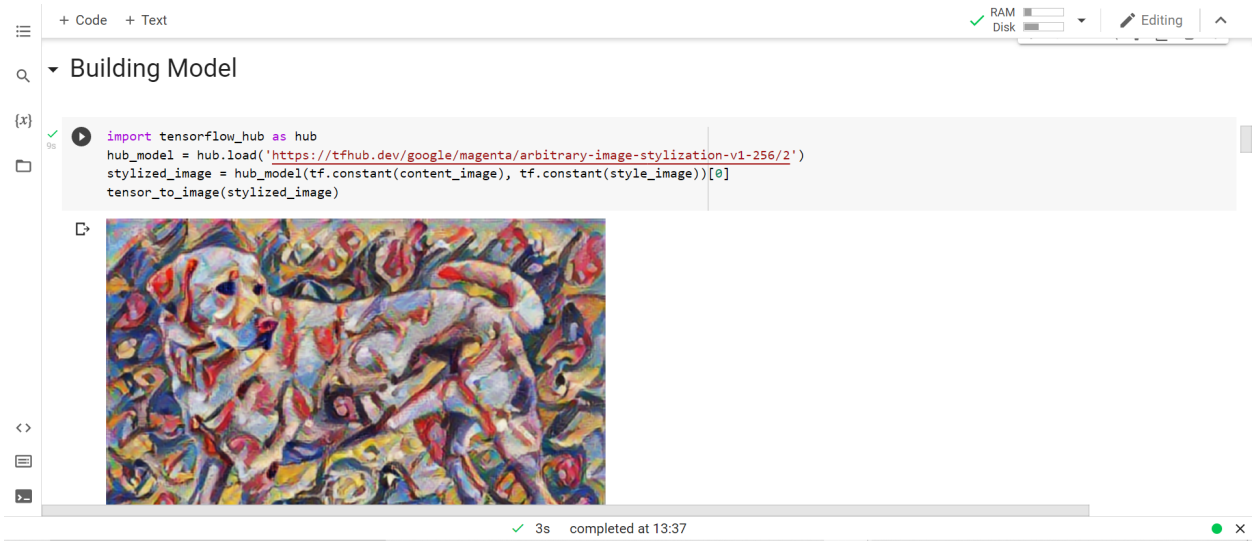
3s completed at 13:37

c. Visualizing input images



II. BUILDING MODEL

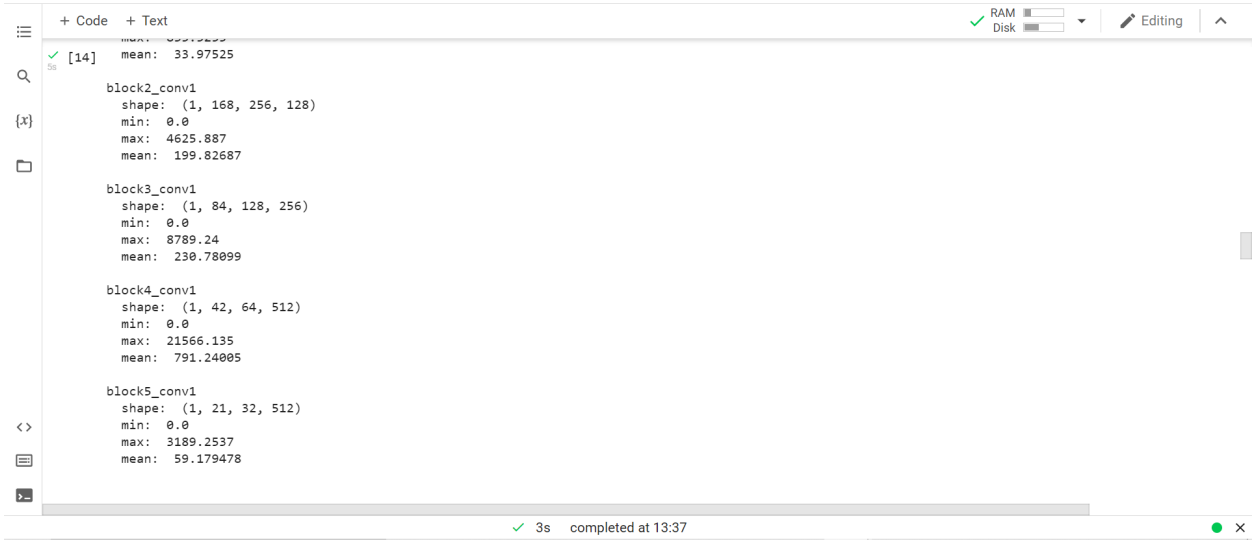
a. Obtain style and content representations




+ Code
+ Text
RAM
Disk
Editing
[9] x = tf.keras.applications.vgg19.preprocess_input(content_image*255)
x = tf.image.resize(x, (224, 224))
vgg = tf.keras.applications.VGG19(include_top=True, weights='imagenet')
prediction_probabilities = vgg(x)
prediction_probabilities.shape
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels.h5
574717952/574710816 [==============================] - 8s 0us/step
574726144/574710816 [==============================] - 8s 0us/step
TensorShape([1, 1000])
[10] predicted_top_5 = tf.keras.applications.vgg19.decode_predictions(prediction_probabilities.numpy())[0]
[(class_name, prob) for (number, class_name, prob) in predicted_top_5]
Downloading data from https://storage.googleapis.com/download.tensorflow.org/data/imagenet_class_index.json
40960/35363 [==================================] - 0s 0us/step
49152/35363 [==] - 0s 0us/step
[('Labrador_retriever', 0.4931729),
('golden_retriever', 0.23665176),
('kuvasz', 0.036357265),
('Chesapeake_Bay_retriever', 0.024182765),
('Greater_Swiss_Mountain_dog', 0.018646065)]
[11] vgg = tf.keras.applications.VGG19(include_top=False, weights='imagenet')
3s completed at 13:37

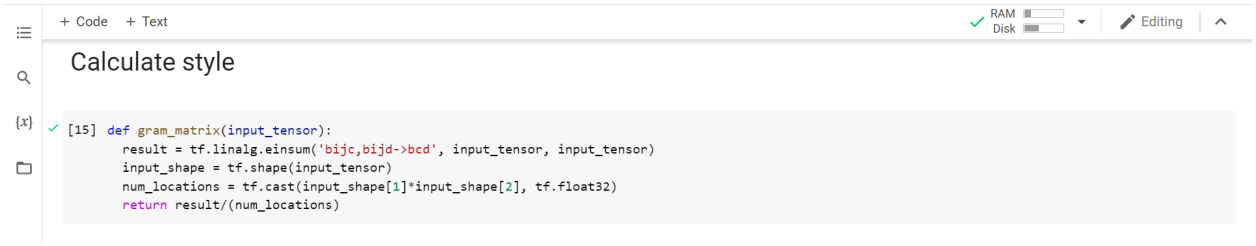

+ Code
+ Text
RAM
Disk
Editing
[11] vgg = tf.keras.applications.VGG19(include_top=False, weights='imagenet')
print()
for layer in vgg.layers:
print(layer.name)
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5
80142336/80134624 [==============================] - 1s 0us/step
80150528/80134624 [==============================] - 1s 0us/step
input_2
block1_conv1
block1_conv2
block1_pool
block2_conv1
block2_conv2
block2_pool
block3_conv1
block3_conv2
block3_conv3
block3_conv4
block3_pool
block4_conv1
block4_conv2
block4_conv3
block4_conv4
3s completed at 13:37

b. Build the VGG19 model

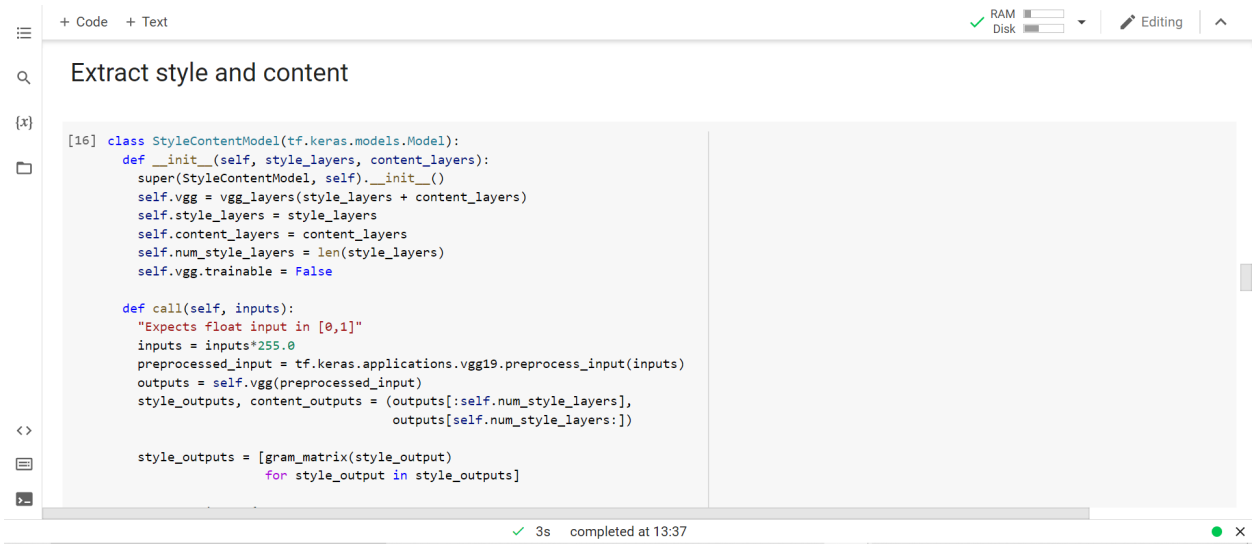


III. STYLE AND CONTENT EXTRACTION

a. Calculate Style



b. Extract style and content



```
[16]     content_dict = {content_name: value
                         for content_name, value
                         in zip(self.content_layers, content_outputs)}

         style_dict = {style_name: value
                       for style_name, value
                       in zip(self.style_layers, style_outputs)}

         return {'content': content_dict, 'style': style_dict}
```

```
[17] extractor = StyleContentModel(style_layers, content_layers)

     results = extractor(tf.constant(content_image))

     print('Styles:')
     for name, output in sorted(results['style'].items()):
       print("  ", name)
       print("    shape: ", output.numpy().shape)
       print("    min: ", output.numpy().min())
       print("    max: ", output.numpy().max())
       print("    mean: ", output.numpy().mean())
       print()

     print("Contents:")
```

3s completed at 13:37

```
[17] print("Contents:")
     for name, output in sorted(results['content'].items()):
       print("  ", name)
       print("    shape: ", output.numpy().shape)
       print("    min: ", output.numpy().min())
       print("    max: ", output.numpy().max())
       print("    mean: ", output.numpy().mean())
```

```
Styles:
   block1_conv1
    shape:  (1, 64, 64)
    min:  0.0055228495
    max:  28014.562
    mean:  263.79022

   block2_conv1
    shape:  (1, 128, 128)
    min:  0.0
    max:  61479.484
    mean:  9100.949

   block3_conv1
    shape:  (1, 256, 256)
    min:  0.0
    max:  545623.4
    mean:  7660.976
```

3s completed at 13:37

```
[17]
         mean:  9100.949
   block3_conv1
    shape:  (1, 256, 256)
    min:  0.0
    max:  545623.4
    mean:  7660.976

   block4_conv1
    shape:  (1, 512, 512)
    min:  0.0
    max:  4320501.0
    mean:  134288.86

   block5_conv1
    shape:  (1, 512, 512)
    min:  0.0
    max:  110005.37
    mean:  1487.0378

Contents:
   block5_conv2
    shape:  (1, 26, 32, 512)
    min:  0.0
    max:  2410.8794
    mean:  13.764149
```

3s completed at 13:37

IV. IMPLEMENT STYLE TRANSFER ALGORITHM

a. Run Gradient Descent

+ Code + Text

RAM Disk

Editing

Run gradient descent

```
[18] style_targets = extractor(style_image)['style']
     content_targets = extractor(content_image)['content']

[19] image = tf.Variable(content_image)

[20] def clip_0_1(image):
       return tf.clip_by_value(image, clip_value_min=0.0, clip_value_max=1.0)

[21] opt = tf.optimizers.Adam(learning_rate=0.02, beta_1=0.99, epsilon=1e-1)

[22] style_weight=1e-2
     content_weight=1e4

[23] def style_content_loss(outputs):
         style_outputs = outputs['style']
         content_outputs = outputs['content']
         style_loss = tf.add_n([tf.reduce_mean((style_outputs[name]-style_targets[name])**2)
```

3s completed at 13:37

+ Code + Text

RAM Disk

Editing

```
[23]     style_outputs = outputs['style']
         content_outputs = outputs['content']
         style_loss = tf.add_n([tf.reduce_mean((style_outputs[name]-style_targets[name])**2)
                                for name in style_outputs.keys()])
         style_loss *= style_weight / num_style_layers

         content_loss = tf.add_n([tf.reduce_mean((content_outputs[name]-content_targets[name])**2)
                                  for name in content_outputs.keys()])
         content_loss *= content_weight / num_content_layers
         loss = style_loss + content_loss
         return loss

[24] @tf.function()
     def train_step(image):
       with tf.GradientTape() as tape:
         outputs = extractor(image)
         loss = style_content_loss(outputs)

       grad = tape.gradient(loss, image)
       opt.apply_gradients([(grad, image)])
       image.assign(clip_0_1(image))

[25] train_step(image)
     train_step(image)
```

3s completed at 13:37

+ Code + Text

RAM Disk

Editing

```
[25] train_step(image)
     train_step(image)
     tensor_to_image(image)
```

3s completed at 13:37

+ Code + Text

RAM Disk

Editing

```
[26] import time
     start = time.time()

     epochs = 10
     steps_per_epoch = 100

     step = 0
     for n in range(epochs):
       for m in range(steps_per_epoch):
         step += 1
         train_step(image)
         print(".", end='', flush=True)
       display.clear_output(wait=True)
       display.display(tensor_to_image(image))
       print("Train step: {}".format(step))

     end = time.time()
     print("Total time: {:.1f}".format(end-start))
```

+ Code + Text

RAM Disk

Editing

[26]

```
Train step: 500
..................................................
```

3s completed at 13:37

b. Perform total variation loss

+ Code + Text

RAM Disk

Editing

Total variation loss

```
[27] def high_pass_x_y(image):
       x_var = image[:, :, 1:, :] - image[:, :, :-1, :]
       y_var = image[:, 1:, :, :] - image[:, :-1, :, :]

       return x_var, y_var
```

```
[28] x_deltas, y_deltas = high_pass_x_y(content_image)

     plt.figure(figsize=(14, 10))
     plt.subplot(2, 2, 1)
     imshow(clip_0_1(2*y_deltas+0.5), "Horizontal Deltas: Original")

     plt.subplot(2, 2, 2)
     imshow(clip_0_1(2*x_deltas+0.5), "Vertical Deltas: Original")

     x_deltas, y_deltas = high_pass_x_y(image)

     plt.subplot(2, 2, 3)
     imshow(clip_0_1(2*y_deltas+0.5), "Horizontal Deltas: Styled")
```

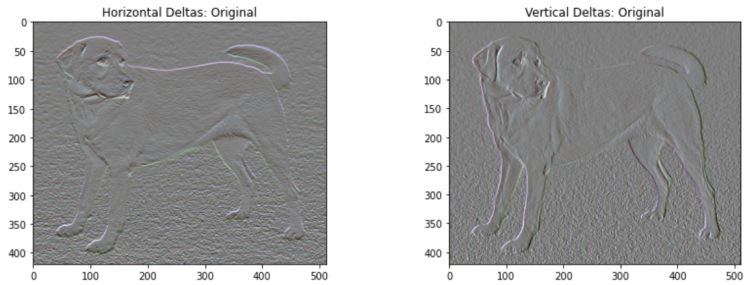
3s completed at 13:37

+ Code + Text

RAM Disk

Editing

```
[28] plt.subplot(2, 2, 4)
     imshow(clip_0_1(2*x_deltas+0.5), "Vertical Deltas: Styled")
```

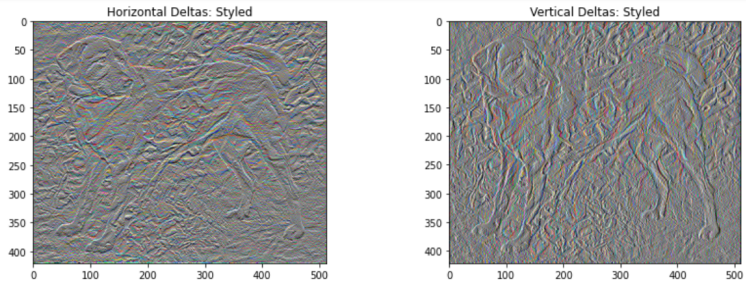


+ Code + Text

RAM Disk

Editing

[28]



```
[ ]  def total_variation_loss(image):
       x_deltas, y_deltas = high_pass_x_y(image)
       return tf.reduce_sum(tf.abs(x_deltas)) + tf.reduce_sum(tf.abs(y_deltas))
```

```
[ ]  total_variation_loss(image).numpy()
```

```
149353.12
```

```
[ ]  tf.image.total_variation(image).numpy()
```

```
array([149353.12], dtype=float32)
```

c. Re-run the optimization

Re-run the optimization

```
[ ]  total_variation_weight=30
```

```
@tf.function()
def train_step(image):
  with tf.GradientTape() as tape:
    outputs = extractor(image)
    loss = style_content_loss(outputs)
    loss += total_variation_weight*tf.image.total_variation(image)

  grad = tape.gradient(loss, image)
  opt.apply_gradients([(grad, image)])
  image.assign(clip_0_1(image))
```

```
[ ]  image = tf.Variable(content_image)
```

```
import time
start = time.time()

epochs = 10
steps_per_epoch = 100

step = 0
for n in range(epochs):
  for m in range(steps_per_epoch):
    step += 1
    train_step(image)
    print(".", end='', flush=True)
  display.clear_output(wait=True)
  display.display(tensor_to_image(image))
  print("Train step: {}".format(step))

end = time.time()
print("Total time: {:.1f}".format(end-start))
```

V. VISUALIZING NEURAL STYLED IMAGE

a.Display styled image

```
Train step: 1000
Total time: 259.1
```

b.Download image

```
file_name = 'stylized_image.png'
tensor_to_image(image).save(file_name)

try:
  from google.colab import files
except ImportError:
  pass
else:
  files.download(file_name)
```

stylized_image.png

7.DATASETS

- In this project VGG19 model, with weights pre-trained on ImageNet is used.
- ImageNet, is a dataset of over 15 millions labeled high-resolution images with around 22,000 categories.
- Also as input a style image and a content image is uploaded.

8.PERFORMANCE MEASURES

I.Content Loss

Let p and x be the original image and the image that is generated and P and F their respective feature representation in layer i .The squared-error loss between the two feature representations is defined as the content loss.

$$\mathcal{L}_{content}(\vec{p}, \vec{x}, l) = \frac{1}{2} \sum_{i,j} \left(F_{ij}^{l} - P_{ij}^{l}\right)^2$$

II.Style Loss

let a and x be the original image and the image that is generated and Ai and Gi their respective style representations in layer i . The contribution of that layer to the total loss is then

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} \left(G_{ij}^l - A_{ij}^l\right)^2$$

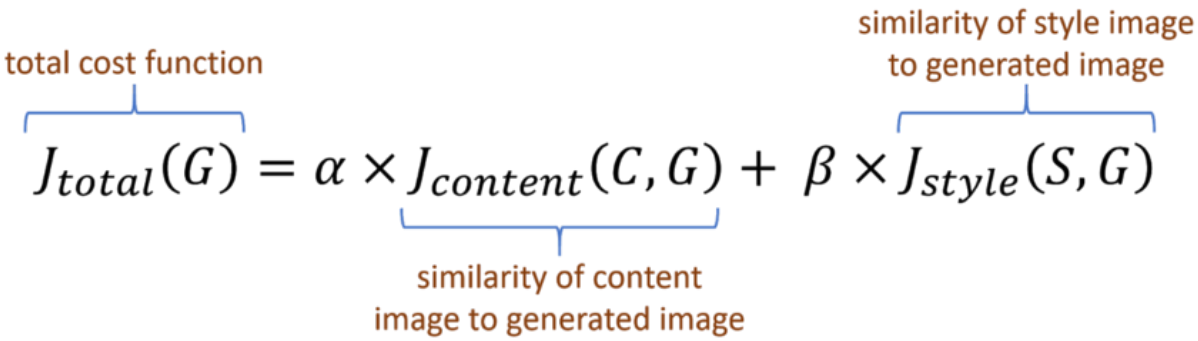
$$\mathcal{L}_{style}(\vec{a}, \vec{x}) = \sum_{l=0}^{L} w_l E_l$$

G - Gram matrix

$$G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l.$$

III.Cost Function For Style Transfer

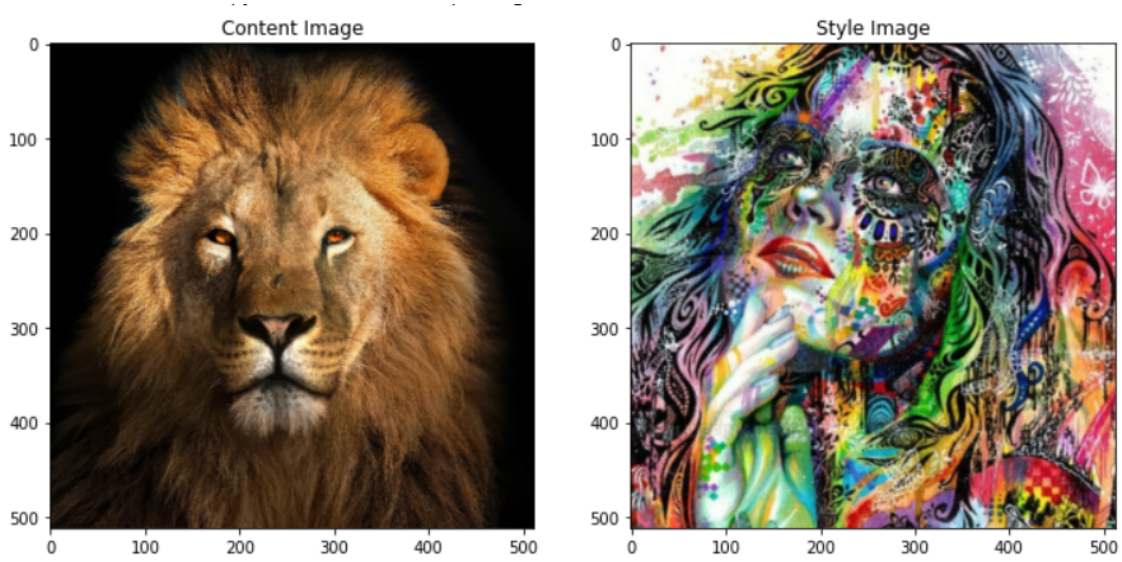
The total cost function is the weighted sum of two separate costs namely, style cost and content cost calculated separately.



Here, α is called the content weight and β is called the style weight. Generally, style weight is much higher than the content weight as to emphasize more on the “style transfer”.

9.TEST CASE:

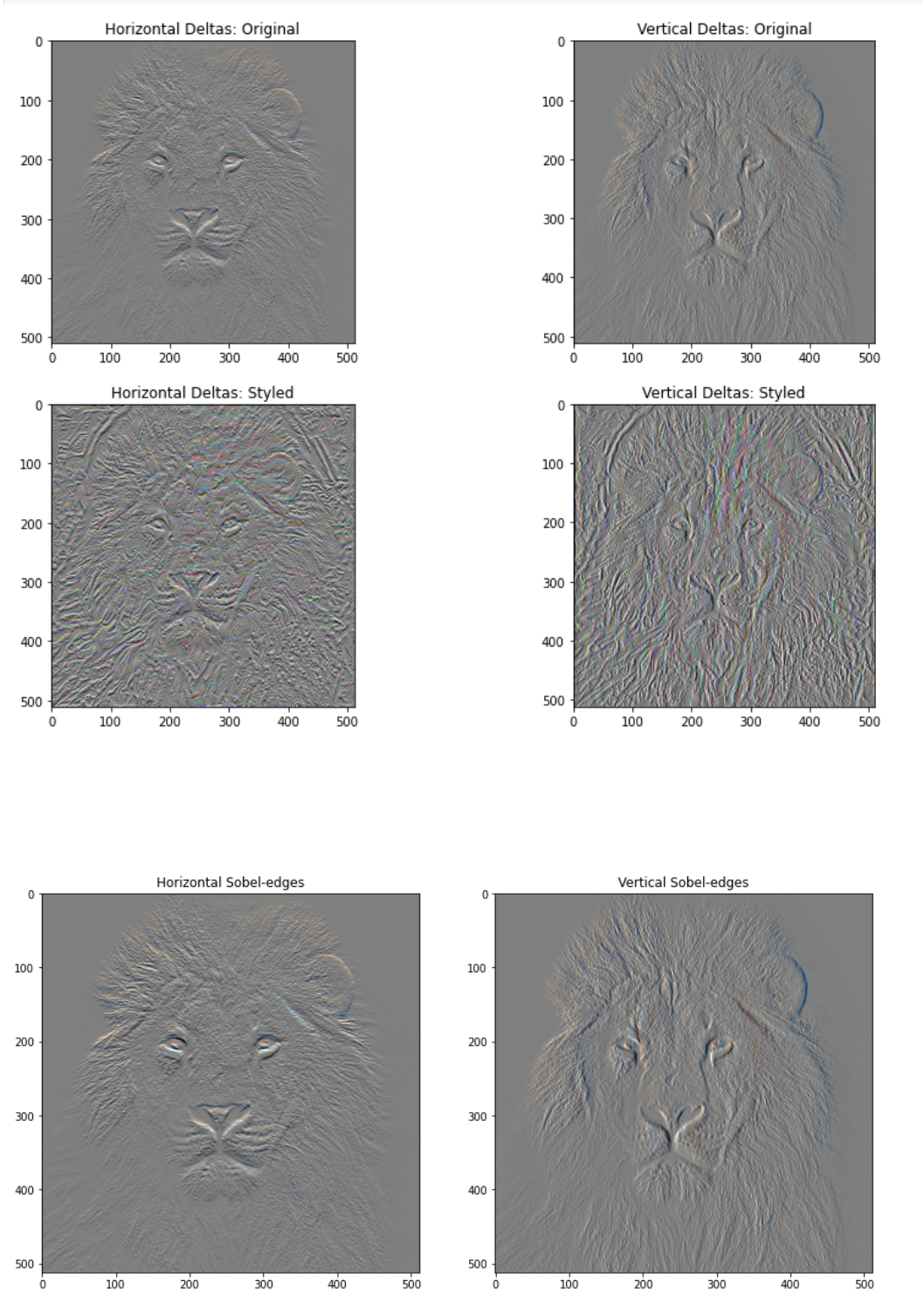
a) Content image and Style image



Implementing style transfer algorithm – Outputs

On Running Gradient Descent

On performing total variation loss function:

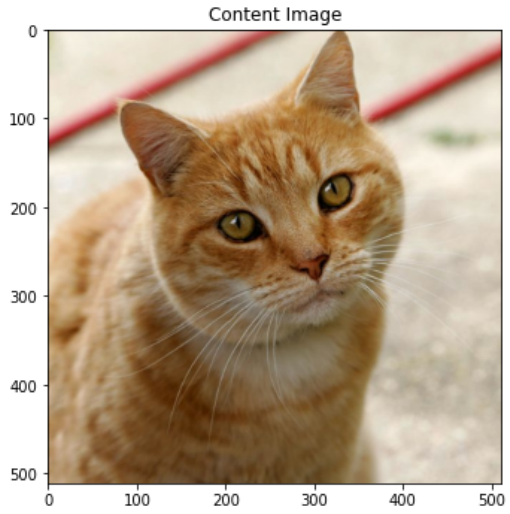


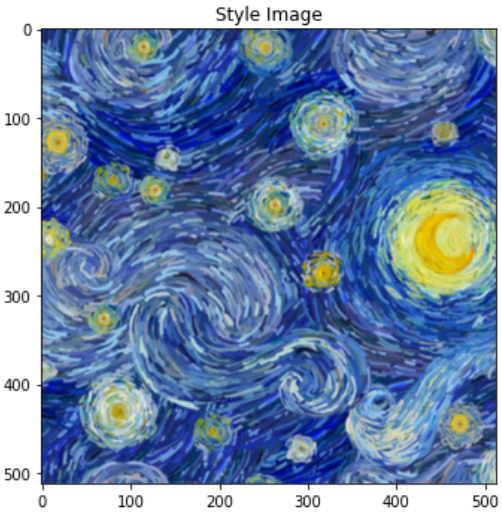
After re-running the optimization
Final stylized image :-

```
Train step: 1000
Total time: 8281.6
```

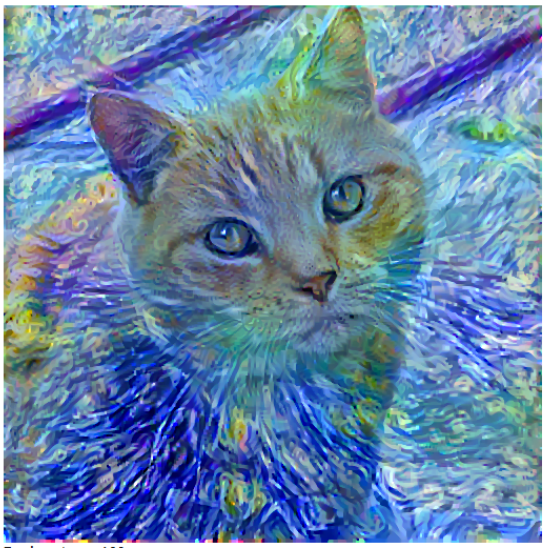
b) Content image and Style image

Implementing style transfer algorithm – Outputs

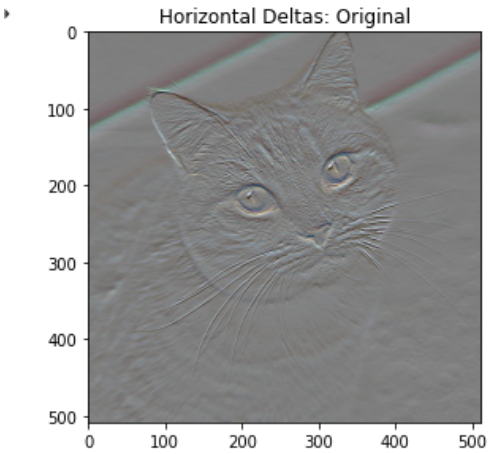


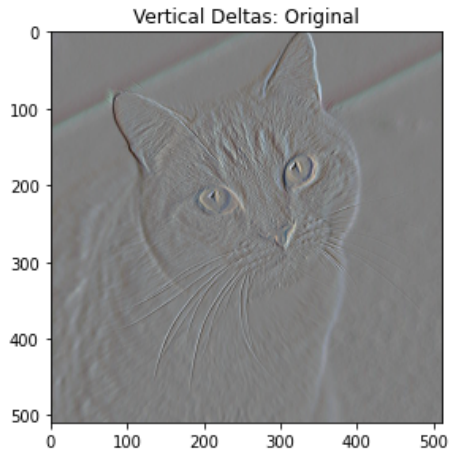


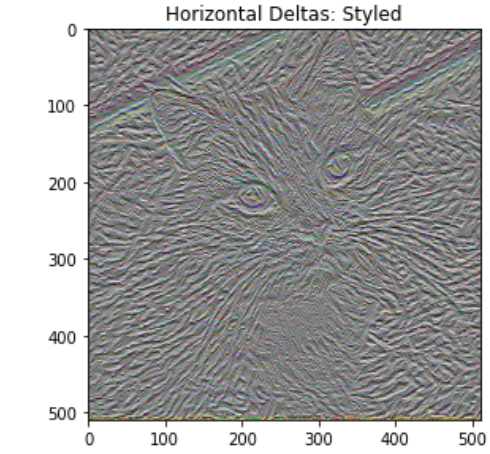
On Running Gradient Descent

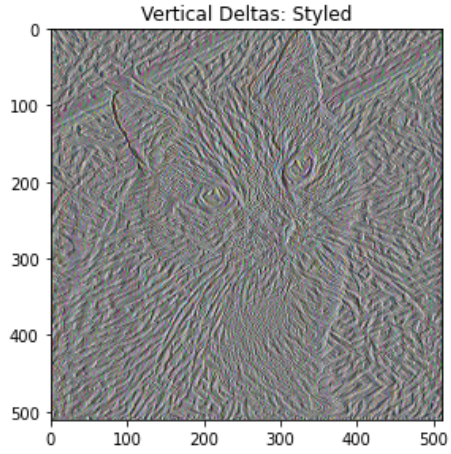


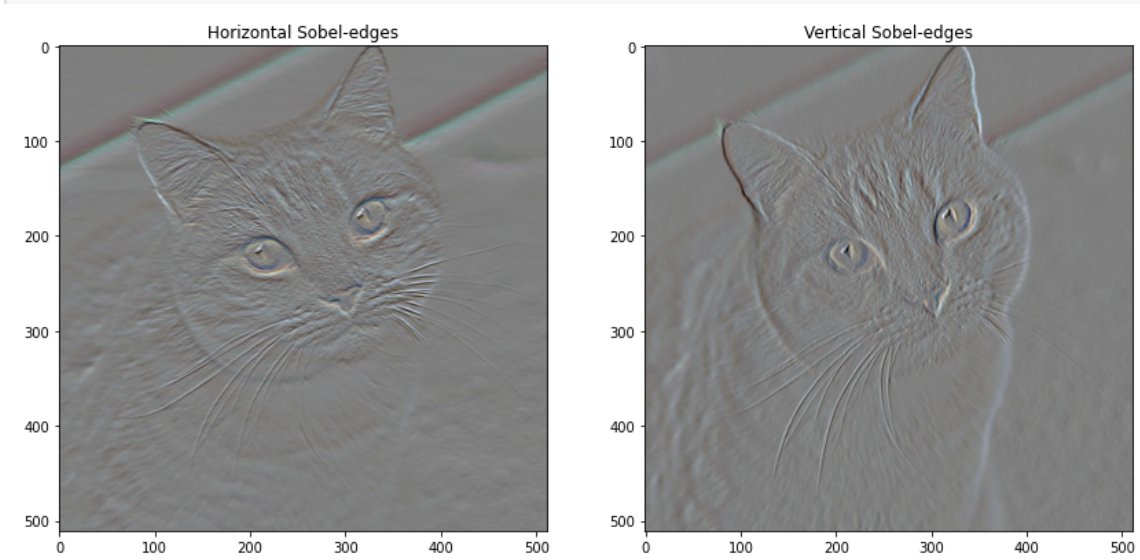
On performing total variation loss function:











After re-running the optimization

Final stylized image :-

```
Train step: 1000
Total time: 6113.3
```

10.EVALUATION

The most common approach in quantitative evaluation is to compute the runtime or the convergence speed(how long it takes the loss function to converge) of different algorithms.

For the test cases:

1) For 1000 training steps
   Run time : 6113.3 seconds (101 minutes approximately)

2)For 1000 training steps
   Run time : 8281.6 seconds (138 minutes approximately)

11.REFERENCES

- Neural Style Transfer: Everything You Need to Know [Guide] (v7labs.com)
- https://sci-hub.hkvisa.net/https://ieeexplore.ieee.org/document/9275956
- Style in Computer Vision — Neural Style Transfer | by Siddhartha Gairola | DataDrivenInvestor