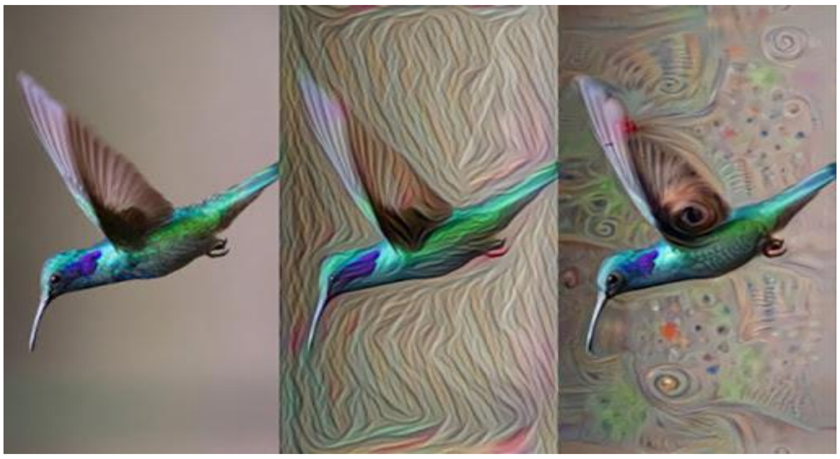
2017122023

JONGYEON PARK

## Introduction

“Once upon a time, I, Zhuangzi, dreamt I was a butterfly, fluttering hither and thither, to all intents and purposes a butterfly. I was conscious only of my happiness as a butterfly, unaware that I was Zhuangzi. Soon I awakened, and there I was, veritably myself again. Now I do not know whether I was then a man dreaming I was a butterfly, or whether I am now a butterfly, dreaming I am a man.”

This is the famous Butterfly Dream parable, written by a renowned Chinese philosopher, Zhuangzi. With the advent of machine learning, the distinction between real and digital is fading just as Zhuangzi confused himself of being a butterfly. One example of this confusion is Deep Dream, an artificial intelligence service that generates artistically transformed copies of an image.



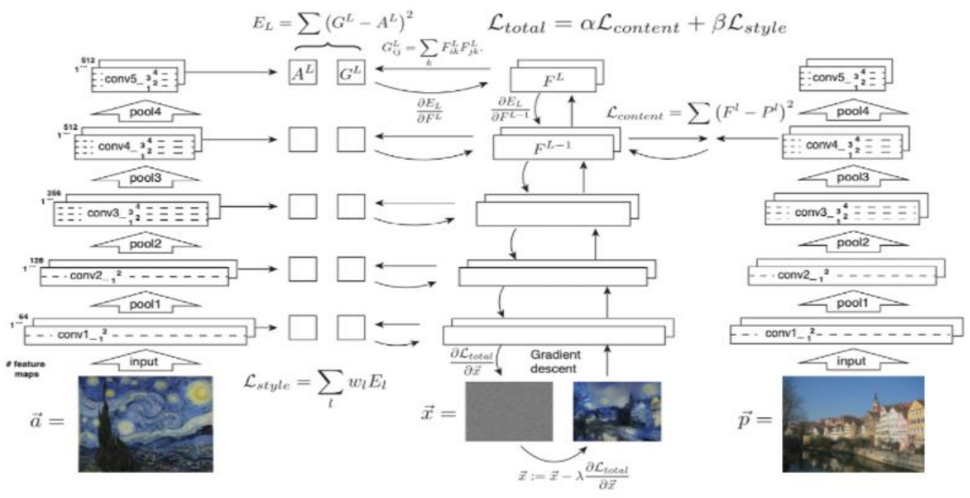
## Style Transfer Algorithm

In this final project, I have created my own model to generate new pictures using neural network. According "Image Style Transfer Using Convolutional Neural Networks" (Gatys et al., CVPR 2015), we can implement the style transfer technique by dissecting style and content, and reconfiguring them into a new image.

The Style Transfer Algorithm follows these procedures:

1. Create a Pre-trained Network.
2. Learning from random white noise image.
3. Calculate the total loss (Style Loss and Content Loss).
4. Use Gradient Descent to implement Backpropagation.
5. Implement Gradient onto input image with the chain rule.
6. Repeat the process until local optima is obtained.

By looking at CVPR 2015(Gatys et al), we can see as layer level of CNN increases, the detail of image fades while generic feature, which represents the characteristic of the image, remains. So, using this methodology, we can obtain content loss with generic image and obtain style loss from the other image, and linearly combining them into total loss. In other words, we minimize the content loss of the noise image and maximize the content of original image (minimize the style loss). The complicate process is depicted in the following figure.



$E_L = \sum (G^L - A^L)^2$
$\mathcal{L}_{total} = \alpha\mathcal{L}_{content} + \beta\mathcal{L}_{style}$
$G^L_{ij} = \sum_k F^L_{ik}F^L_{jk}$
$A^L$
$G^L$
$F^L$
$F^{L-1}$
$\frac{\partial E_L}{\partial F^L}$
$\frac{\partial E_L}{\partial F^{L-1}}$
$\mathcal{L}_{content} = \sum (F^l - P^l)^2$
conv5_1
pool4
conv4_1
pool3
conv3_1
pool2
conv2_1
pool1
conv1_1
input
512
256
128
64
# feature maps
$\mathcal{L}_{style} = \sum_l w_l E_l$
$\frac{\partial \mathcal{L}_{total}}{\partial \vec{x}}$
Gradient descent
$\vec{a} =$
$\vec{x} =$
$\vec{p} =$
$\vec{x} := \vec{x} - \lambda \frac{\partial \mathcal{L}_{total}}{\partial \vec{x}}$

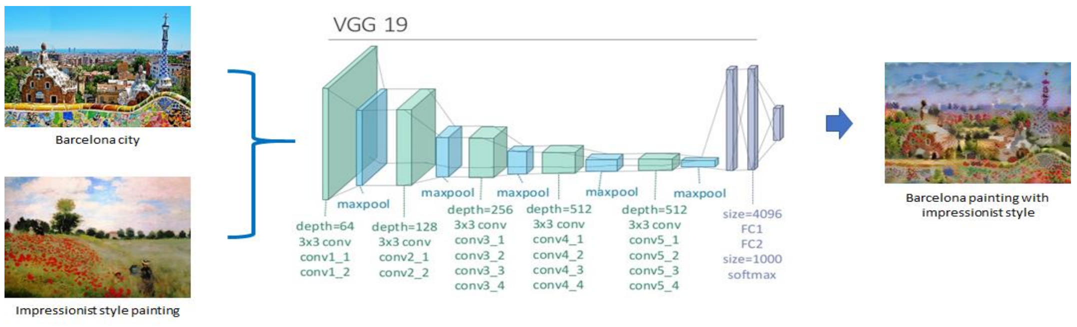
1. Implemented random noise image.
2. Content Image only transfers high level layer loss.
3. Style Image transfers each layers' loss.
4. Total Loss is the weighted sum of two losses.

For this project, I used VGG-19 model. VGG-19, developed by University of Oxford, is constructed with 19 layers.

```
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
```

```
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]
```

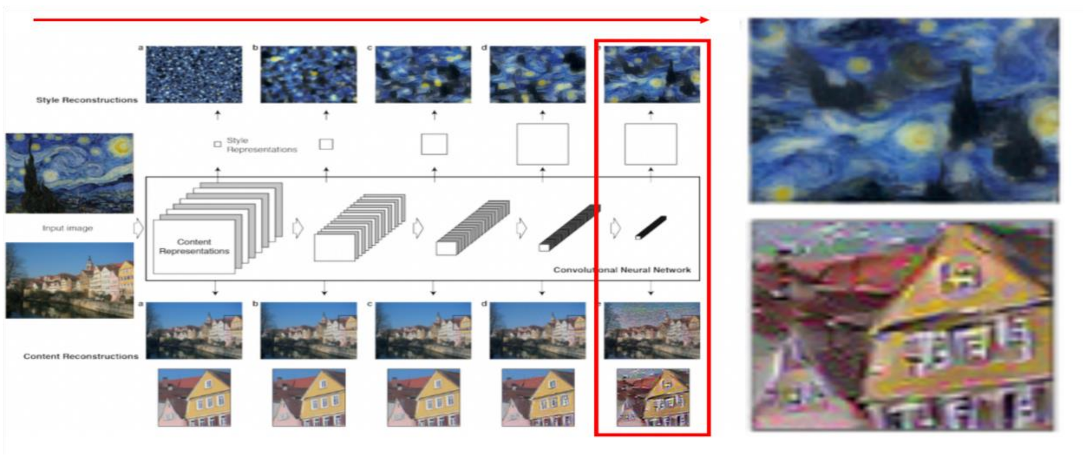
```
#The actual layers of VGG19 that I use for this project.
vgg = tf.keras.applications.VGG19(include_top=False, weights='imagenet')

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



VGG-19 is formed with one input layer followed by two conv3-64 layers. Then, one maxpool (Gatys used average pooling for the project) layer with two conv3-128 layers. The same structure repeats until it has four conv3-512 layers. The rest if just fc layers and a softmax layer.

The main difference between VGG structure and other structures is that it only uses 3*3 filter with stride value of 1, which enables the model to use a smaller number of parameters. For instance, convoluting three times with 3*3 filter only requires 27 (3*3*3) weights but 7*7 needs 49 weights. With lower number, the speed of calculation is faster.

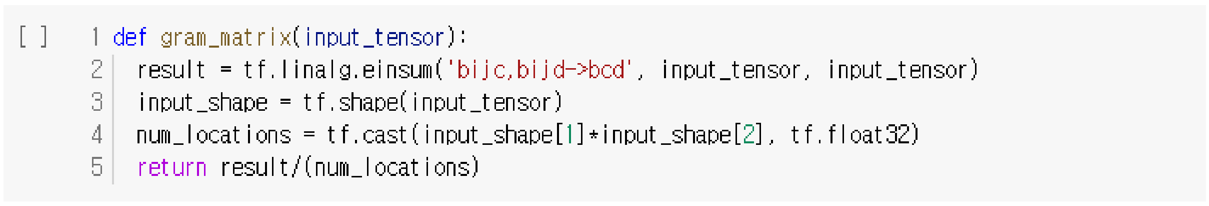
With CNN, I can find the features to implement style transfer. For content image, as the layer deepens, the detailed pixel information fades while high-level image, such as structure of the image remains. Style image, on the other hand, as the layer deepens, the style factor becomes more like the original image.



This algorithm works because of the Gram Matrix. This matrix contains the correlation between the feature map's channel in each layer. The formula as follows:

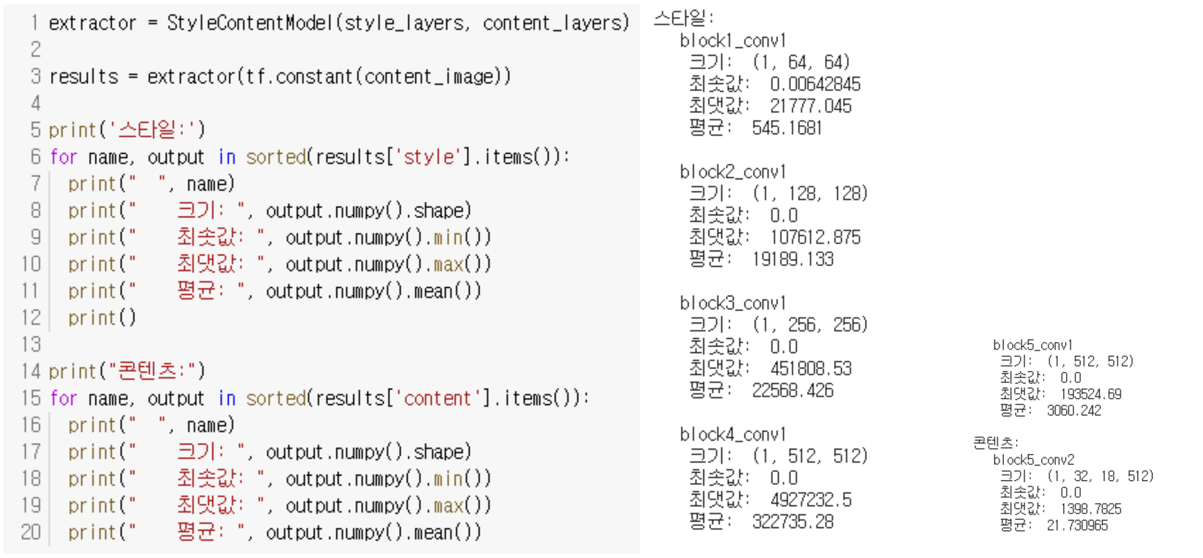
$$G^l_{cd} = \frac{\sum_{ij} F^l_{ijc}(x) F^l_{ijd}(x)}{IJ}$$

Or we can use tf.linalg.einsum function



```
[ ]  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)
```

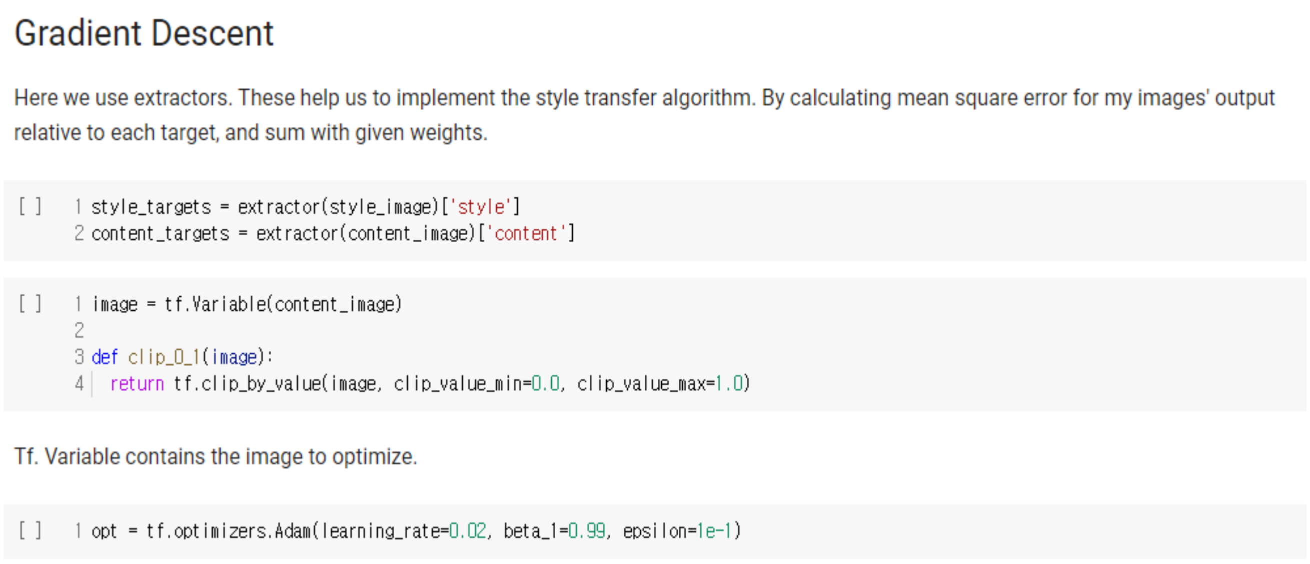
Because style and content image should be uncorrelated, the Gram Matrix should be minimized, and this results in minimizing the loss function of style loss function. As for total loss of style and content, we implement gradient descent to find a local minimum (note that it might not a global minimum because I have two minimums to consider). The minimum is obtained with the concept of mean square error and I have used Adam for compiler (for simplicity).



```
extractor = StyleContentModel(style_layers, content_layers)

results = extractor(tf.constant(content_image))

print('스타일:')
for name, output in sorted(results['style'].items()):
  print("  ", name)
  print("    크기: ", output.numpy().shape)
  print("    최솟값: ", output.numpy().min())
  print("    최댓값: ", output.numpy().max())
  print("    평균: ", output.numpy().mean())
  print()

print("콘텐츠:")
for name, output in sorted(results['content'].items()):
  print("  ", name)
  print("    크기: ", output.numpy().shape)
  print("    최솟값: ", output.numpy().min())
  print("    최댓값: ", output.numpy().max())
  print("    평균: ", output.numpy().mean())
```

```
스타일:
   block1_conv1
    크기:  (1, 64, 64)
    최솟값:  0.00642845
    최댓값:  21777.045
    평균:  545.1681

   block2_conv1
    크기:  (1, 128, 128)
    최솟값:  0.0
    최댓값:  107612.875
    평균:  19189.133

   block3_conv1
    크기:  (1, 256, 256)
    최솟값:  0.0
    최댓값:  451808.53
    평균:  22568.426

   block4_conv1
    크기:  (1, 512, 512)
    최솟값:  0.0
    최댓값:  4927232.5
    평균:  322735.28

   block5_conv1
    크기:  (1, 512, 512)
    최솟값:  0.0
    최댓값:  193524.69
    평균:  3060.242

콘텐츠:
   block5_conv2
    크기:  (1, 32, 18, 512)
    최솟값:  0.0
    최댓값:  1398.7825
    평균:  21.730965
```

The calculated loss values are printed next to the code implementation.



## Gradient Descent

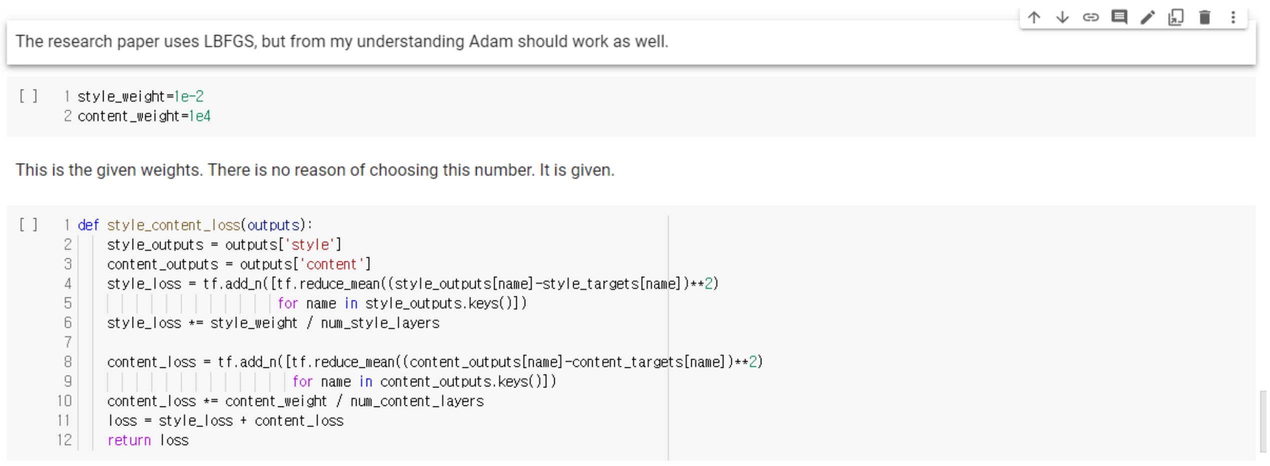
Here we use extractors. These help us to implement the style transfer algorithm. By calculating mean square error for my images' output relative to each target, and sum with given weights.

```
[ ]  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)
```

Tf. Variable contains the image to optimize.

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



The research paper uses LBFGS, but from my understanding Adam should work as well.

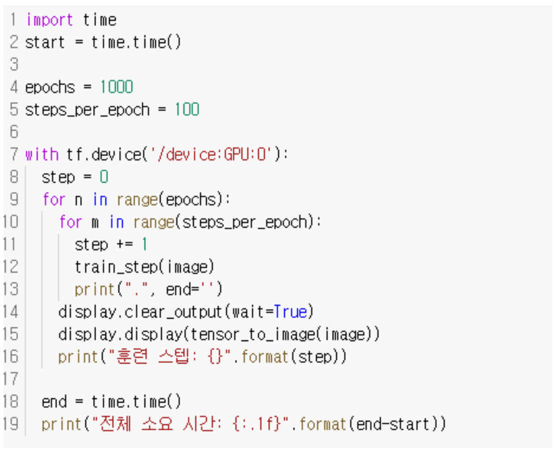
```
[ ]  style_weight=1e-2
     content_weight=1e4
```

This is the given weights. There is no reason of choosing this number. It is given.

```
[ ]  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
```

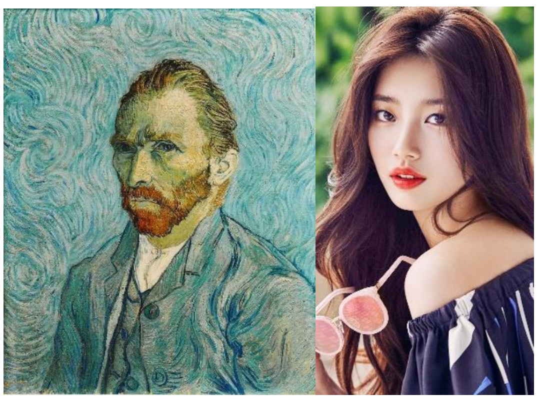
This is the implementation of gradient descent.

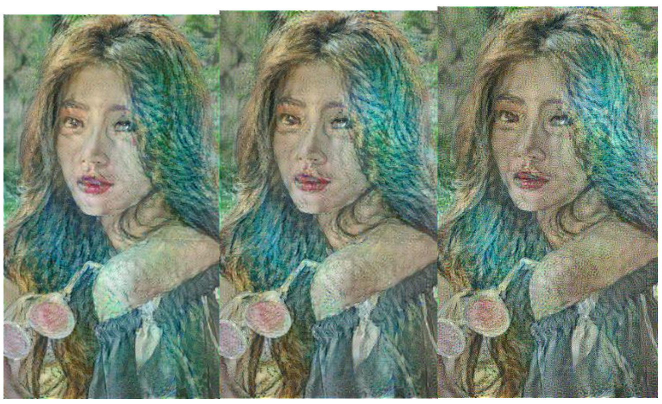
After that, I have run the code to implement style transfer. As the number of epochs increases, the two images mix!



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

epochs = 1000
steps_per_epoch = 100

with tf.device('/device:GPU:0'):
  step = 0
  for n in range(epochs):
    for m in range(steps_per_epoch):
      step += 1
      train_step(image)
      print(".", end='')
    display.clear_output(wait=True)
    display.display(tensor_to_image(image))
    print("훈련 스텝: {}".format(step))

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

Nothing Special, just running the code. I have tried different number of epochs. As epochs increase, the effect transferred becomes better.





50 epochs -> 100 epochs ->1000 epochs

To sum up, I have used the feature of CNN to mix two images. As the layer of CNN increases, I learned that different features can be extracted. By using loss function and grim matrix, I can implement this transfer. Finally, the number of epochs should be sufficiently large for this algorithm to work.

Reference

Butterfly: https://www.learnreligions.com/butterflies-great-sages-and-valid-cognition-3182587

AI article: https://dbr.donga.com/article/view/1303/article_no/9290

DeepDream: https://deepdreamgenerator.com/

Research Paper: "Image Style Transfer Using Convolutional Neural Networks" (Gatys et al., CVPR 2015).

Code: https://www.tensorflow.org/tutorials/generative/style_transfer?hl=ko

VGG-19: https://keras.io/api/applications/vgg/

VGG-19 Paper: "Very deep convolutional networks for large-scale image recognition" (Karen Simonyan, 2015 ICLR)

My full code: https://colab.research.google.com/drive/1O7Q1Vq4VLOIQuy1d-LsZEwN6pEXtgFd6?usp=sharing