IMAGE STYLIZATION BY AI

PRESENTED BY GOURAV CHOUDHARY - 13, ISHANK VARSHNEY - 15,
ISHITA JAIN - 16 AND VISHVENDRA SINGH - 56

WHAT IS IMAGE STYLIZATION BY AI?

Image stylization is an optimization technique used to take two images—a content image and a style reference image (such as an artwork by a famous painter)—and blend them together so the output image looks like the content image, but “painted” in the style of the style reference image.

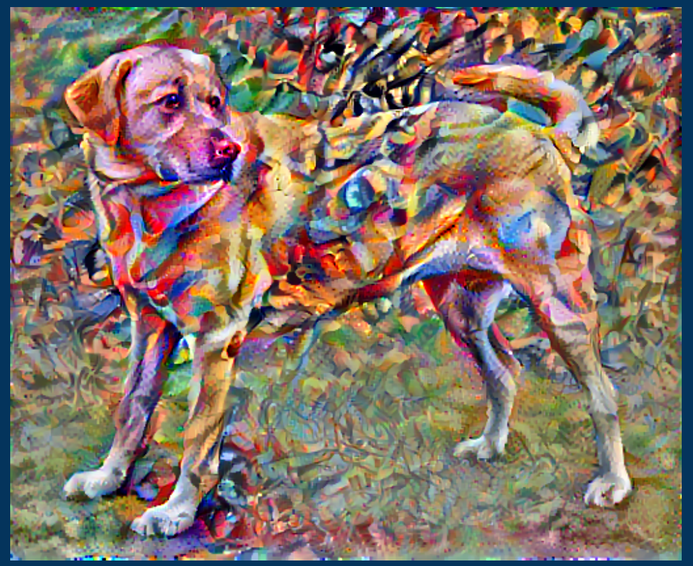
Image Stylization

Content Image

+

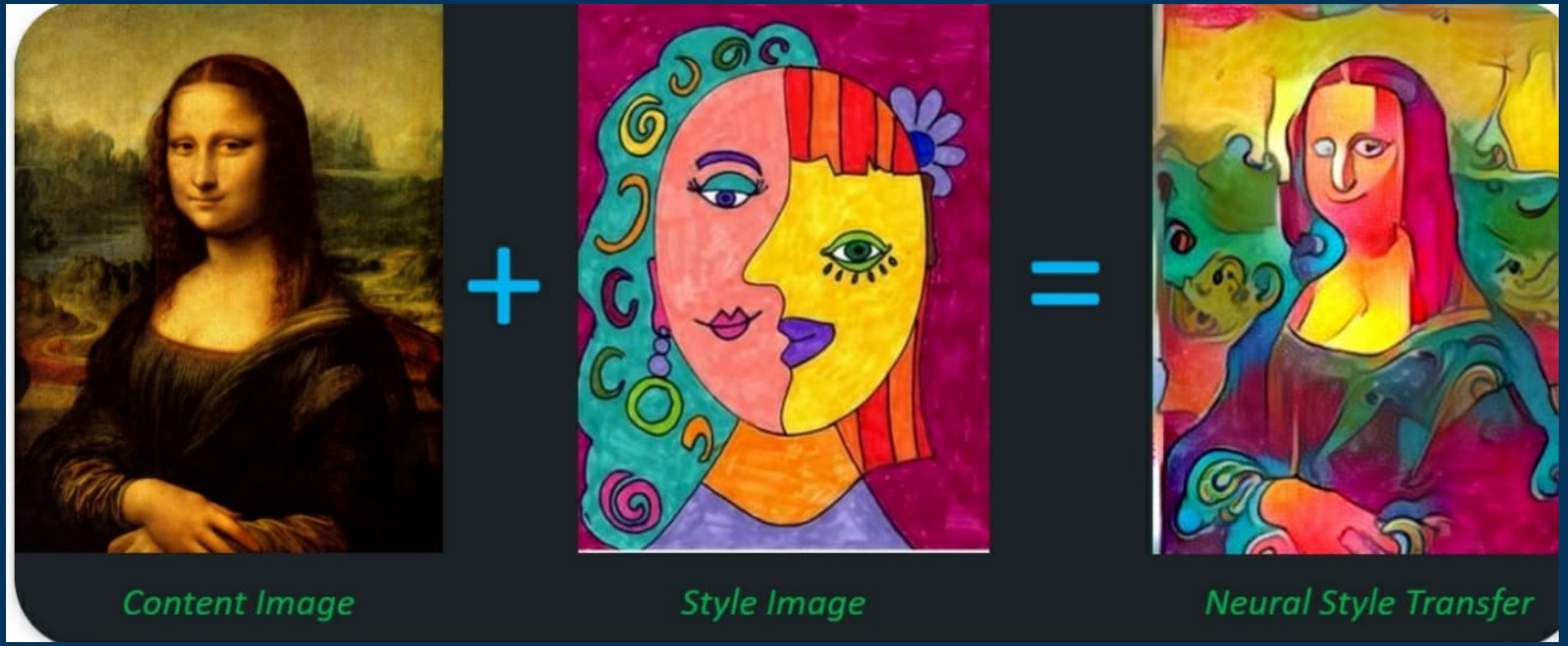
Style Image

=

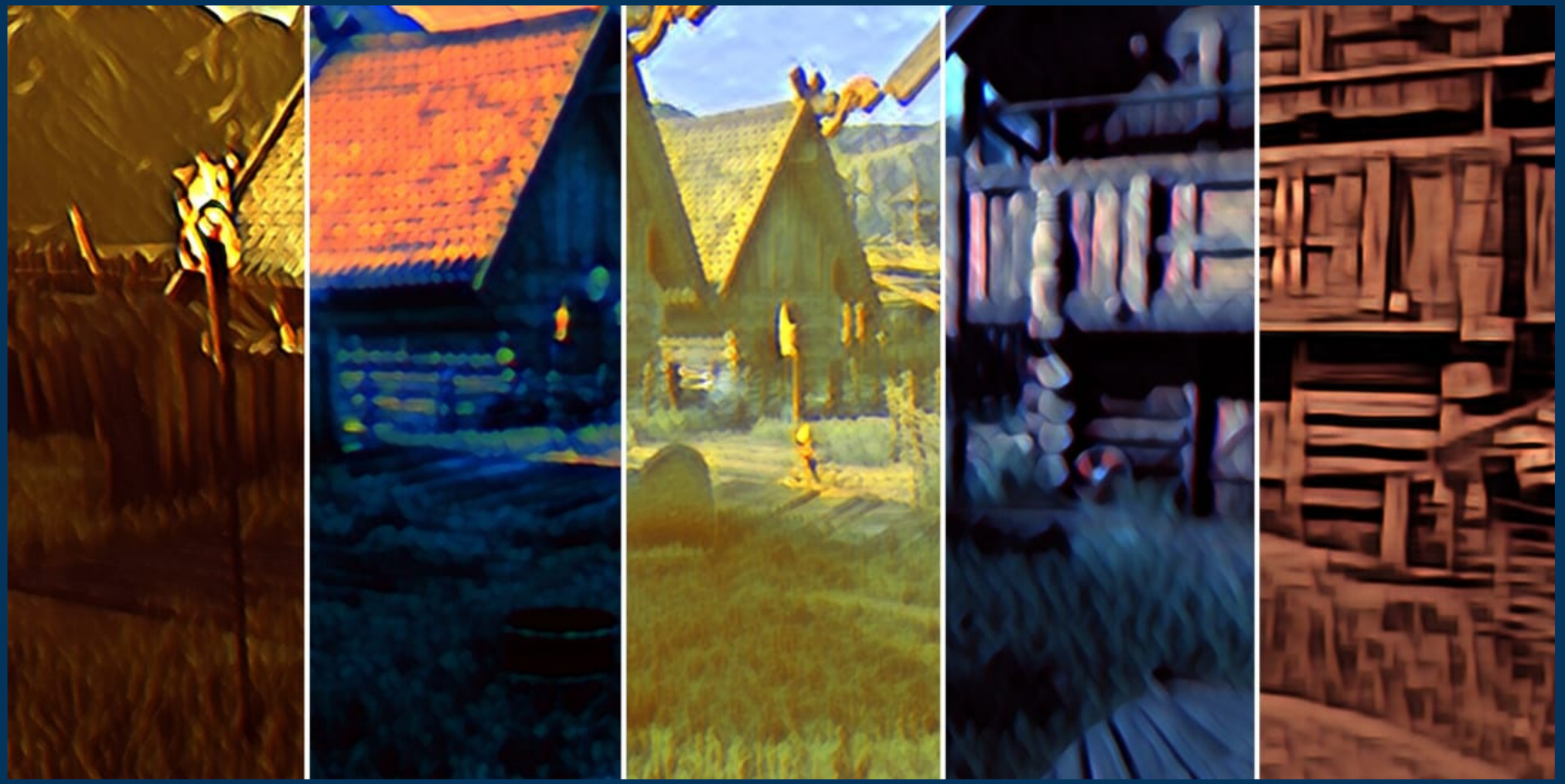


Result

Examples

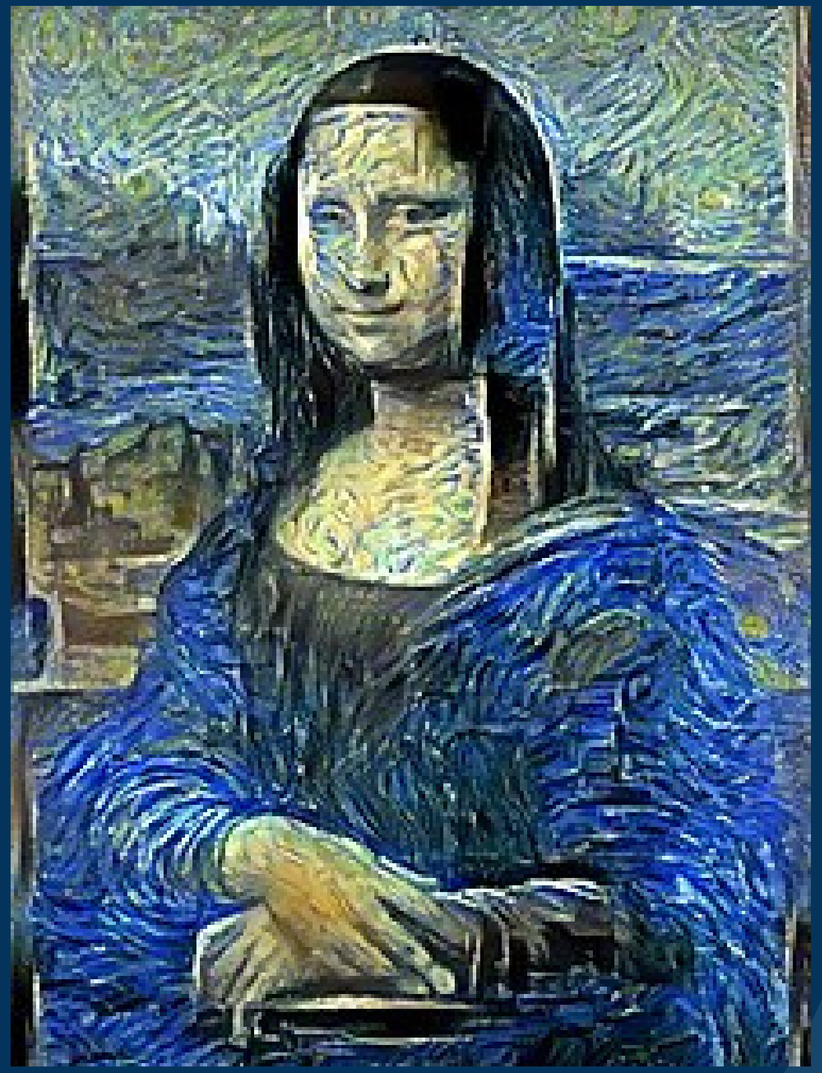


Examples



REAL LIFE APPLICATIONS

WHY IS IT IMPORTANT TO US?



ARTISTIC SATISFACTION

Not everyone is born an artist. Some are more adept at language or other tactile tasks. But with recent advances in technologies like style transfer, almost anyone can enjoy the pleasure that comes along with creating and sharing an artistic masterpiece..

LIVELY ENVIRONMENT

Image stylization helps us make world a beautiful place. It inspires creativity in the art world.

The lens created using this can help people in viewing their surroundings in a new and creative light..

ART EXHIBITIONS FOR AI

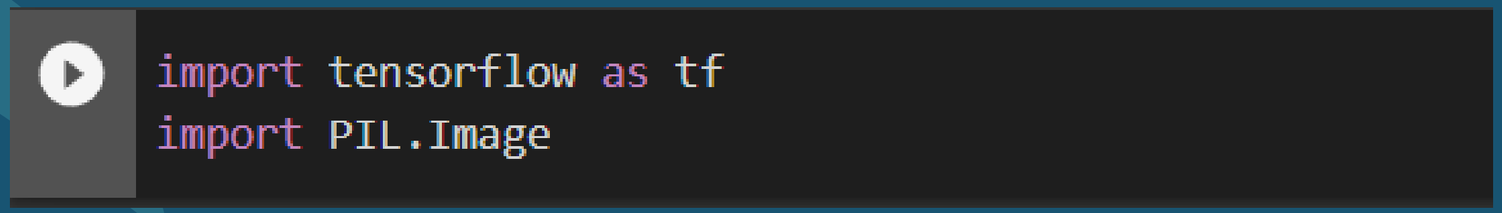
Image stylization can be used by artist to exhibit the art pieces in art galleries that display and exhibition the art pieces that are designed and created by AI..

APPLICATION IN GAMING

Image stylization can be used by game developers to create some attractive and creative effects for various functions in the game. .

LIBRARIES THAT WILL BE USED

- **TensorFlow:** Used for deep learning and neural networks. TensorFlow is a free and open-source software library for machine learning and artificial intelligence. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks.
- **PIL.Image(Python Image Library)**: Used. for image processing and loading. Python Imaging Library is a free and open-source additional library for the Python programming language that adds support for opening, manipulating, and saving many different image file formats. It is available for Windows, Mac OS X and Linux



Importing Python Libraries

[1] 3s

```
import os
import tensorflow as tf
os.environ['TFHUB_MODEL_LOAD_FORMAT'] = 'COMPRESSED'
```

0s

```
import matplotlib.pyplot as plt
import matplotlib as mpl
mpl.rcParams['figure.figsize'] = (12, 12)
mpl.rcParams['axes.grid'] = False

import numpy as np
import PIL.Image
```

Images Used By Our Model

Content Image

Style Image

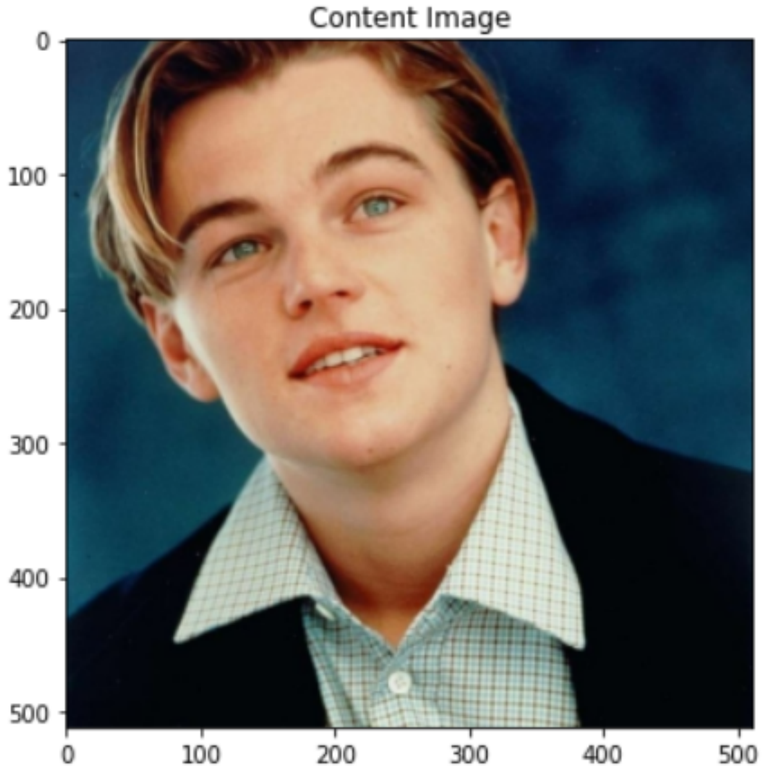
```
[7]  def tensor_image(tensor):
       tensor = tensor*255
       tensor = np.array(tensor, dtype=np.uint8)
       if np.ndim(tensor)>3:
         if tensor.shape[0] == 1:
           tensor = tensor[0]
       return PIL.Image.fromarray(tensor)
```

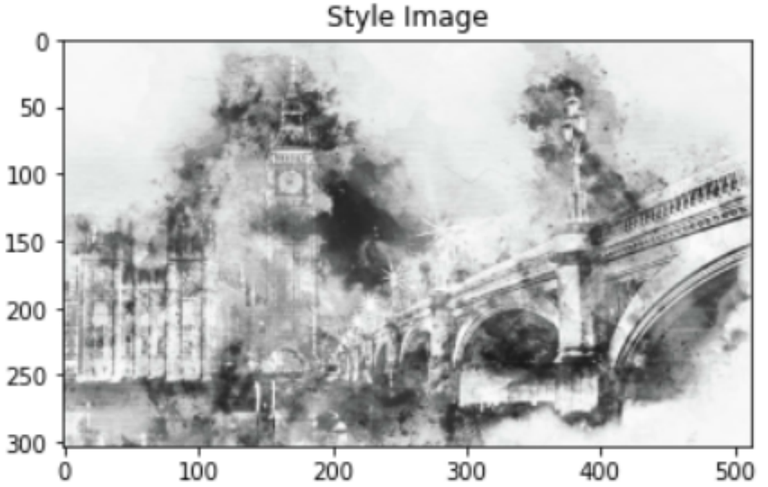
```
[49] content_path = "/content/leonardo.jpg"
     style_path = "/content/london.jpg"
```

```
[50] def load_img(path_to_img):
       max_dim = 512
       img = tf.io.read_file(path_to_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
```

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

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

```
[52] content_image = load_img(content_path)
     style_image = load_img(style_path)

     plt.subplot(1, 2, 1)
     imshow(content_image, 'Content Image')

     plt.subplot(1, 2, 2)
     imshow(style_image, 'Style Image')
```

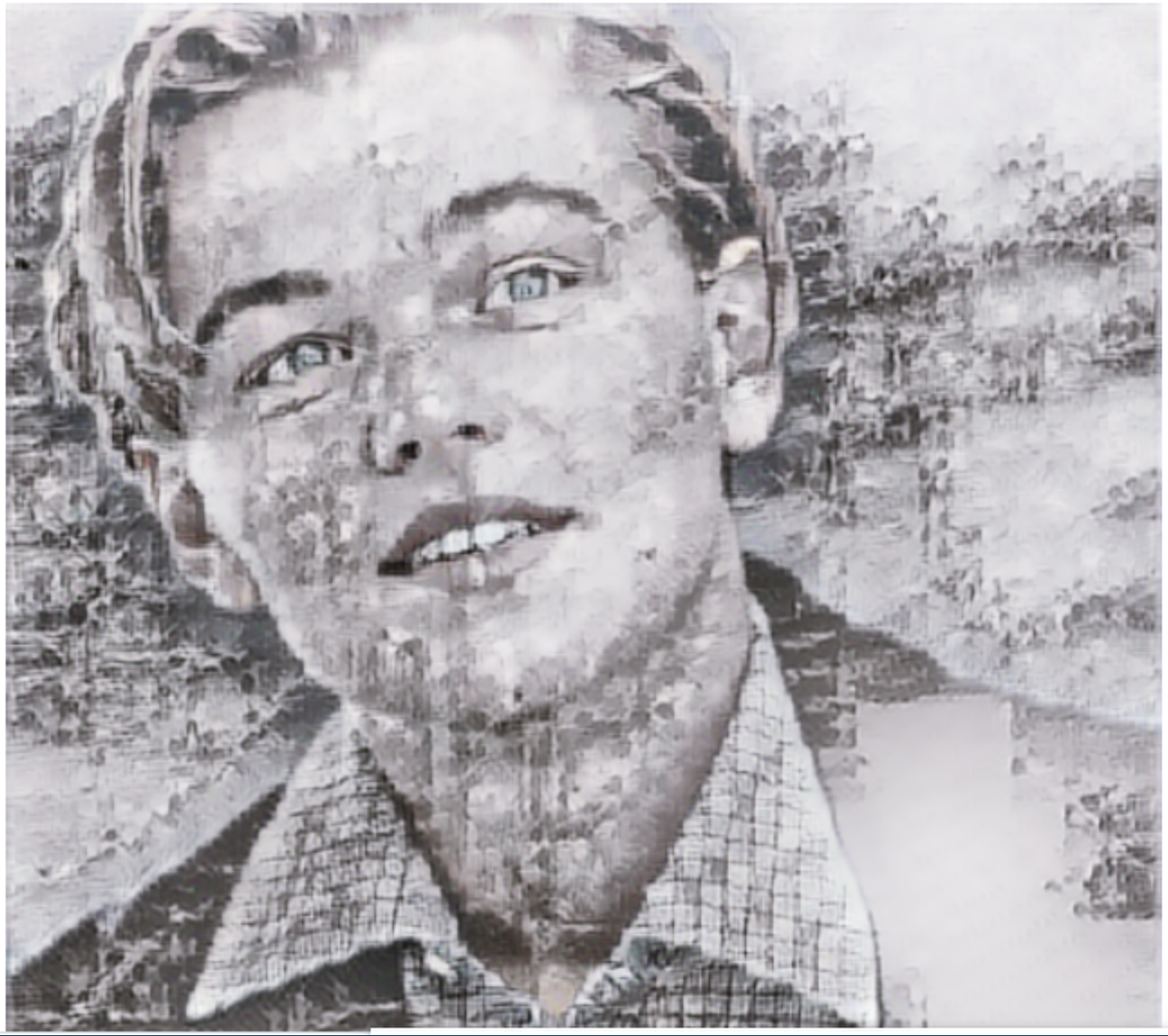




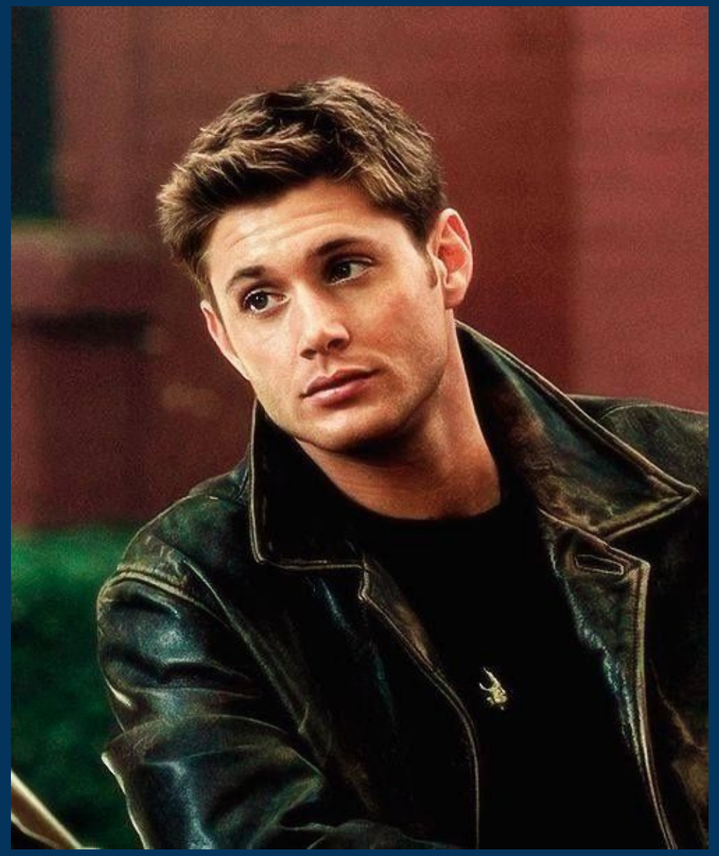
TENSORFLOW HUB

TensorFlow Hub is a repository of trained machine learning models ready for fine-tuning and deployable anywhere. Reuse trained models like BERT and Faster R-CNN with just a few lines of code. dev repository provides many pre-trained models: text embeddings, image classification models, TF. js/TFLite models and much more.

[53]
```
import tensorflow_hub as hub
hub_model = hub.load('https://tfhub.dev/google/magenta/arbitrary-image-stylization-v1-256/2')
stylized_image = hub_model(tf.constant(content_image), tf.constant(style_image))[0]
tensor_image(stylized_image)
```



Why We Need a better model



Content Image

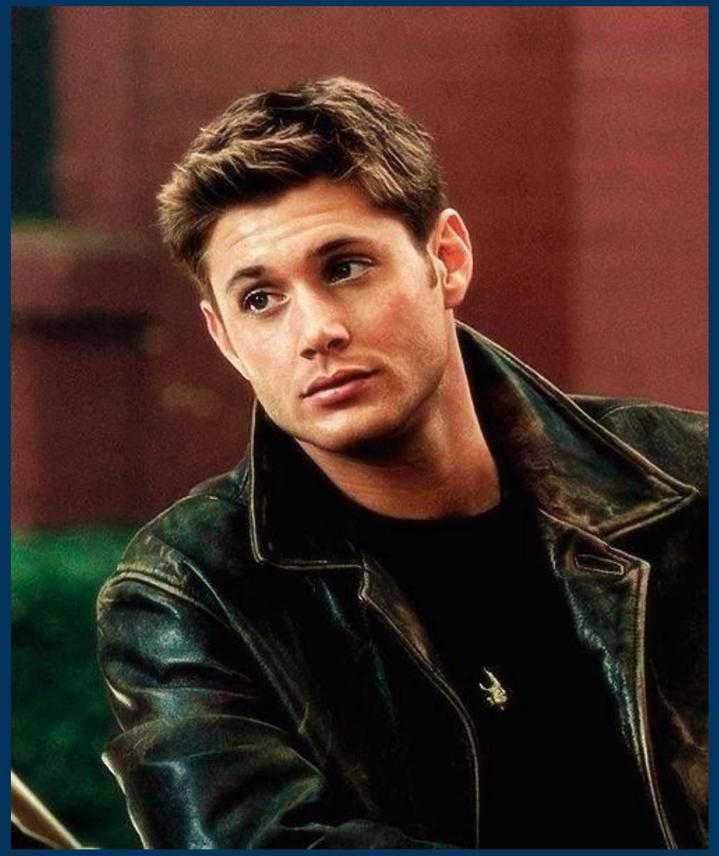
+

Style Image

=

Stylised Image

Why We Need a better model

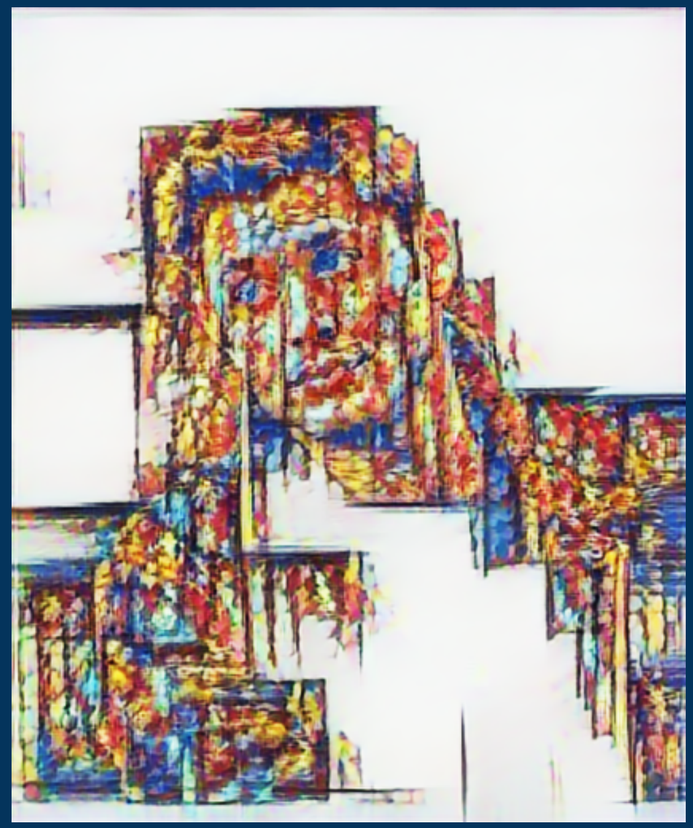


Content Image

+

Style Image

=



Stylised Image

CONVOLUTIONAL NEURAL NETWORK

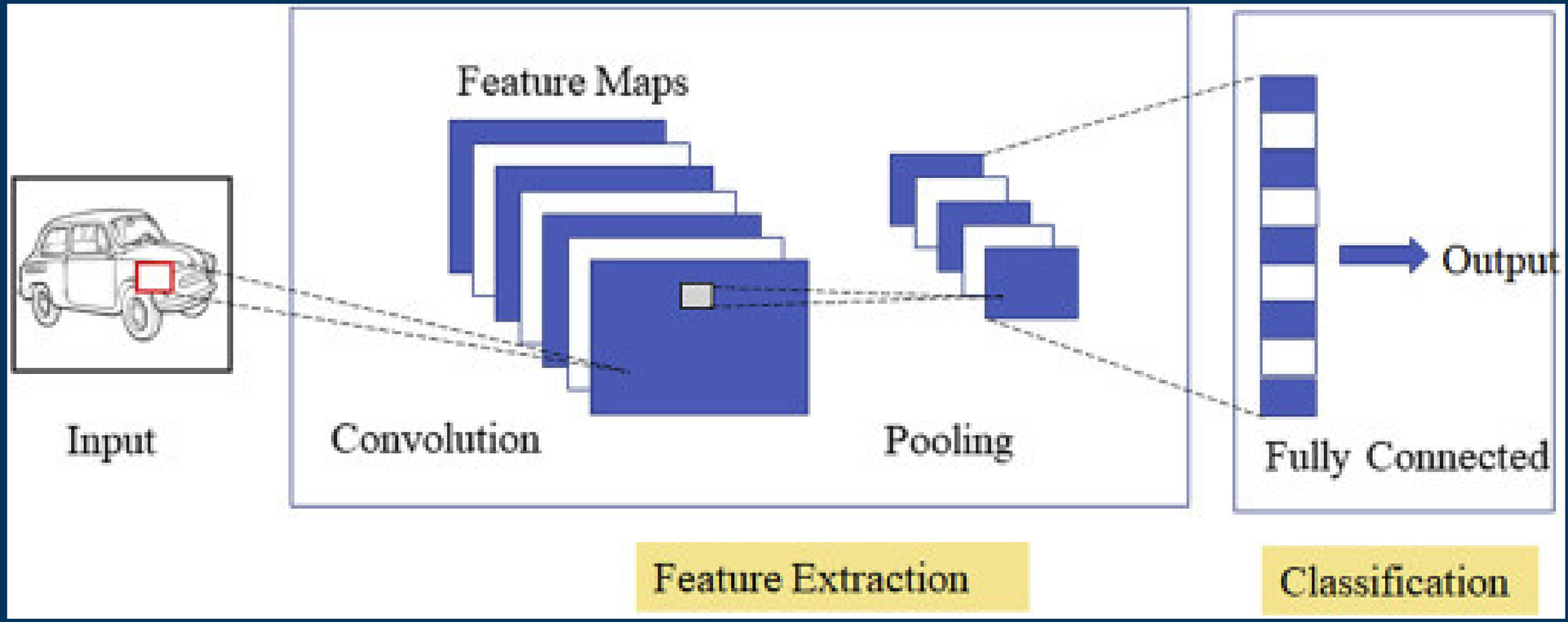
A **Convolutional Neural Network (ConvNet/CNN)** is a Deep Learning algorithm which can take in an input image, assign importance, learnable weights and biases, to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. CNN extract the each and every portion of input image, which is known as receptive field. It assigns weights for each neuron based on the significant role of the receptive field. So that it can discriminate the importance of neurons from one another.

LAYERS OF CNN

Convolutional neural networks are distinguished from other neural networks by their superior performance with image, speech, or audio signal inputs. They have three main types of layers, which are:

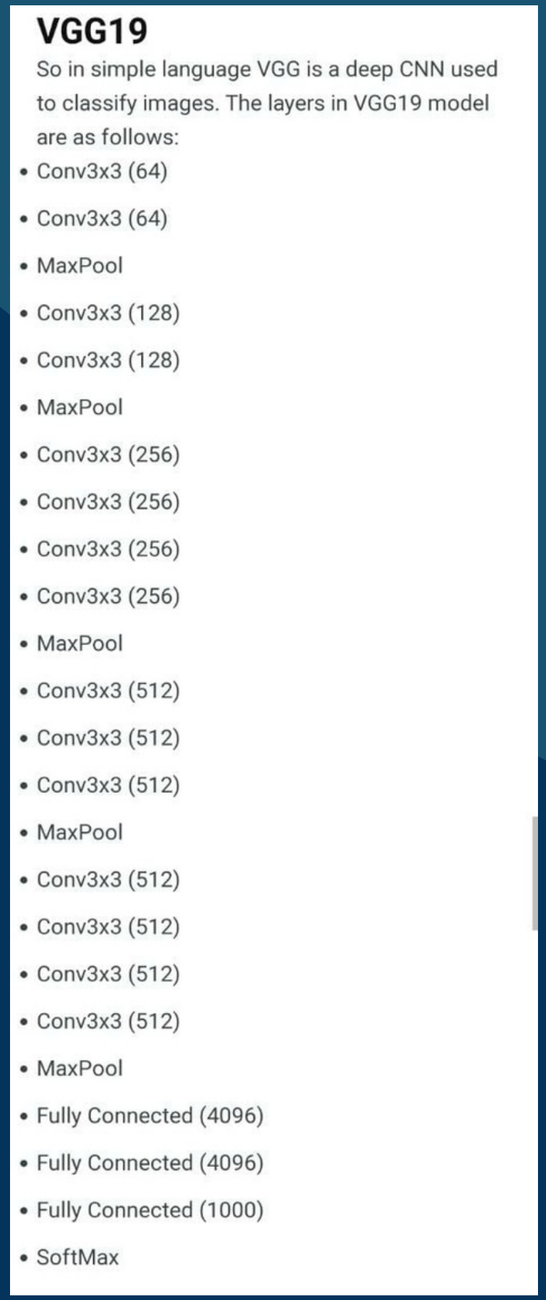
- Convolutional layer
- Pooling layer
- Fully-connected (FC) layer

CNN LAYERS

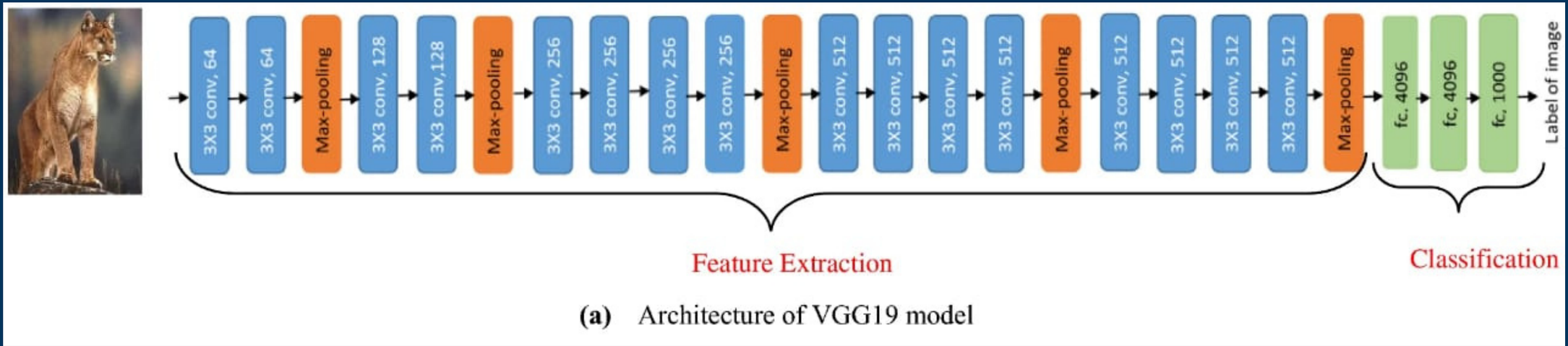

Feature Maps
Input
Convolution
Pooling
Output
Fully Connected
Feature Extraction
Classification

VGG19

VGG19 proposed by Simonyan and Zisserman (2014) is a convolutional neural network that comprises 19 layers with 16 convolution layers and 3 fully connected to classify the images into 1000 object categories. We can load a pretrained version of the network trained on more than a million images from the ImageNet database. The pretrained network can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals. As a result, the network has learned rich feature representations for a wide range of images. The network has an image input size of 224-by-224.



Working of VGG19



(a) Architecture of VGG19 model

Resource: https://link.springer.com/article/10.1007/s12652-021-03488-z/figures/2

Libraries Used

```
import os
import tensorflow as tf
os.environ['TFHUB_MODEL_LOAD_FORMAT'] = 'COMPRESSED'
```

```
import matplotlib.pyplot as plt
import matplotlib as mpl
mpl.rcParams['figure.figsize'] = (12, 12)
mpl.rcParams['axes.grid'] = False

import numpy as np
import PIL.Image
import time
import functools
from IPython.display import display
from IPython.display import clear_output
```

```
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
```

```
TensorShape([1, 1000])
```

```
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]
```

```
[('French_horn', 0.23891814),
 ('trench_coat', 0.106828496),
 ('bassoon', 0.06372758),
 ('sax', 0.052020073),
 ('suit', 0.03468096)]
```

```python
vgg = tf.keras.applications.VGG19(include_top=False, weights='imagenet')

print()
for layer in vgg.layers:
  print(layer.name)
```

```
input_14
block1_conv1
block1_conv2
block1_pool
block2_conv1
block2_conv2
block2_pool
block3_conv1
block3_conv2
block3_conv3
block3_conv4
```

```
[101] content_layers = ['block5_conv2']

      style_layers = ['block1_conv1',
                      'block2_conv1',
                      'block3_conv1',
                      'block4_conv1',
                      'block5_conv1']

      num_content_layers = len(content_layers)
      num_style_layers = len(style_layers)
```

```
[102] def vgg_layers(layer_names):
        # Load our model. Load pretrained VGG, trained on imagenet data
        vgg = tf.keras.applications.VGG19(include_top=False, weights='imagenet')
        vgg.trainable = False

        outputs = [vgg.get_layer(name).output for name in layer_names]

        model = tf.keras.Model([vgg.input], outputs)
        return model
```

```
style_extractor = vgg_layers(style_layers)
style_outputs = style_extractor(style_image*255)

#Look at the statistics of each layer's output
for name, output in zip(style_layers, style_outputs):
  print(name)
  print("  shape: ", output.numpy().shape)
  print("  min: ", output.numpy().min())
  print("  max: ", output.numpy().max())
  print("  mean: ", output.numpy().mean())
  print()
```

```
block1_conv1
  shape:  (1, 512, 512, 64)
  min:  0.0
  max:  872.6996
  mean:  39.810673

block2_conv1
  shape:  (1, 256, 256, 128)
  min:  0.0
  max:  4270.8975
  mean:  228.66785
```

```
def gram_matrix(input_tensor):
  result = tf.linalg.einsum('bijc,bijd->bcd', input_tensor, input_tensor)
  input_shape = tf.shape(input_tensor)
  num_locations = tf.cast(input_shape[1]*input_shape[2], tf.float32)
  return result/(num_locations)
```

```
class StyleContentModel(tf.keras.models.Model):
  def __init__(self, style_layers, content_layers):
    super(StyleContentModel, self).__init__()
    self.vgg = vgg_layers(style_layers + content_layers)
    self.style_layers = style_layers
    self.content_layers = content_layers
    self.num_style_layers = len(style_layers)
    self.vgg.trainable = False

  def call(self, inputs):
    "Expects float input in [0,1]"
    inputs = inputs*255.0
    preprocessed_input = tf.keras.applications.vgg19.preprocess_input(inputs)
    outputs = self.vgg(preprocessed_input)
    style_outputs, content_outputs = (outputs[:self.num_style_layers],
                                      outputs[self.num_style_layers:])

    style_outputs = [gram_matrix(style_output)
                     for style_output in style_outputs]

    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}
```

```
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:")
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:  0.0001758667
    max:  37996.324
    mean:  506.44077
```

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

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

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

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

```
style_weight=1e-3
content_weight=1e6
```

```python
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)
                           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
```

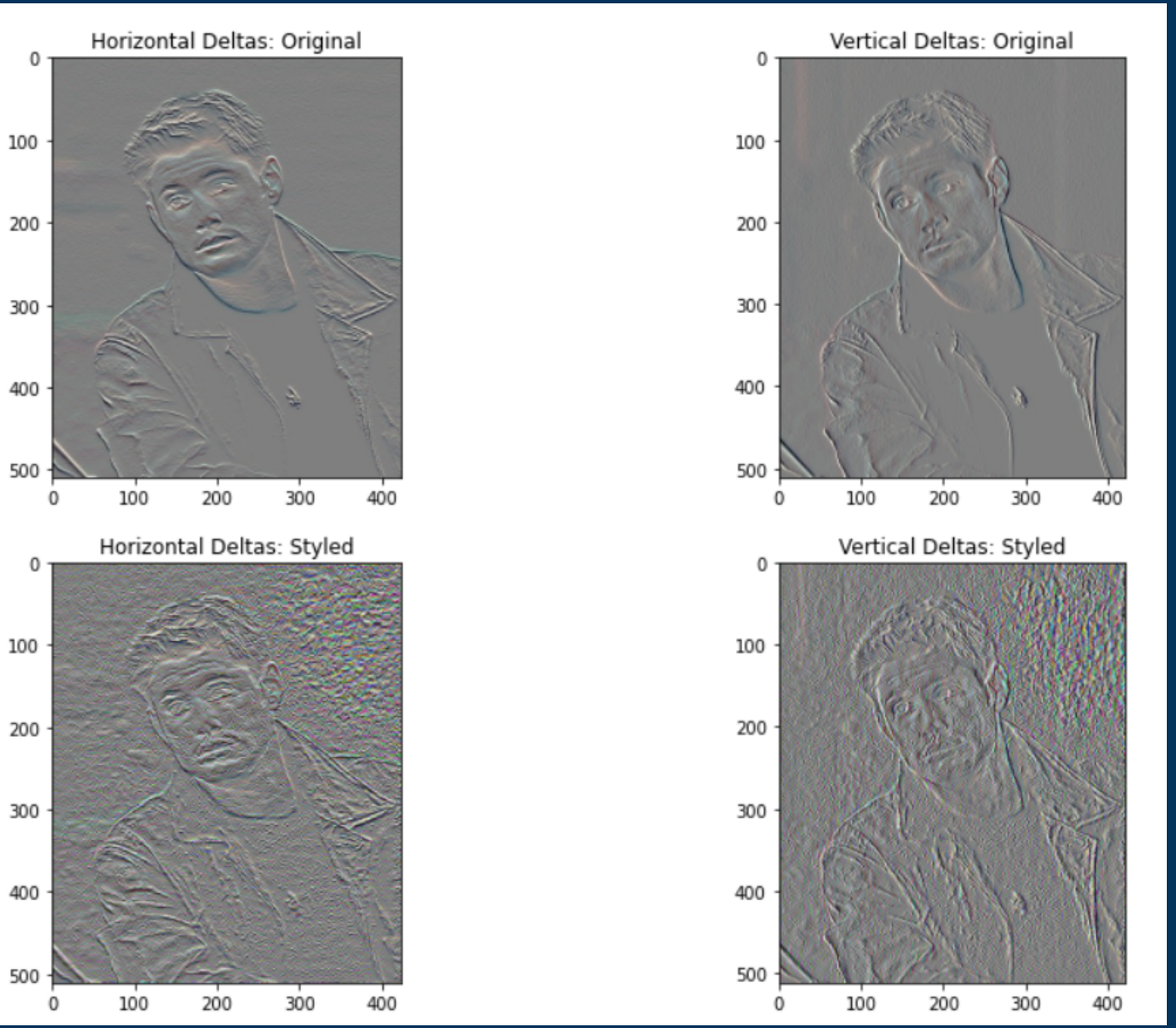
```python
@tf.function()
def train_step(image):
  with tf.GradientTape() as tape:
    outputs = extractor(image)
    loss = style_content_loss(outputs)

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

```
train_step(image)
train_step(image)
train_step(image)
tensor_image(image)
```

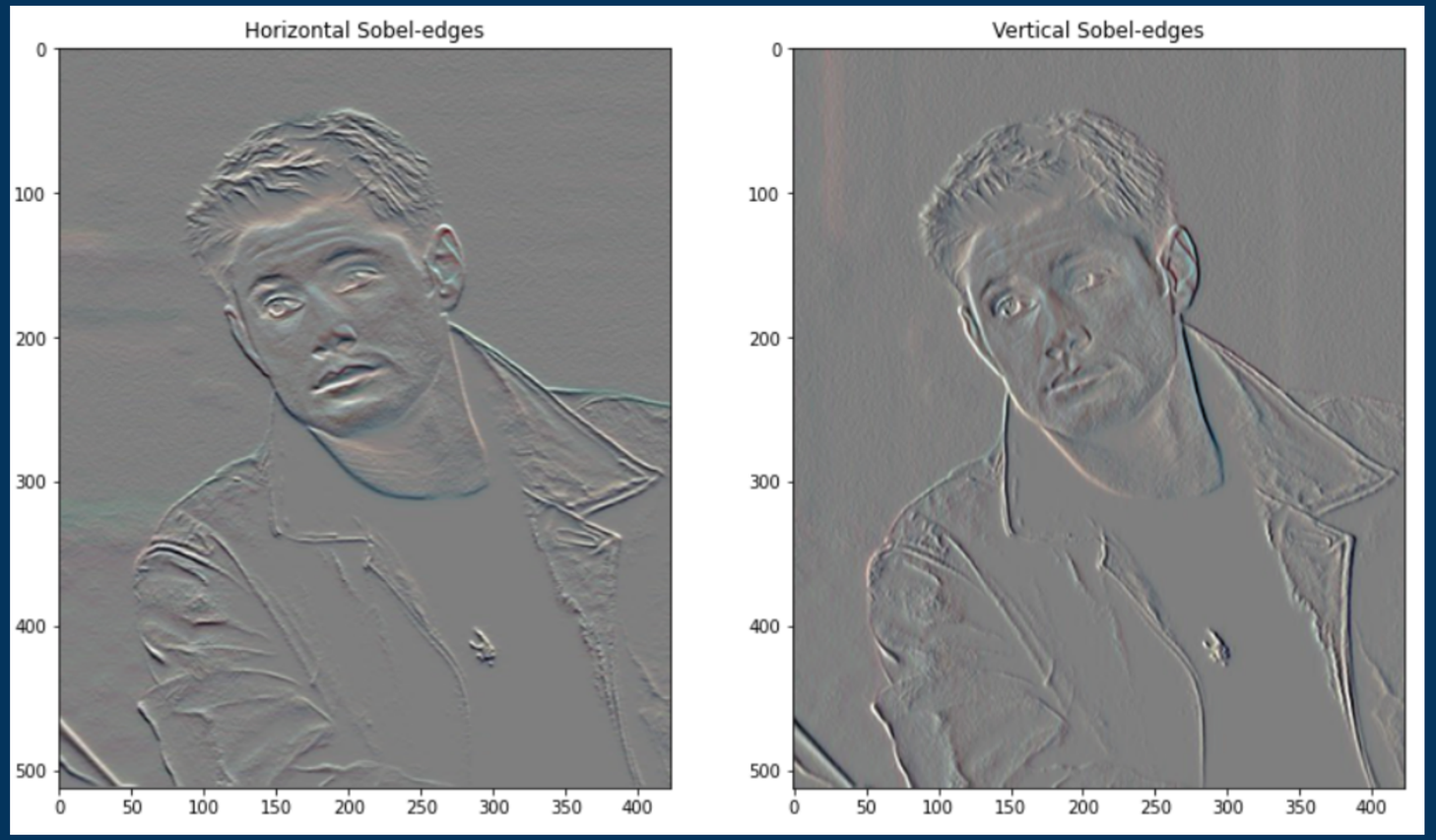
```python
import time
start = time.time()

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)
  clear_output(wait=True)
  display(tensor_image(image))
  print("Train step: {}".format(step))

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

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

  return x_var, y_var
```

```python
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")

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


Horizontal Deltas: Original
Vertical Deltas: Original
Horizontal Deltas: Styled
Vertical Deltas: Styled
0
100
200
300
400
500

```
plt.figure(figsize=(14, 10))

sobel = tf.image.sobel_edges(content_image)
plt.subplot(1, 2, 1)
imshow(clip_0_1(sobel[..., 0]/4+0.5), "Horizontal Sobel-edges")
plt.subplot(1, 2, 2)
imshow(clip_0_1(sobel[..., 1]/4+0.5), "Vertical Sobel-edges")
```



```
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()
```

```
142813.94
```

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

```
array([142813.94], dtype=float32)
```

```
total_variation_weight=30
```

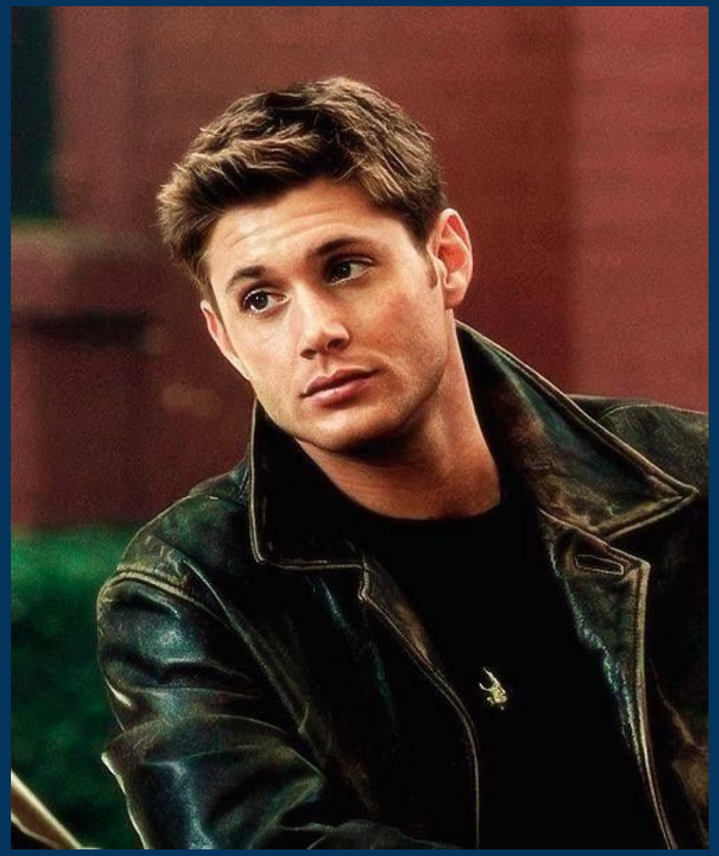
```python
@tf.function()
def train_step(image):
  with tf.GradientTape() as tape:
    outputs = extractor(image)
    loss = style_content_loss(outputs)
    loss += total_variation_weight*tf.image.total_variation(image)

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

```python
image = tf.Variable(content_image)
```

```python
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)
  clear_output(wait=True)
  display(tensor_image(image))
  print("Train step: {}".format(step))
```

```
file_name = 'stylized-image.png'
tensor_image(image).save(file_name)

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

Result with VGG19 model



Content Image

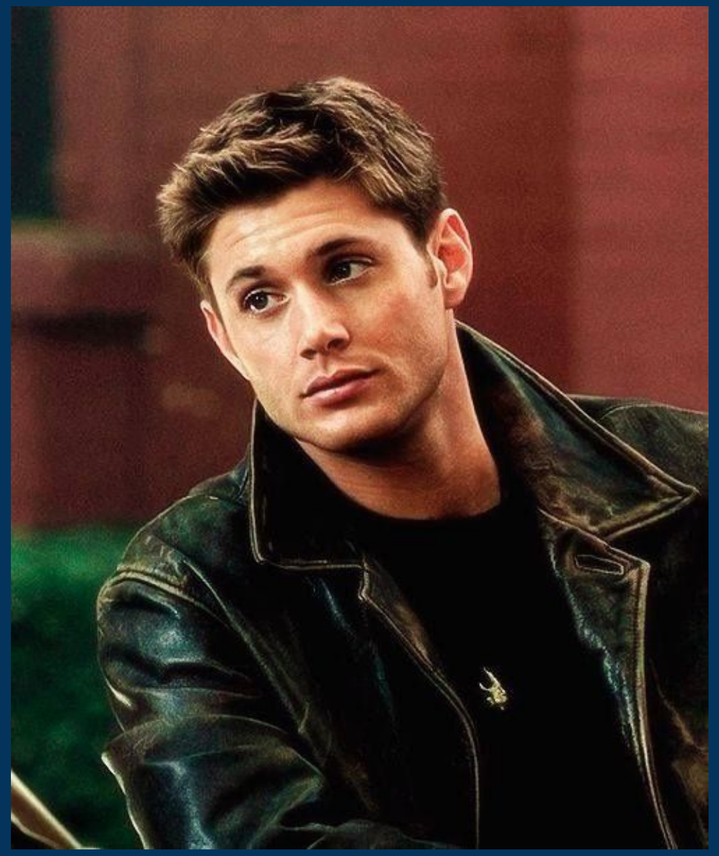
+

Style Image

=

Stylised Image

Result with TF Hub Model

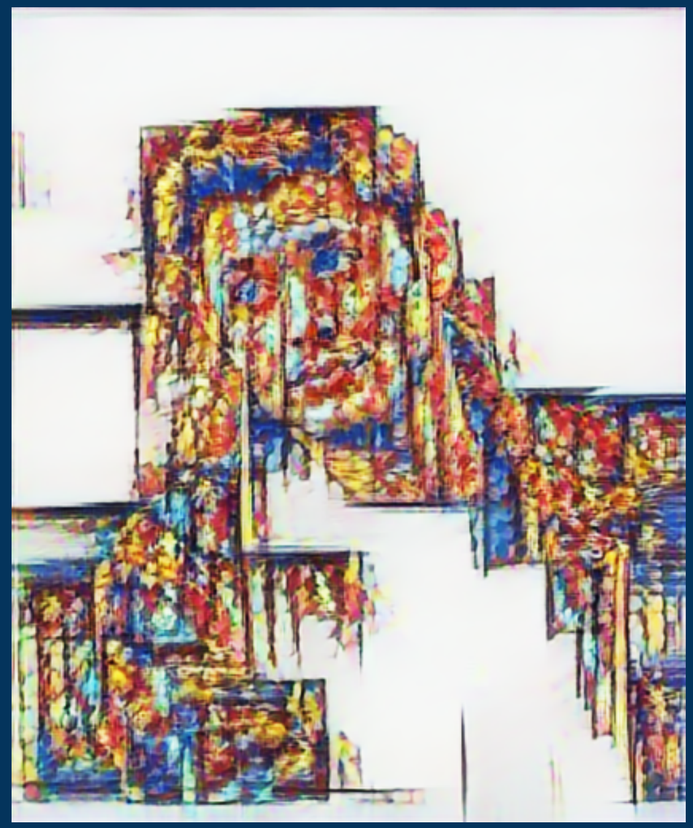


Content Image

+

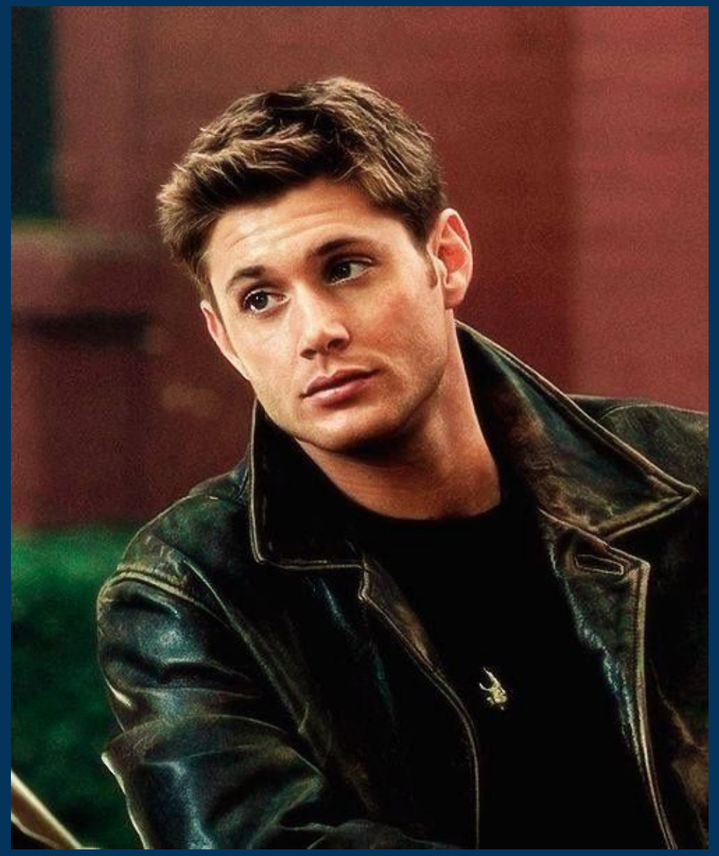
Style Image

=



Stylised Image

Result with VGG19 model



Content Image

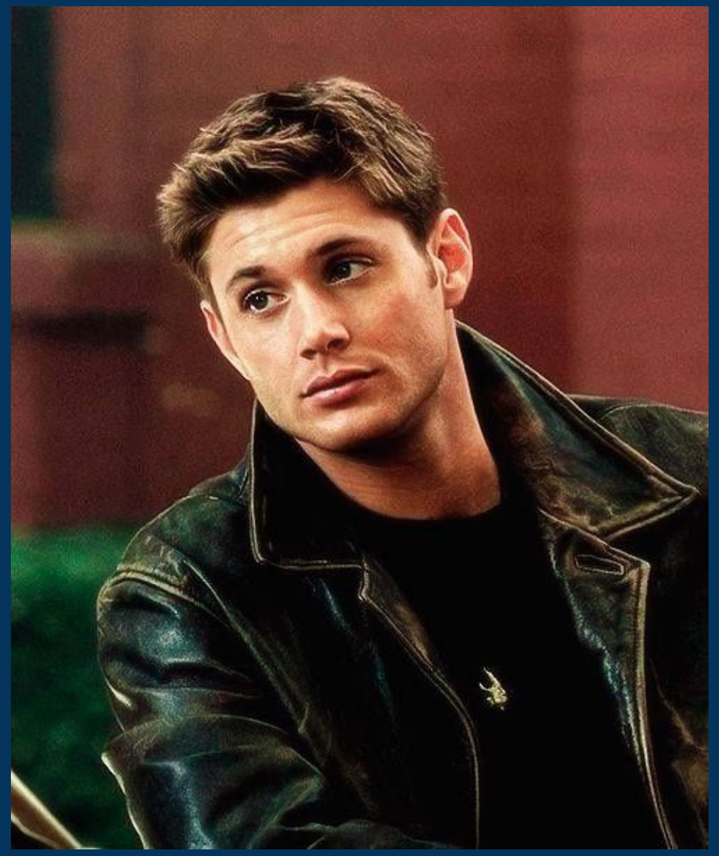
+

Style Image

=

Stylised Image

Result with TF Hub Model



Content Image

+

Style Image

=

Stylised Image

Design is the silent ambassador of your brand.