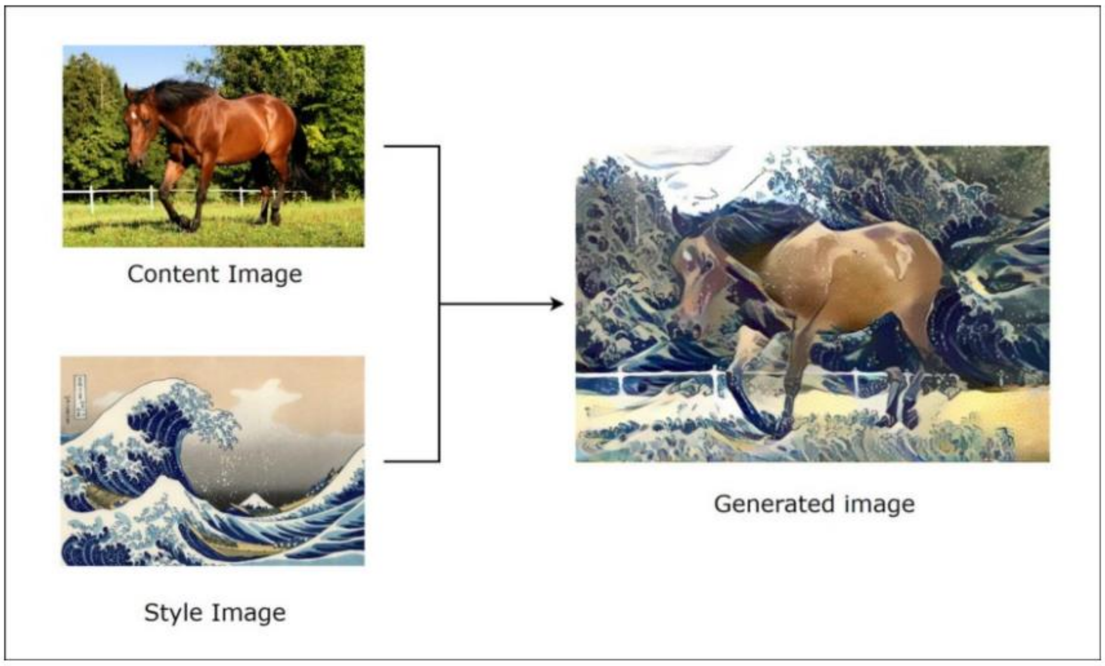
Project Name: Image Style Transfer with Link Only

Fast Neural Style Transfer with TensorFlow Hub & Magenta

Style Transfer:

During the past few years, we've seen a slew of apps like prisma and other similar apps popping up which style your photos in a way wherein they look like paintings. Offering you a variety of beautiful styles some of which are paintings by famous artists like Starry Night by Van Gogh. Trying to explain this concept just with words might be difficult.



Convolutional Neural Networks were originally created for classification of images and have lately been used in a variety of other tasks like Image Segmentation, Neural Style and other computer vision and Natural Language Processing tasks as well.

What insights can convolutional neural network provide?

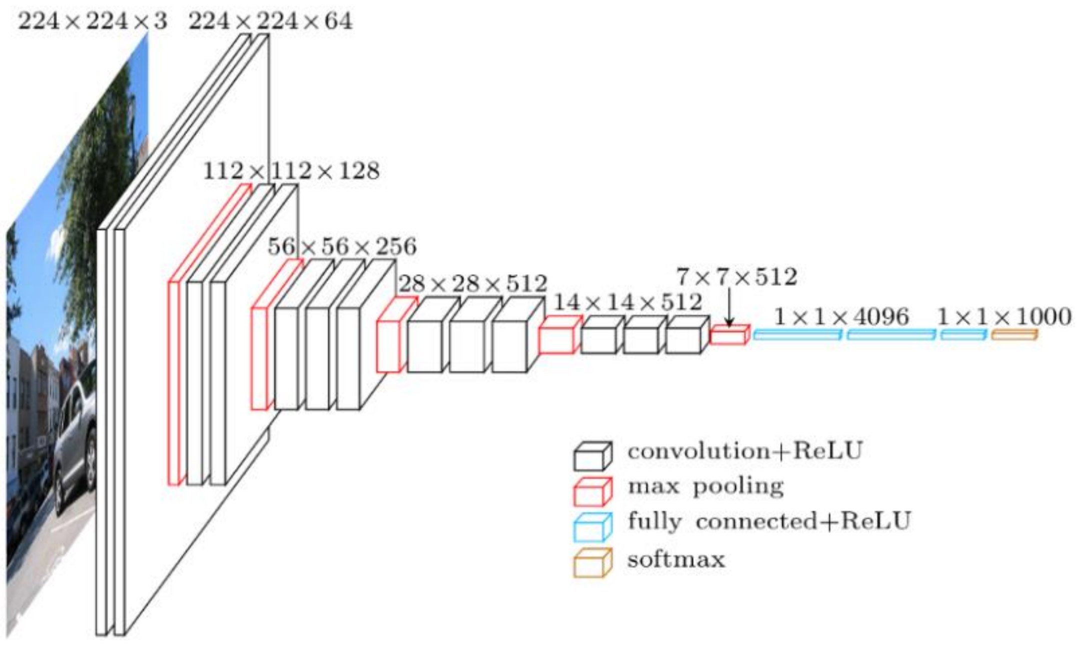


Image having some pattern similar to three intersecting lines.

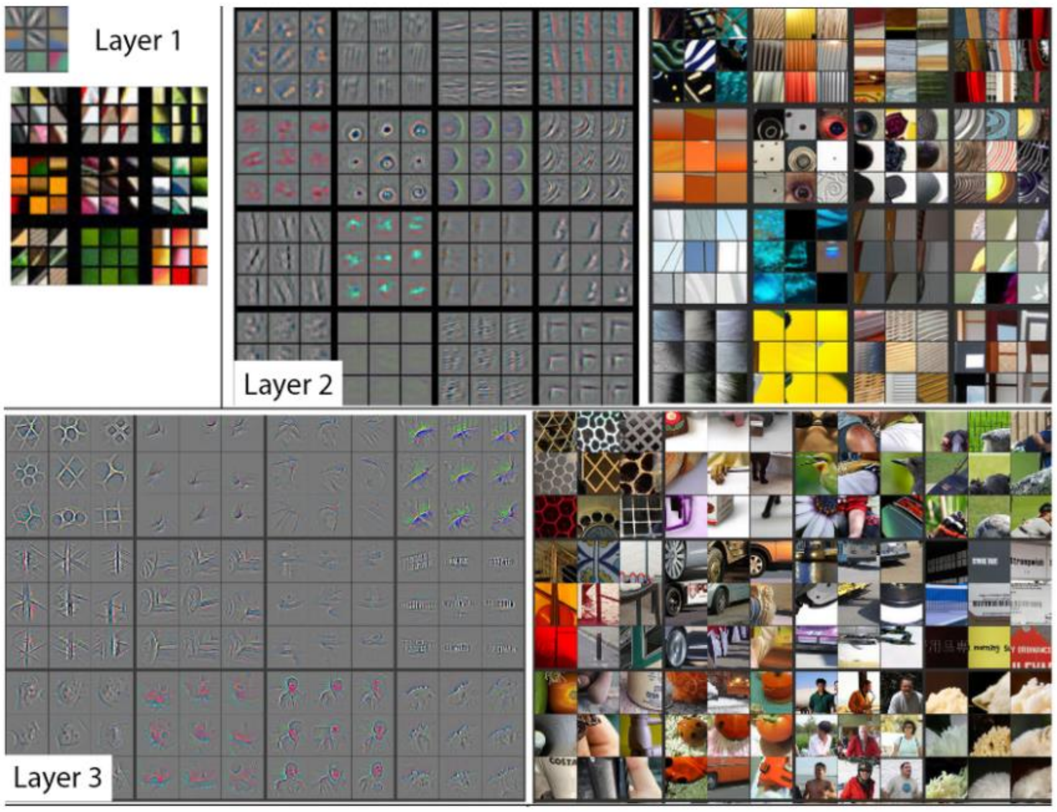


Figure 4 Visualizing and understanding convolutional networks

How these image representations help in style transfer?

Encoding representations itself is the key to style transfer it is used to calculate loss between the generated image with respect to content and style image. As training the model over a ten thousand of images per class the model is able to generate similar feature representation for many different images given, they belong to same class or have similar content or style. Hence it makes sense to use the difference in value of feature representation of generated image w.r.t content and style image to guide the iterations through which we produce the generated image itself but how do we make sure that content image (C) and generated image (G) are similar with respect to their content and not style, while on other hand how do we make sure that generated image only inherits similar style representation from style image (S) and not the entire style image itself. This is solved by dividing the loss function into two parts, one is the Content loss and the other is the Style loss and soon enough we will understand how they are different from each other and how they overcome the problems which we have put forth.

Loss function

$$L_{total}(S, C, G) = \alpha L_{content}(C, G) + \beta L_{style}(S, G)$$

There are two things we need to calculate to get overall loss i.e content loss and style loss, alpha and beta hyperparameters which are used to provide weights to each type of loss i.e these parameters can be thought of simply as knobs to control how much of the content/style we want to inherit in the generated image. So, let’s get to understand what each of this loss term entails.

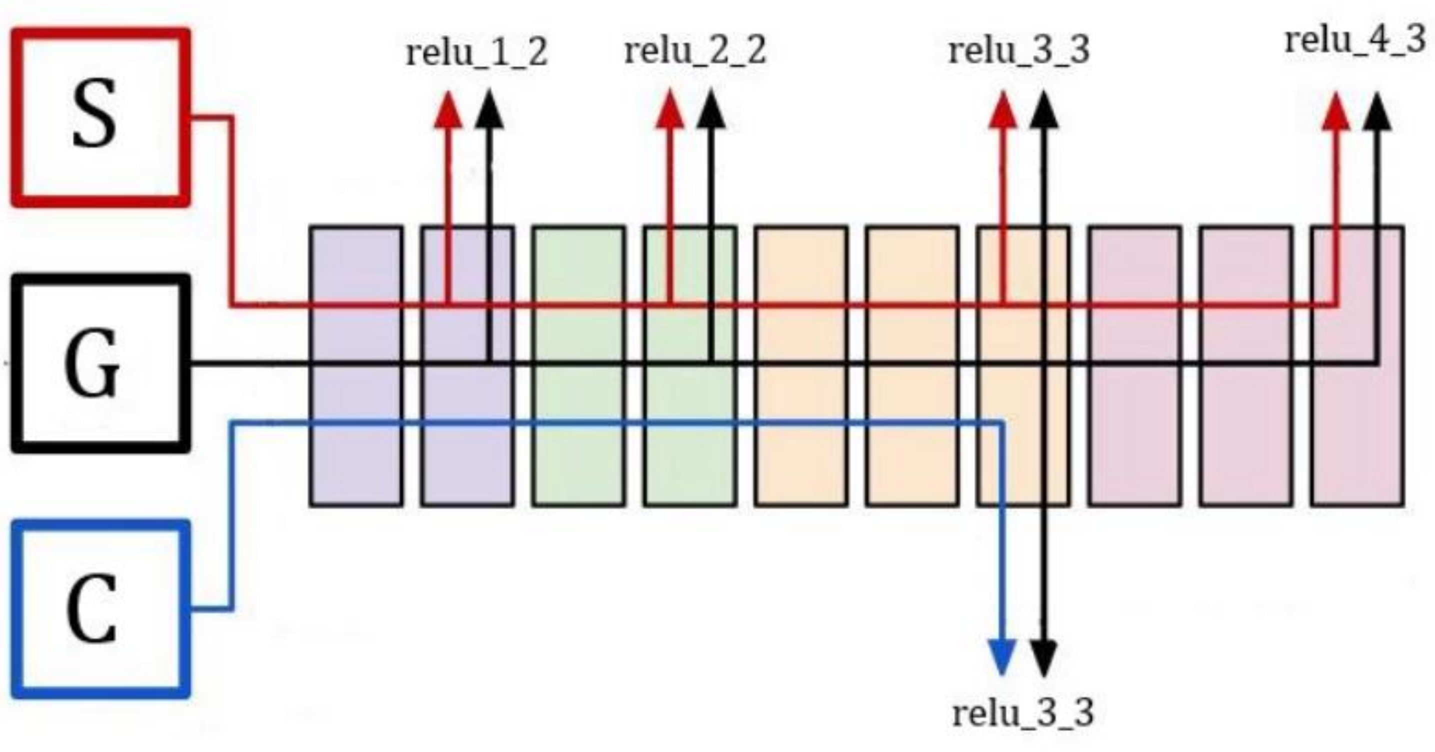


Figure 6 - Different Representations used for Content, Style and Generated Image

Neural style transfer is a method to blend two images and create a new image from a content image by copying the style of another image, called style image. This newly created image is often referred to as the stylized image.
TensorFlow Hub is a collection of trained machines learning models that you can use with ease. Using Magenta, you can create songs, paintings, sounds, and more.
After observing that the original work for NST proposes a slow optimization for style transfer, the Magenta team developed a fast artistic style transfer method, which can work in real-time. Even though the customizability of the model is limited, it is satisfactory enough to perform a non-photorealistic rendering work with NST. Arbitrary Image Stylization under TensorFlow Hub is a module that can perform fast artistic style transfer that may work on arbitrary painting styles.

1. Get the Image Path

   We will start by selecting two image files. I will directly load these image files from URLs. You are free to choose any photo you want. Just change the filename and URL in the code.
   I would like to transfer the style of van Gogh. So, I chose one of his famous paintings Bedroom in Arles, which he painted in 1889. You can even use your own drawings.

The below code sets the path to get the image files.

```
import tensorflow as tf

content_path = tf.keras.utils.get_file('photo-1501820488136-72669149e0d4',
                                       'https://images.unsplash.com/photo-1501820488136-72669149e0d4')

style_path = tf.keras.utils.get_file('Vincent_van_gogh%2C_la_camera_da_letto%2C_1889%2C_02.jpg',
                                     'https://upload.wikimedia.org/wikipedia/commons/8/8c/Vincent_van_gogh%2C_la_camera_da_letto%2C_1889%2C_02.jpg')
```

2. Custom Function for Image Scaling

One thing I noticed that, even though we are very limited with model customization, by rescaling the images, we can change the style transferred to the photo. In fact, I found out that the smaller the images, the better the model transfers the style. Just play with the *max_dim* parameter if you would like to experiment. Just note that a larger *max_dim* means, it will take slightly longer to generate the stylized image.

```
def img_scaler(image, max_dim = 512):

  # Casts a tensor to a new type.
  original_shape = tf.cast(tf.shape(image)[:-1], tf.float32)

  # Creates a scale constant for the image
  scale_ratio = max_dim / max(original_shape)

  # Casts a tensor to a new type.
  new_shape = tf.cast(original_shape * scale_ratio, tf.int32)

  # Resizes the image based on the scaling constant generated above
  return tf.image.resize(image, new_shape)
```

3. Custom Function for Pre-processing the Image

Now that we set our image paths to load and img_scaler function to scale the loaded image, we can actually load our image files with the custom function below.

```
def load_img(path_to_img):

  # Reads and outputs the entire contents of the input filename.
  img = tf.io.read_file(path_to_img)

  # Detect whether an image is a BMP, GIF, JPEG, or PNG, and
  # performs the appropriate operation to convert the input
  # bytes string into a Tensor of type dtype
  img = tf.image.decode_image(img, channels=3)

  # Convert image to dtype, scaling (MinMax Normalization) its values if needed.
  img = tf.image.convert_image_dtype(img, tf.float32)

  # Scale the image using the custom function we created
  img = img_scaler(img)

  # Adds a fourth dimension to the Tensor because
  # the model requires a 4-dimensional Tensor
  return img[tf.newaxis, :]
```

4. Load the Content and Style Images

For content image and style image, we need to call the *load_img* function once and the result will be a 4-dimensional Tensor, which is what will be required by our model.

```
content_image = load_img(content_path)
style_image = load_img(style_path)
```

We can plot them with matplotlib.

```
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 12))
plt.subplot(1, 2, 1)
plt.imshow(content_image[0])
plt.title('Content Image')
plt.subplot(1, 2, 2)
plt.imshow(style_image[0])
plt.title('Style Image')

plt.show()
```

5. Load the Arbitrary Image Stylization Network

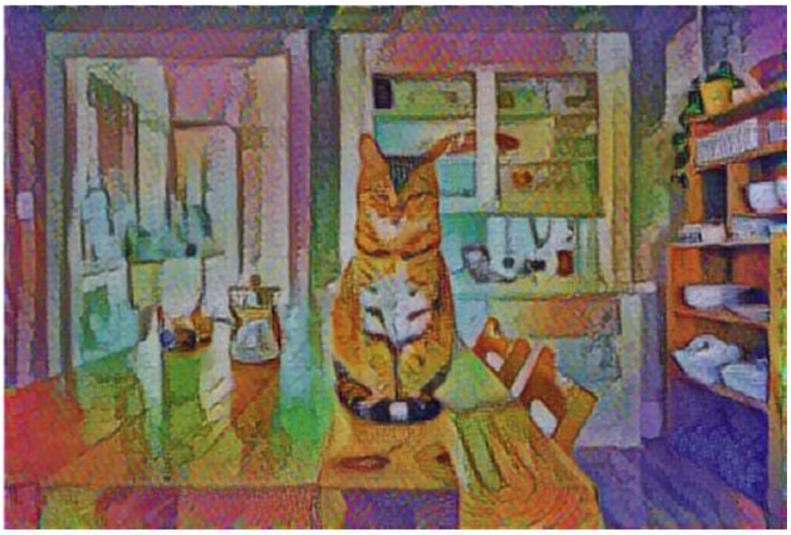
We need to import the tensorflow_hub library so that we can use the modules containing the pre-trained models. After importing tensorflow_hub, we can use the load function to load the Arbitrary Image Stylization module. We can pass the content and style images as arguments in tf.constant object format. The module returns our stylized image in an array format.

```
import tensorflow_hub as hub

# Load Magenta's Arbitrary Image Stylization network from TensorFlow Hub
hub_module = hub.load('https://tfhub.dev/google/magenta/arbitrary-image-stylization-v1-256/1')

# Pass content and style images as arguments in TensorFlow Constant object format
stylized_image = hub_module(tf.constant(content_image), tf.constant(style_image))[0]
```

All we have to do is to use this array and plot it with matplotlib.

```
# Set the size of the plot figure
plt.figure(figsize=(12, 12))

# Plot the stylized image
plt.imshow(stylized_image[0])

# Add title to the plot
plt.title('Stylized Image')

# Hide axes
plt.axis('off')

# Show the plot
plt.show()
```

And here is our stylized image.



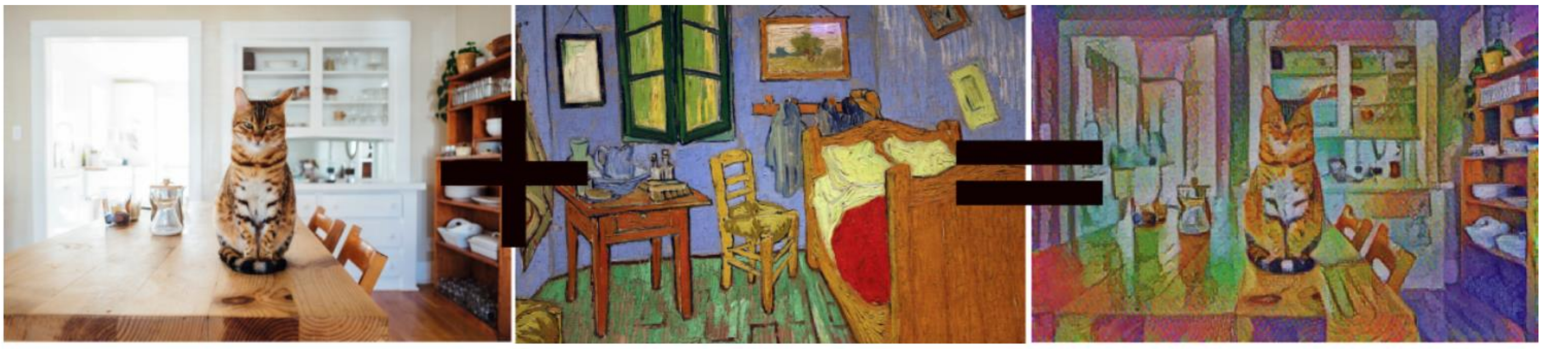


Figure 9. A Neural Style Transfer Example made with Arbitrary Image Stylization Network