Copyright Notice

Style Transfer

Style Image

Wassily Kandinsky (wikipedia)
https://upload.wikimedia.org/wikipedia/commons/b/b4/Vassily_Kandinsky%2C_1913_-_Composition_7.jpg

Content Image

https://cdn.pixabay.com/photo/2017/02/28/23/00/swan-2107052_1280.jpg

Stylized Image

Approaches to Style Transfer

1. Supervised Learning.
2. Neural Style Transfer
3. Fast Neural Style Transfer

Approaches to Style Transfer

1. Supervised Learning.
2. Neural Style Transfer
3. Fast Neural Style Transfer

Approaches to Style Transfer

1. Supervised Learning.
2. Neural Style Transfer
3. Fast Neural Style Transfer

1. Supervised Learning

- Pairs: original & stylized image.
  - Need lots of pairs!

1. Supervised Learning

- Pairs: original & stylized image.
  - Need lots of pairs!

2. Neural Style Transfer

- pre-trained model
- Inputs: a single pair of images
  - Extract style from image 1
  - Extract content from image 2
- Generate image to match style and content.
  - In a loop: minimize loss

2. Neural Style Transfer

- pre-trained model
- Inputs: a single pair of images
  - Extract style from image 1
  - Extract content from image 2
- Generate image to match style and content.
  - In a loop: minimize loss

2. Neural Style Transfer

- pre-trained model
- Inputs: a single pair of images
  - Extract style from image 1
  - Extract content from image 2
- Generate image to match style and content.
  - In a loop: minimize loss

2. Neural Style Transfer

- pre-trained model
- Inputs: a single pair of images
  - Extract style from image 1
  - Extract content from image 2
- Generate image to match style and content.
  - In a loop: minimize loss

2. Neural Style Transfer

- pre-trained model
- Inputs: a single pair of images
  - Extract style from image 1
  - Extract content from image 2
- Generate image to match style and content.
  - In a loop: minimize loss

2. Neural Style Transfer

- pre-trained model
- Inputs: a single pair of images
  - Extract style from image 1
  - Extract content from image 2
- Generate image to match style and content.
  - In a loop: minimize loss

A Neural Algorithm of Artistic Style

Leon Gatys
Alexander Ecker
Matthias Bethge

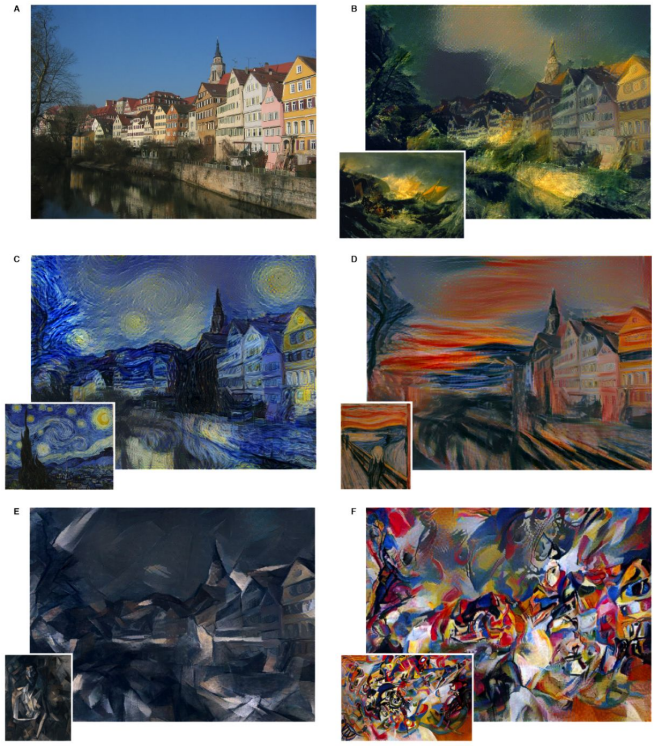
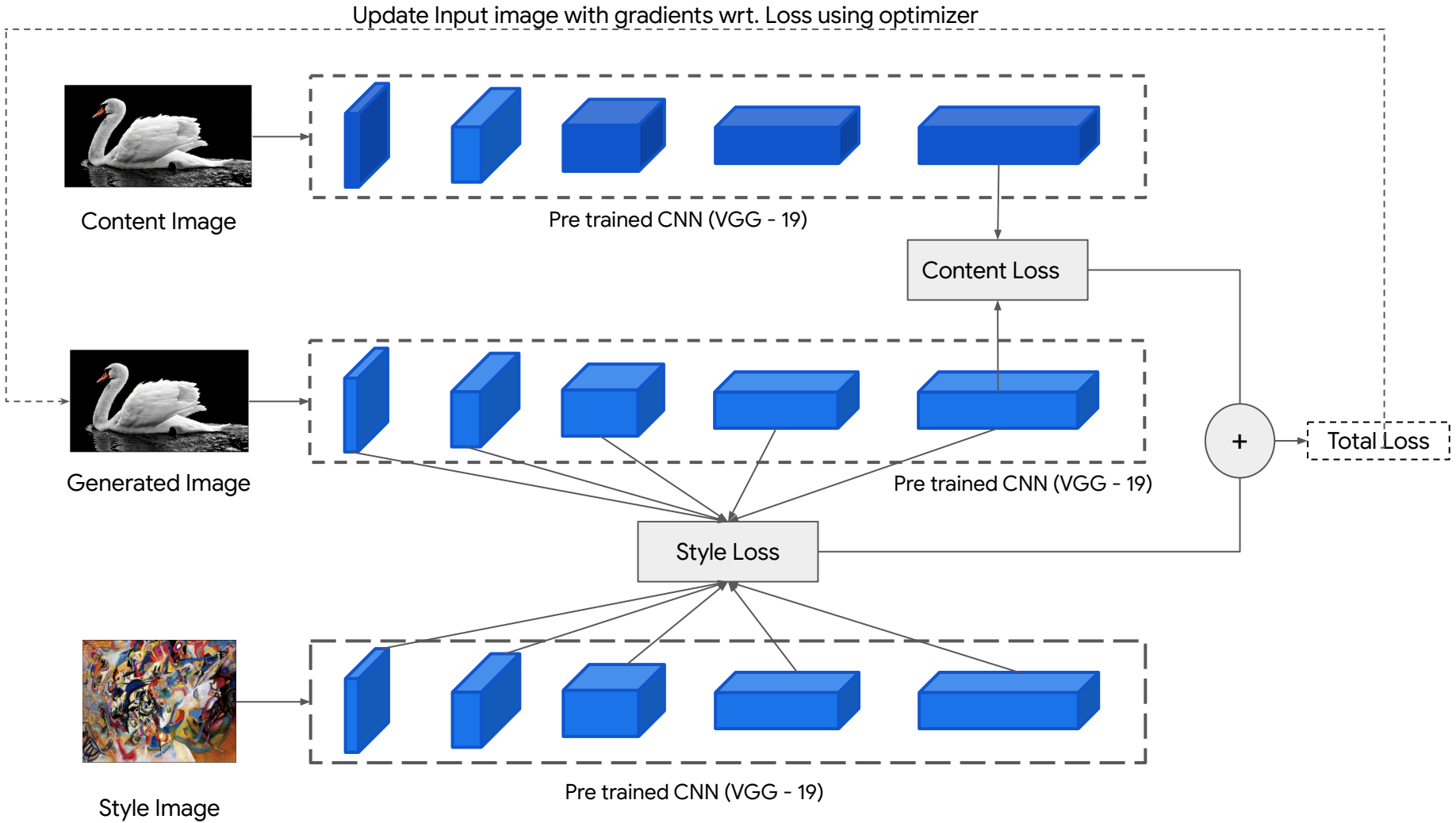
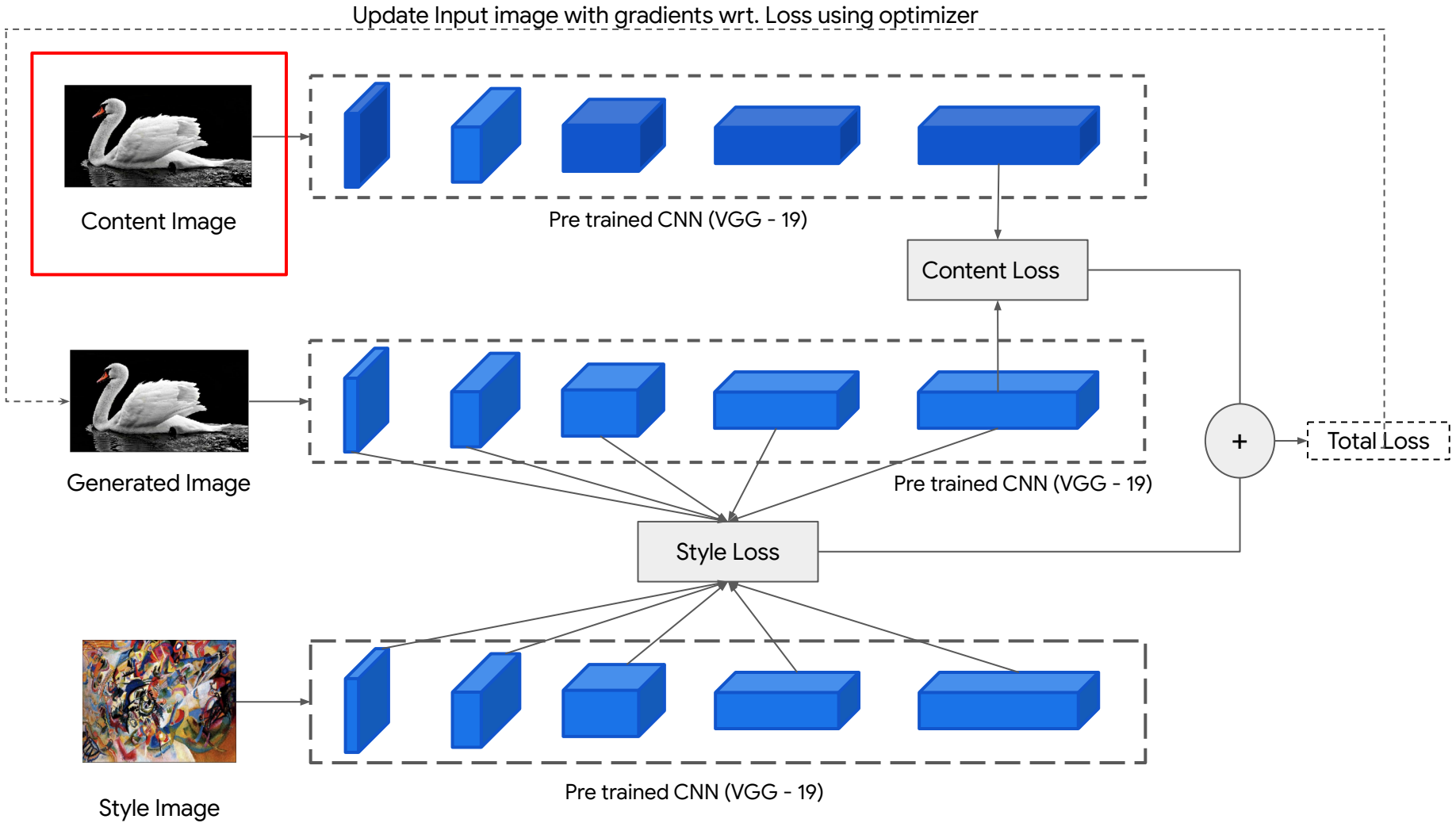
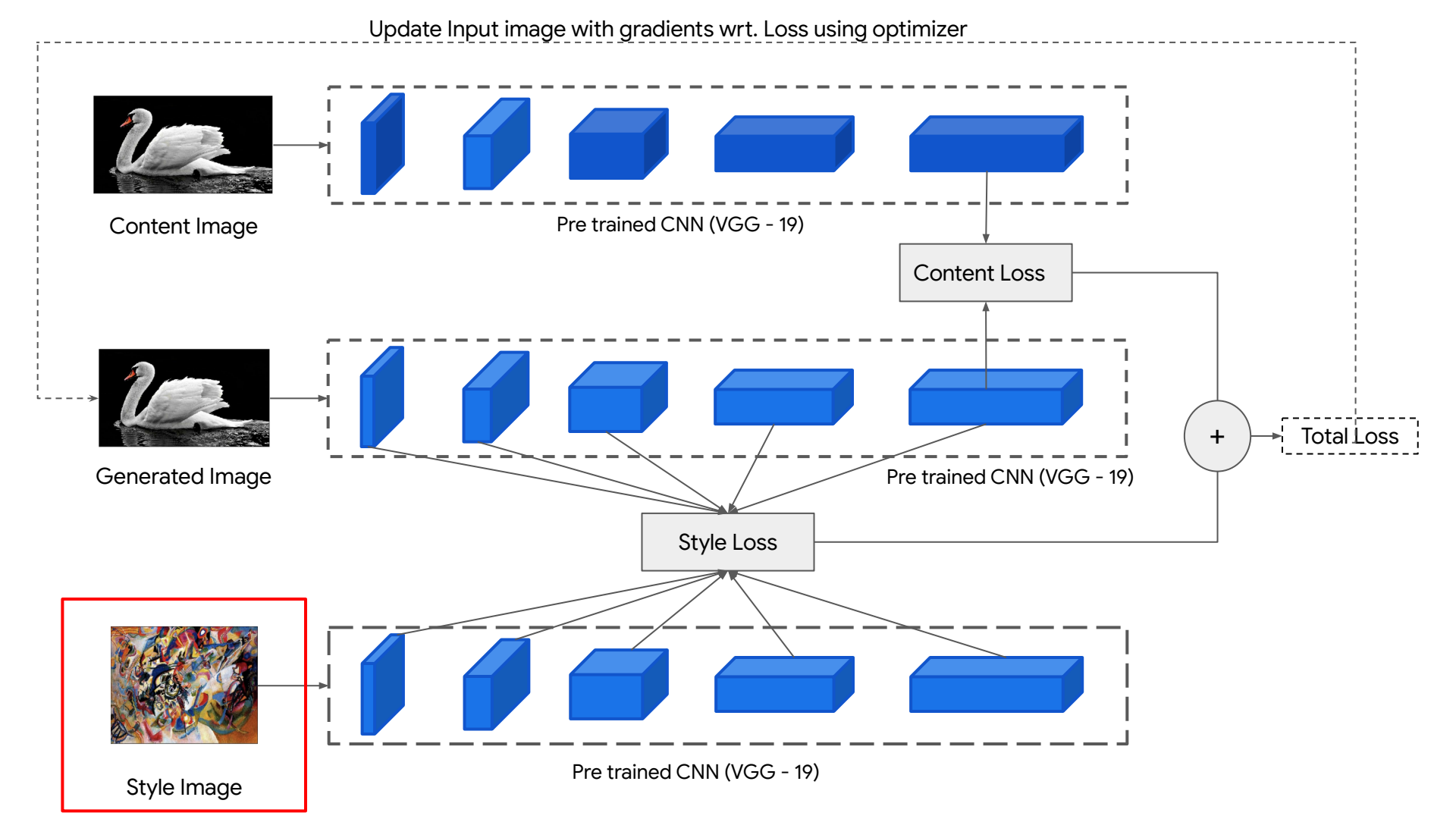


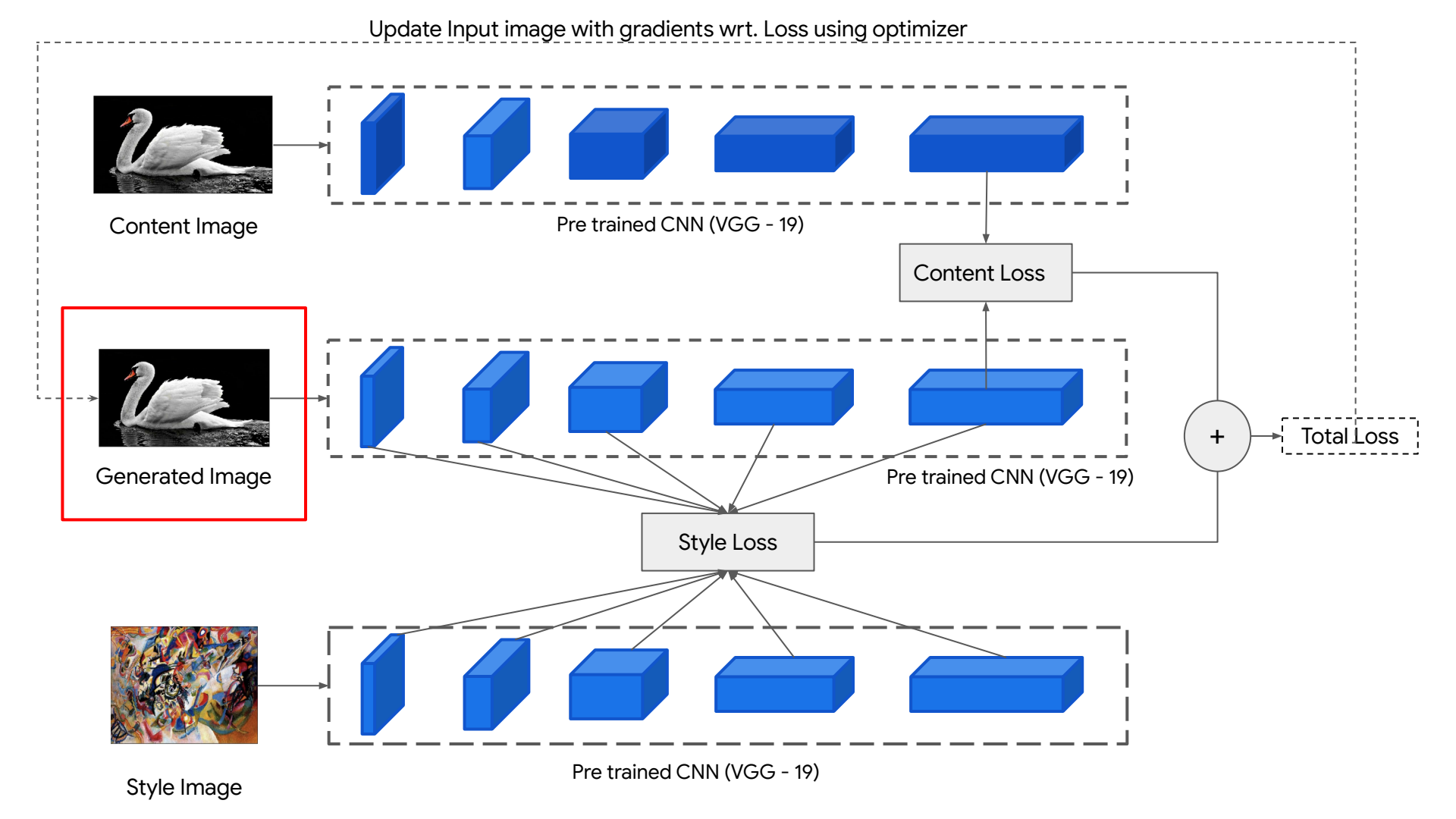
Figure 2: Images that combine the content of a photograph with the style of several well-known artworks. The images were created by finding an image that simultaneously matches the content representation of the photograph and the style representation of the artwork (see Methods). The

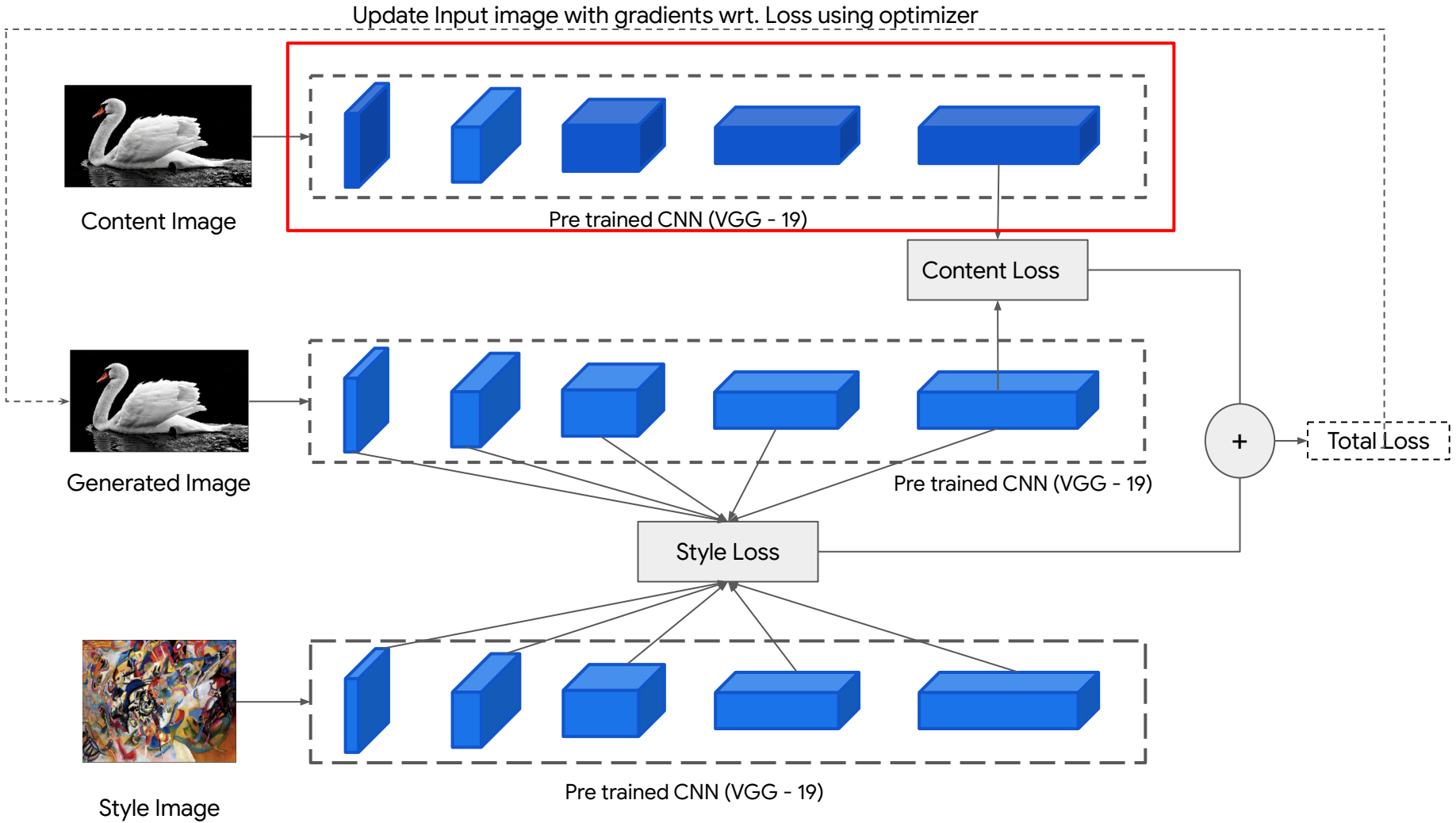
https://arxiv.org/abs/1508.06576

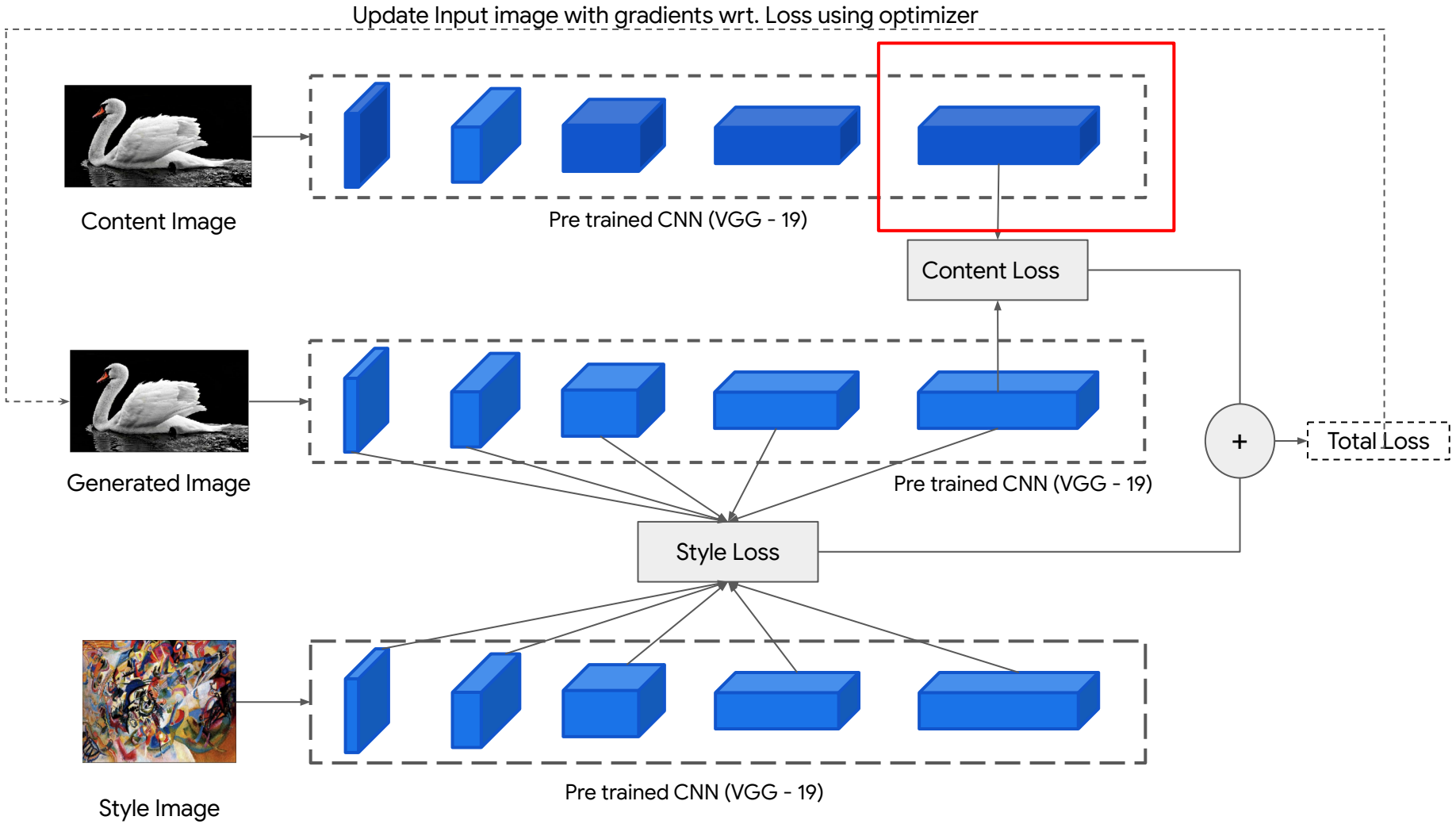

Update Input image with gradients wrt. Loss using optimizer
Content Image
Pre trained CNN (VGG - 19)
Content Loss
Generated Image
Pre trained CNN (VGG - 19)
+
Total Loss
Style Loss
Style Image
Pre trained CNN (VGG - 19)

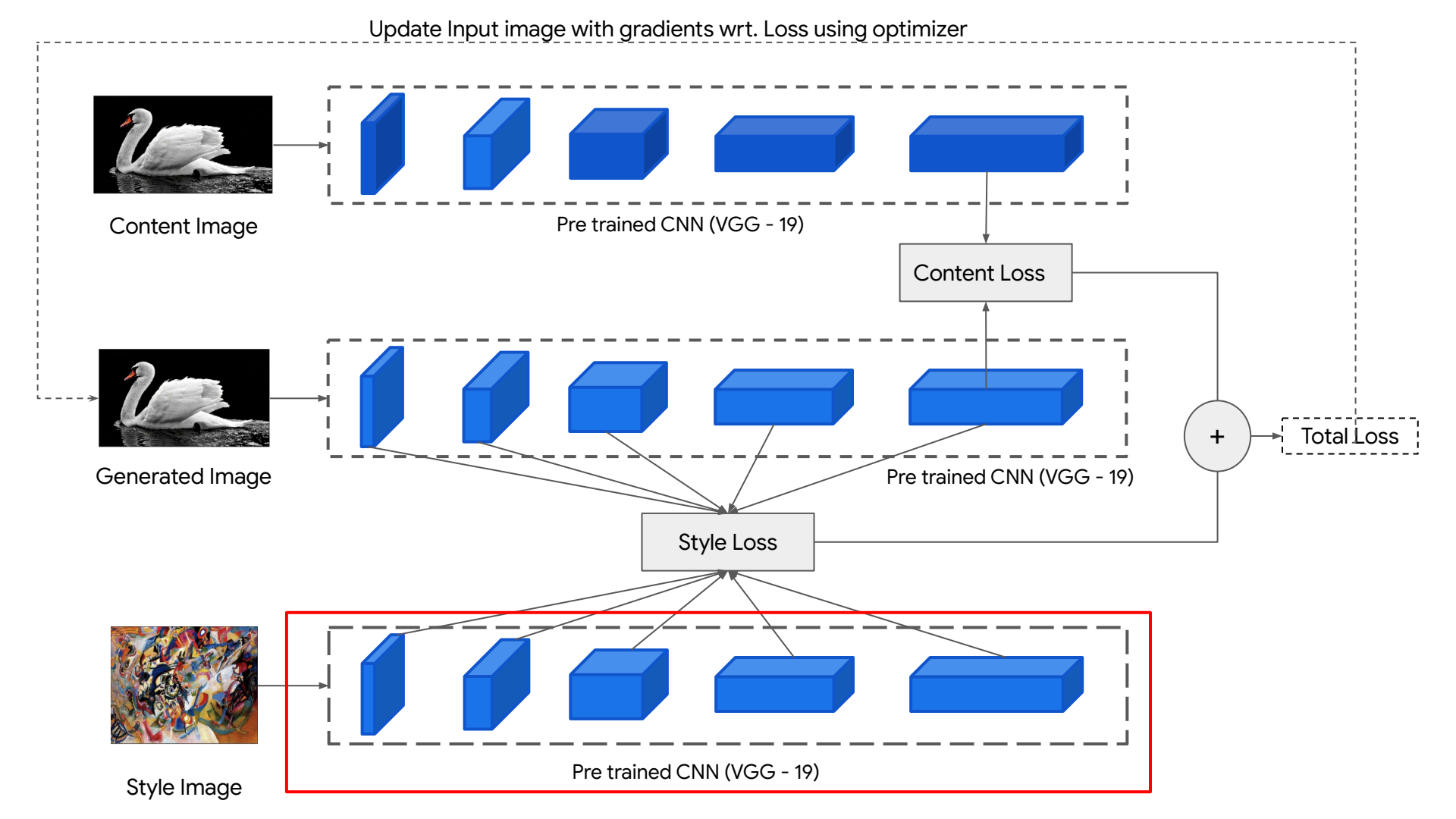

Update Input image with gradients wrt. Loss using optimizer
Content Image
Pre trained CNN (VGG - 19)
Content Loss
Generated Image
Pre trained CNN (VGG - 19)
+
Total Loss
Style Loss
Style Image
Pre trained CNN (VGG - 19)

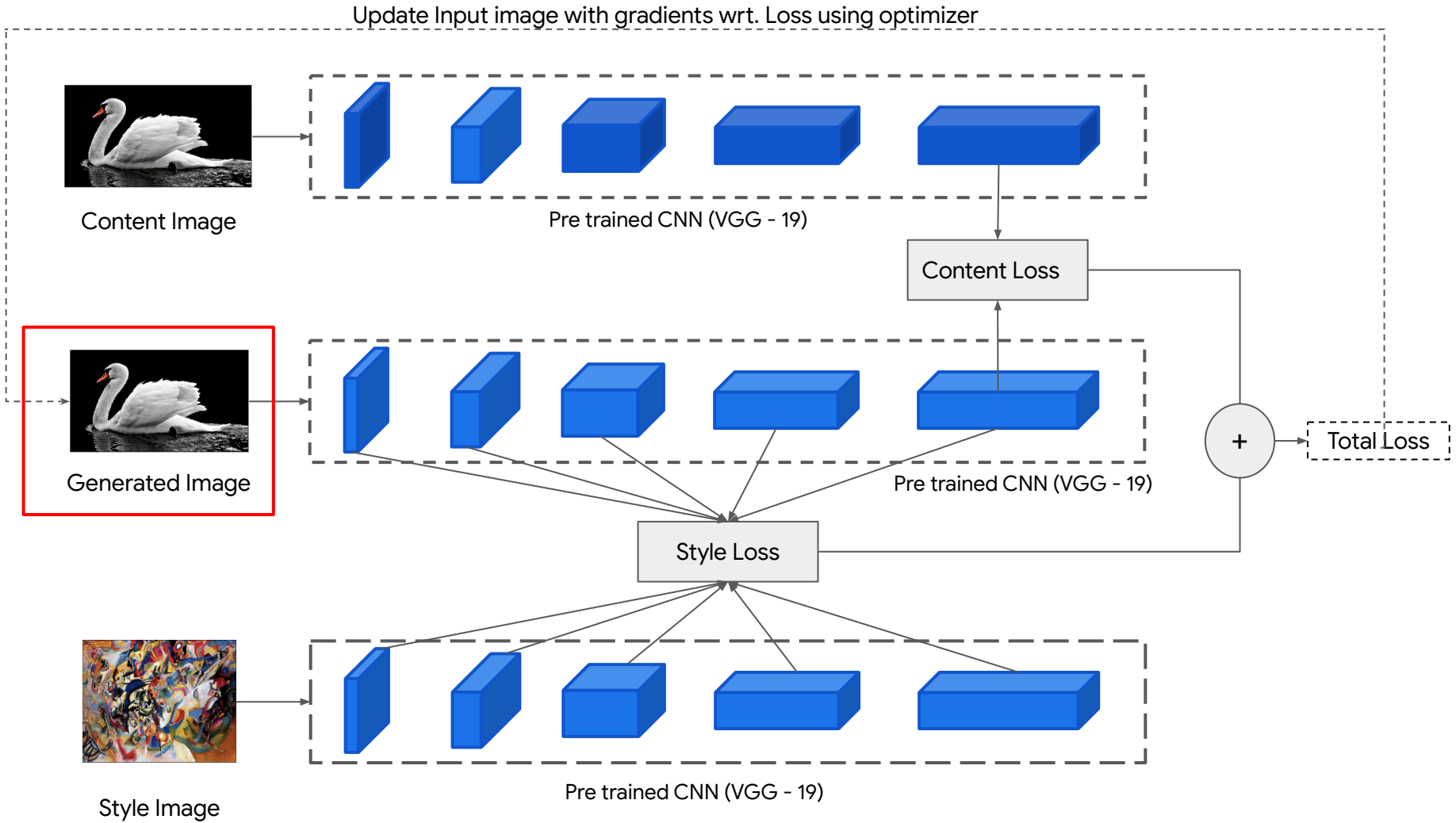

Update Input image with gradients wrt. Loss using optimizer
Content Image
Pre trained CNN (VGG - 19)
Content Loss
Generated Image
Pre trained CNN (VGG - 19)
+
Total Loss
Style Loss
Style Image
Pre trained CNN (VGG - 19)

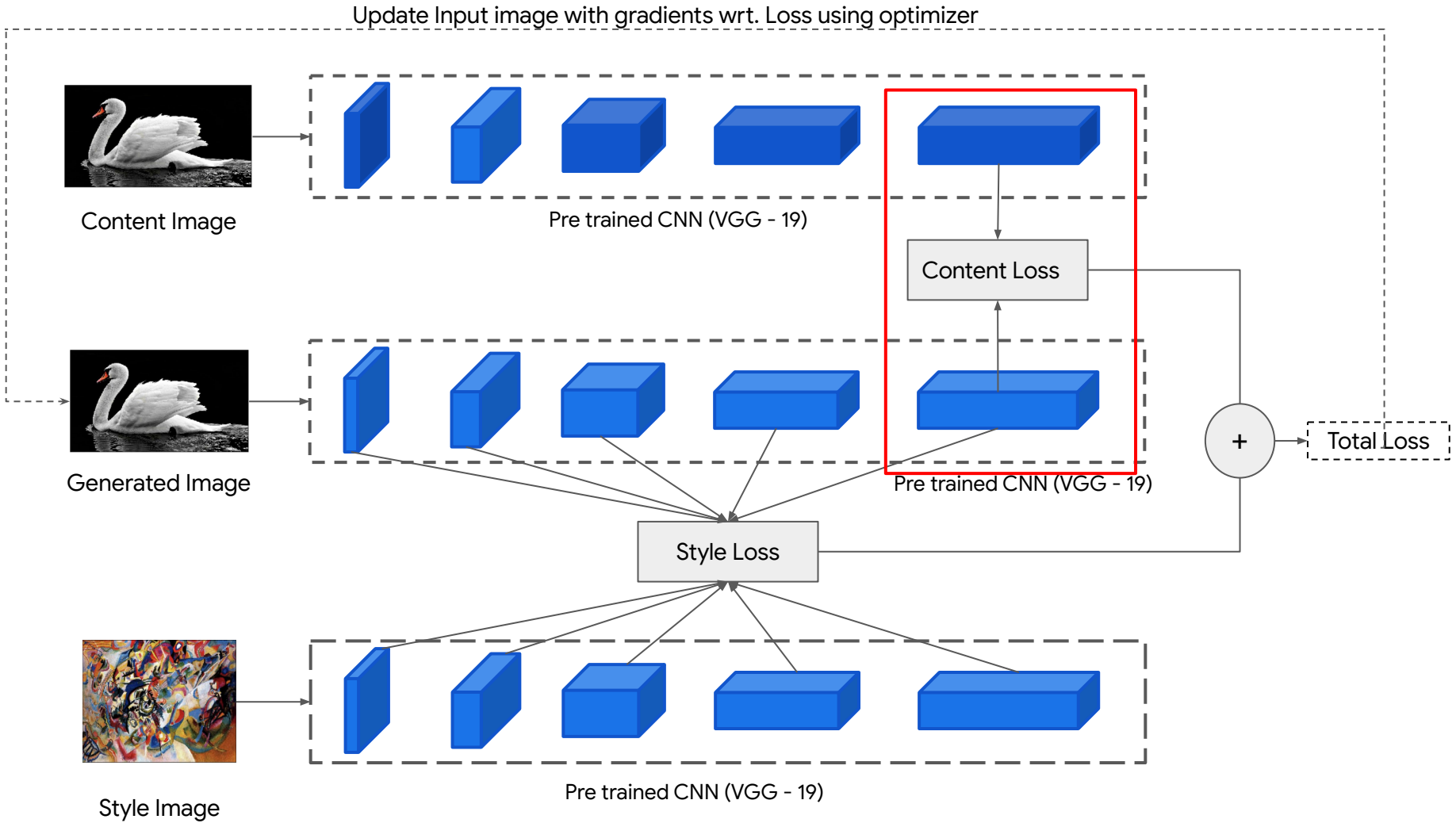

Update Input image with gradients wrt. Loss using optimizer
Content Image
Pre trained CNN (VGG - 19)
Content Loss
Generated Image
Pre trained CNN (VGG - 19)
+
Total Loss
Style Loss
Style Image
Pre trained CNN (VGG - 19)

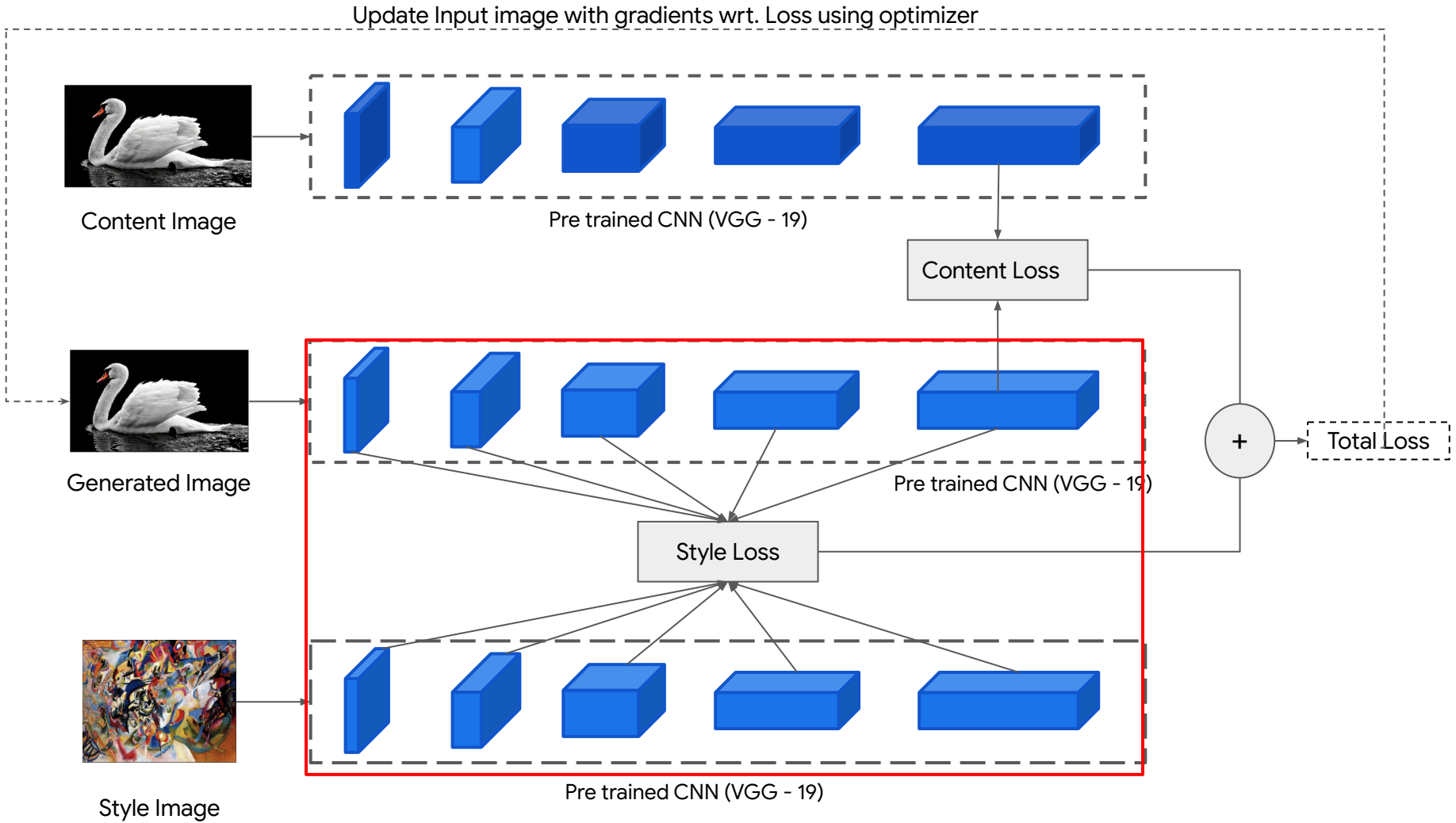

Update Input image with gradients wrt. Loss using optimizer
Content Image
Pre trained CNN (VGG - 19)
Content Loss
Generated Image
Pre trained CNN (VGG - 19)
+
Total Loss
Style Loss
Style Image
Pre trained CNN (VGG - 19)

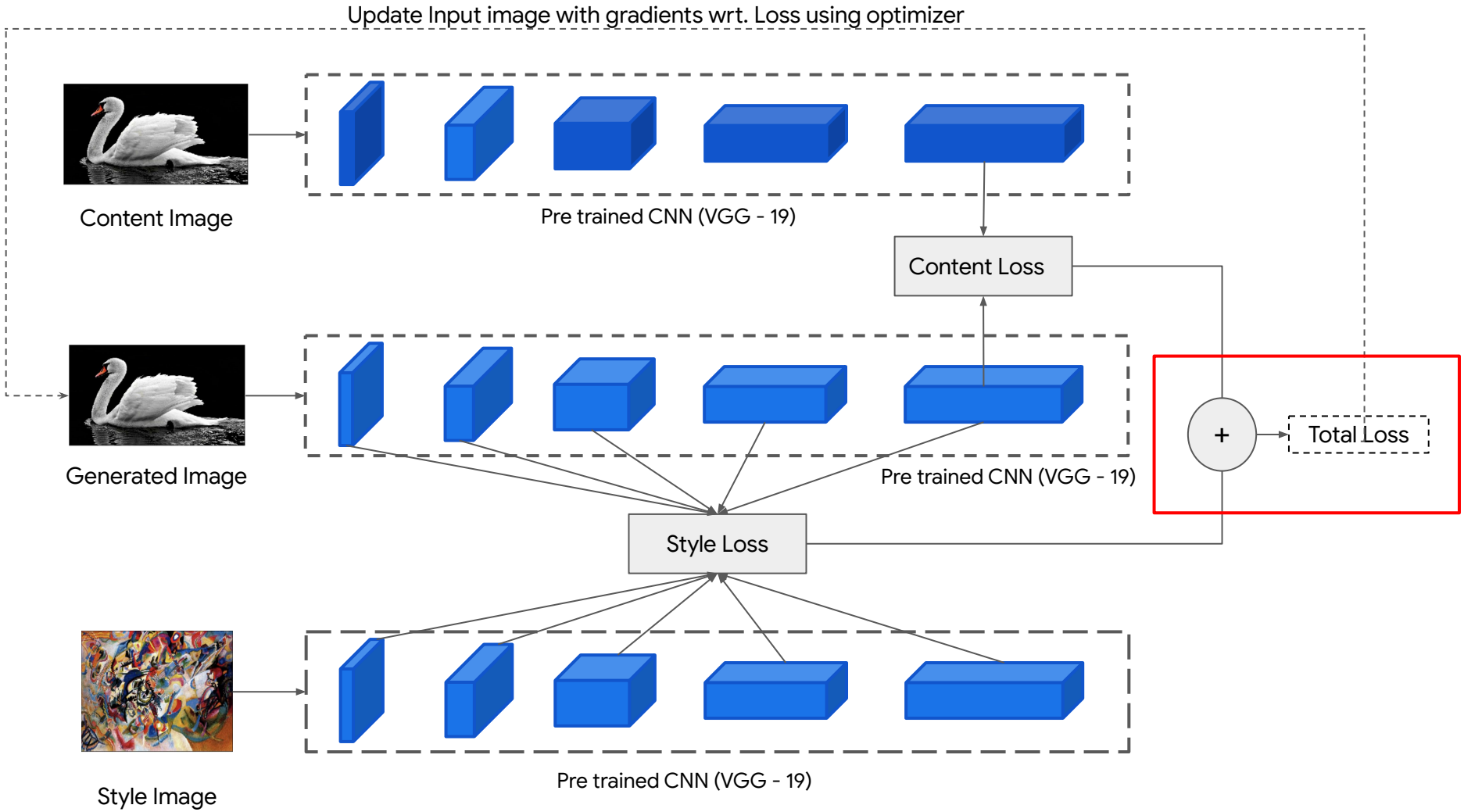

Update Input image with gradients wrt. Loss using optimizer
Content Image
Pre trained CNN (VGG - 19)
Content Loss
Generated Image
Pre trained CNN (VGG - 19)
+
Total Loss
Style Loss
Style Image
Pre trained CNN (VGG - 19)

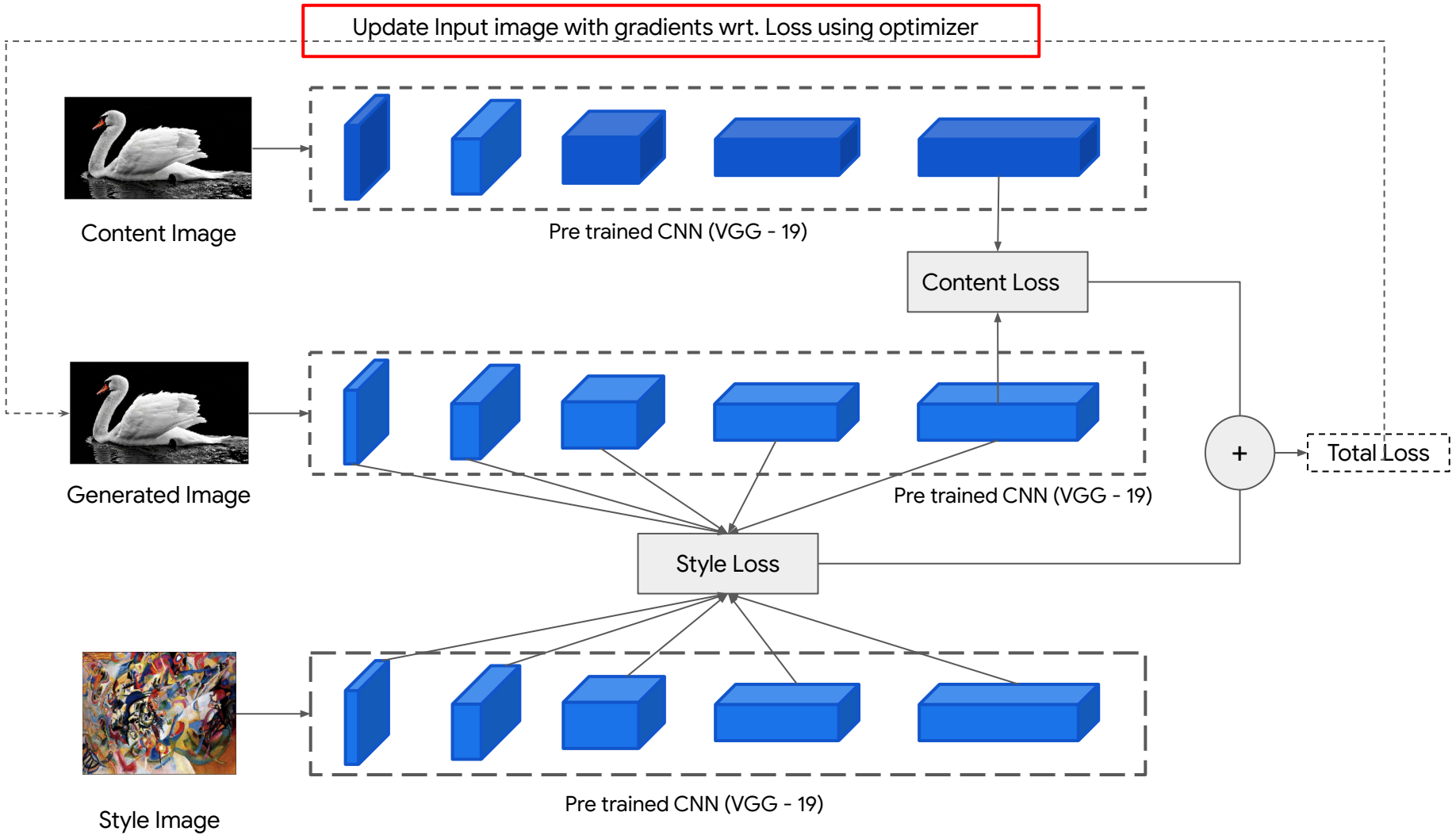

Update Input image with gradients wrt. Loss using optimizer
Content Image
Pre trained CNN (VGG - 19)
Content Loss
Generated Image
Pre trained CNN (VGG - 19)
+
Total Loss
Style Loss
Style Image
Pre trained CNN (VGG - 19)

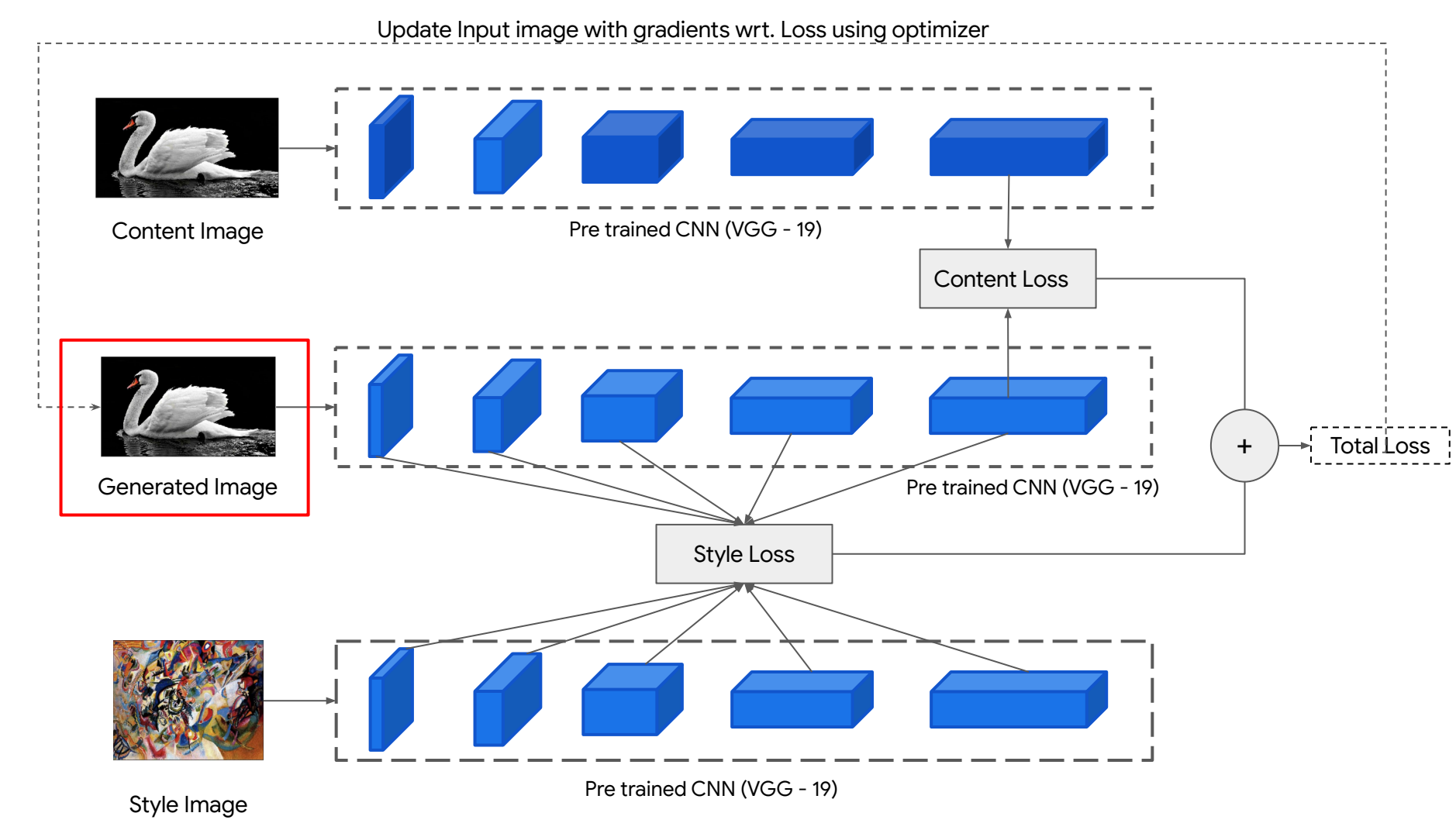

Update Input image with gradients wrt. Loss using optimizer
Content Image
Pre trained CNN (VGG - 19)
Content Loss
Generated Image
Pre trained CNN (VGG - 19)
+
Total Loss
Style Loss
Style Image
Pre trained CNN (VGG - 19)


Update Input image with gradients wrt. Loss using optimizer
Content Image
Pre trained CNN (VGG - 19)
Content Loss
Generated Image
Pre trained CNN (VGG - 19)
+
Total Loss
Style Loss
Style Image
Pre trained CNN (VGG - 19)


Update Input image with gradients wrt. Loss using optimizer
Content Image
Pre trained CNN (VGG - 19)
Content Loss
Generated Image
Pre trained CNN (VGG - 19)
+
Total Loss
Style Loss
Style Image
Pre trained CNN (VGG - 19)


Update Input image with gradients wrt. Loss using optimizer
Content Image
Pre trained CNN (VGG - 19)
Content Loss
Generated Image
Pre trained CNN (VGG - 19)
+
Total Loss
Style Loss
Style Image
Pre trained CNN (VGG - 19)


Update Input image with gradients wrt. Loss using optimizer
Content Image
Pre trained CNN (VGG - 19)
Content Loss
Generated Image
Pre trained CNN (VGG - 19)
+
Total Loss
Style Loss
Style Image
Pre trained CNN (VGG - 19)


Update Input image with gradients wrt. Loss using optimizer
Content Image
Pre trained CNN (VGG - 19)
Content Loss
Generated Image
Pre trained CNN (VGG - 19)
+
Total Loss
Style Loss
Style Image
Pre trained CNN (VGG - 19)

2. Neural Style Transfer

Advantages

- Requires only single pair of content image and style image.

Drawbacks

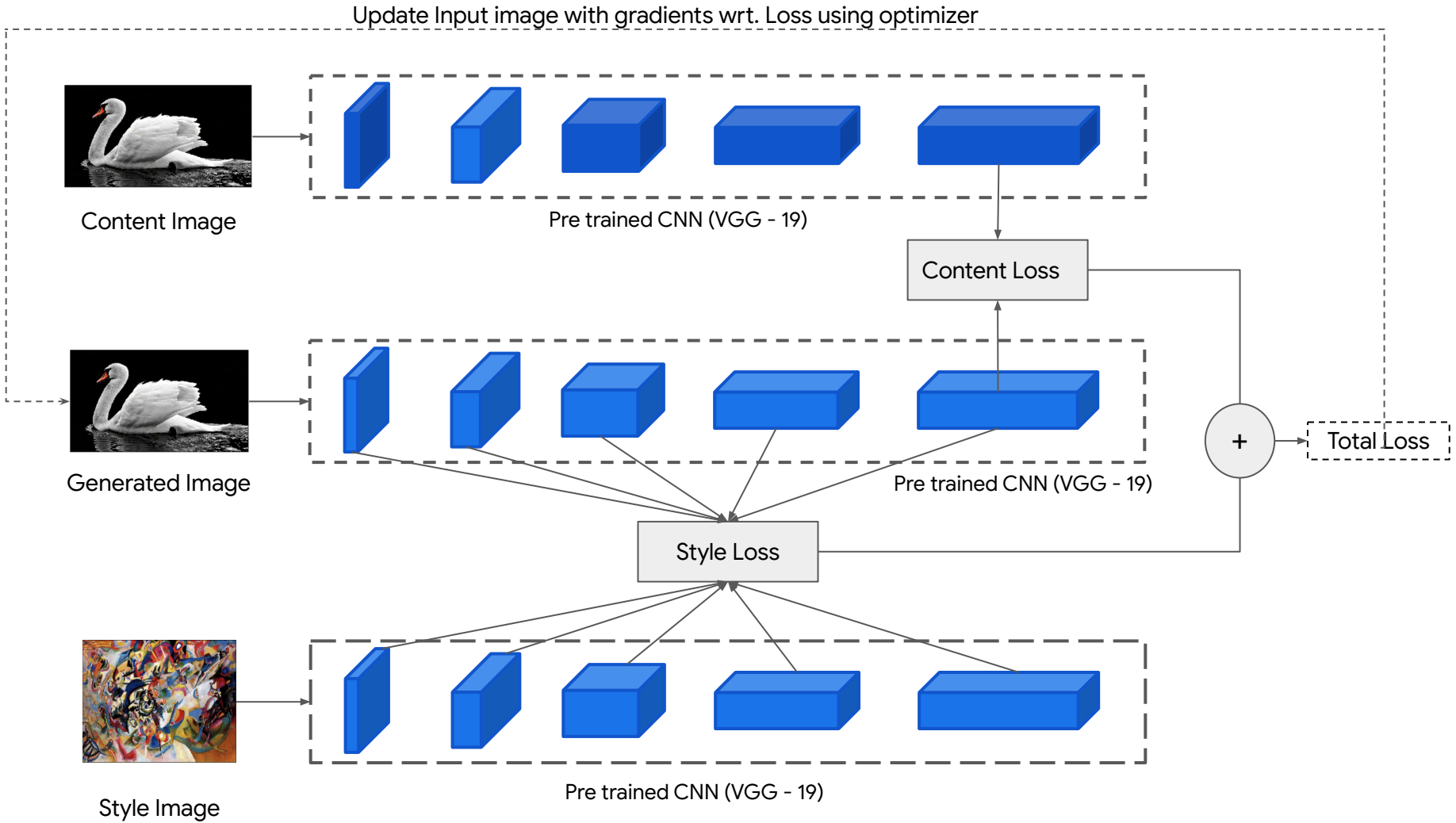
- Requires many passes for stylizing each image.
- Takes time in stylizing every time an input is given.

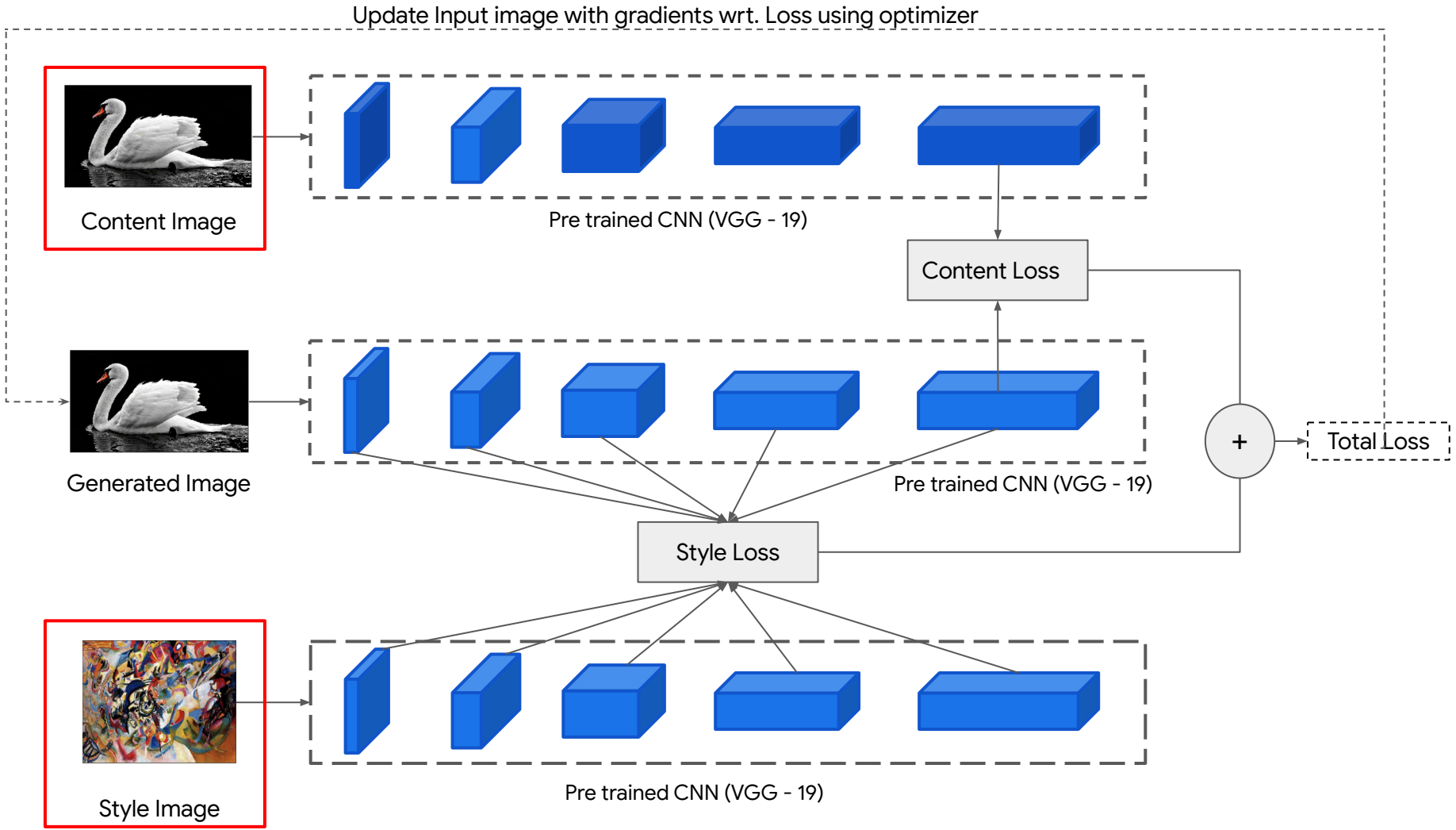
Perceptual Losses for Real-Time Style Transfer and Super Resolution

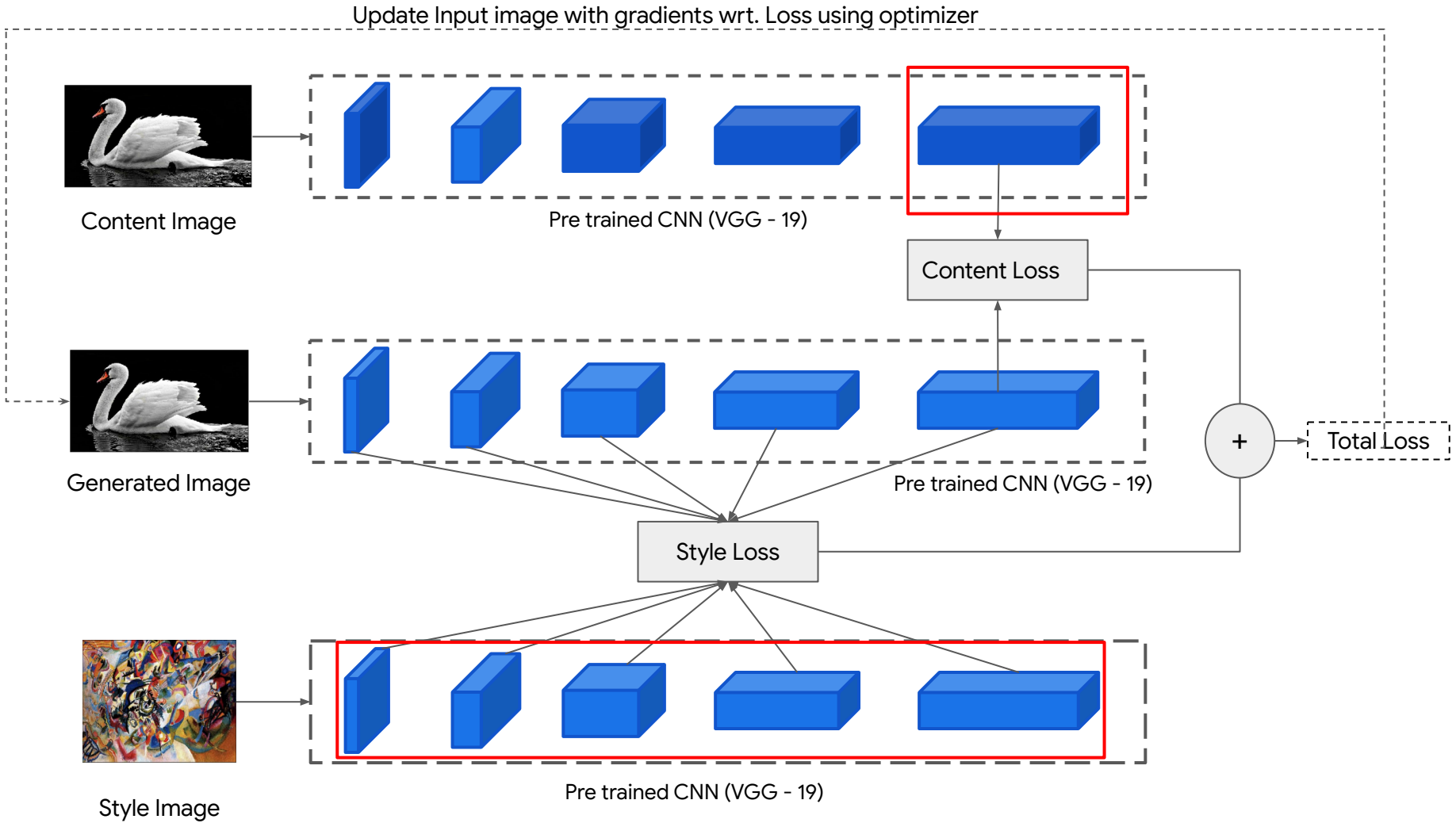
Justin Johnson / Alexandre Alahi / Fei Fei Li

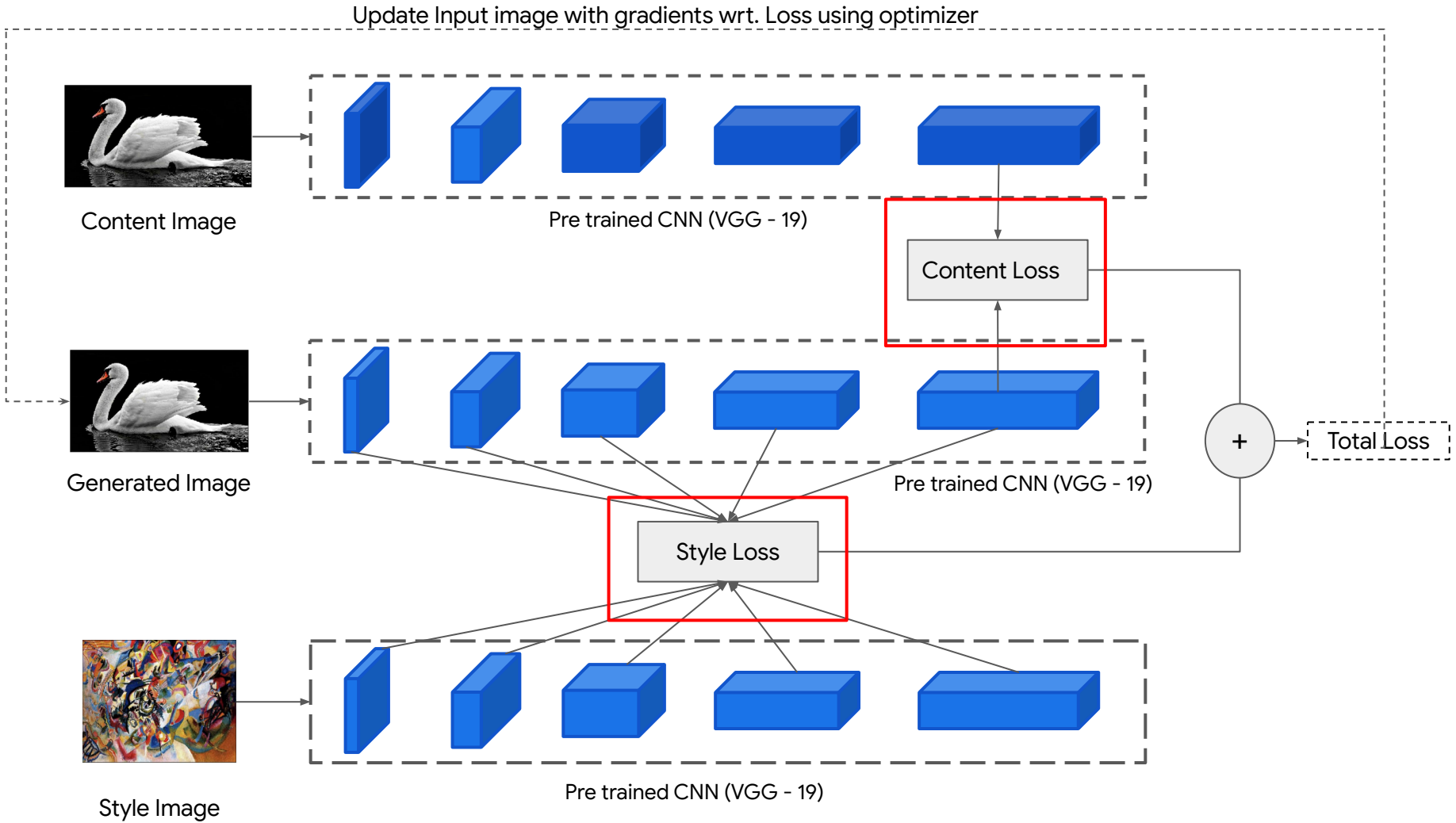
https://cs.stanford.edu/people/jcjohns/eccv16/

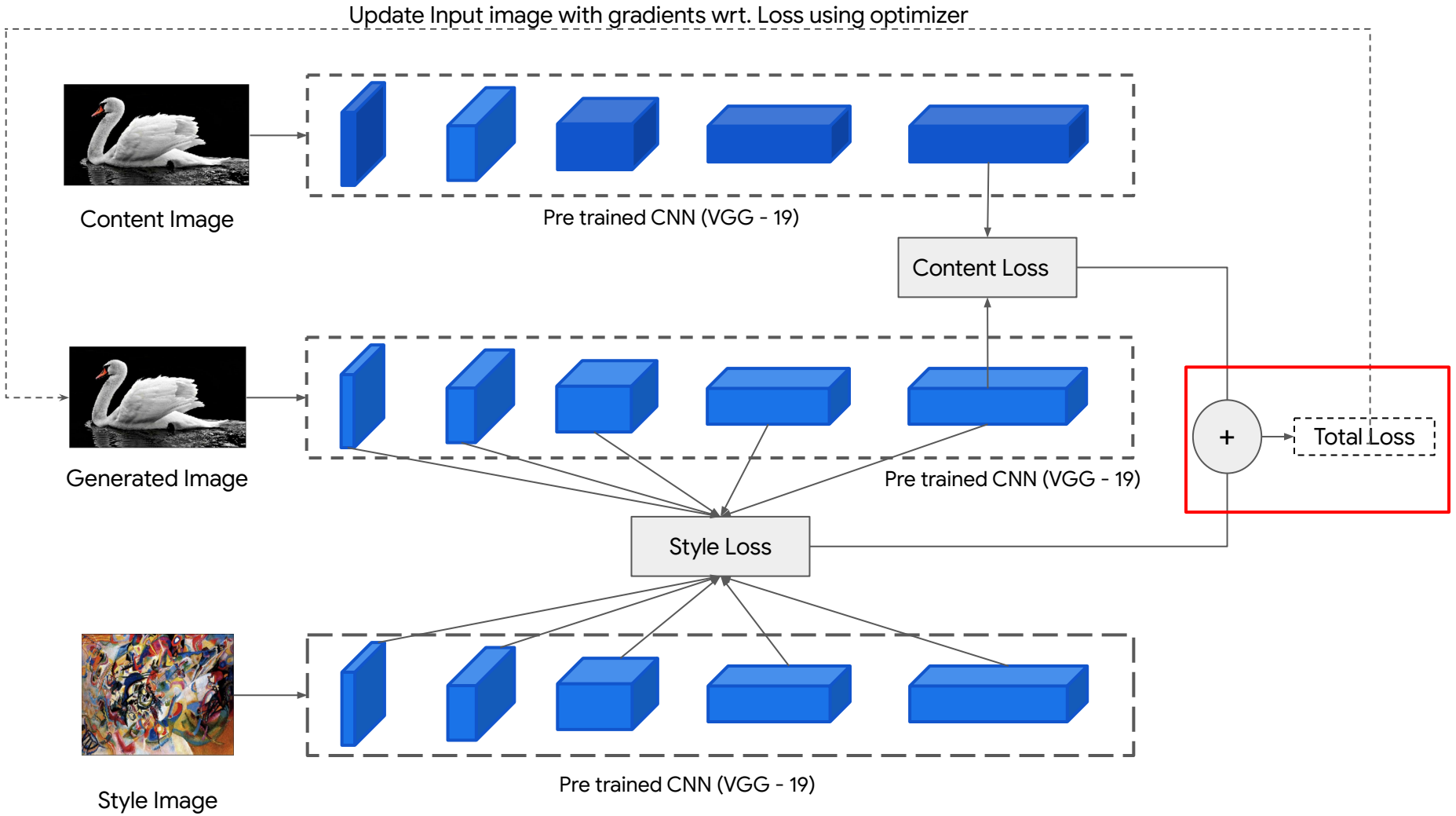
Fig. 7. Example results for style transfer on 512 × 512 images by applying models trained on 256 × 256 images. The style images are the same as Figure 6.

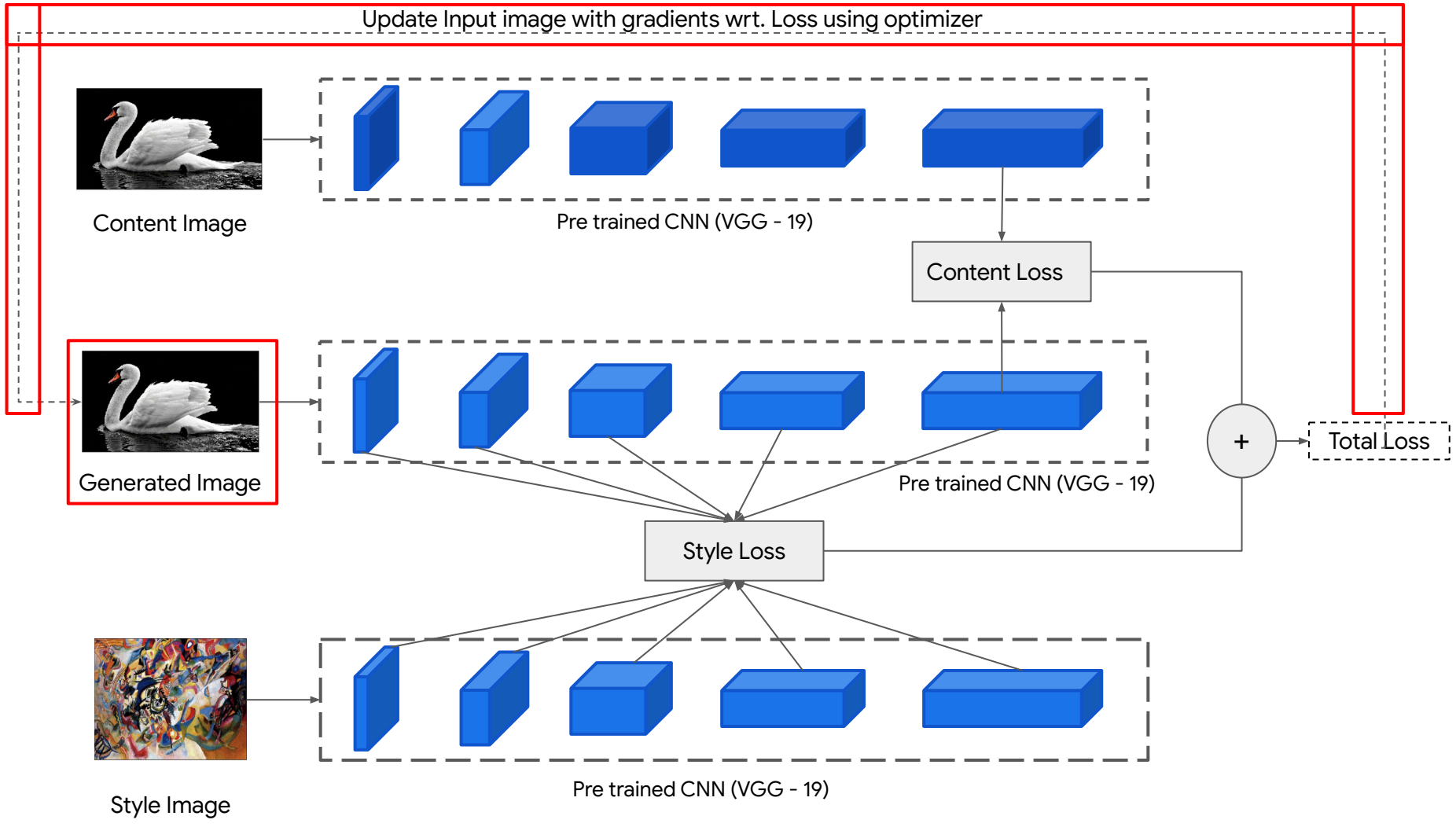

Update Input image with gradients wrt. Loss using optimizer
Content Image
Pre trained CNN (VGG - 19)
Content Loss
Generated Image
Pre trained CNN (VGG - 19)
+
Total Loss
Style Loss
Style Image
Pre trained CNN (VGG - 19)


Update Input image with gradients wrt. Loss using optimizer
Content Image
Pre trained CNN (VGG - 19)
Content Loss
Generated Image
Pre trained CNN (VGG - 19)
+
Total Loss
Style Loss
Style Image
Pre trained CNN (VGG - 19)


Update Input image with gradients wrt. Loss using optimizer
Content Image
Pre trained CNN (VGG - 19)
Content Loss
Generated Image
Pre trained CNN (VGG - 19)
+
Total Loss
Style Loss
Style Image
Pre trained CNN (VGG - 19)


Update Input image with gradients wrt. Loss using optimizer
Content Image
Pre trained CNN (VGG - 19)
Content Loss
Generated Image
Pre trained CNN (VGG - 19)
+
Total Loss
Style Loss
Style Image
Pre trained CNN (VGG - 19)


Update Input image with gradients wrt. Loss using optimizer
Content Image
Pre trained CNN (VGG - 19)
Content Loss
Generated Image
Pre trained CNN (VGG - 19)
+
Total Loss
Style Loss
Style Image
Pre trained CNN (VGG - 19)


Update Input image with gradients wrt. Loss using optimizer
Content Image
Pre trained CNN (VGG - 19)
Content Loss
Generated Image
Pre trained CNN (VGG - 19)
+
Total Loss
Style Loss
Style Image
Pre trained CNN (VGG - 19)

Steps to Develop Neural Style Transfer

1. Preprocess, content & style images
2. Load pre trained model, define loss functions
3. Loop:
    a. Optimize and generate new image
    b. Visualize Outputs

Steps to Develop Neural Style Transfer

1. Preprocess, content & style images
2. Load pre trained model, define loss functions
3. Loop:
    a. Optimize and generate new image
    b. Visualize Outputs

Steps to Develop Neural Style Transfer

1. Preprocess, content & style images
2. Load pre trained model, define loss functions
3. Loop:
    a. Optimize and generate new image
    b. Visualize Outputs

Steps to Develop Neural Style Transfer

1. Preprocess, content & style images
2. Load pre trained model, define loss functions
3. Loop:
    a. Optimize and generate new image
    b. Visualize Outputs

Steps to Develop Neural Style Transfer

1. Preprocess, content & style images
2. Load pre trained model, define loss functions
3. Loop:
    a. Optimize and generate new image
    b. Visualize Outputs

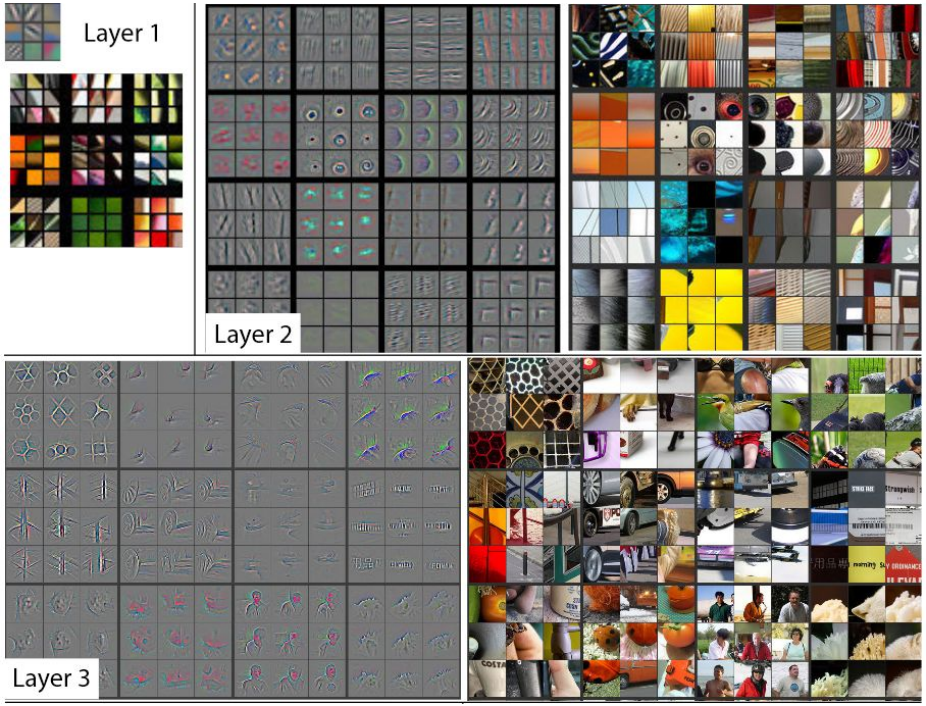
Load and Preprocess Images

- VGG-19 pre trained on ‘imagenet’
- Accepts pixels values [0...255]
    - Don’t normalize to [0 ... 1]
- Expects centered pixel values

Keras Preprocess_input

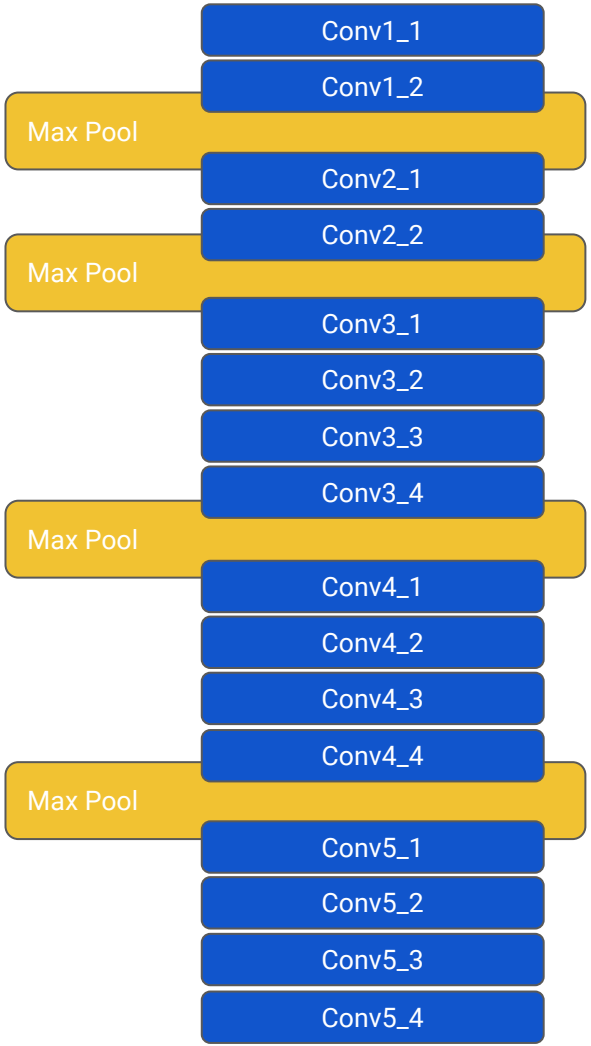
```
def preprocess_image(image):
    ...
    image = tf.keras.applications.vgg19.preprocess_input(image)

    return image
```

How CNN visualizes images?

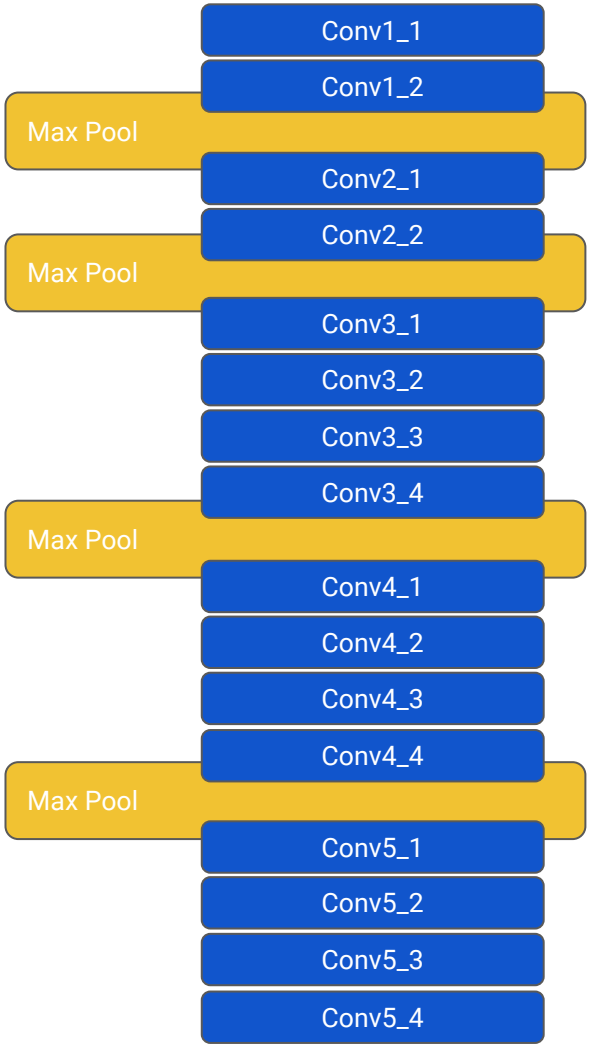


https://arxiv.org/pdf/1311.2901.pdf

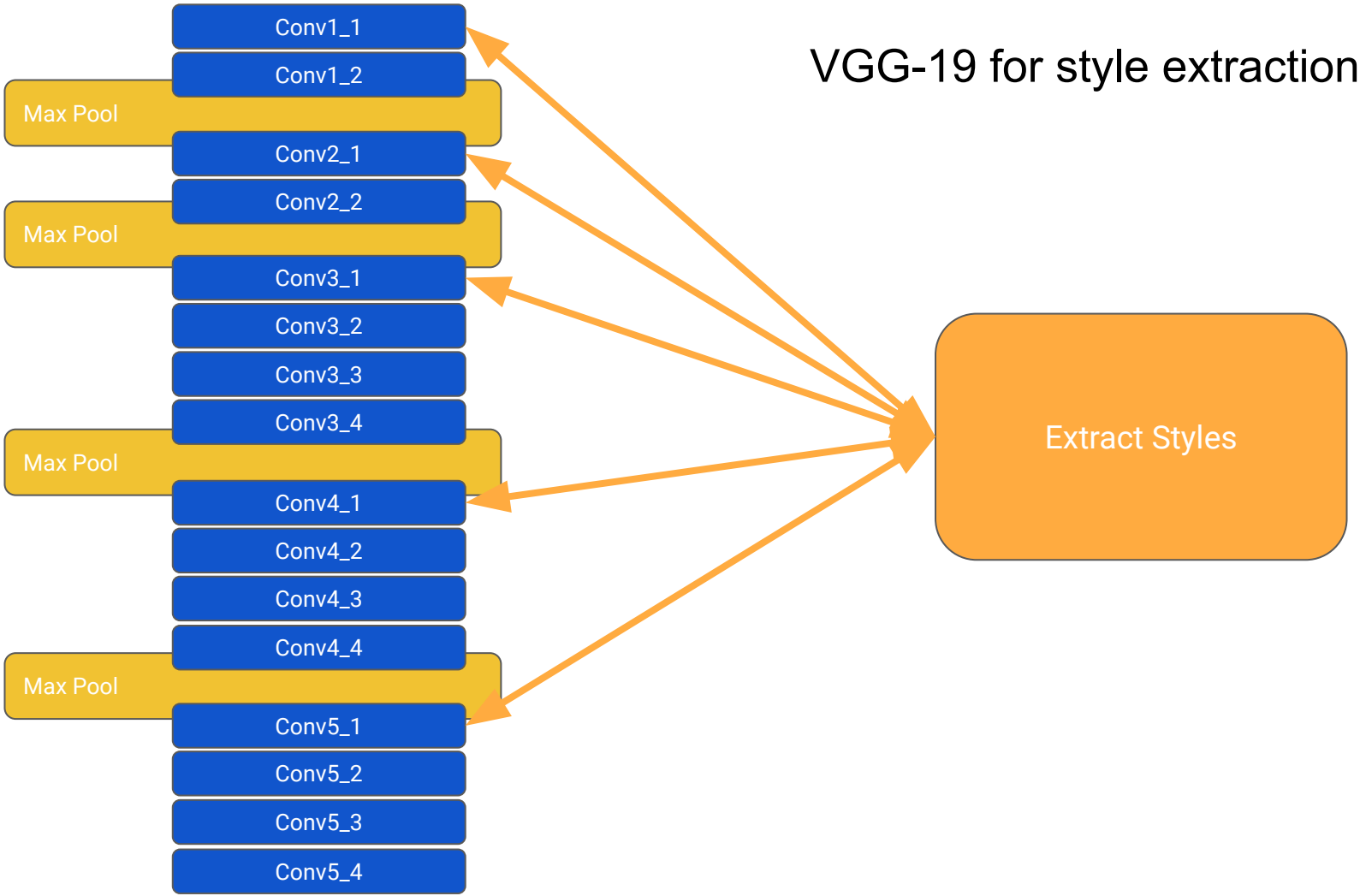
VGG-19 for style extraction



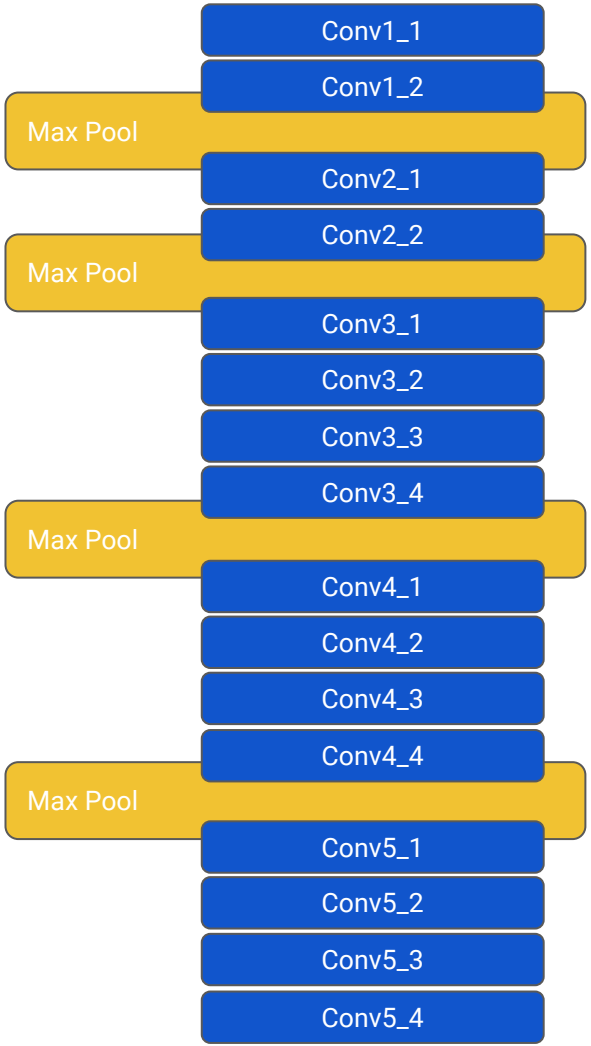
VGG-19 for style extraction



Extract Styles

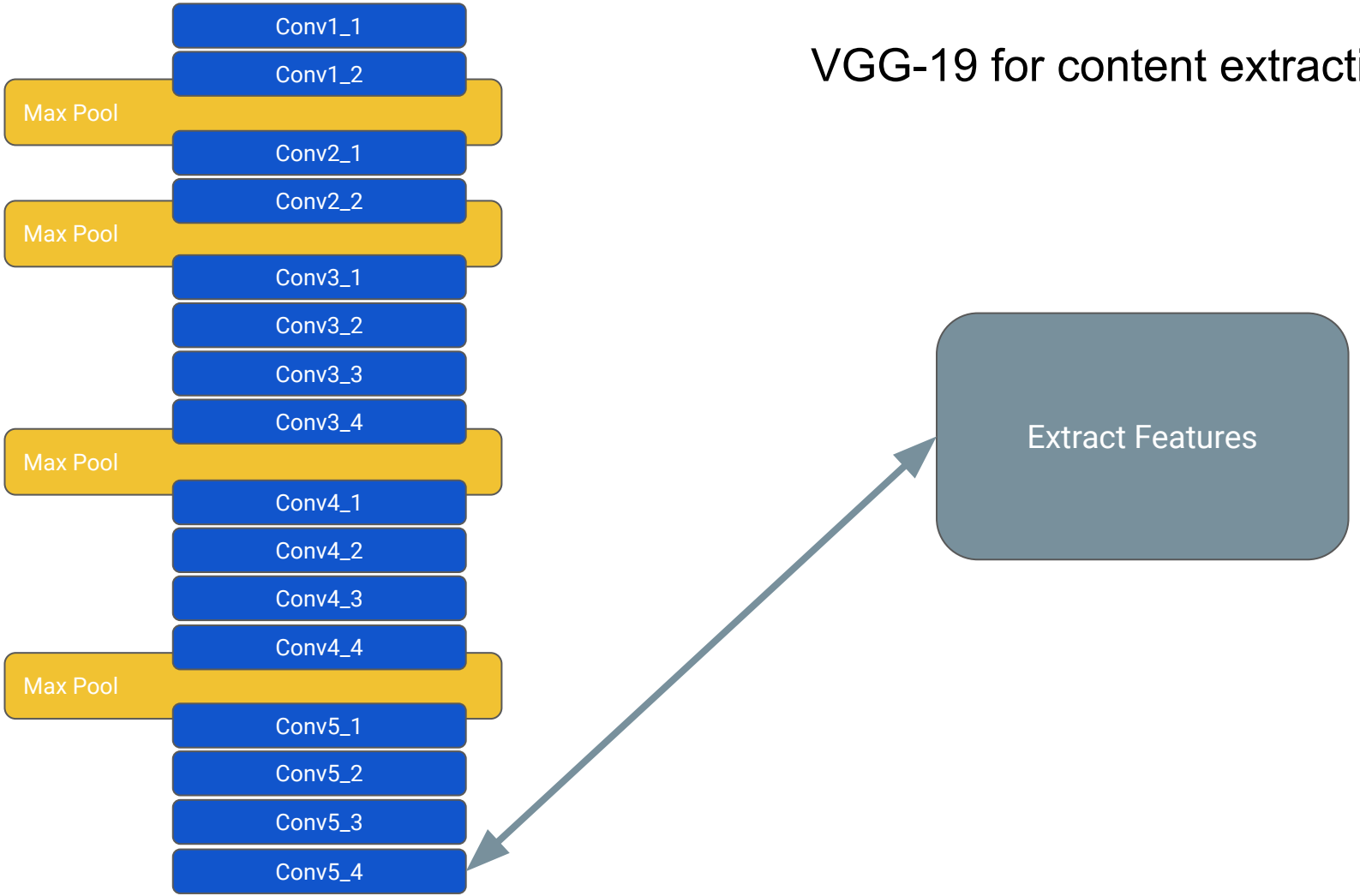

VGG-19 for style extraction
Conv1_1
Conv1_2
Max Pool
Conv2_1
Conv2_2
Max Pool
Conv3_1
Conv3_2
Conv3_3
Conv3_4
Max Pool
Conv4_1
Conv4_2
Conv4_3
Conv4_4
Max Pool
Conv5_1
Conv5_2
Conv5_3
Conv5_4
Extract Styles

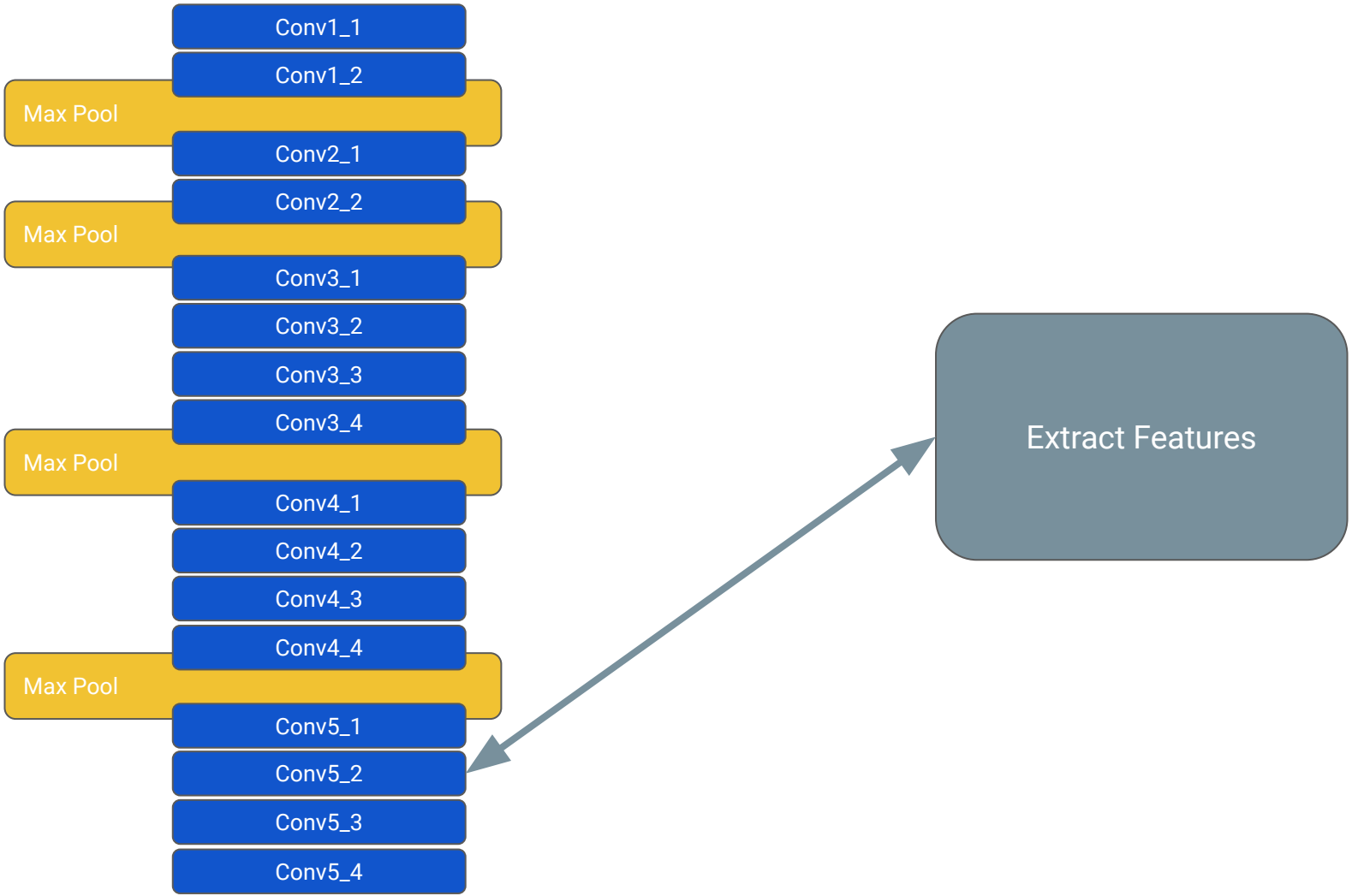
VGG-19 for content extraction



Extract Features

VGG-19 for content extraction




Conv1_1
Conv1_2
Max Pool
Conv2_1
Conv2_2
Max Pool
Conv3_1
Conv3_2
Conv3_3
Conv3_4
Max Pool
Conv4_1
Conv4_2
Conv4_3
Conv4_4
Max Pool
Conv5_1
Conv5_2
Conv5_3
Conv5_4
Extract Features

```
# Content layer where will pull our feature maps
content_layers = ['block5_conv2']

# Style layer of interest
style_layers = ['block1_conv1',
                'block2_conv1',
                'block3_conv1',
                'block4_conv1',
                'Block5_conv1']

layer_names = content_layers + style_layers
```

```
# Content layer where will pull our feature maps
content_layers = ['block5_conv2']

# Style layer of interest
style_layers = ['block1_conv1',
                'block2_conv1',
                'block3_conv1',
                'block4_conv1',
                'Block5_conv1']

layer_names = content_layers + style_layers
```

```
# Content layer where will pull our feature maps
content_layers = ['block5_conv2']

# Style layer of interest
style_layers = ['block1_conv1',
                'block2_conv1',
                'block3_conv1',
                'block4_conv1',
                'Block5_conv1']

layer_names = content_layers + style_layers
```

```
# Content layer where we will pull our feature maps
content_layers = ['block5_conv2']

# Style layer of interest
style_layers = ['block1_conv1',
                'block2_conv1',
                'block3_conv1',
                'block4_conv1',
                'block5_conv1']

layer_names = content_layers + style_layers
```

```
def vgg_model(layer_names):
  """ Creates a vgg model that returns a list of intermediate output values."""
  vgg = tf.keras.applications.vgg19.VGG19(include_top=False, weights='imagenet')
  vgg.trainable = False

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

  model = tf.keras.Model(inputs=vgg.input, outputs=outputs)

  return model
```

```
def vgg_model(layer_names):
  """ Creates a vgg model that returns a list of intermediate output values."""
  vgg = tf.keras.applications.vgg19.VGG19(include_top=False, weights='imagenet')
  vgg.trainable = False

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

  model = tf.keras.Model(inputs=vgg.input, outputs=outputs)

  return model
```

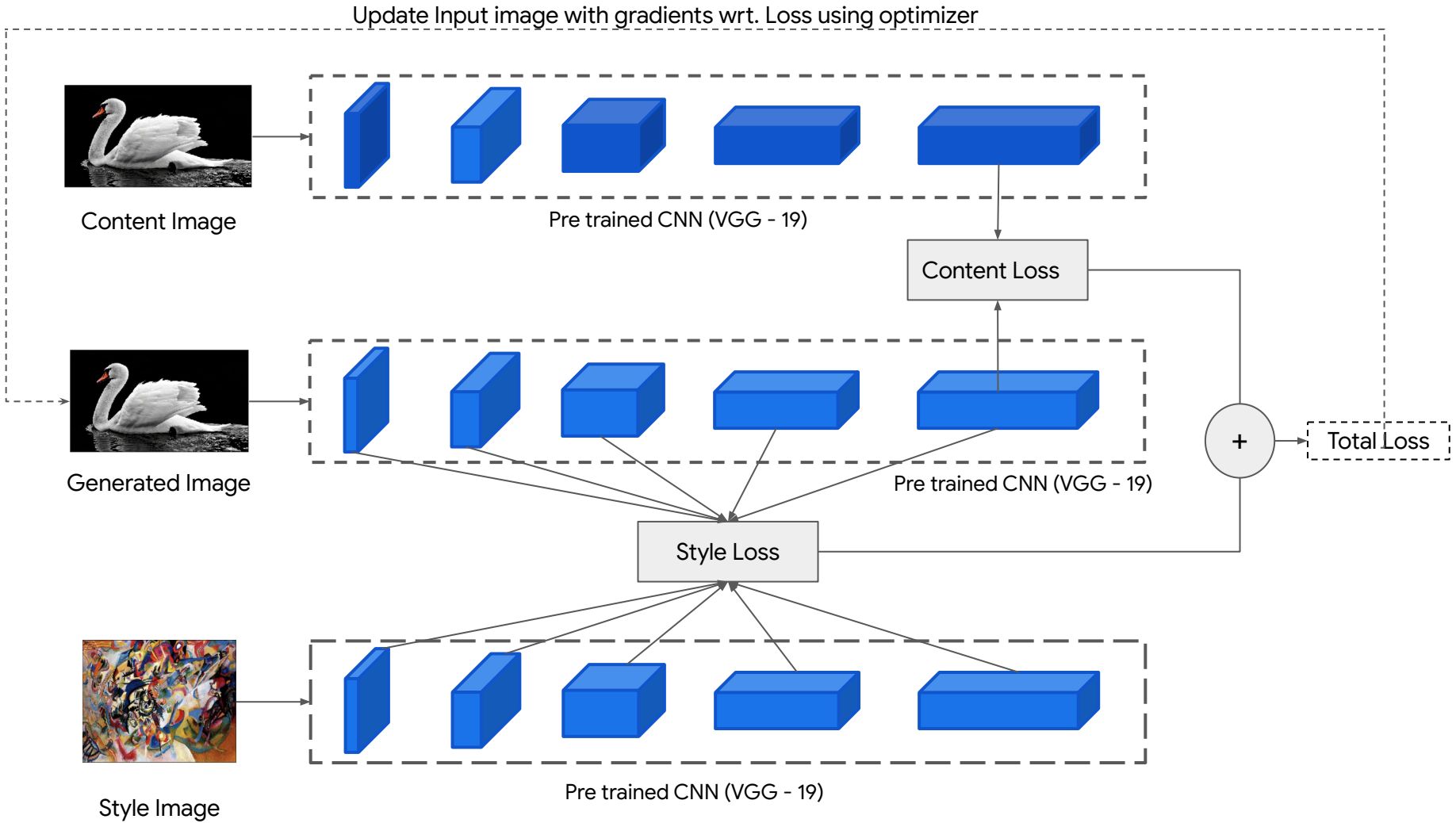
```
def vgg_model(layer_names):
  """ Creates a vgg model that returns a list of intermediate output values."""
  vgg = tf.keras.applications.vgg19.VGG19(include_top=False, weights='imagenet')
  vgg.trainable = False

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

  model = tf.keras.Model(inputs=vgg.input, outputs=outputs)

  return model
```

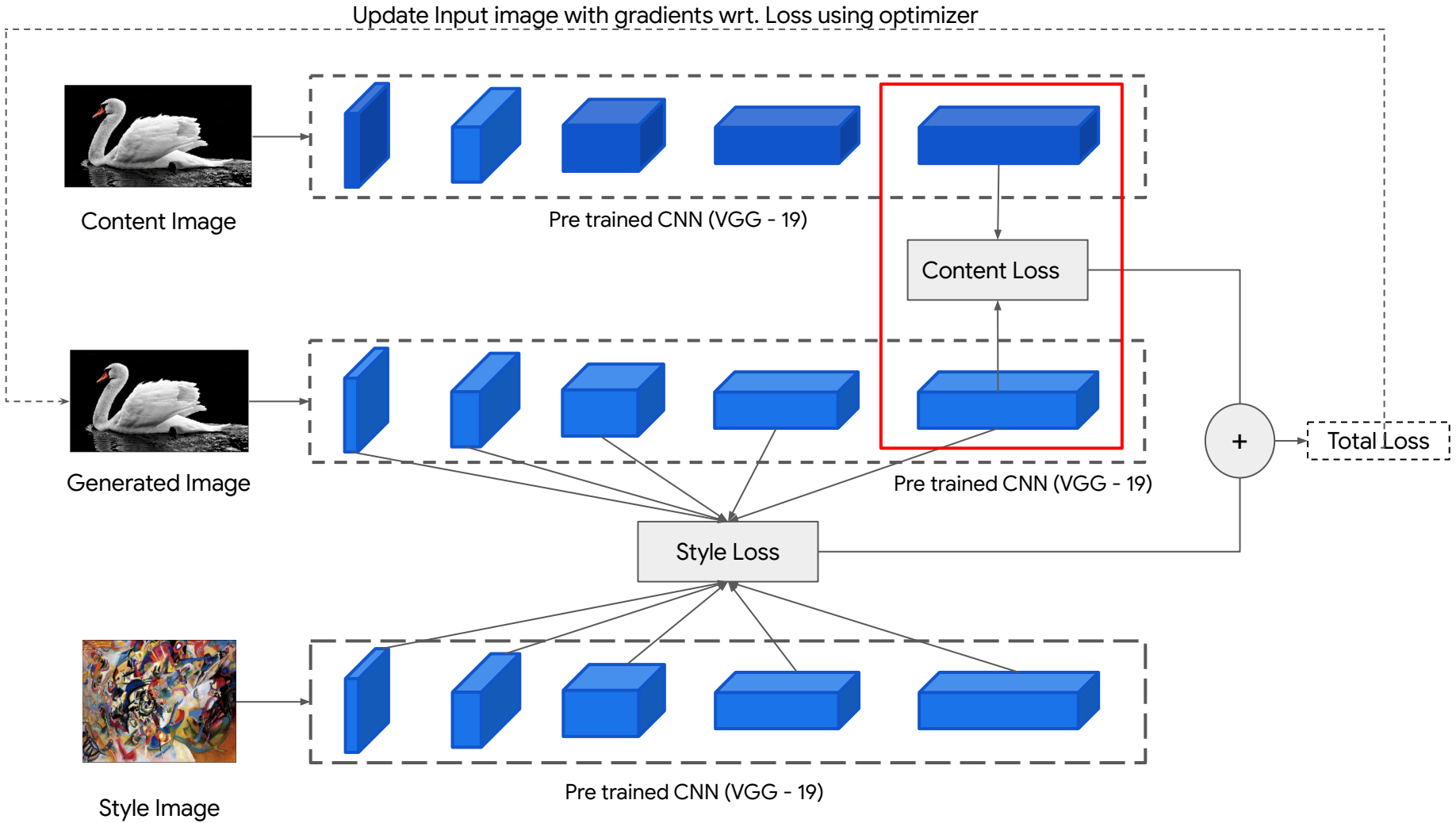
```
def vgg_model(layer_names):
  """ Creates a vgg model that returns a list of intermediate output values."""
  vgg = tf.keras.applications.vgg19.VGG19(include_top=False, weights='imagenet')
  vgg.trainable = False

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

  model = tf.keras.Model(inputs=vgg.input, outputs=outputs)

  return model
```

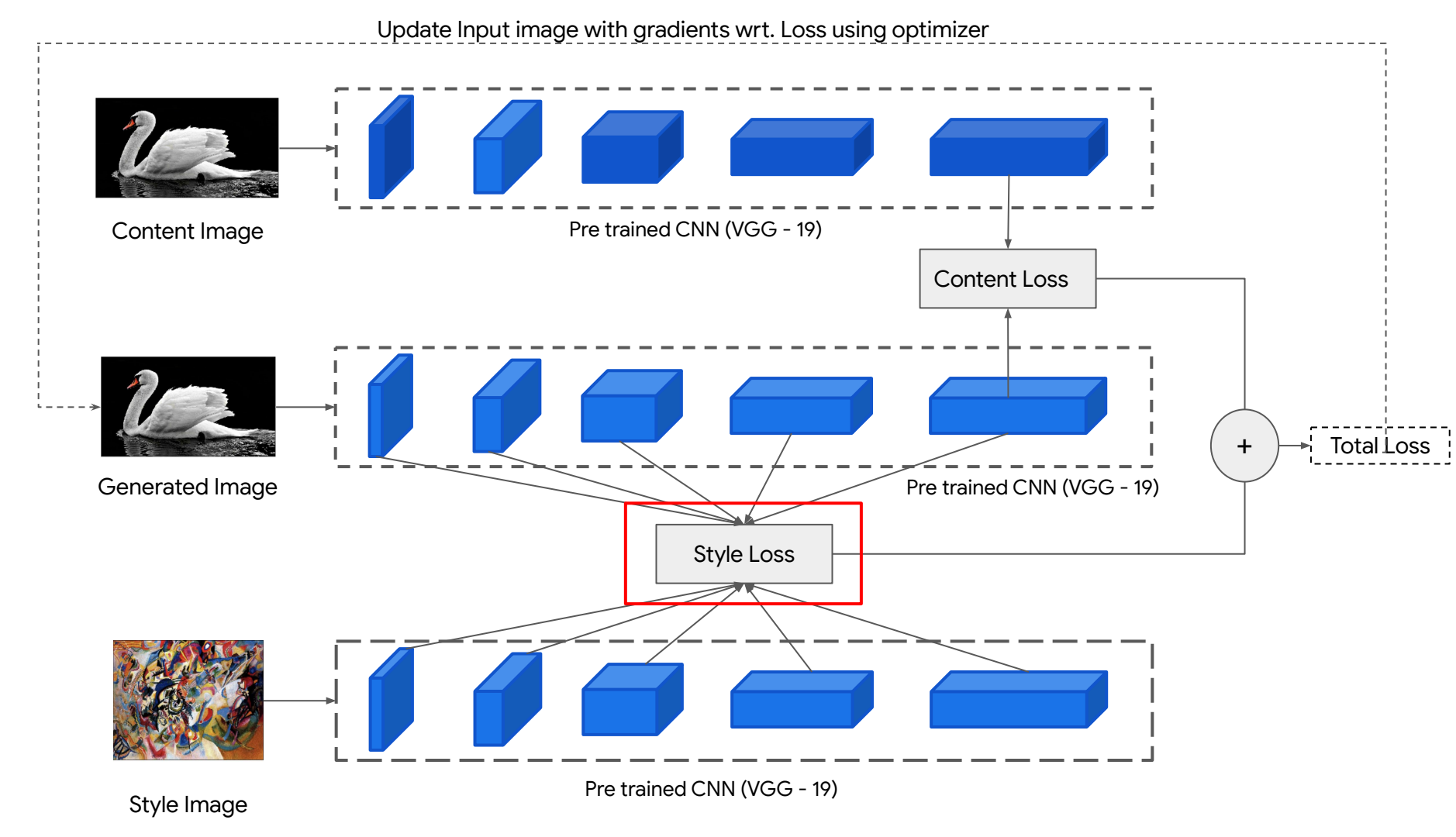
```
def vgg_model(layer_names):
  """ Creates a vgg model that returns a list of intermediate output values."""
  vgg = tf.keras.applications.vgg19.VGG19(include_top=False, weights='imagenet')
  vgg.trainable = False

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

  model = tf.keras.Model(inputs=vgg.input, outputs=outputs)

  return model
```

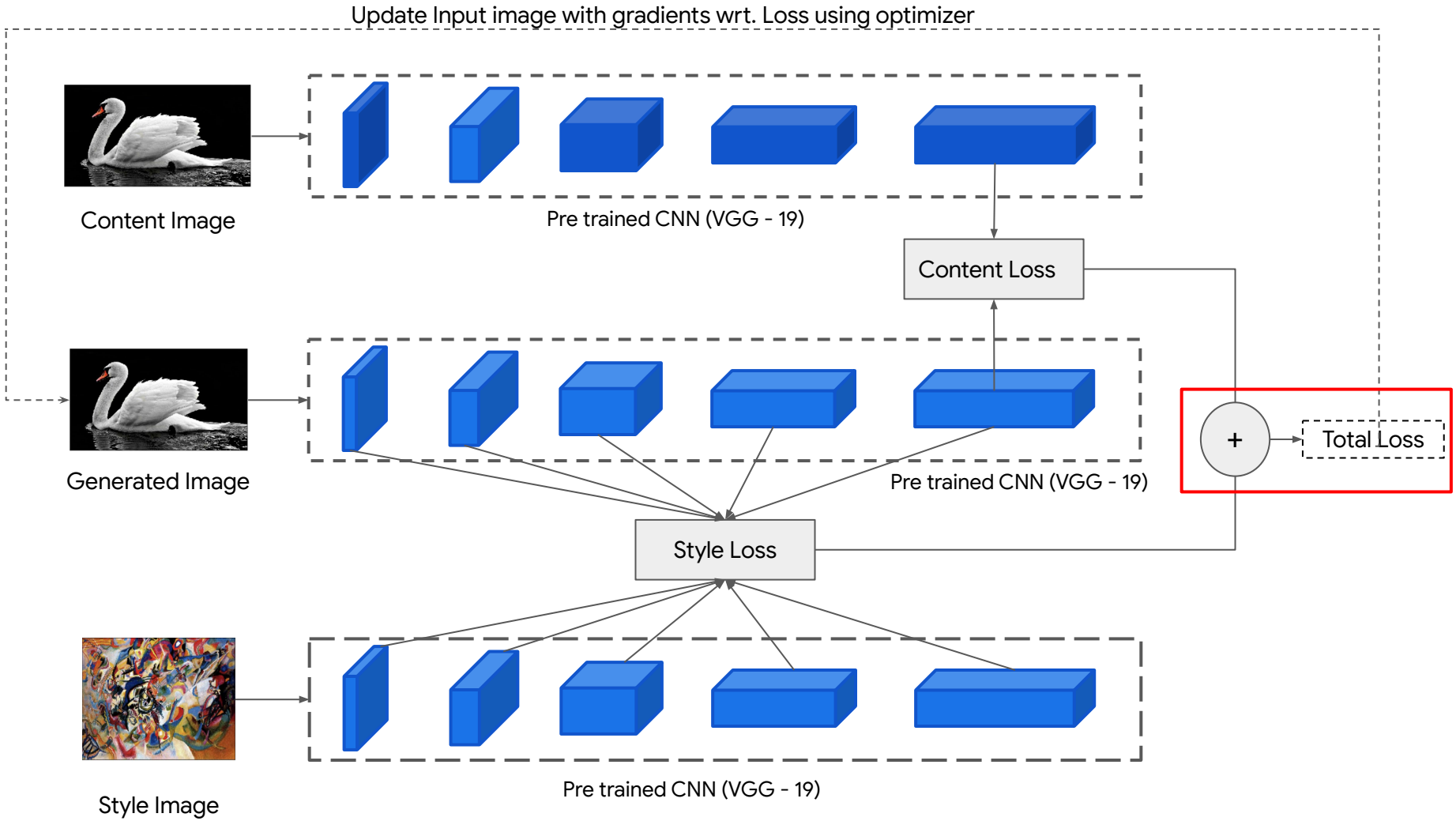
```
def vgg_model(layer_names):
  """ Creates a vgg model that returns a list of intermediate output values."""
  vgg = tf.keras.applications.vgg19.VGG19(include_top=False, weights='imagenet')
  vgg.trainable = False

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

  model = tf.keras.Model(inputs=vgg.input, outputs=outputs)

  return model
```

```
def vgg_model(layer_names):
  """ Creates a vgg model that returns a list of intermediate output values."""
  vgg = tf.keras.applications.vgg19.VGG19(include_top=False, weights='imagenet')
  vgg.trainable = False

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

  model = tf.keras.Model(inputs=vgg.input, outputs=outputs)

  return model
```


Update Input image with gradients wrt. Loss using optimizer
Content Image
Pre trained CNN (VGG - 19)
Content Loss
Generated Image
Pre trained CNN (VGG - 19)
+
Total Loss
Style Loss
Style Image
Pre trained CNN (VGG - 19)


Update Input image with gradients wrt. Loss using optimizer
Content Image
Pre trained CNN (VGG - 19)
Content Loss
Generated Image
Pre trained CNN (VGG - 19)
+
Total Loss
Style Loss
Style Image
Pre trained CNN (VGG - 19)


Update Input image with gradients wrt. Loss using optimizer
Content Image
Pre trained CNN (VGG - 19)
Content Loss
Generated Image
Pre trained CNN (VGG - 19)
+
Total Loss
Style Loss
Style Image
Pre trained CNN (VGG - 19)


Update Input image with gradients wrt. Loss using optimizer
Content Image
Pre trained CNN (VGG - 19)
Content Loss
Generated Image
Pre trained CNN (VGG - 19)
+
Total Loss
Style Loss
Style Image
Pre trained CNN (VGG - 19)

Total Loss

$$L_{total} = L_{content} + L_{style}$$

Total Loss

$$L_{total} = \alpha L_{content} + \beta L_{style}$$

α : Content Weight

β : Style Weight

Total Loss

$$L_{total}(\vec{p}, \vec{a}, \vec{x}) = \alpha L_{content}(\vec{p}, \vec{x}) + \beta L_{style}(\vec{a}, \vec{x})$$

α : Content Weight

β : Style Weight

Total Loss

$$L_{total}(\vec{p}, \vec{a}, \vec{x}) = \alpha L_{content}(\vec{p}, \vec{x}) + \beta L_{style}(\vec{a}, \vec{x})$$

$\vec{p}$: Content Image (Original Photograph)

$\vec{x}$: Generated Image initialized to Input Image

α : Content Weight

β : Style Weight

Total Loss

$$L_{total}(\vec{p}, \vec{a}, \vec{x}) = \alpha L_{content}(\vec{p}, \vec{x}) + \beta L_{style}(\vec{a}, \vec{x})$$

$\vec{a}$: Style Image

$\vec{x}$: Generated Image initialized to Input Image

α : Content Weight

β : Style Weight

Total Loss

$$L_{total}(\vec{p}, \vec{a}, \vec{x}) = \alpha L_{content}(\vec{p}, \vec{x}) + \beta L_{style}(\vec{a}, \vec{x})$$

$\vec{p}$: Content Image (Original Photograph)

$\vec{a}$: Style Image

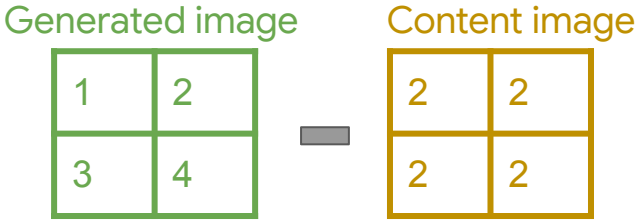
$\vec{x}$: Generated Image initialized to Input Image

α : Content Weight

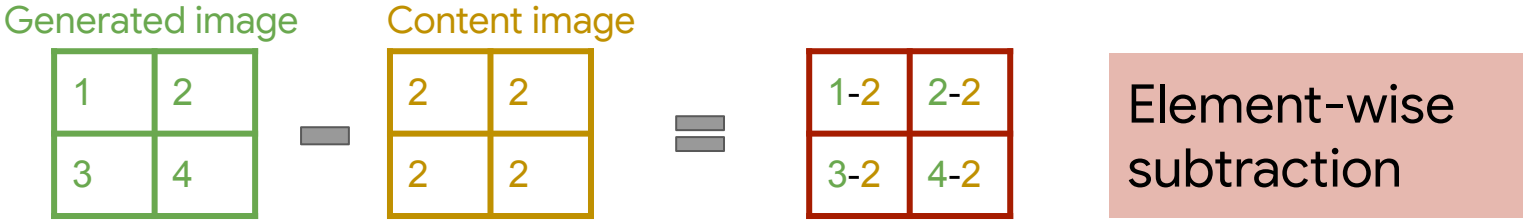
β : Style Weight

Content Loss

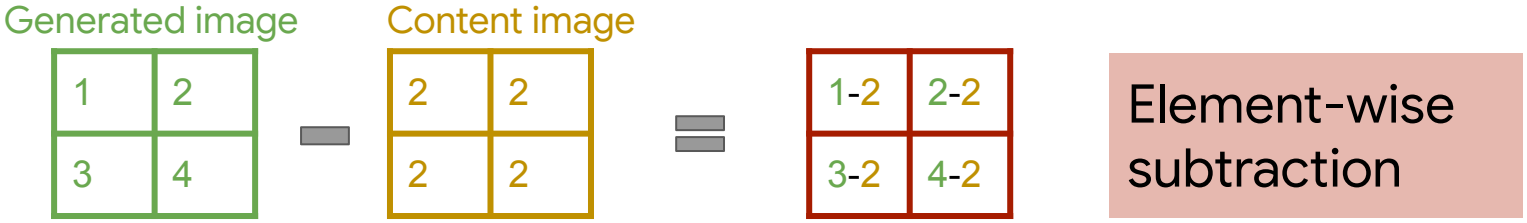
Content Loss



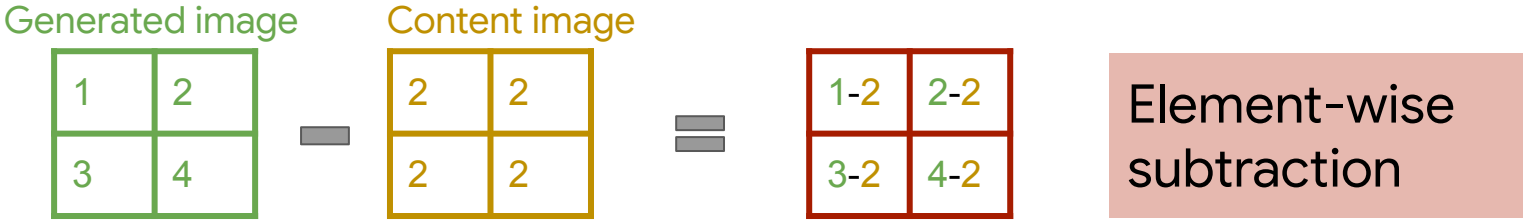
Content Loss



Content Loss



Content Loss



Content Loss

Generated image

1	2
3	4

−

Content image

2	2
2	2

=

1-2	2-2
3-2	4-2

Element-wise subtraction

$(1-2)^2$	$(2-2)^2$
$(3-2)^2$	$(4-2)^2$

Element-wise square

$1^2 + 0^2 + 1^2 + 2^2 =$ 6

Reduce sum

$(½)\ 6 =$ 3

weight

Content Loss

$$L_{content}(\overrightarrow{p}, \overrightarrow{x}, l) = \frac{1}{2}\sum_{i,j}(F^l_{i,j} - P^l_{i,j})^2$$

l: layer *l*

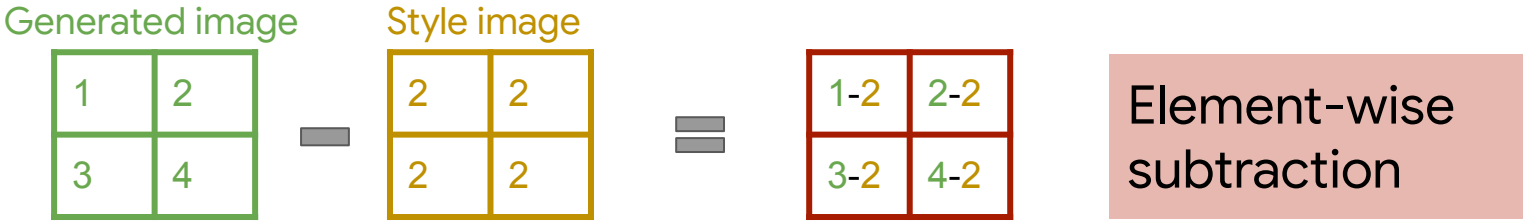
$\mathbf{F}^{l}_{ij}$: Content representation of generated image x in layer l (Activation of *i*th feature map at position *j* in layer *l of generated image)*.

$\mathbf{P}^{l}_{ij}$: Content representation of content image p in layer l. (Activation of the *i* th feature map at position *j* in layer *l* of content image.)

Content Loss - In Practice

```
def get_content_loss(features, targets):
   return 0.5 * tf.reduce_sum(tf.square(features - targets))
```

Style Loss


Generated image
1
2
3
4
−
Style image
2
2
2
2
=
1-2
2-2
3-2
4-2
Element-wise subtraction
$(1-2)^2$
$(2-2)^2$
$(3-2)^2$
$(4-2)^2$
Element-wise square
$1^2 + 0^2 + 1^2 + 2^2 =$
6
Reduce sum
$(?/_?)\ 6 =$
?
weight

Style Loss

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

l: layer *l*

$\mathbf{A}^l_{ij}$: Style Representation(Gram Matrix) of style image a.

$\mathbf{G}^l_{ij}$: Style Representation(Gram Matrix) of generated image x

Style Loss

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

l: layer *l*

$\mathbf{A}^l_{ij}$: Style Representation(Gram Matrix) of style image a.

$\mathbf{G}^l_{ij}$: Style Representation(Gram Matrix) of generated image x

Style Loss

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

l: layer *l*

$\mathbf{A}^l_{ij}$: Style Representation(Gram Matrix) of style image a.

$\mathbf{G}^l_{ij}$: Style Representation(Gram Matrix) of generated image x

Gram Matrix

$$G_{ij}^{l} = \sum_{k} F_{ik}^{l} F_{jk}^{l}$$

G_{ij}^{l}: inner product between vectorized feature maps i and j in layer l.

Gram Matrix - In Practice

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

https://numpy.org/doc/stable/reference/generated/numpy.einsum.html

Get Style Feature Representation at Layers

```
def get_style_image_features(image):
  preprocessed_style_image = preprocess_image(image)
  style_outputs = vgg(preprocessed_style_image)
  gram_style_features =
     [gram_matrix(style_layer) for style_layer in style_outputs[:num_style_layers]]

  return gram_style_features
```

Get Style Feature Representation at Layers

```
def get_style_image_features(image):
  preprocessed_style_image = preprocess_image(image)
  style_outputs = vgg(preprocessed_style_image)
  gram_style_features =
     [gram_matrix(style_layer) for style_layer in style_outputs[:num_style_layers]]

  return gram_style_features
```

Get Style Feature Representation at Layers

```
def get_style_image_features(image):
  preprocessed_style_image = preprocess_image(image)
  style_outputs = vgg(preprocessed_style_image)
  gram_style_features =
     [gram_matrix(style_layer) for style_layer in style_outputs[:num_style_layers]]

  return gram_style_features
```

Get Style Feature Representation at Layers

```
def get_style_image_features(image):
  preprocessed_style_image = preprocess_image(image)
  style_outputs = vgg(preprocessed_style_image)
  gram_style_features =
     [gram_matrix(style_layer) for style_layer in style_outputs[:num_style_layers]]

  return gram_style_features
```

Get Style Feature Representation at Layers

```
def get_style_loss(features, targets):
  return tf.reduce_mean(tf.square(features - targets))
```

Total Loss

$$L_{total}(\vec{p}, \vec{a}, \vec{x}) = \alpha L_{content}(\vec{p}, \vec{x}) + \beta L_{style}(\vec{a}, \vec{x})$$

```
def get_style_content_loss(style_targets, style_outputs,
                           content_targets, content_outputs,
                           style_weight, content_weight):

  style_loss = tf.add_n([get_style_loss(style_output, style_target)
                           for style_output, style_target in zip(style_outputs, style_targets)])

  style_loss *= style_weight / num_style_layers

  content_loss = tf.add_n([get_content_loss(content_output, content_target)
                  for content_output, content_target in zip(content_outputs, content_targets)])
  content_loss *= content_weight / num_content_layers

  loss = style_loss + content_loss
  return loss
```

```
def get_style_content_loss(style_targets, style_outputs,
                           content_targets, content_outputs,
                           style_weight, content_weight):

  style_loss = tf.add_n([get_style_loss(style_output, style_target)
                         for style_output, style_target in zip(style_outputs, style_targets)])

  style_loss *= style_weight / num_style_layers

  content_loss = tf.add_n([get_content_loss(content_output, content_target)
                  for content_output, content_target in zip(content_outputs, content_targets)])
  content_loss *= content_weight / num_content_layers

  loss = style_loss + content_loss
  return loss
```

```
def get_style_content_loss(style_targets, style_outputs,
                           content_targets, content_outputs,
                           style_weight, content_weight):

  style_loss = tf.add_n([get_style_loss(style_output, style_target)
                         for style_output, style_target in zip(style_outputs, style_targets)])

  style_loss *= style_weight / num_style_layers

  content_loss = tf.add_n([get_content_loss(content_output, content_target)
                  for content_output, content_target in zip(content_outputs, content_targets)])
  content_loss *= content_weight / num_content_layers

  loss = style_loss + content_loss
  return loss
```

```
def get_style_content_loss(style_targets, style_outputs,
                           content_targets, content_outputs,
                           style_weight, content_weight):

  style_loss = tf.add_n([get_style_loss(style_output, style_target)
                         for style_output, style_target in zip(style_outputs, style_targets)])

  style_loss *= style_weight / num_style_layers

  content_loss = tf.add_n([get_content_loss(content_output, content_target)
                  for content_output, content_target in zip(content_outputs, content_targets)])
  content_loss *= content_weight / num_content_layers

  loss = style_loss + content_loss
  return loss
```

```
def get_style_content_loss(style_targets, style_outputs,
                           content_targets, content_outputs,
                           style_weight, content_weight):

  style_loss = tf.add_n([get_style_loss(style_output, style_target)
                         for style_output, style_target in zip(style_outputs, style_targets)])

  style_loss *= style_weight / num_style_layers

  content_loss = tf.add_n([get_content_loss(content_output, content_target)
                 for content_output, content_target in zip(content_outputs, content_targets)])
  content_loss *= content_weight / num_content_layers

  loss = style_loss + content_loss
  return loss
```

```
def get_style_content_loss(style_targets, style_outputs,
                           content_targets, content_outputs,
                           style_weight, content_weight):

  style_loss = tf.add_n([get_style_loss(style_output, style_target)
                           for style_output, style_target in zip(style_outputs, style_targets)])

  style_loss *= style_weight / num_style_layers

  content_loss = tf.add_n([get_content_loss(content_output, content_target)
                  for content_output, content_target in zip(content_outputs, content_targets)])
  content_loss *= content_weight / num_content_layers

  loss = style_loss + content_loss
  return loss
```

```
def calculate_gradients(image, content_targets,
                        style_targets, style_weight,
                        content_weight,with_regularization=False):

    with tf.GradientTape() as tape:
        style_features = get_style_image_features(image)
        content_features = get_content_image_features(image)
        loss = get_style_content_loss(style_targets, style_features,
                                      content_targets, content_features,
                                      style_weight, content_weight)

    gradients = tape.gradient(loss, image)

    return gradients
```

```
def calculate_gradients(image, content_targets,
                        style_targets, style_weight,
                        content_weight,with_regularization=False):

    with tf.GradientTape() as tape:
        style_features = get_style_image_features(image)
        content_features = get_content_image_features(image)
        loss = get_style_content_loss(style_targets, style_features,
                                      content_targets, content_features,
                                      style_weight, content_weight)

    gradients = tape.gradient(loss, image)

    return gradients
```

```
def calculate_gradients(image, content_targets,
                        style_targets, style_weight,
                        content_weight,with_regularization=False):

    with tf.GradientTape() as tape:
        style_features = get_style_image_features(image)
        content_features = get_content_image_features(image)
        loss = get_style_content_loss(style_targets, style_features,
                                      content_targets, content_features,
                                      style_weight, content_weight)

    gradients = tape.gradient(loss, image)

    return gradients
```

```
def update_image_with_style(image, content_targets, style_targets,
                            optimizer, style_weight, content_weight,
                            with_regularization=False):

  gradients = calculate_gradients(image, content_targets,
                                  style_targets, style_weight,
                                  content_weight,with_regularization)

  optimizer.apply_gradients([(gradients, image)]) #Apply gradients on image
```

```
def update_image_with_style(image, content_targets, style_targets,
                            optimizer, style_weight, content_weight,
                            with_regularization=False):

  gradients = calculate_gradients(image, content_targets,
                                  style_targets, style_weight,
                                  content_weight,with_regularization)

  optimizer.apply_gradients([(gradients, image)]) #Apply gradients on image
```

```
def update_image_with_style(image, content_targets, style_targets,
                            optimizer, style_weight, content_weight,
                            with_regularization=False):

  gradients = calculate_gradients(image, content_targets,
                                  style_targets, style_weight,
                                  content_weight,with_regularization)

  optimizer.apply_gradients([(gradients, image)]) #Apply gradients on image
```

```
def fit_style_transfer(input_image, style_image, optimizer,
                       epochs=1, steps_per_epoch=1,
                       with_regularization=False, style_weight = 0.01):


  for n in range(epochs):
    for m in range(steps_per_epoch):
         update_image_with_style(input_image, content_targets,
                                 style_targets, optimizer,
                                 style_weight, content_weight,
                                with_regularization=with_regularization)

  return input_image, images
```

Visualizing Outputs - Final Results

Style Image

https://upload.wikimedia.org/wikipedia/commons/b/b4/Vassily_Kandinsky%2C_1913_-_Composition_7.jpg

Content Image

https://cdn.pixabay.com/photo/2017/02/28/23/00/swan-2107052_1280.jpg

Stylized Image

Style Loss

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

l: layer *l*

$\mathbf{A}^l_{ij}$: Style Representation(Gram Matrix) of style image a.

$\mathbf{G}^l_{ij}$: Style Representation(Gram Matrix) of generated image x

Gram Matrix

$$G_{ij}^{l} = \sum_{k} F_{ik}^{l} F_{jk}^{l}$$

G_{ij}^{l}: inner product between vectorized feature maps i and j in layer l.

http://mlwiki.org/index.php/Gram_Matrices

Style Loss (one layer)

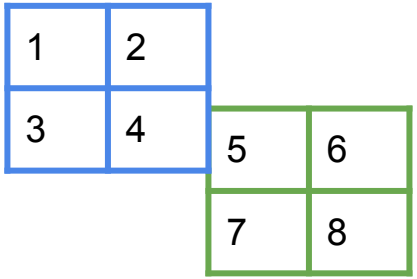
Style layer

H
W
F

Style Loss (one layer)

Style layer

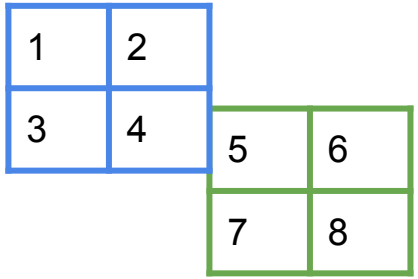
H =2
W = 2
F = 2

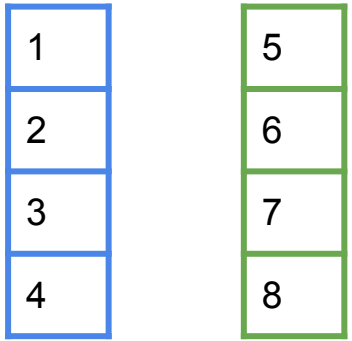


Style Loss (one layer)

Style layer

H =2
W = 2
F = 2

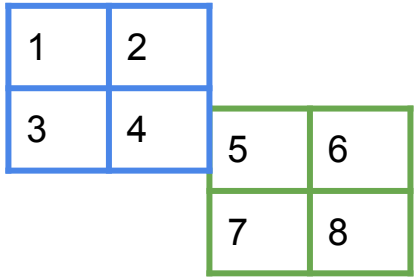




Style Loss (one layer)

Style layer

H =2
W = 2
F = 2



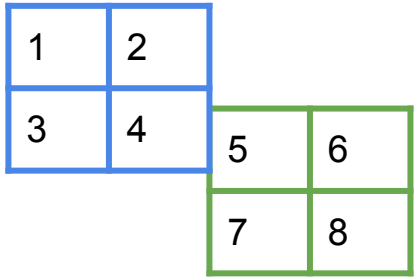
$$A = \begin{bmatrix} 1 & 5 \\ 2 & 6 \\ 3 & 7 \\ 4 & 8 \end{bmatrix}$$

a_1 a_2

Style Loss (one layer)

Style layer

H =2
W = 2
F = 2



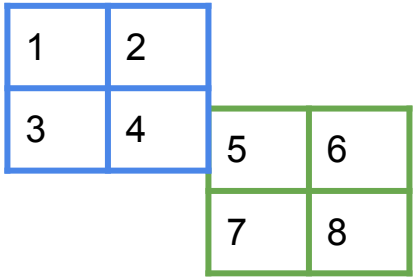
$$A = \left[\begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} \quad \begin{matrix} 5 \\ 6 \\ 7 \\ 8 \end{matrix} \right] \quad G = \left[\quad \right]$$

a_1 a_2

Style Loss (one layer)

Style layer

H =2
W = 2
F = 2



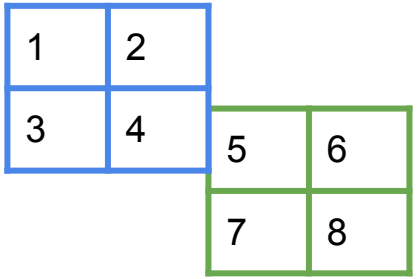
$$A = \left[\begin{array}{cc} 1 & 5 \\ 2 & 6 \\ 3 & 7 \\ 4 & 8 \end{array}\right] \quad G = \left[\begin{array}{cc} a_1 \cdot a_1 & \\ & \end{array}\right]$$

a_1 a_2

Style Loss (one layer)

Style layer

H =2
W = 2
F = 2



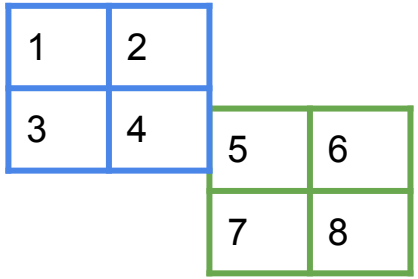
$$A = \begin{bmatrix} 1 & 5 \\ 2 & 6 \\ 3 & 7 \\ 4 & 8 \end{bmatrix} G = \begin{bmatrix} a_1 \cdot a_1 & a_1 \cdot a_2 \end{bmatrix}$$

a_1 a_2

Style Loss (one layer)

Style layer

H =2
W = 2
F = 2



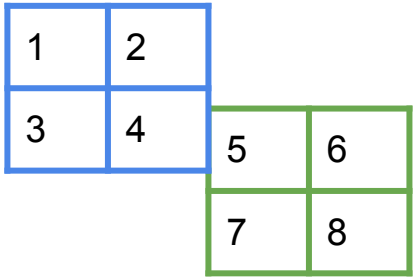
$$A = \begin{bmatrix} 1 & 5 \\ 2 & 6 \\ 3 & 7 \\ 4 & 8 \end{bmatrix} \quad G = \begin{bmatrix} a_1 \cdot a_1 & a_1 \cdot a_2 \\ a_2 \cdot a_1 & \end{bmatrix}$$

a_1 a_2

Style Loss (one layer)

Style layer

H =2
W = 2
F = 2



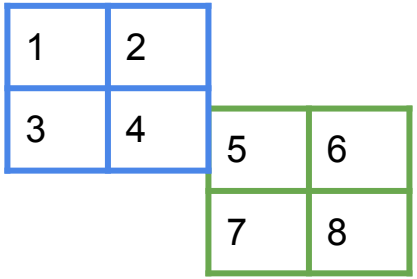
$$A = \begin{bmatrix} 1 & 5 \\ 2 & 6 \\ 3 & 7 \\ 4 & 8 \end{bmatrix} \quad G = \begin{bmatrix} a_1 \cdot a_1 & a_1 \cdot a_2 \\ a_2 \cdot a_1 & a_2 \cdot a_2 \end{bmatrix}$$

a_1 a_2

Style Loss (one layer)

Style layer

H =2
W = 2
F = 2



Gram matrix

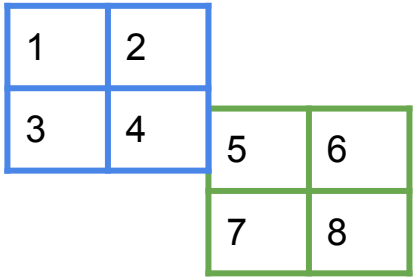
$$A = \begin{bmatrix} 1 & 5 \\ 2 & 6 \\ 3 & 7 \\ 4 & 8 \end{bmatrix} \quad G = \begin{bmatrix} a_1 \cdot a_1 & a_1 \cdot a_2 \\ a_2 \cdot a_1 & a_2 \cdot a_2 \end{bmatrix}$$

a_1 a_2

Style Loss (one layer)

Style layer

H =2
W = 2
F = 2



Gram matrix

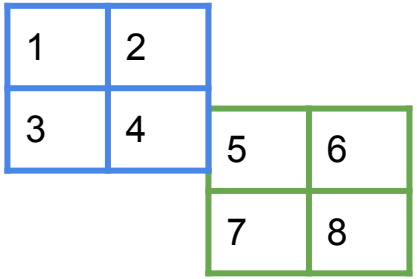
$$A = \begin{bmatrix} 1 & 5 \\ 2 & 6 \\ 3 & 7 \\ 4 & 8 \end{bmatrix} \quad G = \begin{bmatrix} a_1 \cdot a_1 & a_1 \cdot a_2 \\ a_2 \cdot a_1 & a_2 \cdot a_2 \end{bmatrix} = A^T A$$

a_1 a_2

Style Loss (one layer)

Style layer

H =2
W = 2
F = 2



Gram matrix

$$A = \begin{bmatrix} 1 & 5 \\ 2 & 6 \\ 3 & 7 \\ 4 & 8 \end{bmatrix} \quad G = \begin{bmatrix} a_1 \cdot a_1 & a_1 \cdot a_2 \\ a_2 \cdot a_1 & a_2 \cdot a_2 \end{bmatrix} = A^T A$$

a_1 a_2

Style feature

Style Loss (code)

```
style_layer = tf.constant([1,2,3,4,5,6,7,8],
                          shape=(2,2,2))
```

Style Loss (code)

```
style_layer = tf.constant([1,2,3,4,5,6,7,8],
                          shape=(2,2,2))
```

```
[[[1, 2],
  [3, 4]],

 [[5, 6],
  [7, 8]]]
```

Style Loss (code)

```
style_layer = tf.constant([1,2,3,4,5,6,7,8],
                          shape=(2,2,2))
```

```
A = tf.transpose(
        tf.reshape(style_layer,
                   shape=(2,4)))
```

```
[[[1, 2],
  [3, 4]],

 [[5, 6],
  [7, 8]]]
```

Style Loss (code)

```
style_layer = tf.constant([1,2,3,4,5,6,7,8],
                          shape=(2,2,2))


A = tf.transpose(
        tf.reshape(style_layer,
                   shape=(2,4)))
```

```
[[[1, 2],
  [3, 4]],

 [[5, 6],
  [7, 8]]]
```

```
[[1, 5],
 [2, 6],
 [3, 7],
 [4, 8]]
```

Style Loss (code)

```
style_layer = tf.constant([1,2,3,4,5,6,7,8],
                          shape=(2,2,2))


A = tf.transpose(
        tf.reshape(style_layer,
                   shape=(2,4)))
```

```
AT = tf.transpose(A)
```

```
[[[1, 2],
  [3, 4]],

 [[5, 6],
  [7, 8]]]

[[1, 5],
 [2, 6],
 [3, 7],
 [4, 8]]
```

Style Loss (code)

```
style_layer = tf.constant([1,2,3,4,5,6,7,8],
                          shape=(2,2,2))
```

```
[[[1, 2],
  [3, 4]],

 [[5, 6],
  [7, 8]]]
```

```
A = tf.transpose(
        tf.reshape(style_layer,
                   shape=(2,4)))
```

```
[[1, 5],
 [2, 6],
 [3, 7],
 [4, 8]]
```

```
AT = tf.transpose(A)
```

```
[[1, 2, 3, 4],
 [5, 6, 7, 8]]
```

Style Loss (code)

```
style_layer = tf.constant([1,2,3,4,5,6,7,8],
                          shape=(2,2,2))
```

```
[[[1, 2],
  [3, 4]],

 [[5, 6],
  [7, 8]]]
```

```
A = tf.transpose(
        tf.reshape(style_layer,
                   shape=(2,4)))
```

```
[[1, 5],
 [2, 6],
 [3, 7],
 [4, 8]]
```

```
AT = tf.transpose(A)
```

```
[[1, 2, 3, 4],
 [5, 6, 7, 8]]
```

```
G = tf.matmul(AT,A)
```

Style Loss (code)

```
style_layer = tf.constant([1,2,3,4,5,6,7,8],
                          shape=(2,2,2))
```

```
[[[1, 2],
  [3, 4]],

 [[5, 6],
  [7, 8]]]
```

```
A = tf.transpose(
        tf.reshape(style_layer,
                   shape=(2,4)))
```

```
[[1, 5],
 [2, 6],
 [3, 7],
 [4, 8]]
```

```
AT = tf.transpose(A)
```

```
[[1, 2, 3, 4],
 [5, 6, 7, 8]]
```

```
G = tf.matmul(AT,A)
```

```
[[ 30,  70],
 [ 70, 174]]
```

Style Loss (code)

```
style_layer = tf.constant([1,2,3,4,5,6,7,8],
                          shape=(2,2,2))
```

```
[[[1, 2],
  [3, 4]],

 [[5, 6],
  [7, 8]]]
```

```
A = tf.transpose(
        tf.reshape(style_layer,
                   shape=(2,4)))
```

```
[[1, 5],
 [2, 6],
 [3, 7],
 [4, 8]]
```

```
AT = tf.transpose(A)
```

```
[[1, 2, 3, 4],
 [5, 6, 7, 8]]
```

```
G = tf.matmul(AT,A)
```

```
[[ 30,  70],
 [ 70, 174]]
```

```
G = tf.linalg.einsum('cij,dij->cd',
                     style_layer,
                     style_layer)
```

Einstein notation

`tf.linalg.einsum`

Einstein notation

```
x = tf.constant([1,2,3,4],shape=(2,2))
```

Einstein notation

```
x = tf.constant([1,2,3,4],shape=(2,2))
```

```
[1, 2],
[3, 4]
```

Einstein notation

```
x = tf.constant([1,2,3,4],shape=(2,2))
```

```
[1, 2],
[3, 4]
```

```
xT = tf.transpose(x)
```

Einstein notation

```
x = tf.constant([1,2,3,4],shape=(2,2))

 xT = tf.transpose(x)
```

```
[1, 2],
[3, 4]
```

```
[1, 3],
[2, 4]
```

Einstein notation

```
x = tf.constant([1,2,3,4],shape=(2,2))

 xT = tf.transpose(x)

 tf.linalg.einsum('ij,ij->ij', x, x)
```

```
[1, 2],
[3, 4]

[1, 3],
[2, 4]
```

Einstein notation

```
x = tf.constant([1,2,3,4],shape=(2,2))

 xT = tf.transpose(x)

 tf.linalg.einsum('ij,ij->ij', x, x)
```

```
[1, 2],
[3, 4]

[1, 3],
[2, 4]

[ 1,  4],
[ 9, 16]
```

Einstein notation

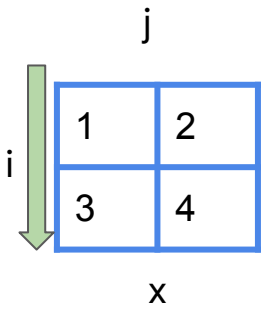
```
x = tf.constant([1,2,3,4],shape=(2,2))

 xT = tf.transpose(x)

 tf.linalg.einsum('ij,ij->ij', x, x)

 tf.square(x)
```

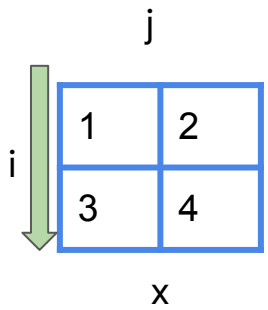
```
[1, 2],
[3, 4]

[1, 3],
[2, 4]

[ 1,  4],
[ 9, 16]
```

Einstein notation

```
x = tf.constant([1,2,3,4],shape=(2,2))

xT = tf.transpose(x)

tf.linalg.einsum('ij,ij->ij', x, x)

tf.square(x)
```

```
[1, 2],
[3, 4]

[1, 3],
[2, 4]

[ 1,  4],
[ 9, 16]
```

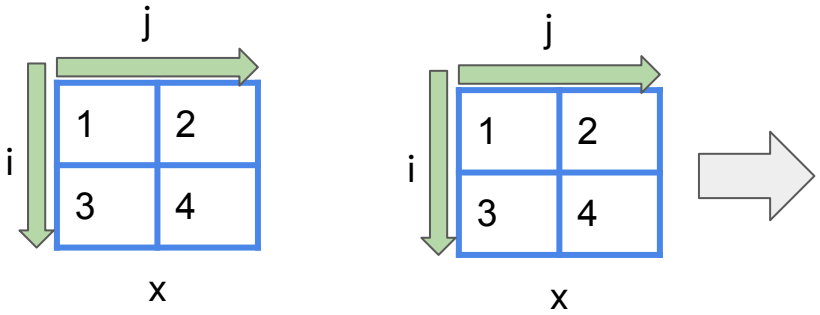




Einstein notation

```
x = tf.constant([1,2,3,4],shape=(2,2))

xT = tf.transpose(x)

tf.linalg.einsum('ij,ij->ij', x, x)

tf.square(x)
```

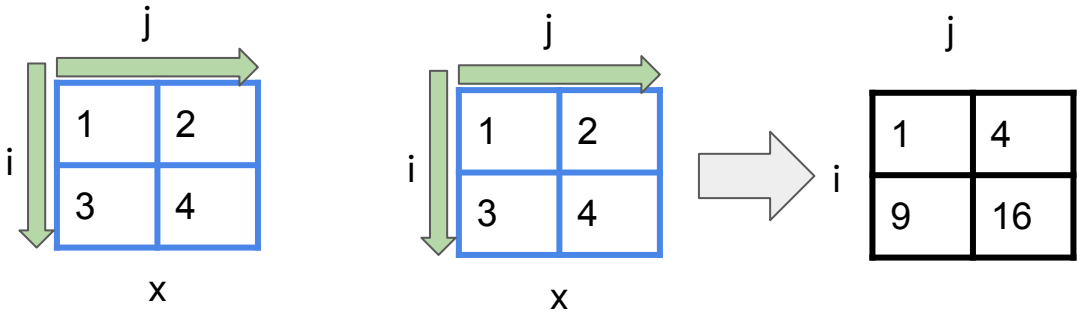
```
[1, 2],
[3, 4]

[1, 3],
[2, 4]

[ 1,  4],
[ 9, 16]
```



Einstein notation

```
x = tf.constant([1,2,3,4],shape=(2,2))

xT = tf.transpose(x)

tf.linalg.einsum('ij,ij->ij', x, x)

tf.square(x)
```

```
[1, 2],
[3, 4]

[1, 3],
[2, 4]

[ 1,  4],
[ 9, 16]
```



Einstein notation

```
x = tf.constant([1,2,3,4],shape=(2,2))

 xT = tf.transpose(x)

 tf.linalg.einsum('ij,ij->ij', x, x)

 tf.square(x)

 tf.linalg.einsum('ij,jk->ik', xT, x)
```

```
[1, 2],
[3, 4]

[1, 3],
[2, 4]

[ 1,  4],
[ 9, 16]
```

Einstein notation

```
x = tf.constant([1,2,3,4],shape=(2,2))
```

```
[1, 2],
[3, 4]
```

```
xT = tf.transpose(x)
```

```
[1, 3],
[2, 4]
```

```
tf.linalg.einsum('ij,ij->ij', x, x)
```

```
[ 1,  4],
[ 9, 16]
```

```
tf.square(x)
```

```
tf.linalg.einsum('ij,jk->ik', xT, x)
```

```
[10, 14],
[14, 20]
```

Einstein notation

```
x = tf.constant([1,2,3,4],shape=(2,2))
```

```
[1, 2],
[3, 4]
```

```
xT = tf.transpose(x)
```

```
[1, 3],
[2, 4]
```

```
tf.linalg.einsum('ij,ij->ij', x, x)
```

```
[ 1,  4],
[ 9, 16]
```

```
tf.square(x)
```

```
tf.linalg.einsum('ij,jk->ik', xT, x)
```

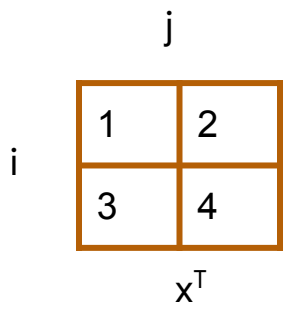
```
[10, 14],
[14, 20]
```

```
tf.matmul(xT, x)
```

Einstein notation

```
x = tf.constant([1,2,3,4],shape=(2,2))

xT = tf.transpose(x)

tf.linalg.einsum('ij,ij->ij', x, x)

tf.square(x)

tf.linalg.einsum('ij,jk->ik', xT, x)

tf.matmul(xT, x)
```

```
[1, 2],
[3, 4]

[1, 3],
[2, 4]

[ 1,  4],
[ 9, 16]

[10, 14],
[14, 20]
```



Einstein notation

```
x = tf.constant([1,2,3,4],shape=(2,2))
```

```
[1, 2],
[3, 4]
```

```
xT = tf.transpose(x)
```

```
[1, 3],
[2, 4]
```

```
tf.linalg.einsum('ij,ij->ij', x, x)
```

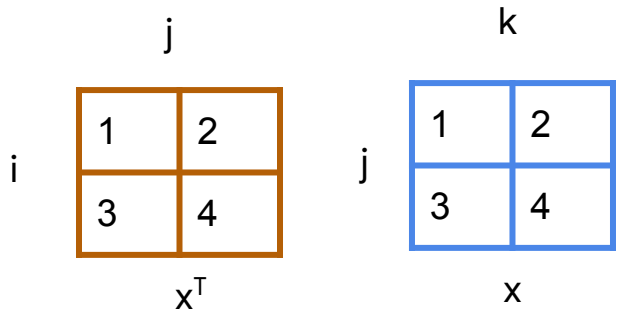
```
[ 1,  4],
[ 9, 16]
```

```
tf.square(x)
```

```
tf.linalg.einsum('ij,jk->ik', xT, x)
```

```
[10, 14],
[14, 20]
```

```
tf.matmul(xT, x)
```



Einstein notation

```
x = tf.constant([1,2,3,4],shape=(2,2))
```

```
[1, 2],
[3, 4]
```

```
xT = tf.transpose(x)
```

```
[1, 3],
[2, 4]
```

```
tf.linalg.einsum('ij,ij->ij', x, x)
```

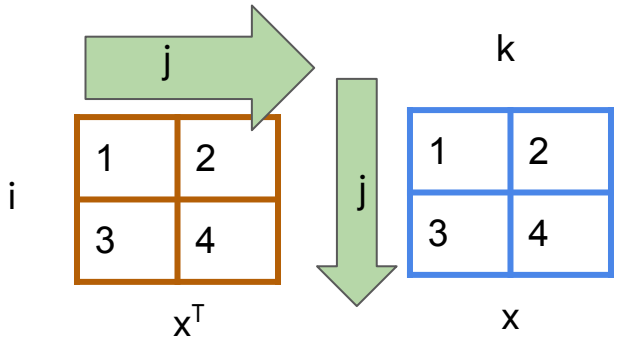
```
[ 1,  4],
[ 9, 16]
```

```
tf.square(x)
```

```
tf.linalg.einsum('ij,jk->ik', xT, x)
```

```
[10, 14],
[14, 20]
```

```
tf.matmul(xT, x)
```



Einstein notation

```
x = tf.constant([1,2,3,4],shape=(2,2))
```

```
[1, 2],
[3, 4]
```

```
xT = tf.transpose(x)
```

```
[1, 3],
[2, 4]
```

```
tf.linalg.einsum('ij,ij->ij', x, x)
```

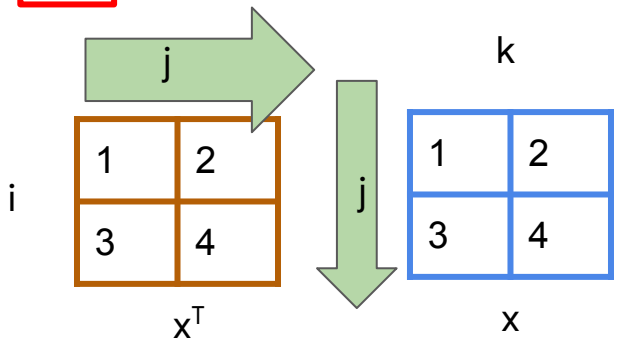
```
[ 1,  4],
[ 9, 16]
```

```
tf.square(x)
```

```
tf.linalg.einsum('ij,jk->ik', xT, x)
```

```
[10, 14],
[14, 20]
```

```
tf.matmul(xT, x)
```



Einstein notation

```
x = tf.constant([1,2,3,4],shape=(2,2))
```

```
[1, 2],
[3, 4]
```

```
xT = tf.transpose(x)
```

```
[1, 3],
[2, 4]
```

```
tf.linalg.einsum('ij,ij->ij', x, x)
```

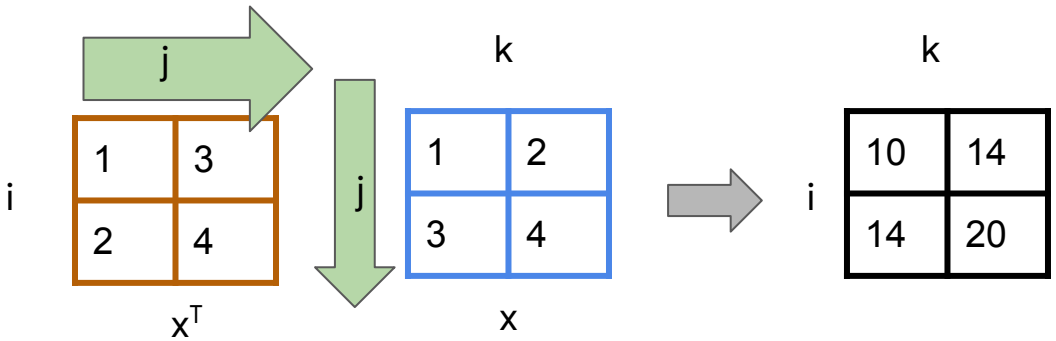
```
[ 1,  4],
[ 9, 16]
```

```
tf.square(x)
```

```
tf.linalg.einsum('ij,jk->ik', xT, x)
```

```
[10, 14],
[14, 20]
```

```
tf.matmul(xT, x)
```



Reduce sum

Reduce sum

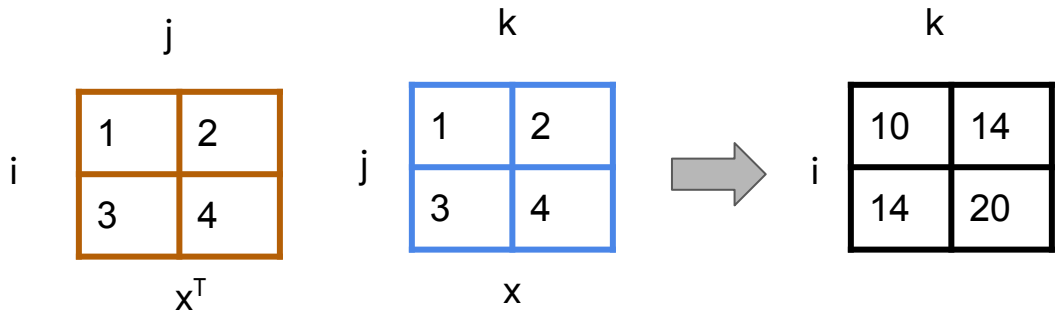
```
tf.linalg.einsum('ij,jk->ik , xT, x)
```

```
[10, 14],
[14, 20]
```

Reduce sum

```
tf.linalg.einsum('ij,jk->ik', xT, x)
```

```
[10, 14],
[14, 20]
```

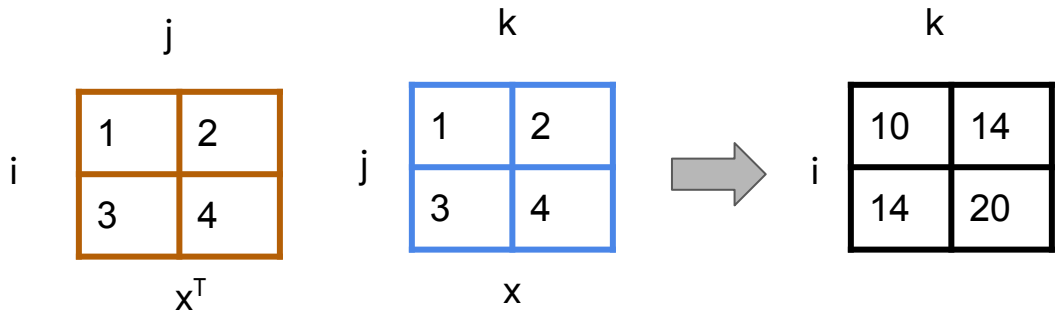



```
tf.linalg.einsum('ij,jk->', xT, x)
```

Reduce sum

```
tf.linalg.einsum('ij,jk->ik', xT, x)
```

```
[10, 14],
[14, 20]
```

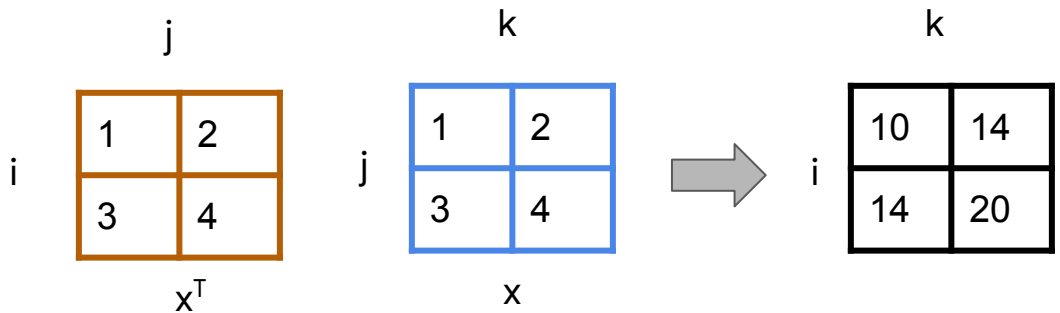



```
tf.linalg.einsum('ij,jk->', xT, x)
```

Reduce sum

```
tf.linalg.einsum('ij,jk->ik', xT, x)
```

```
[10, 14],
[14, 20]
```



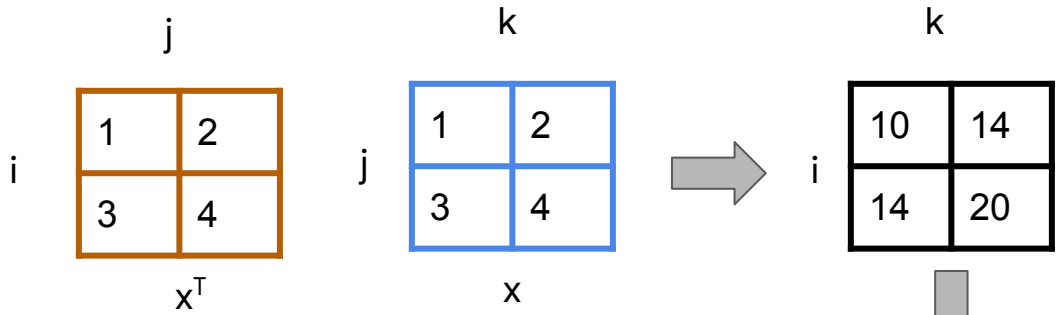

```
tf.linalg.einsum('ij,jk->', xT, x)
```

```
[58]
```

Reduce sum

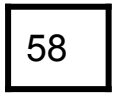
```
tf.linalg.einsum('ij,jk->ik', xT, x)
```

```
[10, 14],
[14, 20]
```




```
tf.linalg.einsum('ij,jk->', xT, x)
```

```
[58]
```



Einstein notation

```
x = tf.constant([1,2,3,4],shape=(2,2))
```

```
[1, 2],
[3, 4]
```

```
 v = tf.reshape(x, shape=(4,1) )
```

```
[1,
 2,
 3,
 4]
```

```
vT = tf.transpose(v)
```

```
[1,2,3,4]
```

```
tf.matmul(vT,v)
```

```
[30]
```

```
tf.linalg.einsum('ij,ij->', x, x)
```

```
[30]
```

Style Loss

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

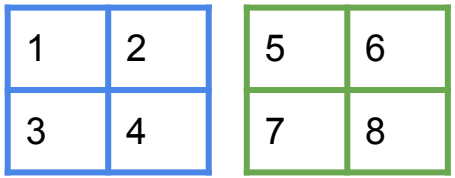
l: layer *l*

$\mathbf{A}^l_{ij}$: Style Representation(Gram Matrix) of style image a.

$\mathbf{G}^l_{ij}$: Style Representation(Gram Matrix) of generated image x)

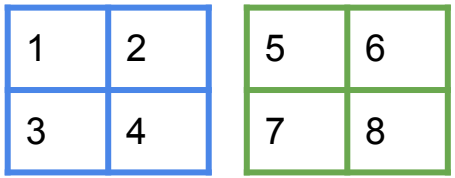
Weighting the style loss

Style layer (height, width, filters)



Weighting the style loss

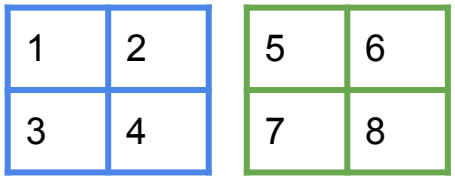
Style layer (height, width, filters)



Height =2
Width = 2
Filters = 2

Weighting the style loss

Style layer (height, width, filters)



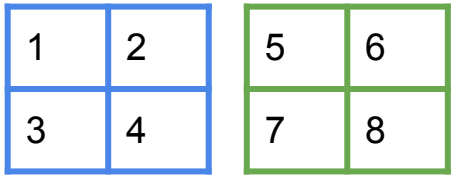
Height =2
Width = 2
Filters = 2

Style weight

$$\frac{1}{(2 \times h \times w \times f)^2}$$

Weighting the style loss

Style layer (height, width, filters)



Height =2
Width = 2
Filters = 2

Style weight

$$\frac{1}{(2 \times h \times w \times f)^2}$$

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

Gram Matrix - In Practice

```
def gram_matrix(input_tensor):
  input_tensorT = tf.transpose(input_tensor,perm[0,2,1,3])
  result = tf.linalg.einsum('bijc,bijd->bcd' input_tensorT, input_tensor)
  input_shape = tf.shape(input_tensor)
  num_locations = tf.cast(input_shape[1]*input_shape[2], tf.float32)
  return result/(num_locations)
```

https://numpy.org/doc/stable/reference/generated/numpy.einsum.html

Gram Matrix - In Practice

(batch, height, width, filters)

```
def gram_matrix(input_tensor):
  input_tensorT = tf.transpose(input_tensor,perm[0,2,1,3])
  result = tf.linalg.einsum('bijc,bijd->bcd', input_tensorT, input_tensor)
  input_shape = tf.shape(input_tensor)
  num_locations = tf.cast(input_shape[1]*input_shape[2], tf.float32)
  return result/(num_locations)
```

Gram Matrix - In Practice

(batch, height, width, filters) (batch, height, width, filters)

```
def gram_matrix(input_tensor):
  input_tensorT = tf.transpose(input_tensor,perm[0,2,1,3])
  result = tf.linalg.einsum('bijc,bijd->bcd', input_tensorT, input_tensor)
  input_shape = tf.shape(input_tensor)
  num_locations = tf.cast(input_shape[1]*input_shape[2], tf.float32)
  return result/(num_locations)
```

Gram Matrix - In Practice

(batch, height, width, filters) (batch, height, width, filters)

b b

```
def gram_matrix(input_tensor):
  input_tensorT = tf.transpose(input_tensor,perm[0,2,1,3])
  result = tf.linalg.einsum('bijc,bijd->bcd', input_tensorT, input_tensor)
  input_shape = tf.shape(input_tensor)
  num_locations = tf.cast(input_shape[1]*input_shape[2], tf.float32)
  return result/(num_locations)
```

Gram Matrix - In Practice

(batch, height, width, filters) (batch, height, width, filters)

b i j b

```
def gram_matrix(input_tensor):
  input_tensorT = tf.transpose(input_tensor,perm[0,2,1,3])
  result = tf.linalg.einsum('bijc,bijd->bcd', input_tensorT, input_tensor)
  input_shape = tf.shape(input_tensor)
  num_locations = tf.cast(input_shape[1]*input_shape[2], tf.float32)
  return result/(num_locations)
```

Gram Matrix - In Practice

(batch, height, width, filters) (batch, height, width, filters)

b i j b i j

```
def gram_matrix(input_tensor):
  input_tensorT = tf.transpose(input_tensor,perm[0,2,1,3])
  result = tf.linalg.einsum('bijc,bijd->bcd', input_tensorT, input_tensor)
  input_shape = tf.shape(input_tensor)
  num_locations = tf.cast(input_shape[1]*input_shape[2], tf.float32)
  return result/(num_locations)
```

Gram Matrix - In Practice

(batch, height, width, filters) (batch, height, width, filters)

b i j c b i j

```
def gram_matrix(input_tensor):
  input_tensorT = tf.transpose(input_tensor,perm[0,2,1,3])
  result = tf.linalg.einsum('bijc,bijd->bcd', input_tensorT, input_tensor)
  input_shape = tf.shape(input_tensor)
  num_locations = tf.cast(input_shape[1]*input_shape[2], tf.float32)
  return result/(num_locations)
```

Gram Matrix - In Practice

(batch, height, width, filters) (batch, height, width, filters)

b i j c b i j d

```
def gram_matrix(input_tensor):
  input_tensorT = tf.transpose(input_tensor,perm[0,2,1,3])
  result = tf.linalg.einsum('bijc,bijd->bcd', input_tensorT, input_tensor)
  input_shape = tf.shape(input_tensor)
  num_locations = tf.cast(input_shape[1]*input_shape[2], tf.float32)
  return result/(num_locations)
```

Gram Matrix - In Practice

(batch, height, width, filters) (batch, height, width, filters) (batch, filters, filters)

b i j c b i j d b c d

```
def gram_matrix(input_tensor):
  input_tensorT = tf.transpose(input_tensor,perm[0,2,1,3])
  result = tf.linalg.einsum('bijc,bijd->bcd', input_tensorT, input_tensor)
  input_shape = tf.shape(input_tensor)
  num_locations = tf.cast(input_shape[1]*input_shape[2], tf.float32)
  return result/(num_locations)
```

Gram Matrix - In Practice

(batch, height, width, filters) (batch, height, width, filters) (batch, filters, filters)

b i j c b i j d b c d

```
def gram_matrix(input_tensor):
  input_tensorT = tf.transpose(input_tensor,perm[0,2,1,3])
  result = tf.linalg.einsum('bijc,bijd->bcd', input_tensorT, input_tensor)
  input_shape = tf.shape(input_tensor)
  num_locations = tf.cast(input_shape[1]*input_shape[2], tf.float32)
  return result/(num_locations)
```

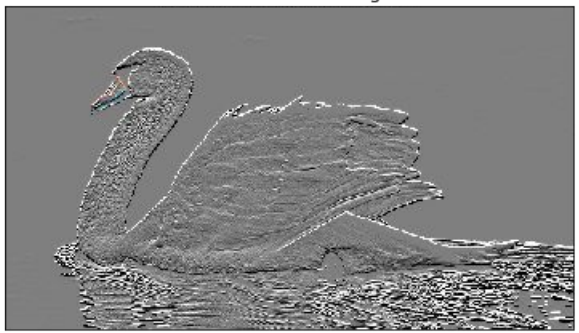
https://numpy.org/doc/stable/reference/generated/numpy.einsum.html

Extract High Frequency Components

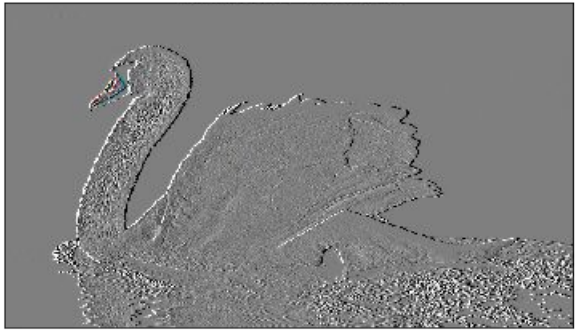
```
def high_pass_x_y(image):
  x_var = image[:,:,1:,:] - image[:,:,:-1,:]
  y_var = image[:,1:,:,:] - image[:,:-1,:,:]

  return x_var, y_var
```

Visualization-High Frequency Artifacts

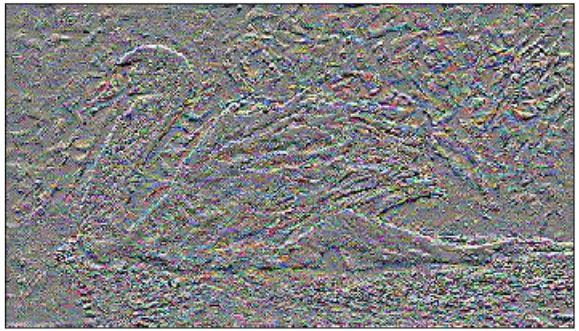
Horizontal Deltas: Original



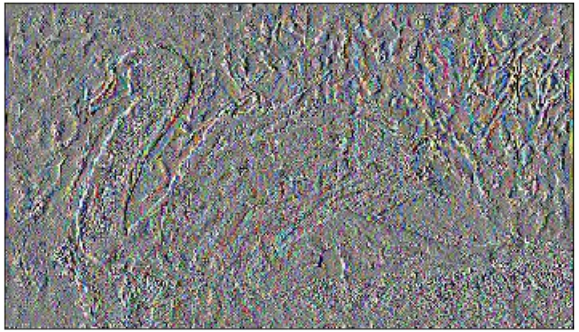
Vertical Deltas: Original



Horizontal Deltas: Stylized Image



Vertical Deltas: Stylized Image



Solution - Total Variation Loss

- Decrease high frequency artefacts.
- Explicit regularization term on high frequency components

Factoring in Total Variation using TensorFlow

```
loss += total_variation_weight * tf.image.total_variation(image)
```

```
def calculate_gradients(image, content_targets,
                        style_targets, style_weight,
                        content_weight,with_regularization=False ):

    total_variation_weight = 30
    with tf.GradientTape() as tape:
      if with_regularization:
        loss += total_variation_weight*tf.image.total_variation(image)

    gradients = tape.gradient(loss, image)
    return gradients
```

```
def calculate_gradients(image, content_targets,
                        style_targets, style_weight,
                        content_weight,with_regularization=False ):

    total_variation_weight = 30
    with tf.GradientTape() as tape:
      if with_regularization:
        loss += total_variation_weight*tf.image.total_variation(image)

    gradients = tape.gradient(loss, image)
    return gradients
```

Comparison

Without Variation Loss

With Variation Loss

https://cdn.pixabay.com/photo/2017/02/28/23/00/swan-2107052_1280.jpg

https://arxiv.org/pdf/1705.06830.pdf

Exploring the structure of a real-time, arbitrary neural artistic stylization network

Golnaz Ghiasi
Google Brain
golnazg@google.com

Honglak Lee
Google Brain
honglak@google.com

Manjunath Kudlur
Google Brain
keveman@google.com

Vincent Dumoulin
MILA, Université de Montréal
vi.dumoulin@gmail.com

Jonathon Shlens
Google Brain
shlens@google.com

August 28, 2017

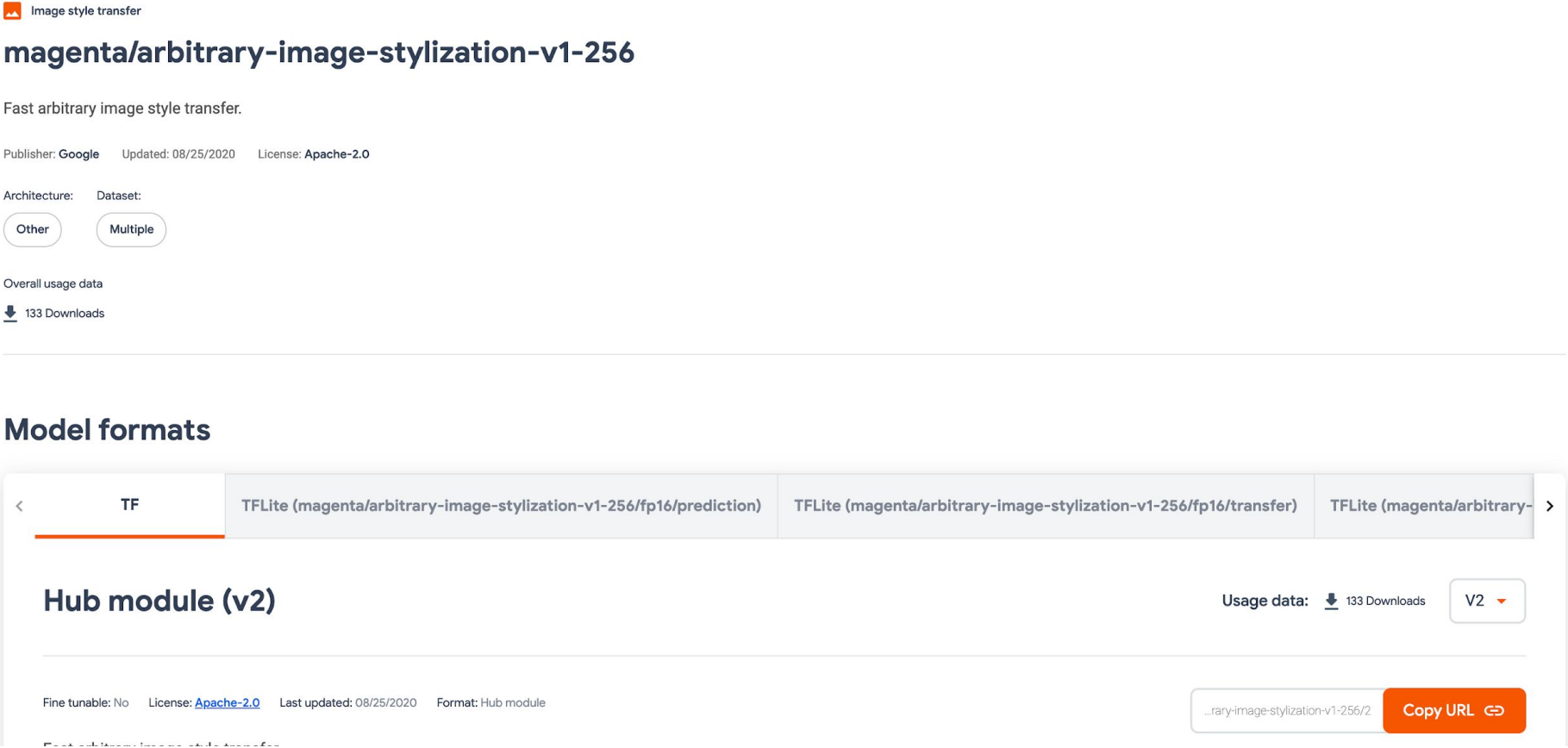
Abstract

In this paper, we present a method which combines the flexibility of the neural algorithm of artistic style with the speed of fast style transfer networks to allow real-time stylization using any content/style image pair. We build upon recent work leveraging conditional instance normalization for multi-style transfer networks by learning to predict the conditional instance normalization parameters directly from a style image. The model is successfully trained on a corpus of roughly 80,000 paintings and is able to generalize to paintings previously unobserved. We demonstrate that the learned embedding space is smooth and contains a rich structure and organizes semantic information associated with paintings in an entirely unsupervised manner.

1 Introduction

Elmyr de Hory gained world-wide fame by forging thousands of pieces of artwork and selling them to art dealers and museums [13]. The forger's skill is a testament to the human talent and intelligence required to reproduce the artistic details of a diverse set of paintings. In computer vision, much work has been invested in teaching computers to likewise capture the artistic style of a painting with the goal of conferring this style in arbitrary photographs in a convincing manner.

https://tfhub.dev/google/magenta/arbitrary-image-stylization-v1-256/2



```
import tensorflow_hub as hub

h_m = hub.load
        ('https://tfhub.dev/google/magenta/arbitrary-image-stylization-v1-256/2')
```

```
import tensorflow_hub as hub

h_m = hub.load
        ('https://tfhub.dev/google/magenta/arbitrary-image-stylization-v1-256/2')


content_image, style_image = load_images("content.jpg", "style.jpg")
```

```
import tensorflow_hub as hub

h_m = hub.load
        ('https://tfhub.dev/google/magenta/arbitrary-image-stylization-v1-256/2')

content_image, style_image = load_images("content.jpg", "style.jpg")

stylized_image = h_m(tf.image.convert_image_dtype(content_image, tf.float32),
                     tf.image.convert_image_dtype(style_image, tf.float32))[0]
```

```
import tensorflow_hub as hub

h_m = hub.load
        ('https://tfhub.dev/google/magenta/arbitrary-image-stylization-v1-256/2')

content_image, style_image = load_images("content.jpg", "style.jpg")

stylized_image = h_m(tf.image.convert_image_dtype(content_image, tf.float32),
                     tf.image.convert_image_dtype(style_image, tf.float32))[0]

tensor_to_image(stylized_image)
```