AI VIETNAM
All-in-One Course

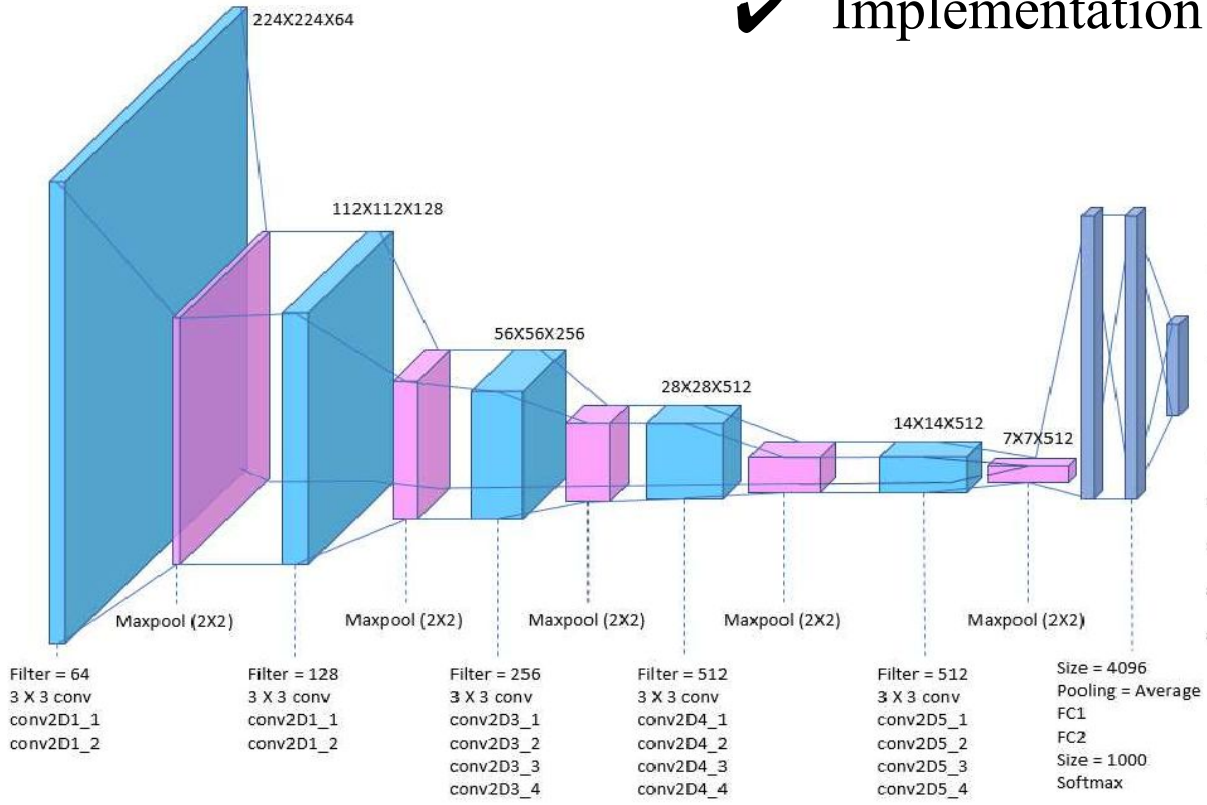
Neural Style Transfer
(TA Session)

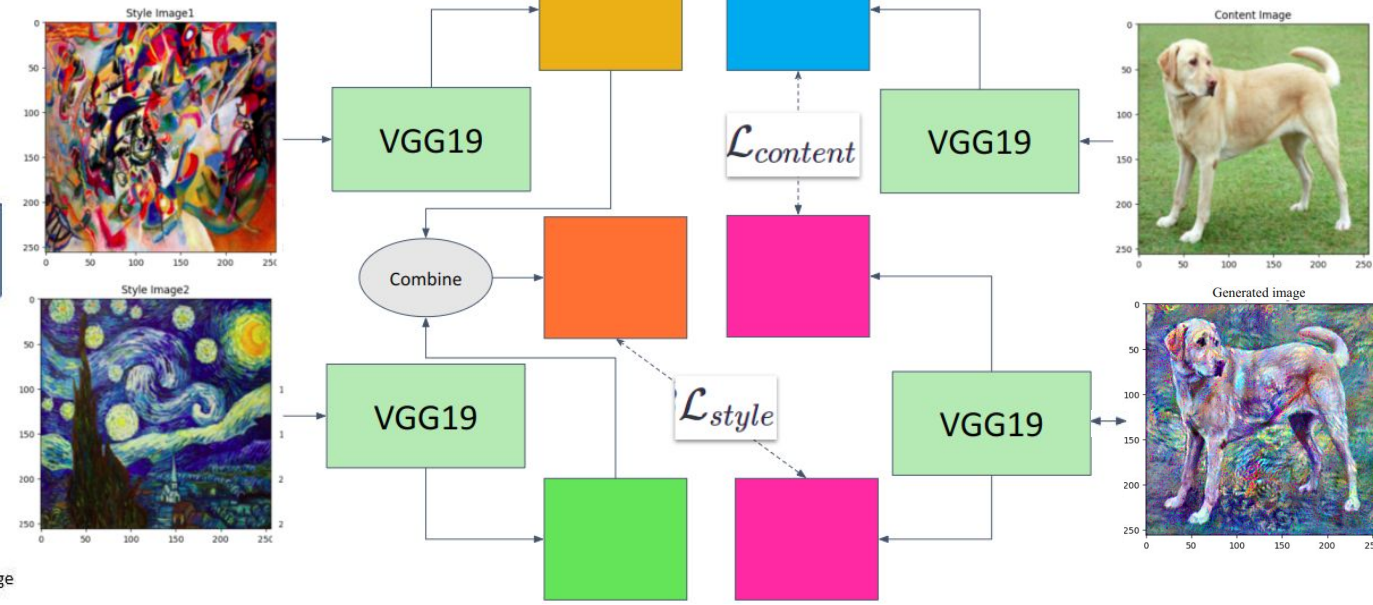
Khoa Nguyen
Ph.D. Student

Year 2024

Objectives

- ✔ Review Style Transfer (Definition, …)
- ✔ Review requisite knowledge
- ✔ Introduce Style Transfer Examples
- ✔ Introduce Milestone Papers
- ✔ Implementation of Style Transfer with 2 Style Images
- ✔ Implementation of Style Transfer with Rotation + DE





Outline

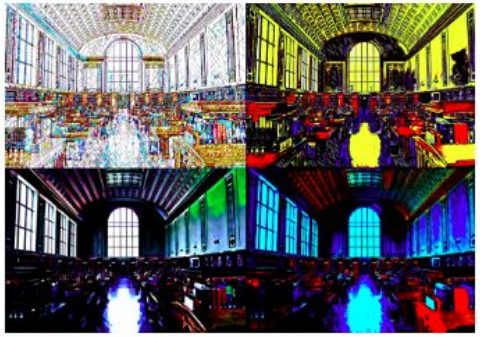
- Introduction
- Basic Knowledge
- Style Transfer Example
- Milestone Papers
- Style Transfer with 2 Style Images
- Style Transfer with Rotation + DE

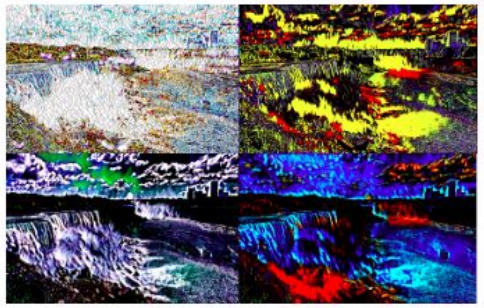
Introduction

❖ Definition

Introduction

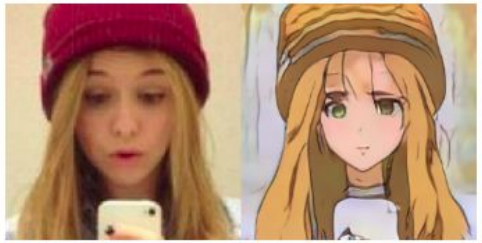











Introduction

❖ Definition

Neural Style Transfer is a fascinating technique in the field of **computer vision** and **artificial intelligence** that allows the **style of one image** to be **applied** to the **content of another image**, effectively blending the two. This is achieved through the use of Convolutional Neural Networks (CNNs), which are a class of deep neural networks commonly used in analyzing visual imagery

Content Image: This is the primary image to which you want to apply a new style. It provides the structure and content.

Style Image: This image provides the artistic style you wish to apply to the content image. It could be a famous painting or any other image with a distinctive artistic style.

Generated Image: This is the output image that starts as a copy of the content image and iteratively transforms to match the style of the style image while retaining the content of the original content image.

Introduction

❖ Definition


Content Style

Generated image


Content Style

Generated image

AI VIETNAM
All-in-One Course

Basic Knowledge

Basic Knowledge

❖ **Generate Image**

Content

Style

Initialize Random Image

- I : CxHxW

Use gradient descent to minimize the loss function

Basic Knowledge

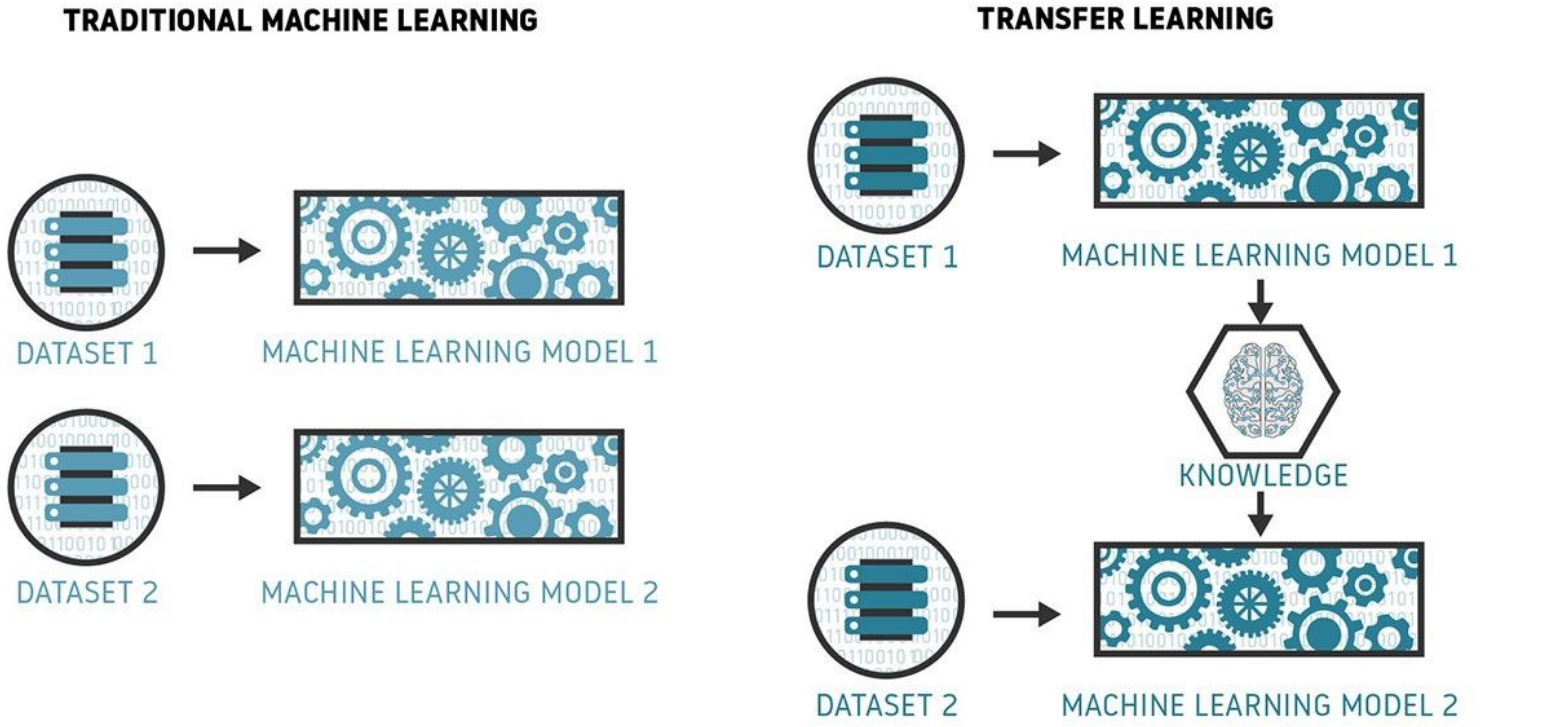
❖ Neural Style Transfer

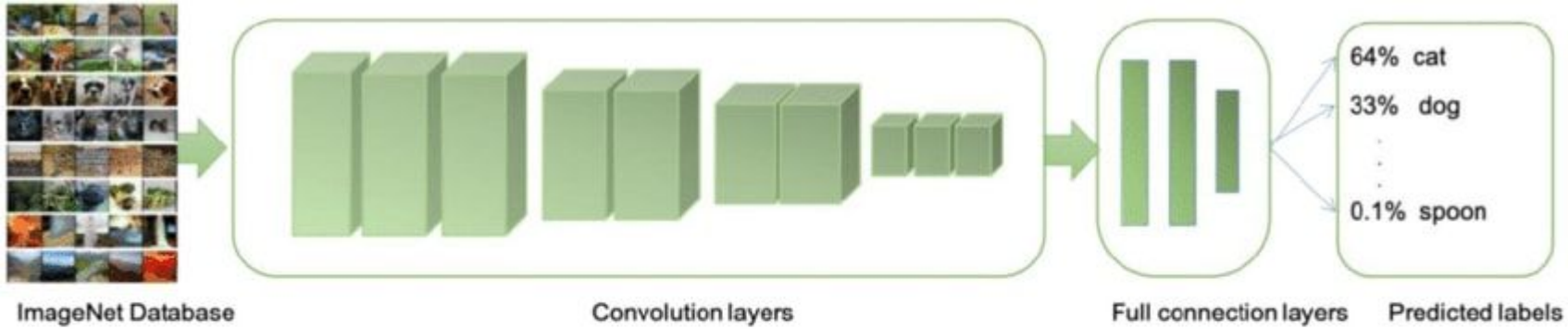
$$\mathcal{L}_{total} = \alpha \mathcal{L}_{content} + \beta \mathcal{L}_{style}$$

using CNN representation (VGG)

Basic Knowledge

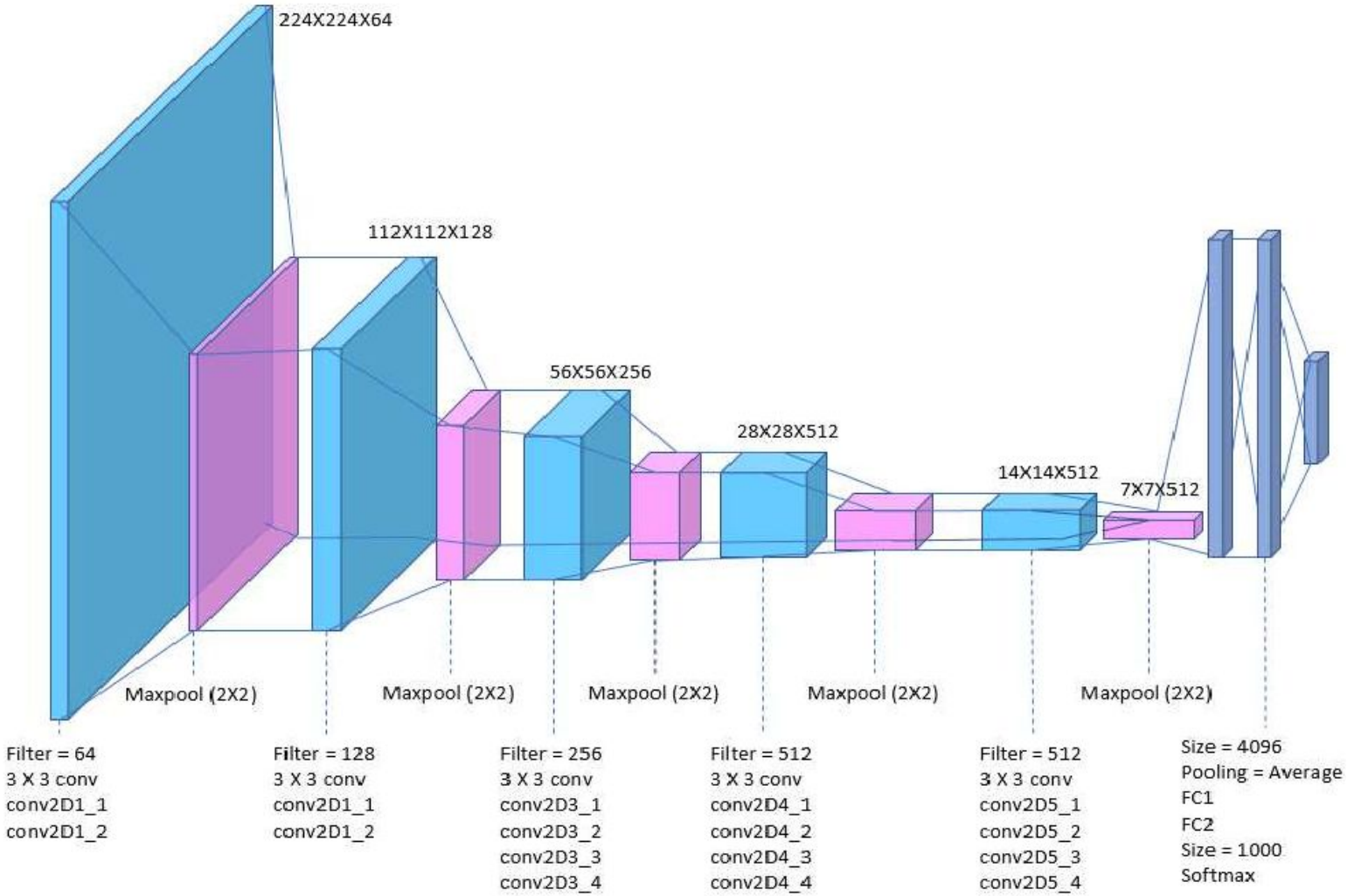
❖ Pre-trained model





Basic Knowledge

VGG



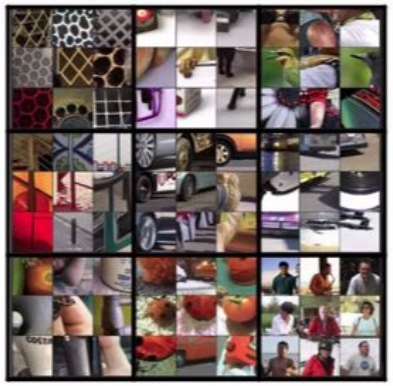
VGG16	VGG19
Sofmax	Sofmax
FC 1000	FC 1000
FC 4096	FC 4096
FC 4096	FC 4096
Pool	Pool
3×3 conv,512	3×3 conv,512
3×3 conv,512	3×3 conv,512
3×3 conv,512	3×3 conv,512
Pool	Pool
3×3 conv,512	3×3 conv,512
3×3 conv,512	3×3 conv,512
3×3 conv,512	3×3 conv,512
Pool	Pool
3×3 conv,512	3×3 conv,512
3×3 conv,512	3×3 conv,512
3×3 conv,512	3×3 conv,512
Pool	Pool
3×3 conv,128	3×3 conv,128
3×3 conv,128	3×3 conv,128
Pool	Pool
3×3 conv,64	3×3 conv,64
3×3 conv,64	3×3 conv,64
Input	Input

Basic Knowledge

❖ Visualizing deep layers: Layer 1

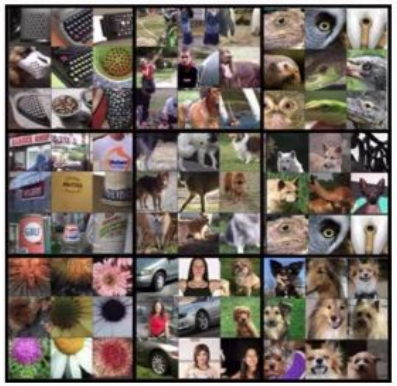
Layer 1

Layer 2

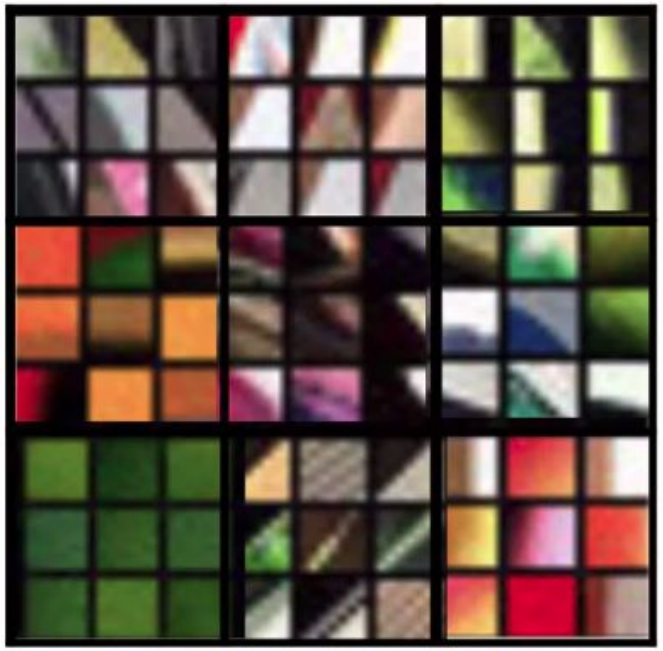


Layer 3

Layer 4



Layer 5

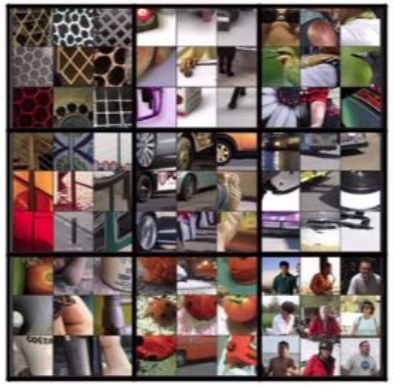


Basic Knowledge

❖ Visualizing deep layers: Layer 2

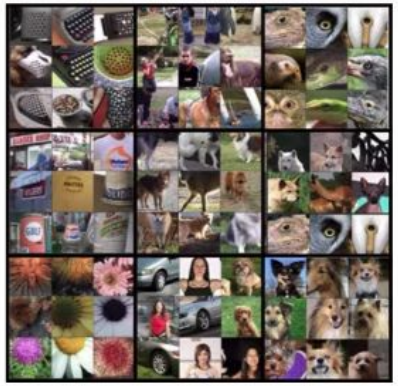
Layer 1

Layer 2

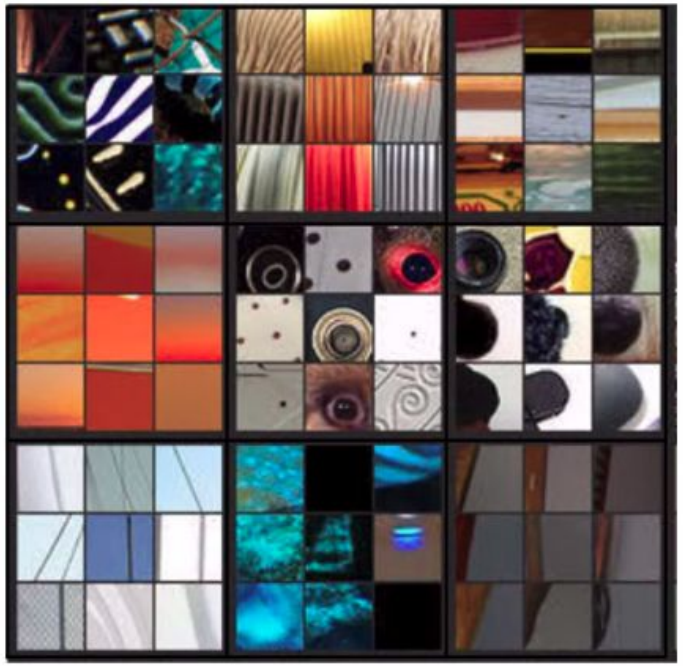


Layer 3

Layer 4



Layer 5

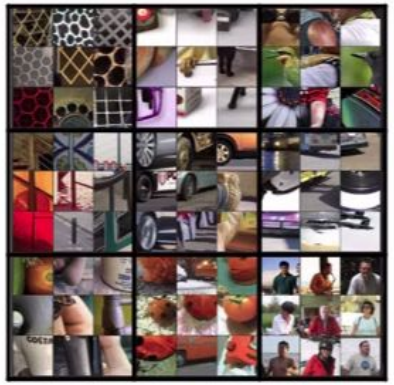


Basic Knowledge

❖ Visualizing deep layers: Layer 3

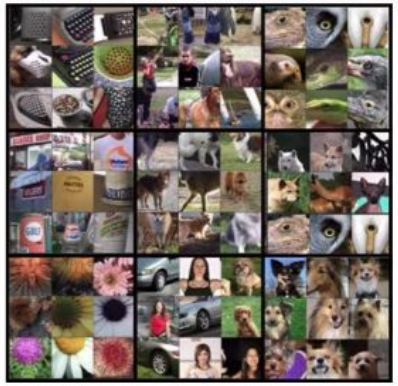
Layer 1

Layer 2

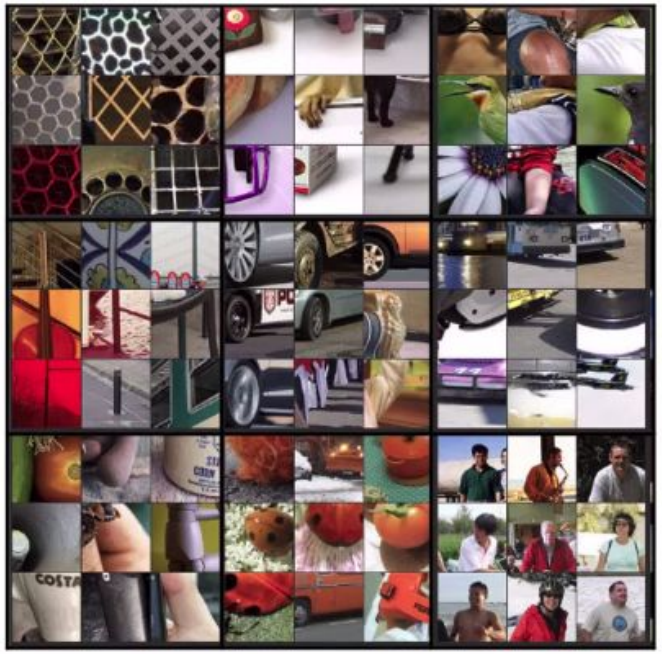


Layer 3

Layer 4



Layer 5

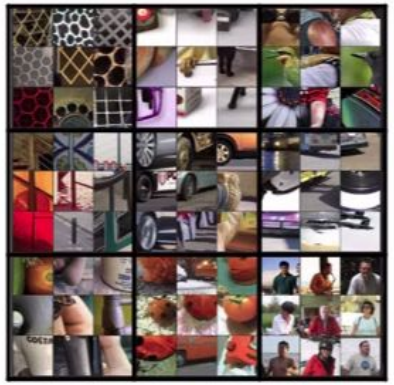


Basic Knowledge

❖ Visualizing deep layers: Layer 4

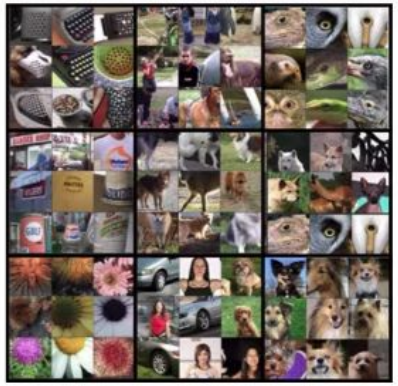
Layer 1

Layer 2

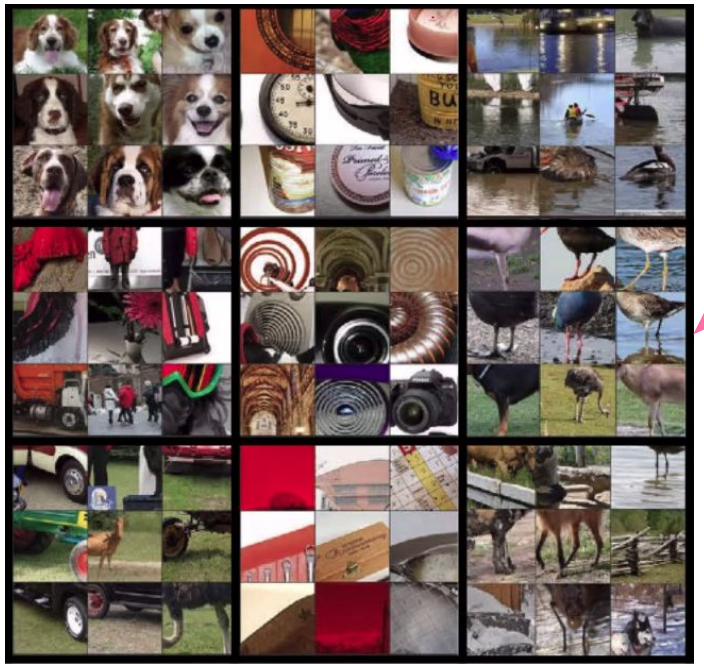


Layer 3

Layer 4



Layer 5

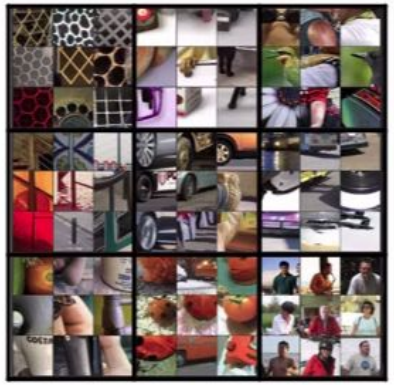


Basic Knowledge

❖ Visualizing deep layers: Layer 5

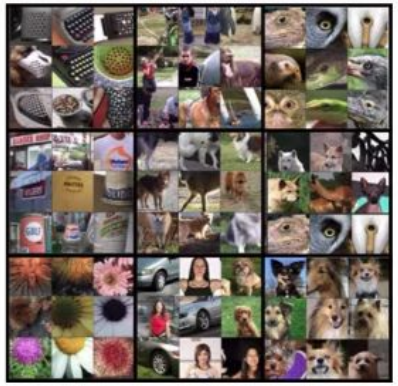
Layer 1

Layer 2

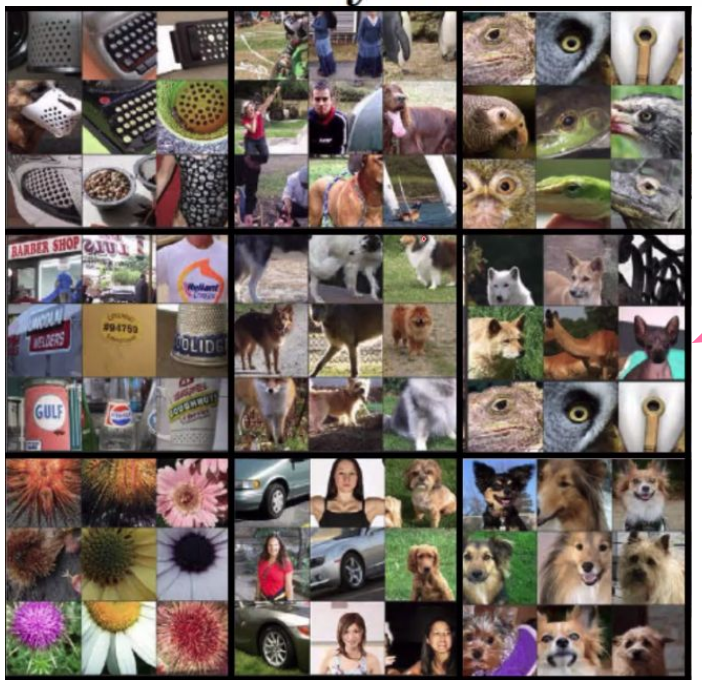


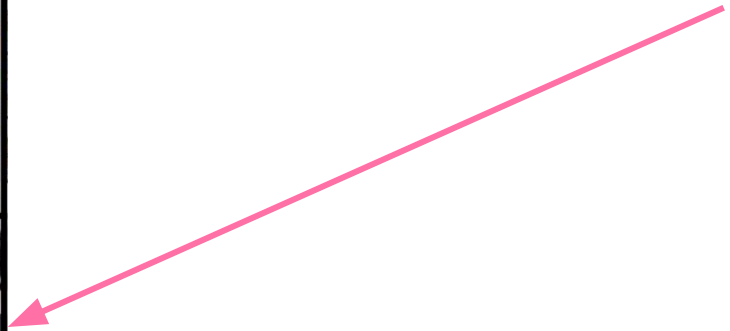
Layer 3

Layer 4



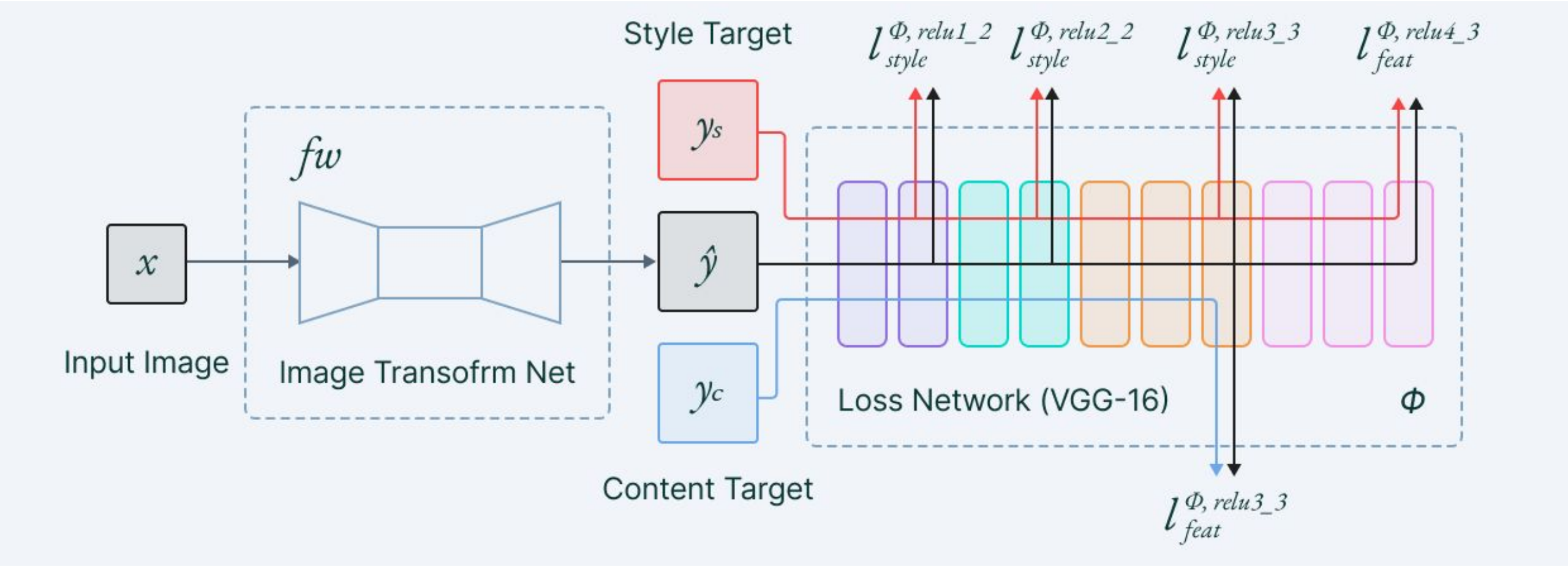
Layer 5





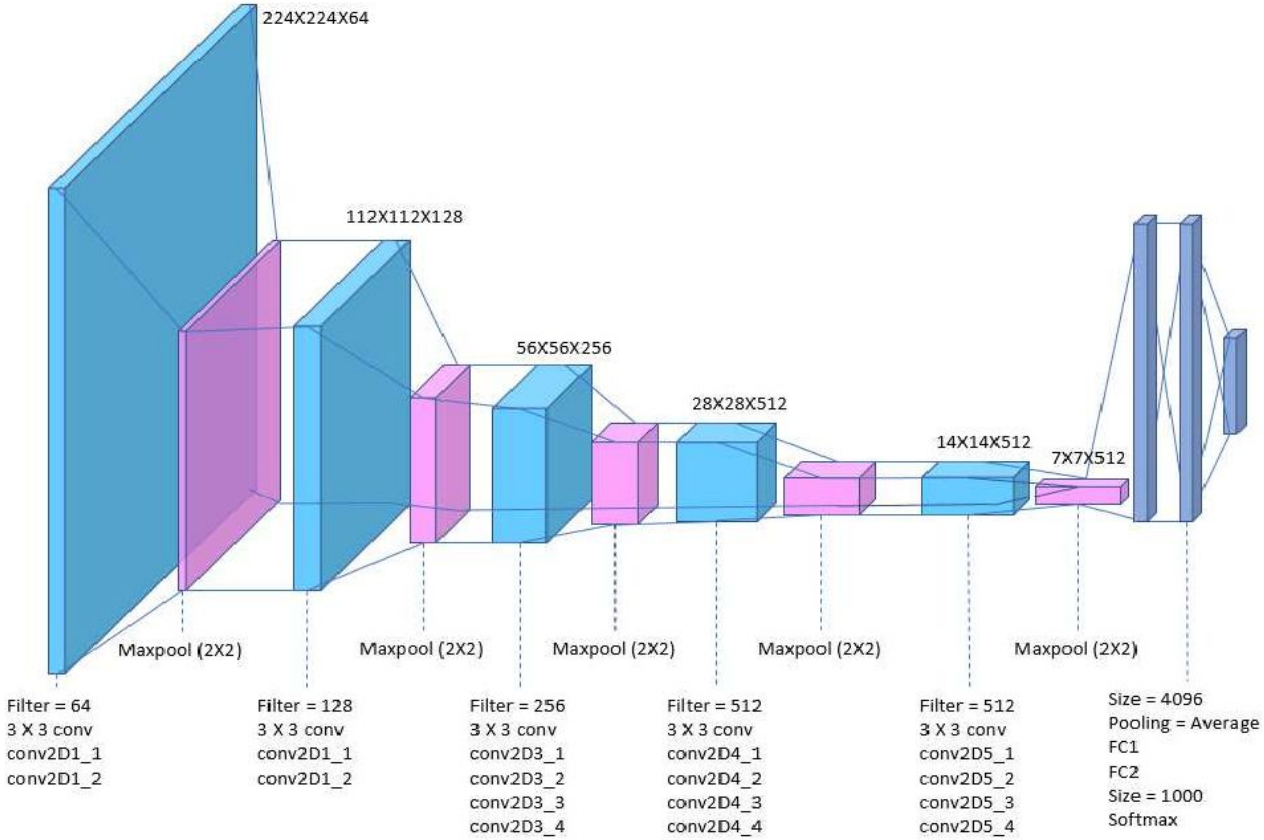
Basic Knowledge

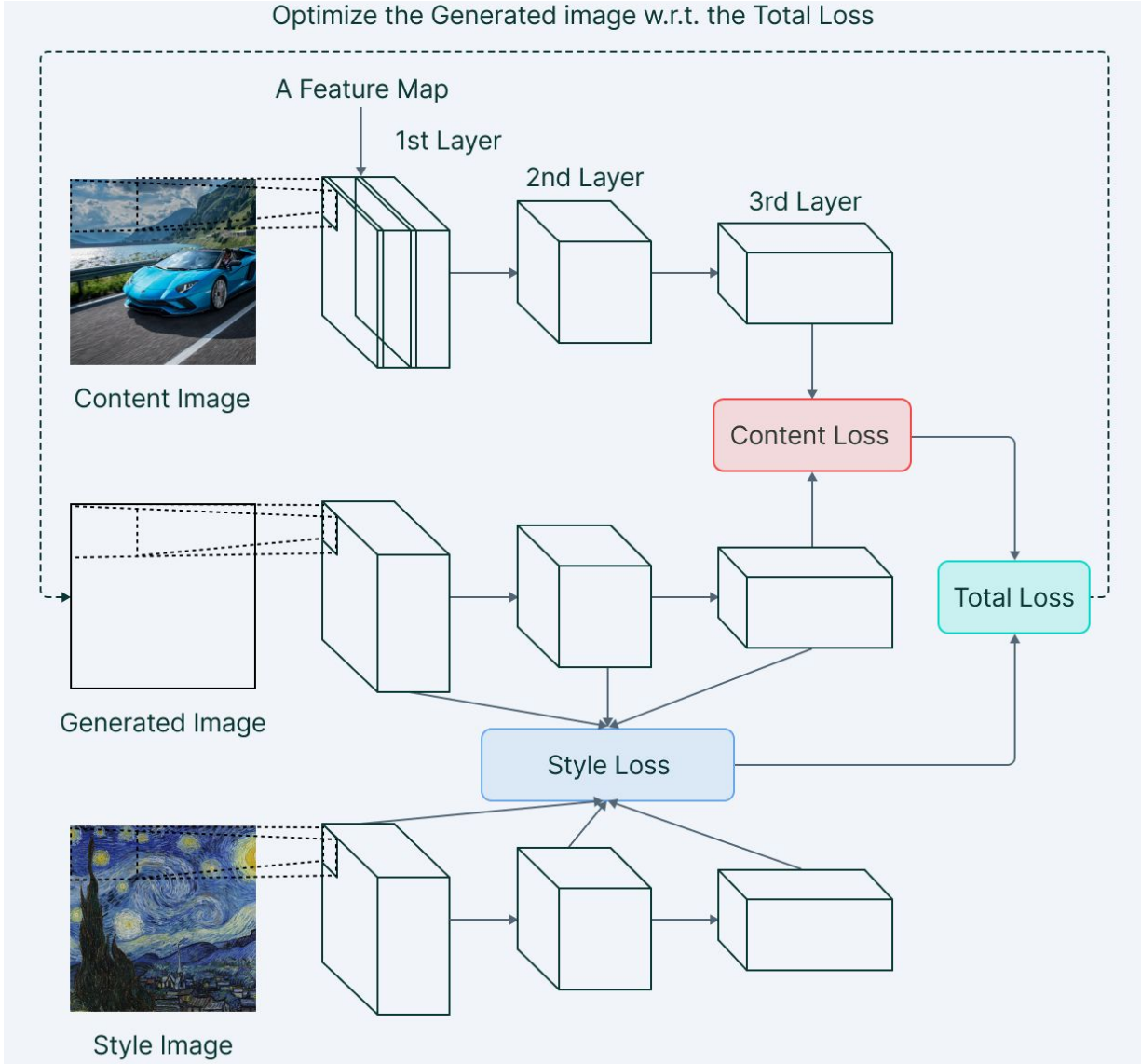
❖ Neural Style Transfer basic structure



Basic Knowledge

❖ Neural Style Transfer basic structure

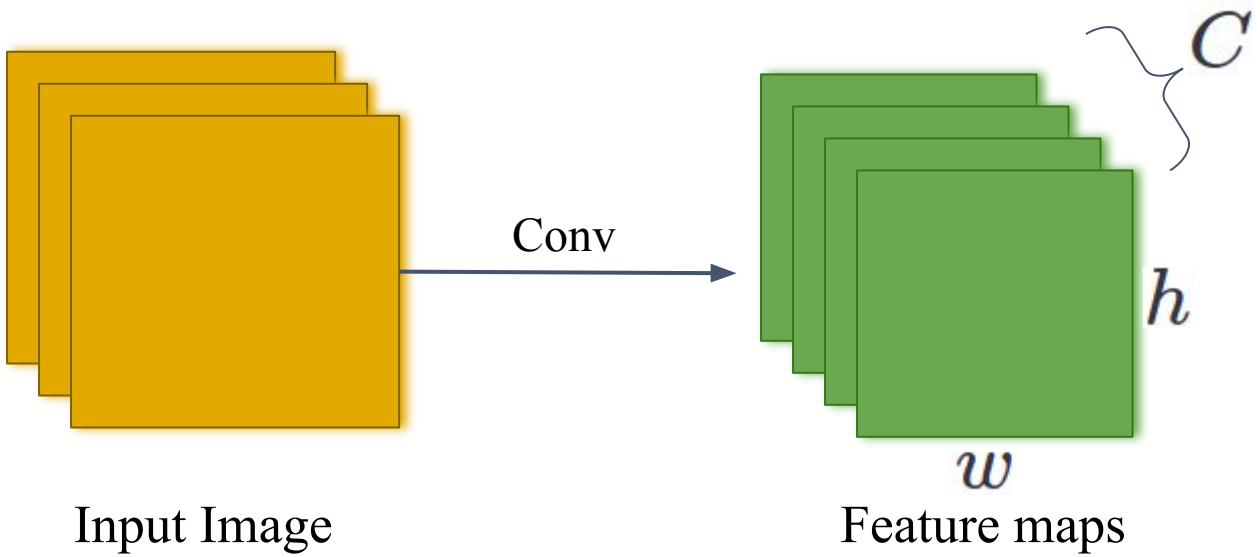




Basic Knowledge

❖ Content Loss

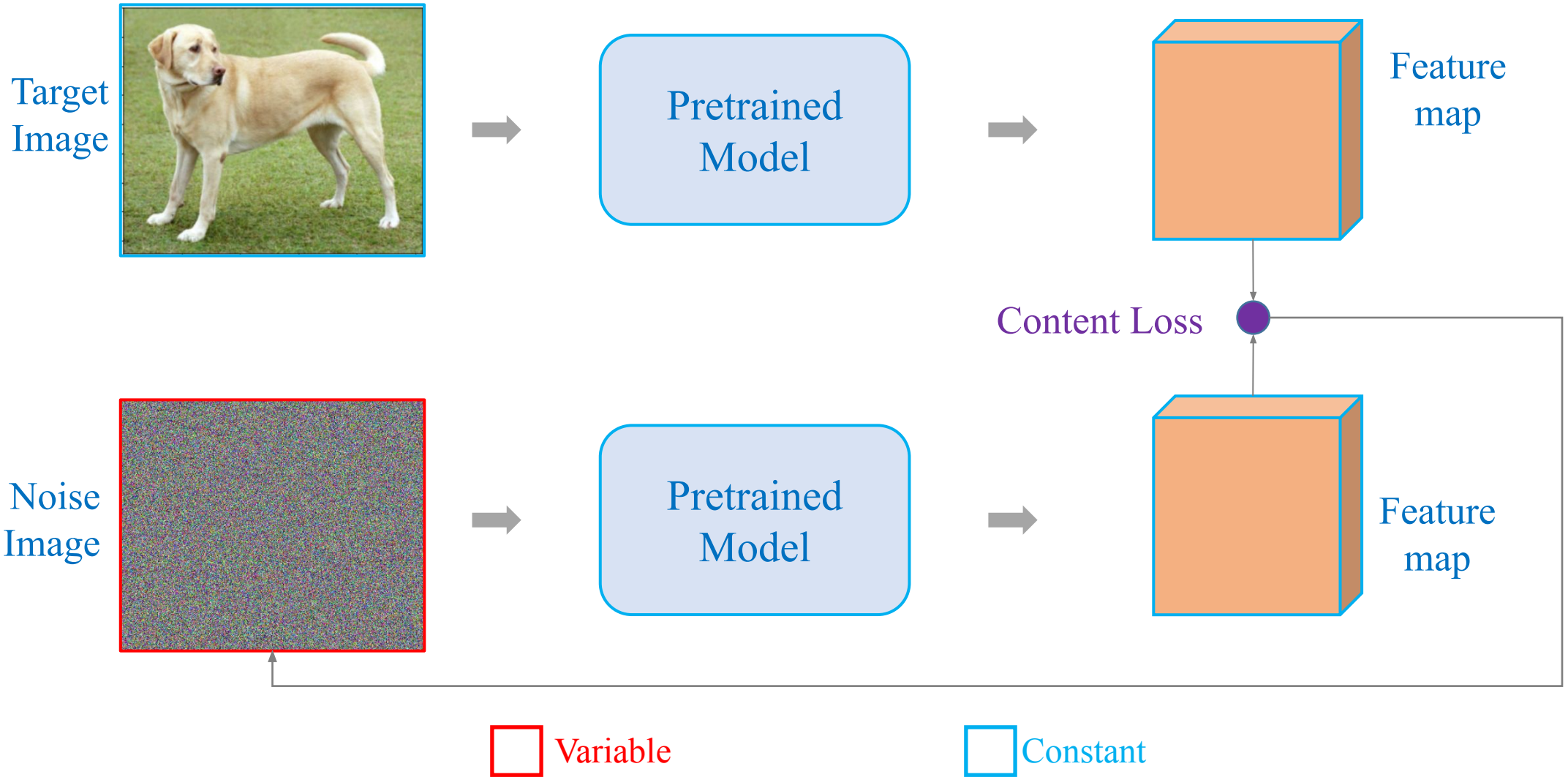
- Layer l
- Number of filters C
- Generated image x
- Generated feature map F
- Content image p
- Content feature map P



$$\mathcal{L}_{content}(p, x, l) = \frac{1}{2} \sum_{i,j,k} (F^l_{ijk} - P^l_{ijk})^2$$

Basic Knowledge

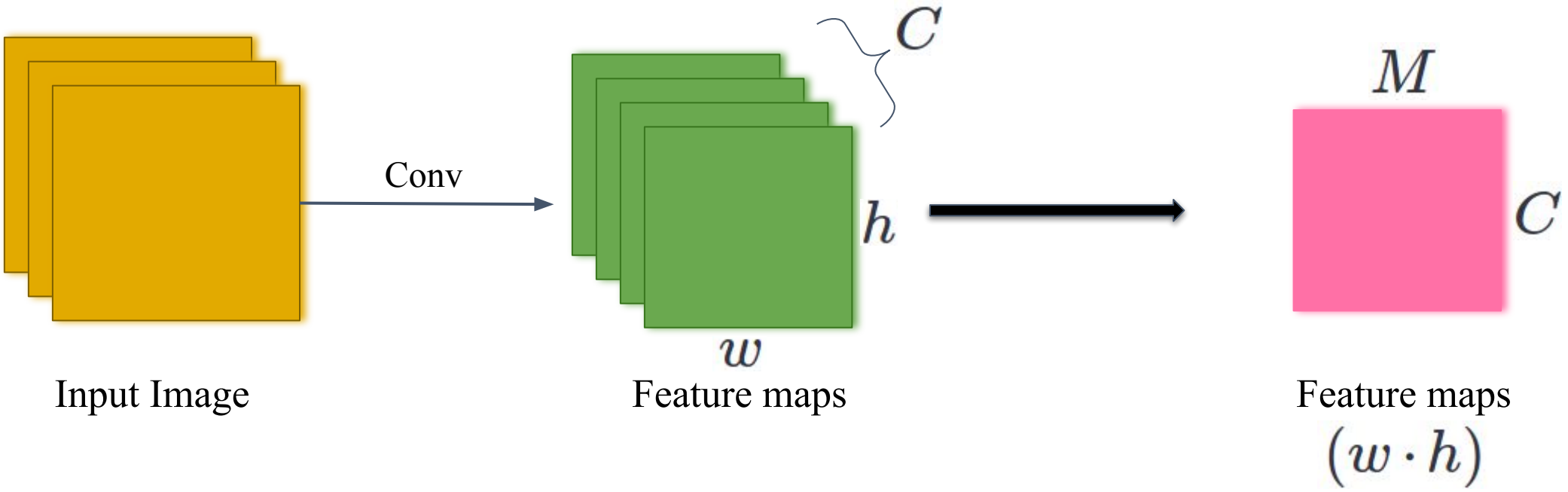
❖ Content Loss



Basic Knowledge

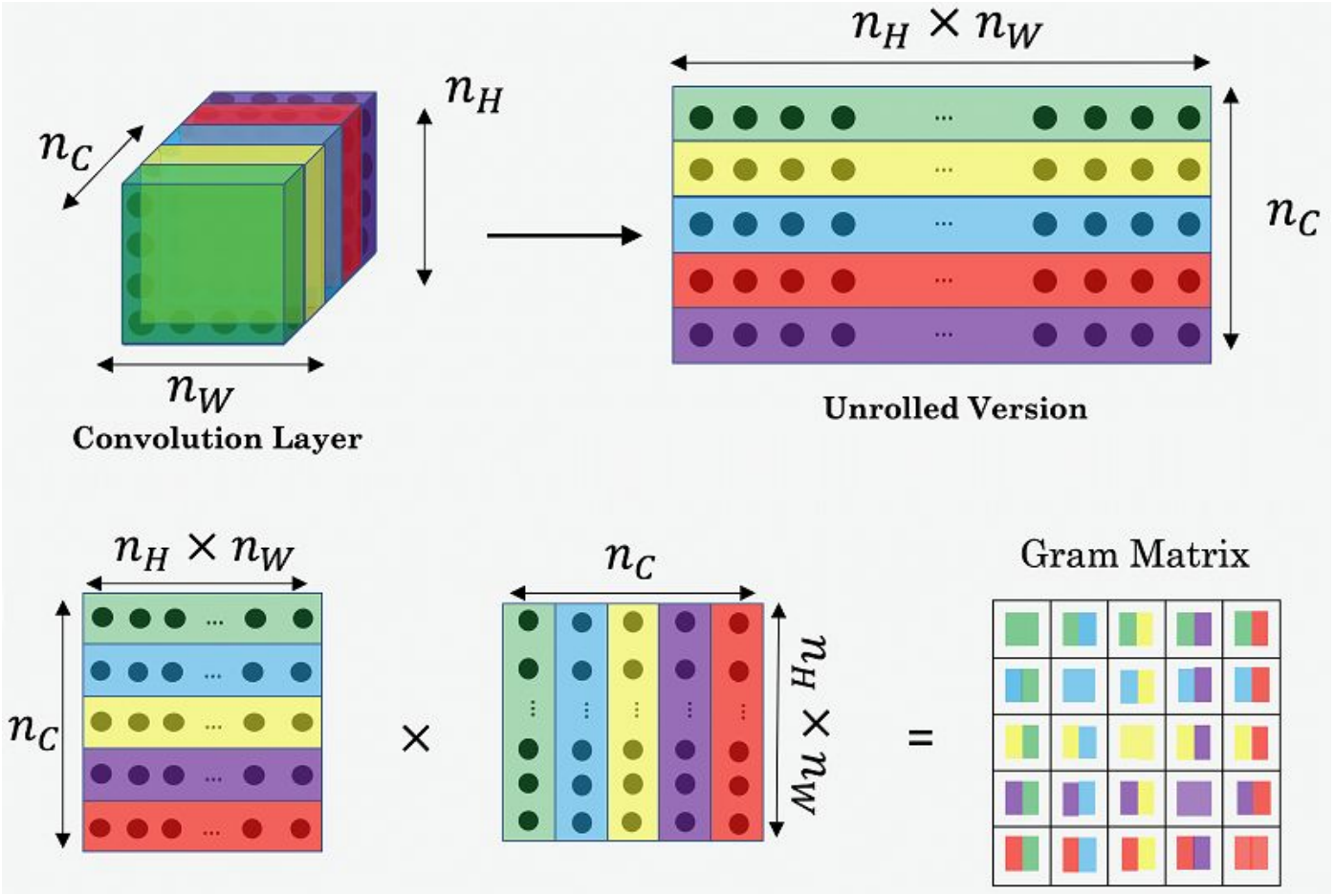
❖ Style Loss

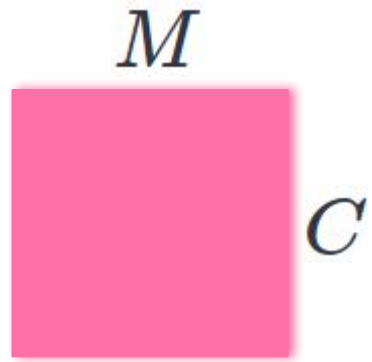
- Layer l
- Number of filters C
- Feature map size $M\ (w \cdot h)$
- Gram Matrix G



Basic Knowledge

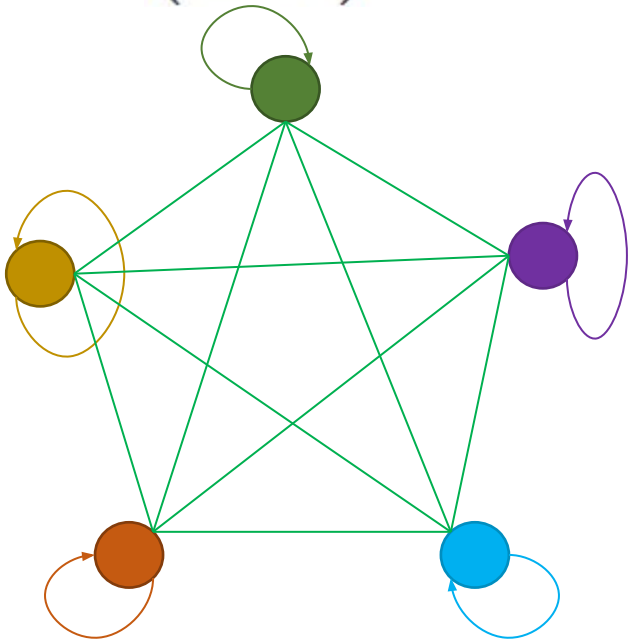
❖ Gram Matrix





Feature maps

$(w \cdot h)$

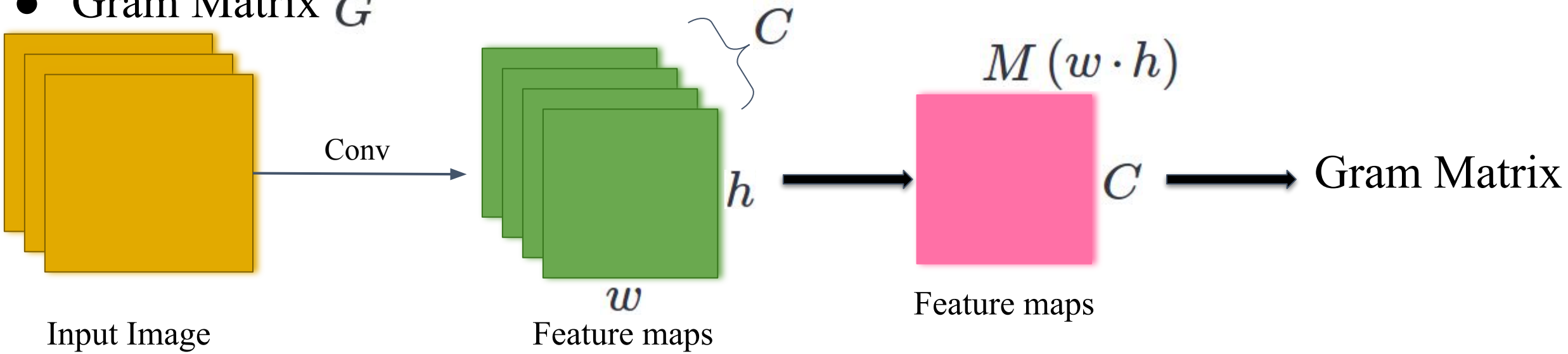


Basic Knowledge

❖ Style Loss

- Layer l
- Number of filters C
- Feature map size $M\ (w \cdot h)$
- Gram Matrix G

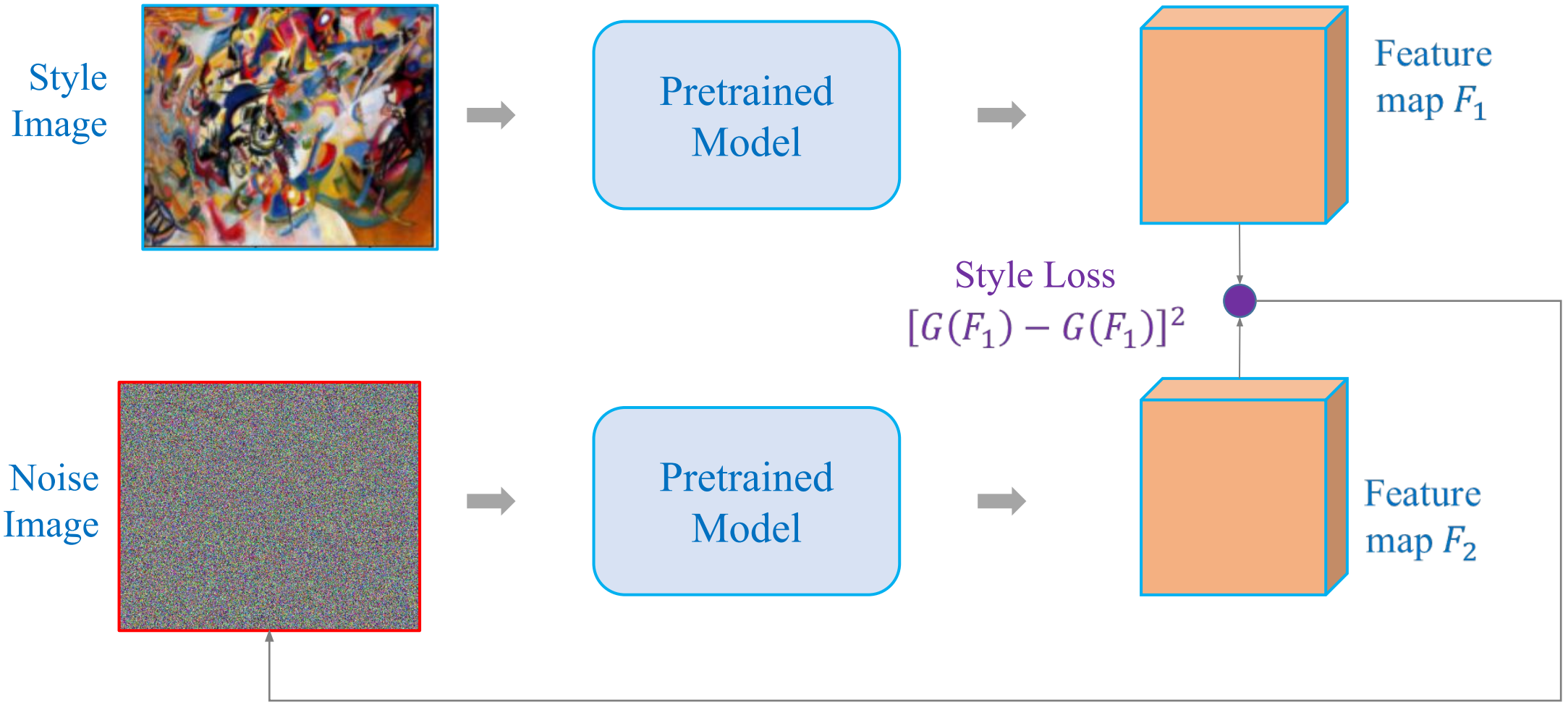
- Generated image x
- Generated feature map F
- Style image s
- Style feature map S



$$\mathcal{L}_{style}(s, x, l) = \frac{1}{2} \sum_{m,n} (G(F)^l_{mn} - G(S)^l_{mn})^2$$

Basic Knowledge

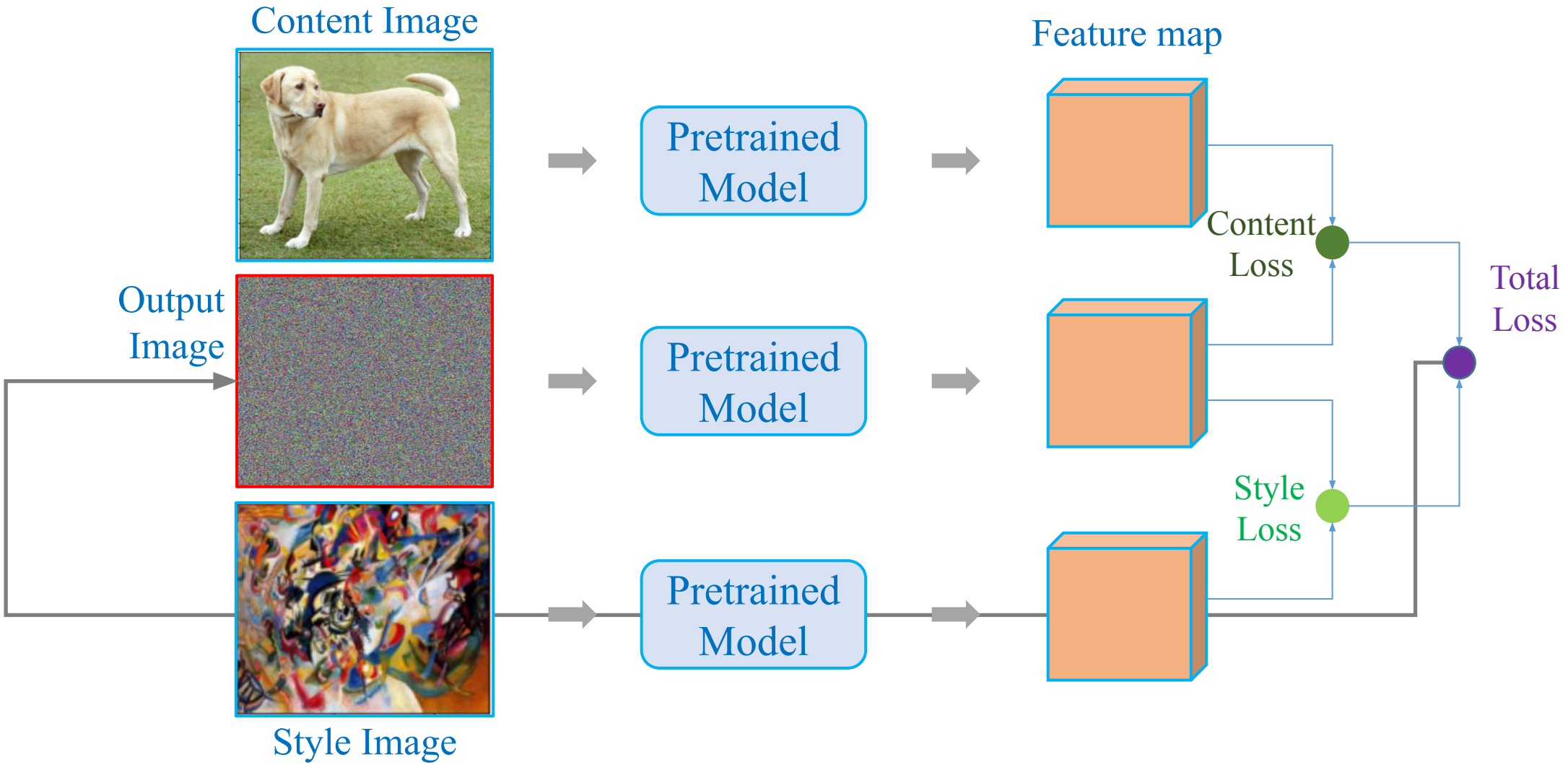
❖ Style Loss



Basic Knowledge

❖ Total Loss

$$\mathcal{L}_{total}(p, s, x) = \alpha \mathcal{L}_{content}(p, x) + \beta \mathcal{L}_{style}(s, x)$$



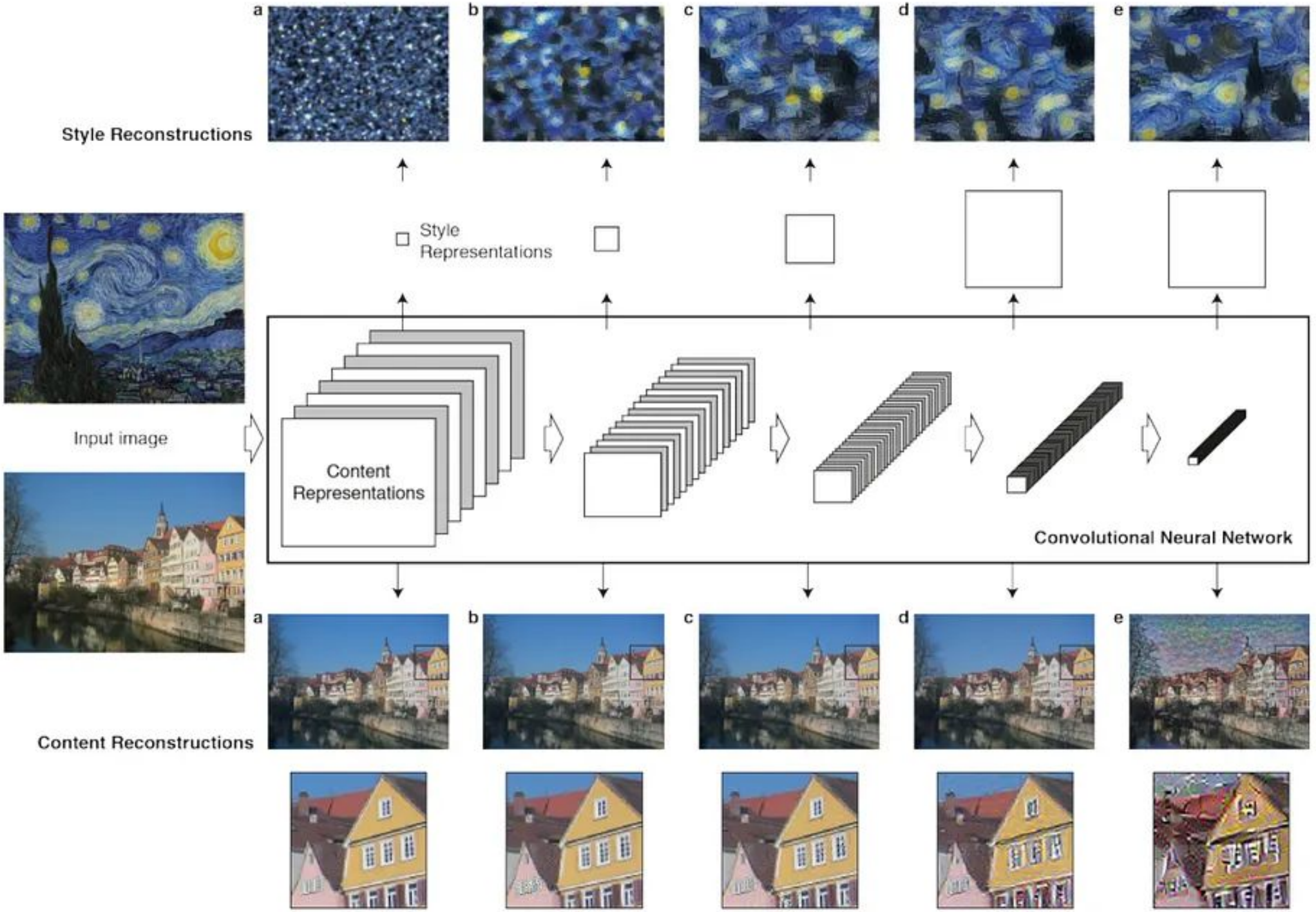
Style Transfer Example

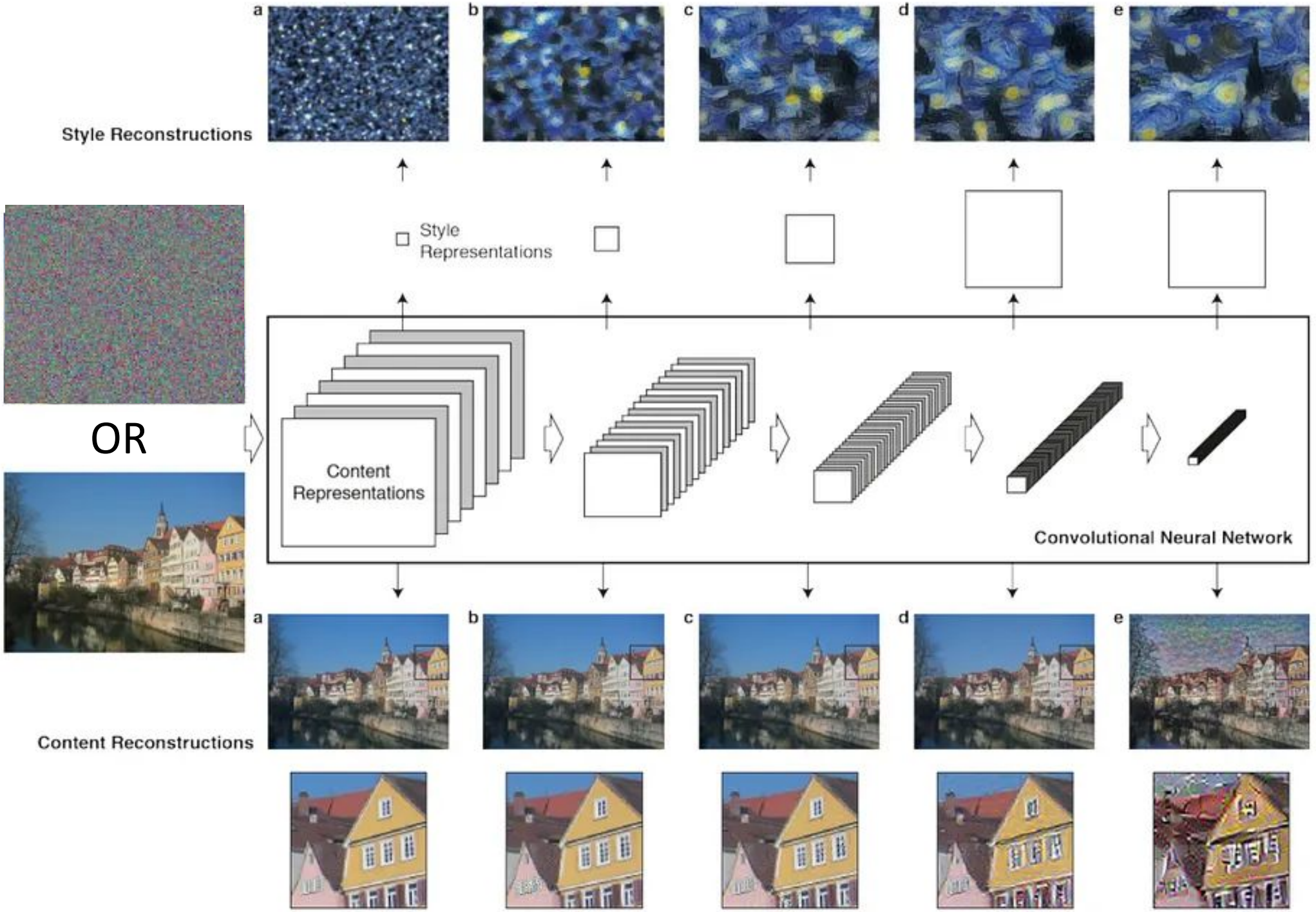
Style Transfer Example

❖ Style Transfer

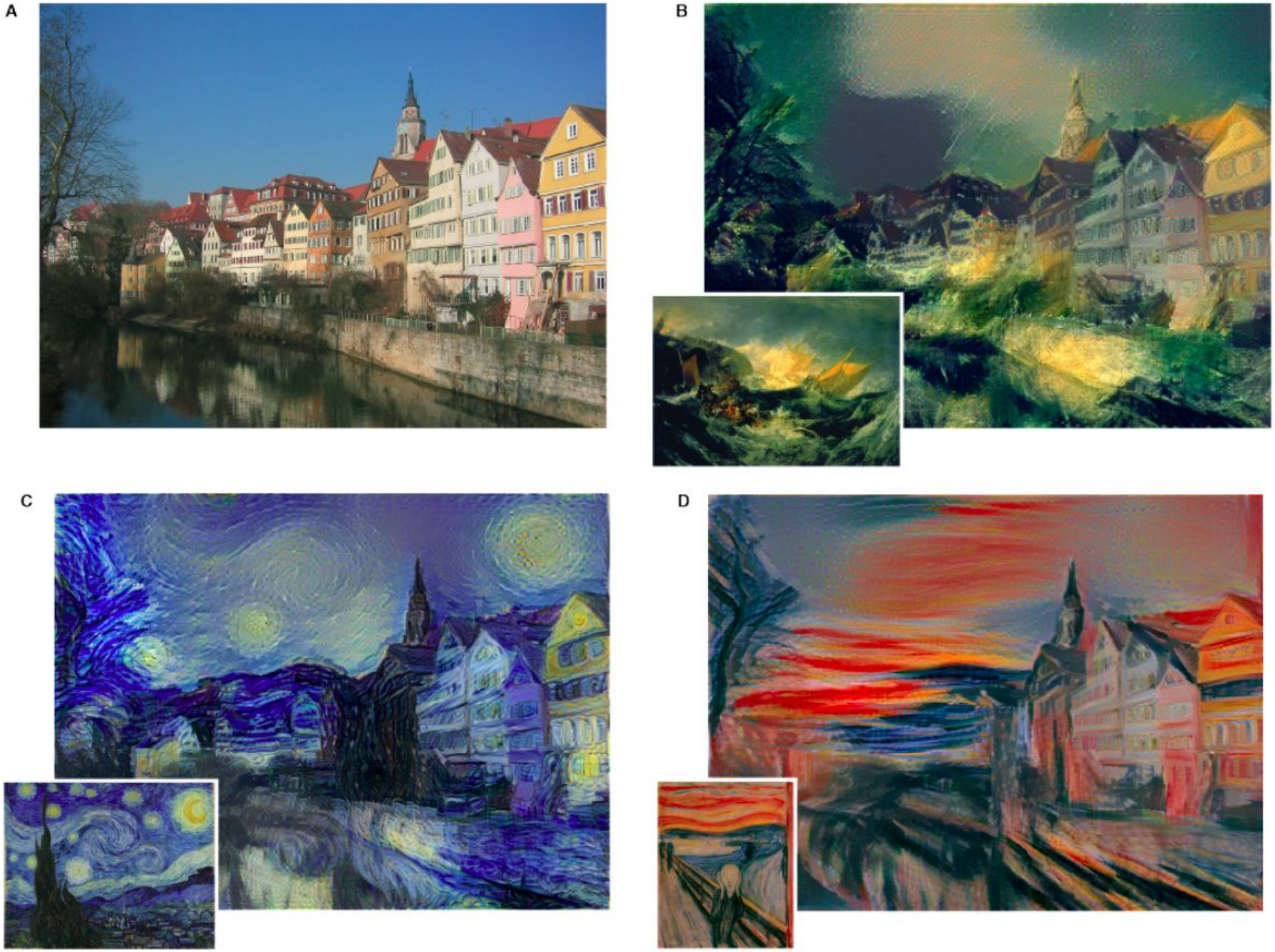
Algorithm 1 Neural Style Transfer

Start with a copy of the content reference image, call this *input* image
Run content and style reference images through VGG-19 model
for num_epochs **do**
  Run input image through VGG-19
  Calculate L using intermediate feature maps
  Backpropagate error through VGG-19 and adjust pixel values of input image
end for


a
b
c
d
e
Style Reconstructions
Style
Representations
Input image
Content
Representations
Convolutional Neural Network
a
b
c
d
e
Content Reconstructions


a
b
c
d
e
Style Reconstructions
Style
Representations
OR
Content
Representations
Convolutional Neural Network
a
b
c
d
e
Content Reconstructions

Style



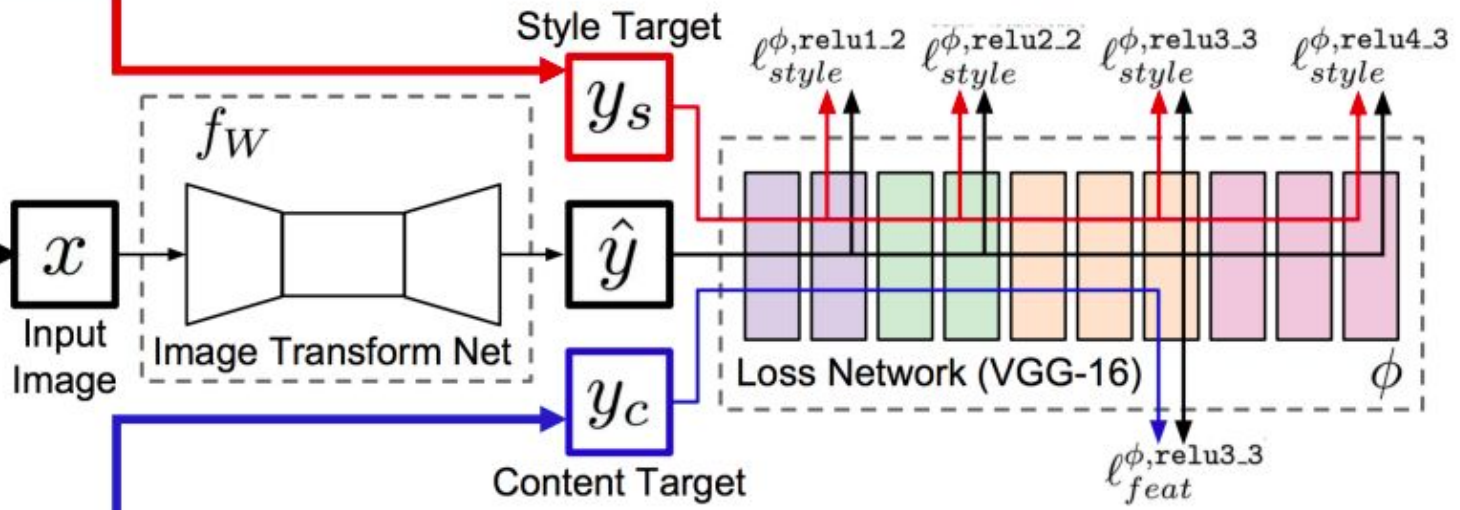
AI VIETNAM
All-in-One Course

Milestone Papers

Milestone Papers

❖ Fast Neural Style Transfer

1. Train a feedforward network for each style
2. Use pretrained CNN to compute same losses as before
3. After training, stylize images using a single forward pass

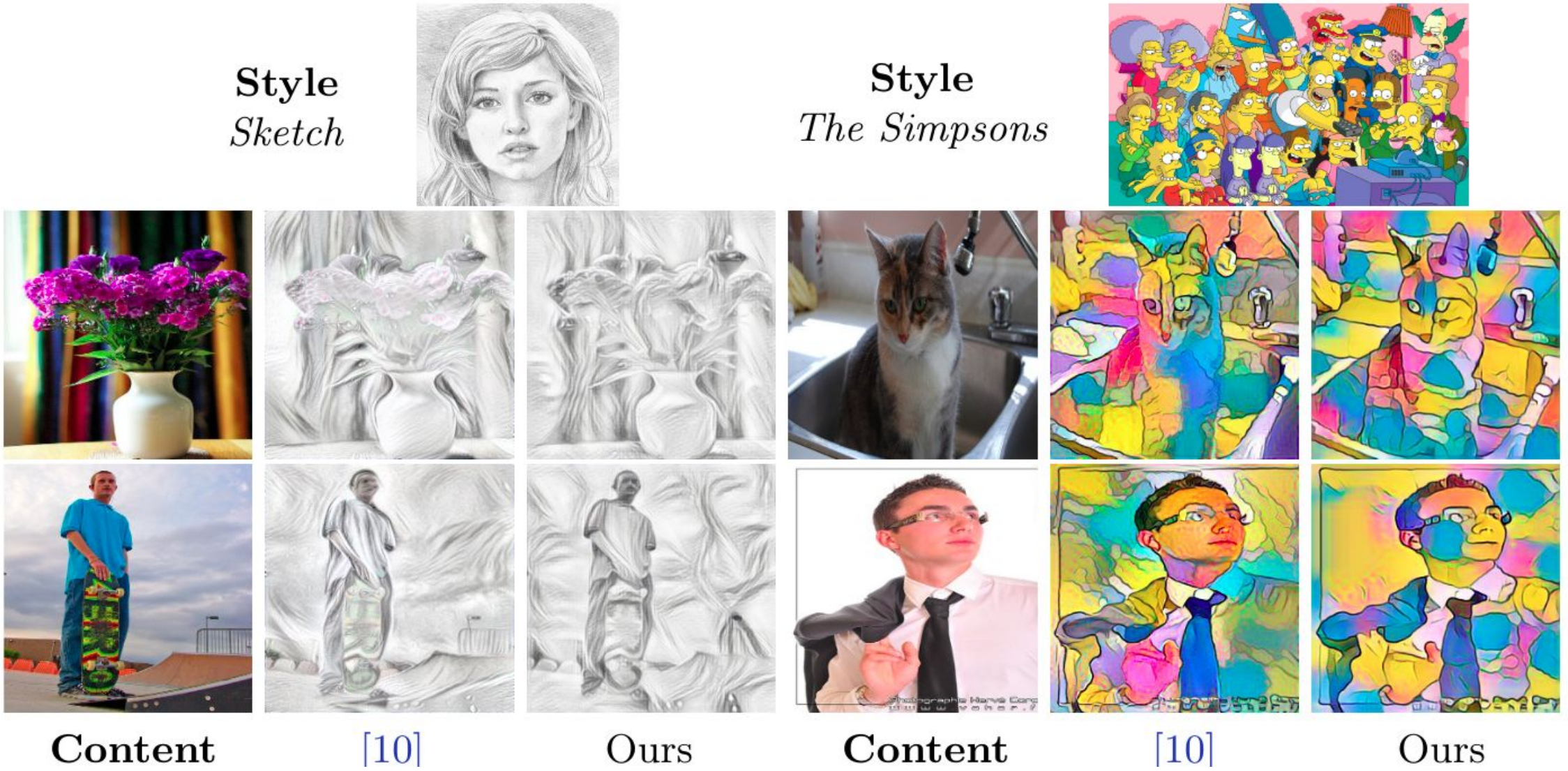


Milestone Papers

❖ Fast Neural Style Transfer

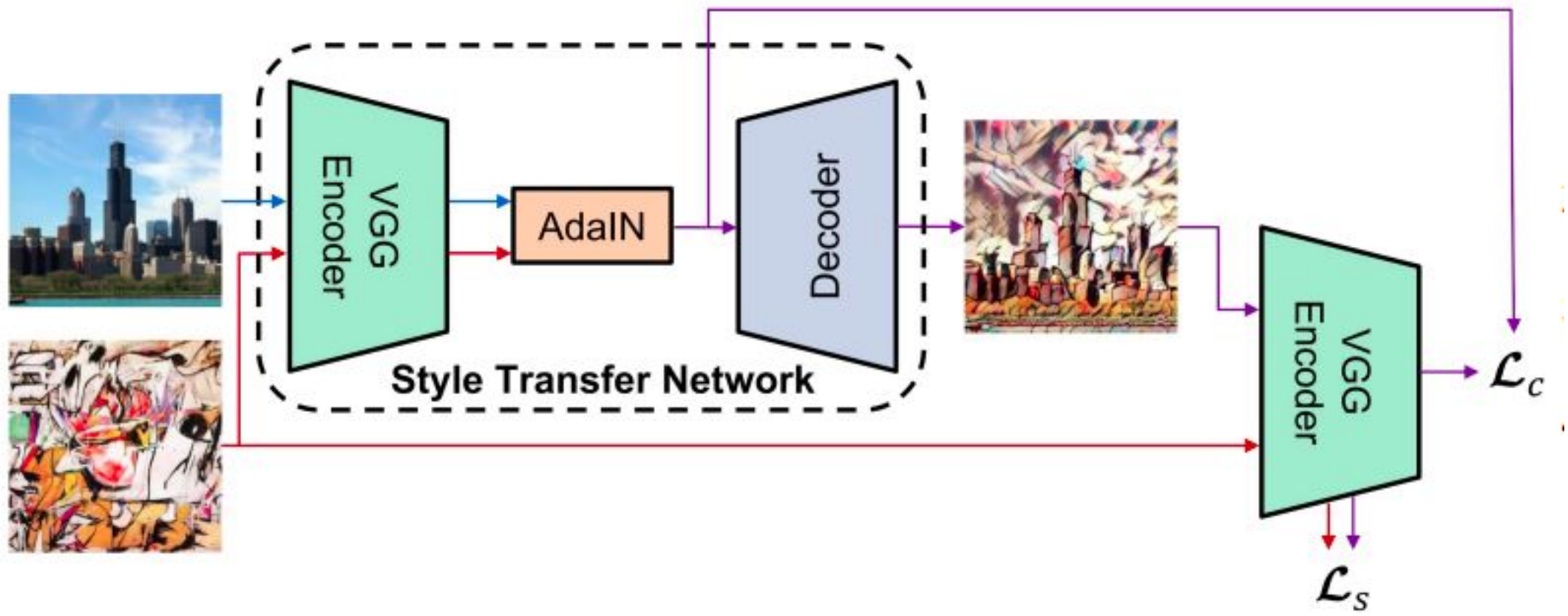
Milestone Papers

❖ Fast Neural Style Transfer



Milestone Papers

❖ Arbitrary Style Transfer



Milestone Papers

❖ Arbitrary Style Transfer

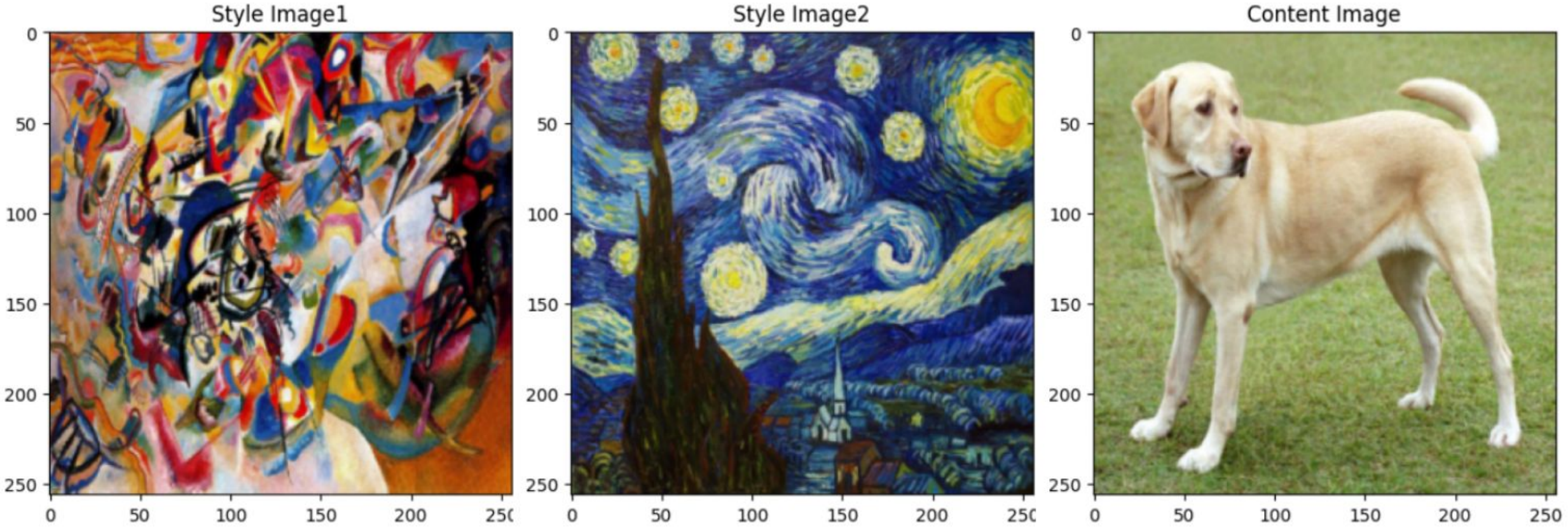
(a) $\alpha = 0$ (b) $\alpha = 0.25$ (c) $\alpha = 0.5$ (d) $\alpha = 0.75$ (e) $\alpha = 1$ (f) Style Image


QUIZ
QUIZ
QUIZ

AI VIETNAM
All-in-One Course

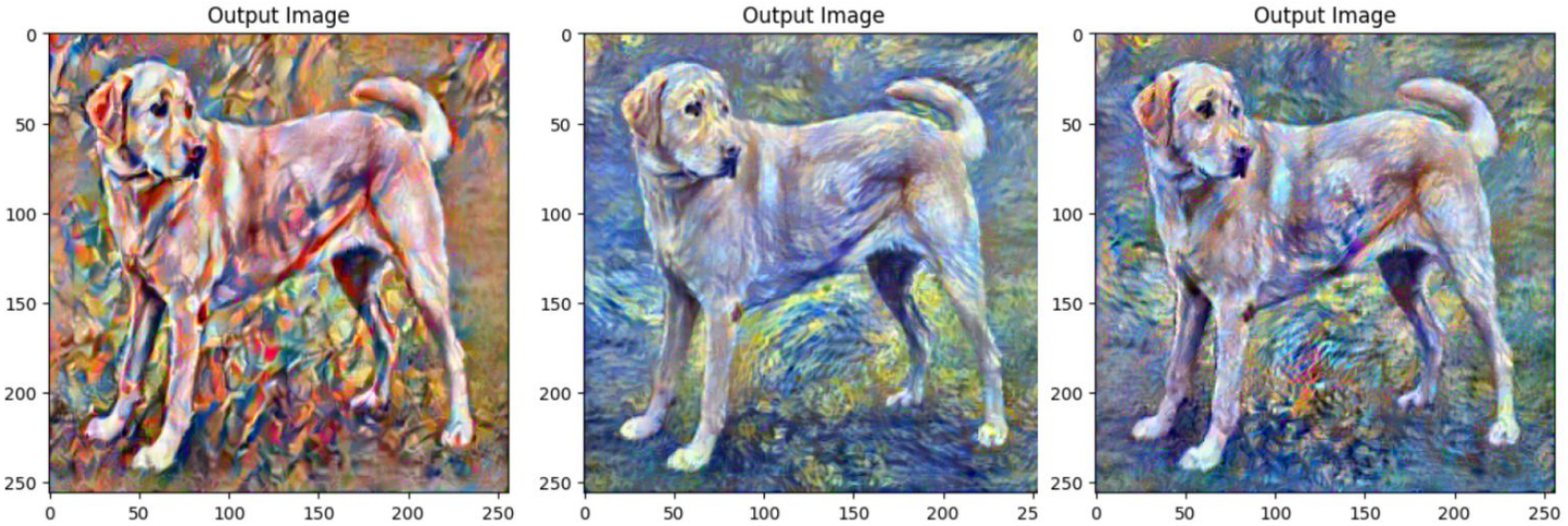
Style Transfer with 2 Style Images

Style Transfer with 2 Style Images



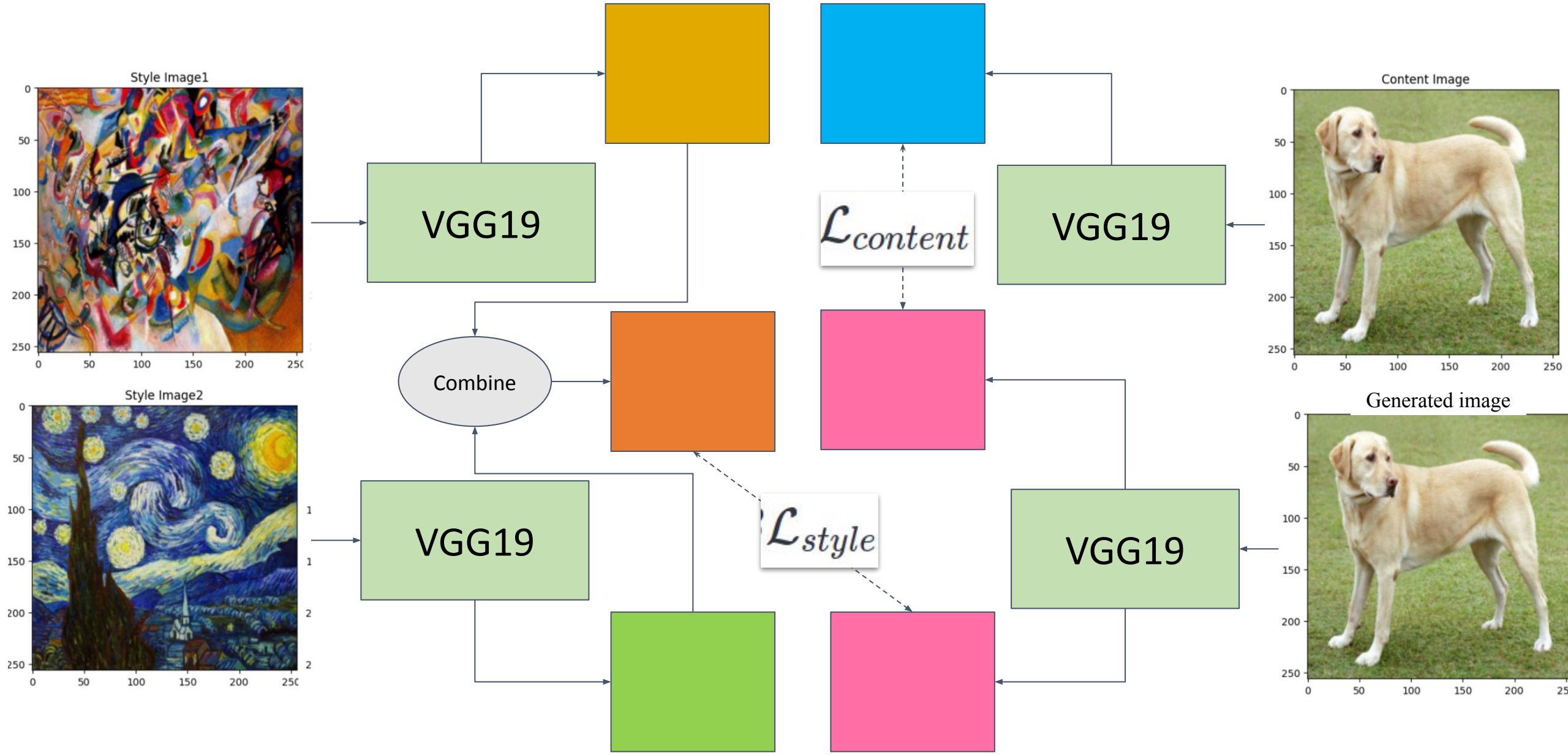
(a) Ảnh đầu vào của bài tập 1, ảnh trái là ảnh style 1, ảnh giữa là ảnh style 2, và ảnh phải là ảnh content

Style Transfer with 2 Style Images

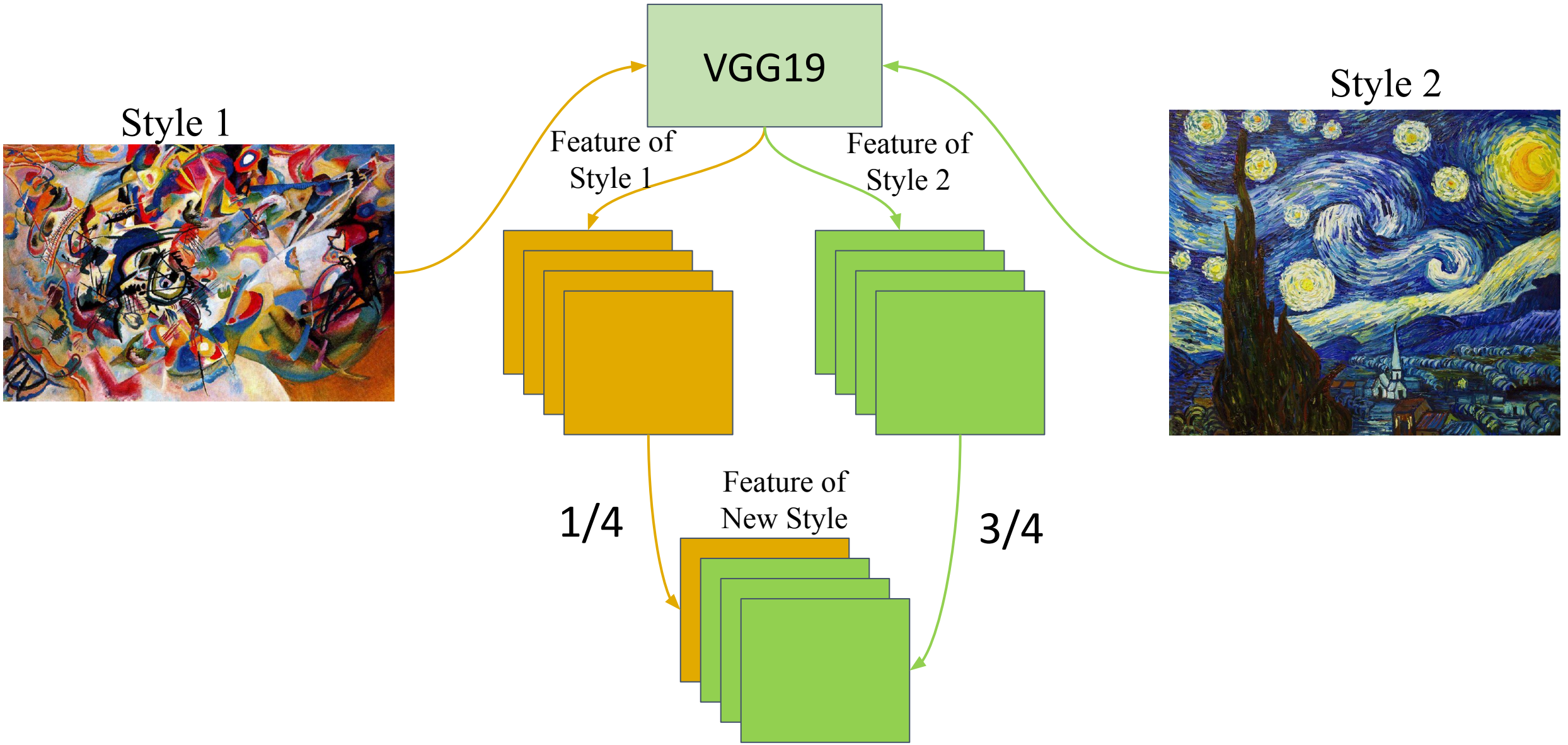


(b) Ảnh trái là ảnh khi transfer theo style 1, ảnh giữa là ảnh khi transfer theo style 2, và ảnh phải là ảnh output của bài tập 1 khi transfer theo 1/4 style1 + 3/4 style 2

Style Transfer with 2 Style Images



Style Transfer with 2 Style Images



Style Transfer with 2 Style Images

❖ Code Overview

0: Import Library → 1: Data Preparation → 2: Modeling → 3: Train and Evaluate → 4: Plot Results

1: Data Preparation	2: Modeling	3: Train and Evaluate	4: Plot Results
1 Load Data	1 Config Model	1 Train model	1 Train/Test Loss
2 Processing Data	2 Build Model	2 Evaluate model	2 Train/Test Acc
	3 Build Loss		

Style Transfer with 2 Style Images

❖ Data

```python
from PIL import Image
import torchvision.transforms as transforms

imsize = 256

img_transforms = transforms.Compose([
    transforms.Resize((imsize, imsize)),
    transforms.ToTensor(),
])
```

```python
def image_loader(image_name):
    image = Image.open(image_name)
    image = img_transforms(image).unsqueeze(0)
    return image.to(device, torch.float)

style_img1 = image_loader("style_img.jpg")
style_img2 = image_loader("style_img2.jpg")
content_img = image_loader("content_img.jpg")
```

Đầu tiên chúng ta cần load ảnh content và style bằng thư viện **PIL**, rồi resize về cùng kích thước **256x256** và convert thành tensor để sử dụng với Pytorch. Đối với bài 1 chúng ta cần load 3 ảnh (2 style và 1 content), còn đối với bài 2 chỉ cần 1 style và 1 content

Style Transfer with 2 Style Images

❖ Loss

3.1 Content Loss

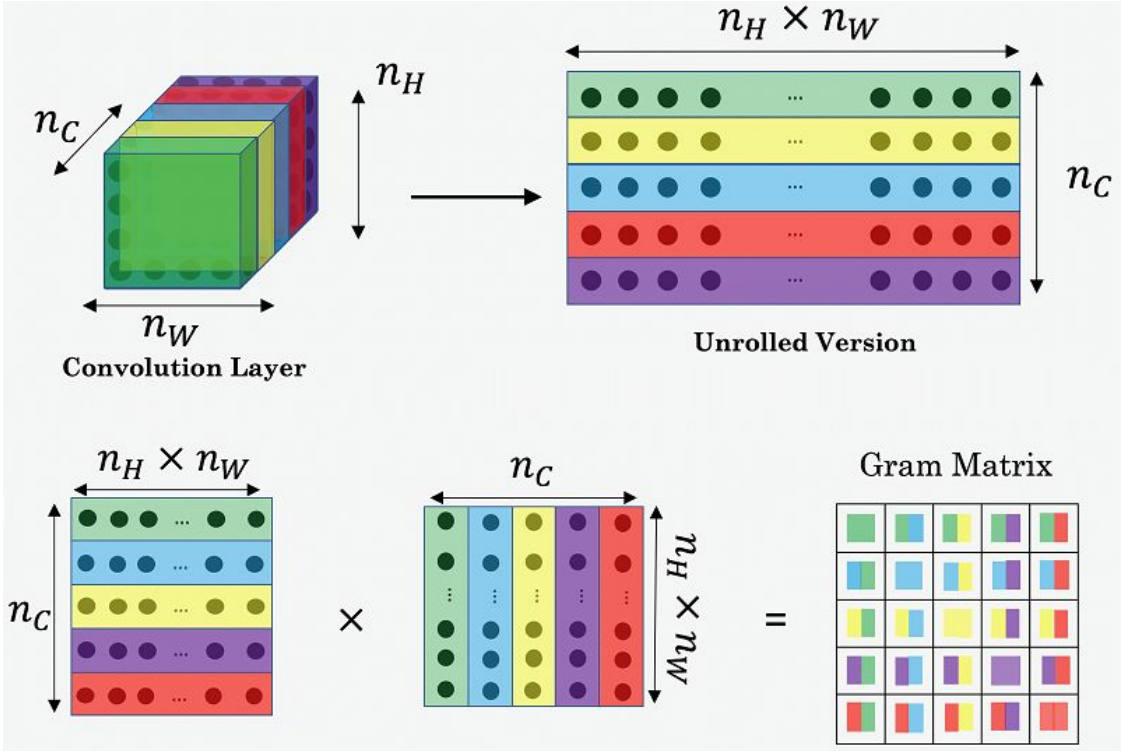
```
[ ]  1  content_weight = 1
     2  ContentLoss = nn.MSELoss()
```

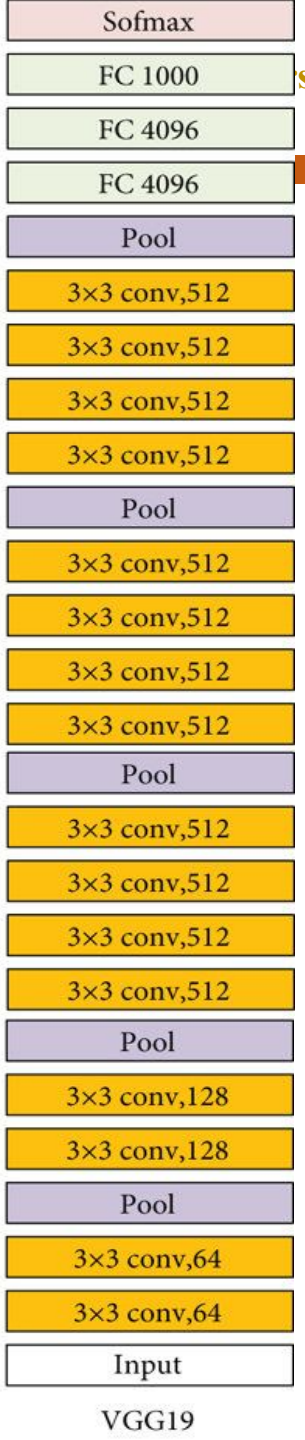
3.2 Style Loss

```
[ ]  1  def gram_matrix(tensor):
     2      a, b, c, d = tensor.size()
     3      tensor = tensor.view(a * b, c * d)
     4      G = torch.mm(tensor, tensor.t())
     5      return G.div(a * b * c * d)
     6
     7  style_weight = 1e6
     8  StyleLoss = nn.MSELoss()
```

$$\mathcal{L}_{content}(p,x,l) = \frac{1}{2}\sum_{i,j,k}(F^l_{ijk} - P^l_{ijk})^2$$

$$\mathcal{L}_{style}(s,x,l) = \frac{1}{2}\sum_{m,n}(G(F)^l_{mn} - G(S)^l_{mn})^2$$





VGG19

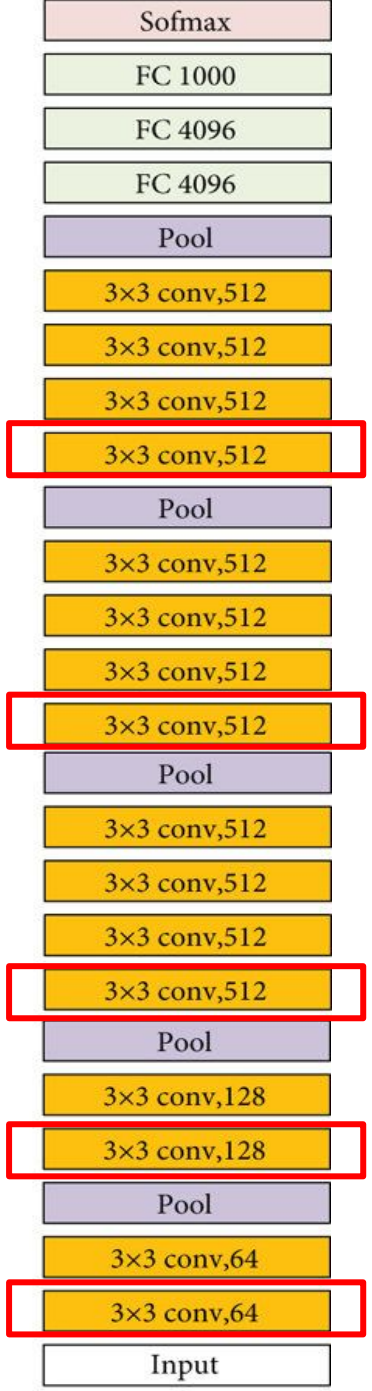
Style Transfer with 2 Style Images

❖ Model

```
1   from torchvision.models import vgg19, VGG19_Weights
2
3   VGG19_pretrained = vgg19(weights=VGG19_Weights.DEFAULT).features.eval()
4   VGG19_pretrained.to(device)
```

```
(0): Conv2d(3, 64, kernel_size=(3, 3), str
(1): ReLU(inplace=True)
(2): Conv2d(64, 64, kernel_size=(3, 3), st
(3): ReLU(inplace=True)
(4): MaxPool2d(kernel_size=2, stride=2, pa
(5): Conv2d(64, 128, kernel_size=(3, 3), s
(6): ReLU(inplace=True)
(7): Conv2d(128, 128, kernel_size=(3, 3),
(8): ReLU(inplace=True)
(9): MaxPool2d(kernel_size=2, stride=2, pa
(10): Conv2d(128, 256, kernel_size=(3, 3),
(11): ReLU(inplace=True)
(12): Conv2d(256, 256, kernel_size=(3, 3),
(13): ReLU(inplace=True)
(14): Conv2d(256, 256, kernel_size=(3, 3),
(15): ReLU(inplace=True)
(16): Conv2d(256, 256, kernel_size=(3, 3),
(17): ReLU(inplace=True)
```

```
(18): MaxPool2d(kernel_size=2, stride=2, p
(19): Conv2d(256, 512, kernel_size=(3, 3),
(20): ReLU(inplace=True)
(21): Conv2d(512, 512, kernel_size=(3, 3),
(22): ReLU(inplace=True)
(23): Conv2d(512, 512, kernel_size=(3, 3),
(24): ReLU(inplace=True)
(25): Conv2d(512, 512, kernel_size=(3, 3),
(26): ReLU(inplace=True)
(27): MaxPool2d(kernel_size=2, stride=2, p
(28): Conv2d(512, 512, kernel_size=(3, 3),
(29): ReLU(inplace=True)
(30): Conv2d(512, 512, kernel_size=(3, 3),
(31): ReLU(inplace=True)
(32): Conv2d(512, 512, kernel_size=(3, 3),
(33): ReLU(inplace=True)
(34): Conv2d(512, 512, kernel_size=(3, 3),
(35): ReLU(inplace=True)
(36): MaxPool2d(kernel_size=2, stride=2, p
```

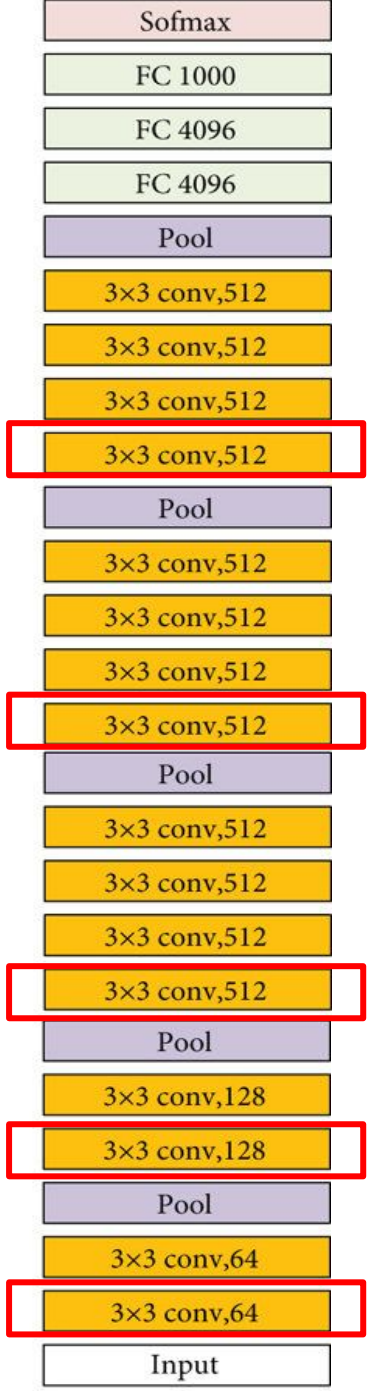


VGG19

```python
def get_features(pretrained_model, image):
    layers = {
        '0': 'conv_1',
        '5': 'conv_2',
        '10': 'conv_3',
        '19': 'conv_4',
        '28': 'conv_5'
    }
```

```
(0): Conv2d(3, 64, kernel_size=(3, 3), str
(1): ReLU(inplace=True)
(2): Conv2d(64, 64, kernel_size=(3, 3), st
(3): ReLU(inplace=True)
(4): MaxPool2d(kernel_size=2, stride=2, pa
(5): Conv2d(64, 128, kernel_size=(3, 3),
(6): ReLU(inplace=True)
(7): Conv2d(128, 128, kernel_size=(3, 3),
(8): ReLU(inplace=True)
(9): MaxPool2d(kernel_size=2, stride=2, pa
(10): Conv2d(128, 256, kernel_size=(3, 3)
(11): ReLU(inplace=True)
(12): Conv2d(256, 256, kernel_size=(3, 3),
(13): ReLU(inplace=True)
(14): Conv2d(256, 256, kernel_size=(3, 3),
(15): ReLU(inplace=True)
(16): Conv2d(256, 256, kernel_size=(3, 3),
(17): ReLU(inplace=True)
```

```
(18): MaxPool2d(kernel_size=2, stride=2, p
(19): Conv2d(256, 512, kernel_size=(3, 3),
(20): ReLU(inplace=True)
(21): Conv2d(512, 512, kernel_size=(3, 3),
(22): ReLU(inplace=True)
(23): Conv2d(512, 512, kernel_size=(3, 3),
(24): ReLU(inplace=True)
(25): Conv2d(512, 512, kernel_size=(3, 3),
(26): ReLU(inplace=True)
(27): MaxPool2d(kernel_size=2, stride=2, p
(28): Conv2d(512, 512, kernel_size=(3, 3),
(29): ReLU(inplace=True)
(30): Conv2d(512, 512, kernel_size=(3, 3),
(31): ReLU(inplace=True)
(32): Conv2d(512, 512, kernel_size=(3, 3),
(33): ReLU(inplace=True)
(34): Conv2d(512, 512, kernel_size=(3, 3),
(35): ReLU(inplace=True)
(36): MaxPool2d(kernel_size=2, stride=2, p
```



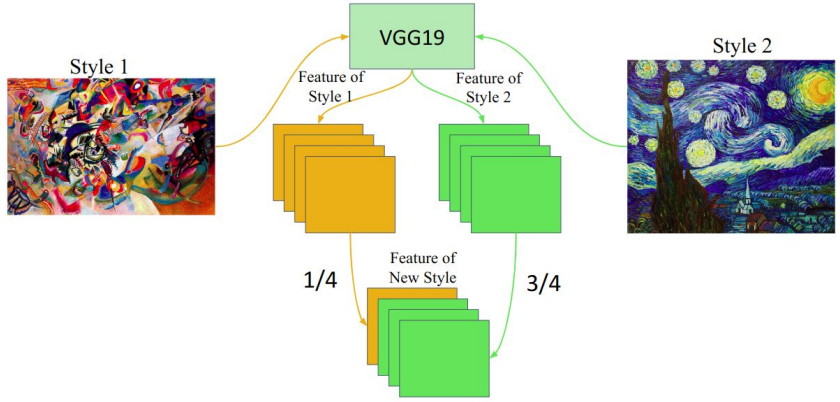
VGG19

```
 9       features = {}
10       x = image
11       for name, pretrained_layer in pretrained_model._modules.items():
12           x = pretrained_layer(x)
13           if name in layers:
14               features[layers[name]] = x
15       return features
16
```

```
(0): Conv2d(3, 64, kernel_size=(3, 3), str
(1): ReLU(inplace=True)
(2): Conv2d(64, 64, kernel_size=(3, 3), st
(3): ReLU(inplace=True)
(4): MaxPool2d(kernel_size=2, stride=2, pa
(5): Conv2d(64, 128, kernel_size=(3, 3),
(6): ReLU(inplace=True)
(7): Conv2d(128, 128, kernel_size=(3, 3),
(8): ReLU(inplace=True)
(9): MaxPool2d(kernel_size=2, stride=2, pa
(10): Conv2d(128, 256, kernel_size=(3, 3)
(11): ReLU(inplace=True)
(12): Conv2d(256, 256, kernel_size=(3, 3),
(13): ReLU(inplace=True)
(14): Conv2d(256, 256, kernel_size=(3, 3),
(15): ReLU(inplace=True)
(16): Conv2d(256, 256, kernel_size=(3, 3),
(17): ReLU(inplace=True)
```

```
(18): MaxPool2d(kernel_size=2, stride=2, p
(19): Conv2d(256, 512, kernel_size=(3, 3),
(20): ReLU(inplace=True)
(21): Conv2d(512, 512, kernel_size=(3, 3),
(22): ReLU(inplace=True)
(23): Conv2d(512, 512, kernel_size=(3, 3),
(24): ReLU(inplace=True)
(25): Conv2d(512, 512, kernel_size=(3, 3),
(26): ReLU(inplace=True)
(27): MaxPool2d(kernel_size=2, stride=2, p
(28): Conv2d(512, 512, kernel_size=(3, 3),
(29): ReLU(inplace=True)
(30): Conv2d(512, 512, kernel_size=(3, 3),
(31): ReLU(inplace=True)
(32): Conv2d(512, 512, kernel_size=(3, 3),
(33): ReLU(inplace=True)
(34): Conv2d(512, 512, kernel_size=(3, 3),
(35): ReLU(inplace=True)
(36): MaxPool2d(kernel_size=2, stride=2, p
```

AI VIE
All-in-Or



er

```
content_layers = ['conv_4']
style_layers = ['conv_1', 'conv_2',
                'conv_3', 'conv_4',
                'conv_5']
```

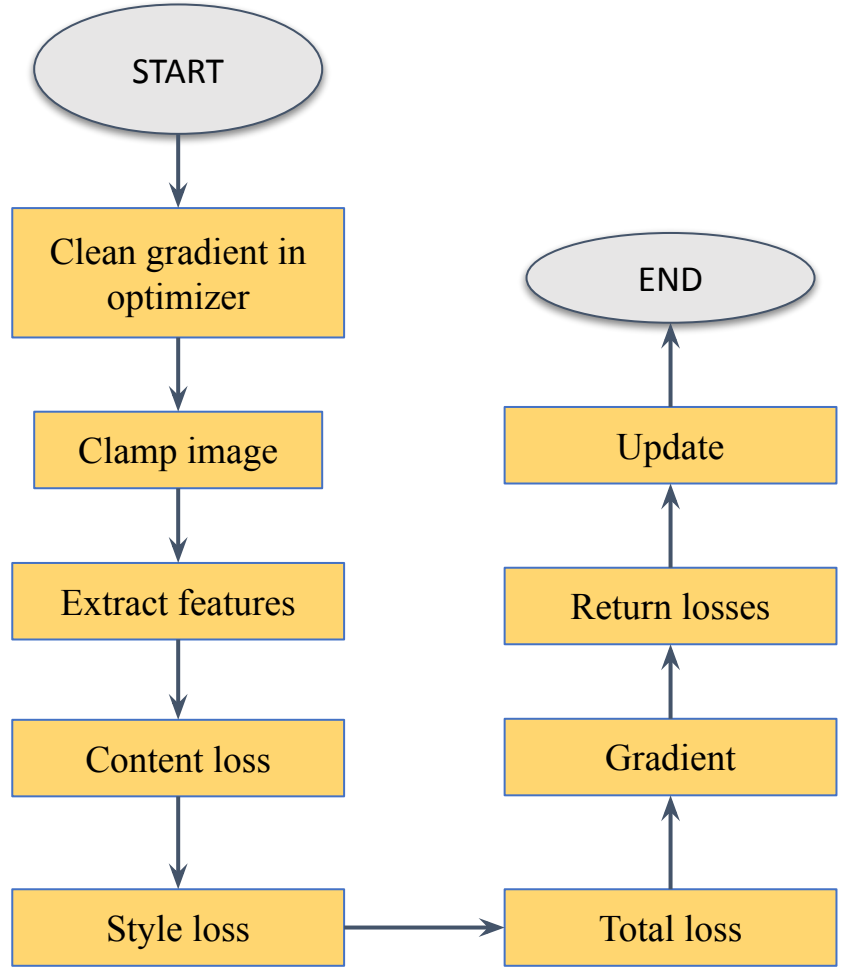
```
def get_dual_style(style_features1, style_features2, style_layers):
    final_style_features = {}
    for layer in style_layers:
        sf1 = style_features1[layer]
        sf2 = style_features2[layer]
        sf1_size = int(sf1.size()[1] / 4)
        final_style_features[layer] = torch.concatenate([
                                          sf1[:,:sf1_size,:,:],
                                          sf2[:,sf1_size:,:,:]
                                      ], dim=1)
    return final_style_features
```

```
content_features = get_features(VGG19_pretrained, content_img)
style_features1 = get_features(VGG19_pretrained, style_img1)
style_features2 = get_features(VGG19_pretrained, style_img2)
final_style_features = get_dual_style(style_features1, style_features2, style_layers)
```

Style Transfer with 2 Style Images

❖ Style Transfer

```python
def style_tranfer_(model, optimizer, target_img,
                   content_features, style_features,
                   style_layers, content_weight, style_weight):

    optimizer.zero_grad()
    with torch.no_grad():
        target_img.clamp_(0, 1)
    target_features = get_features(model, target_img)

    content_loss = ContentLoss(content_features['conv_4'],
                               target_features['conv_4'])

    style_loss = 0
    for layer in style_layers:
        target_gram = gram_matrix(target_features[layer])
        style_gram = gram_matrix(style_features[layer])
        style_loss += StyleLoss(style_gram, target_gram)

    total_loss = content_loss*content_weight + style_loss*style_weight
    total_loss.backward(retain_graph=True)
    optimizer.step()
    return total_loss, content_loss, style_loss
```



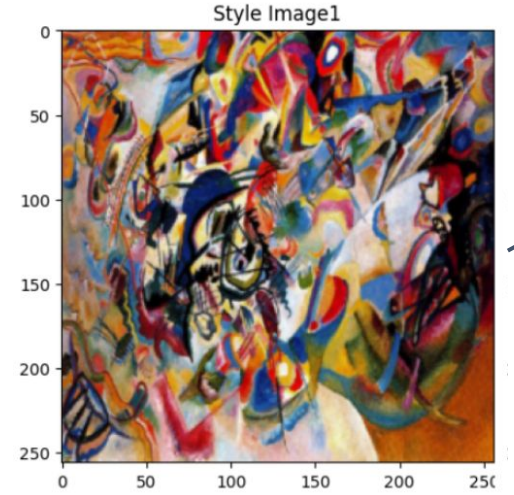
Style Transfer with 2 Style Images

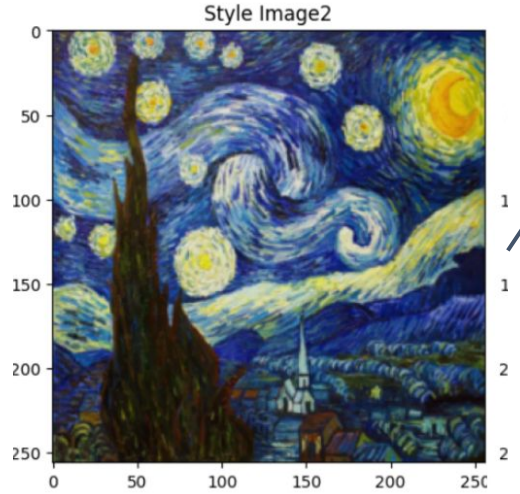
❖ Style Transfer

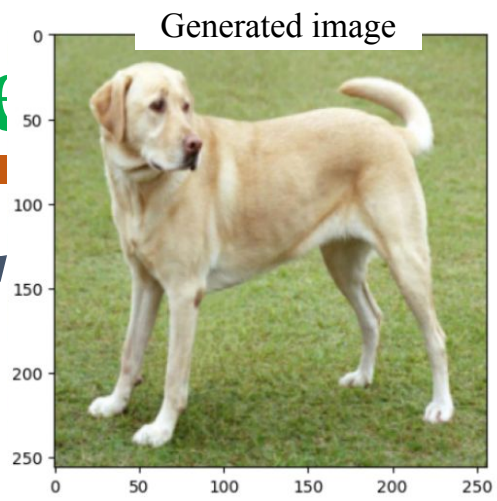
```
3   target_img = content_img.clone().requires_grad_(True).to(device)
4   optimizer = optim.Adam([target_img], lr=0.02)
```

```
1   STEPS = 3000
2
3   for step in range(STEPS):
4       optimizer.zero_grad()
5       with torch.no_grad():
6           target_img.clamp_(0, 1)
7
8       total_loss, content_loss, style_loss = style_tranfer_(VGG19_pretrained, optimizer, target_img,
9                                                             content_features, final_style_features,
10                                                            style_layers, content_weight, style_weight)
11      if step % 100 == 99:
12          print(f"Epoch [{step+1}/{STEPS}] Total loss: {total_loss.item():.6f} - \
13                  Content loss: {content_loss.item():.6f} - Style loss: {style_loss.item():.6f}")
14
15      with torch.no_grad():
16          target_img.clamp_(0, 1)
```

Style Transfe tyle Images

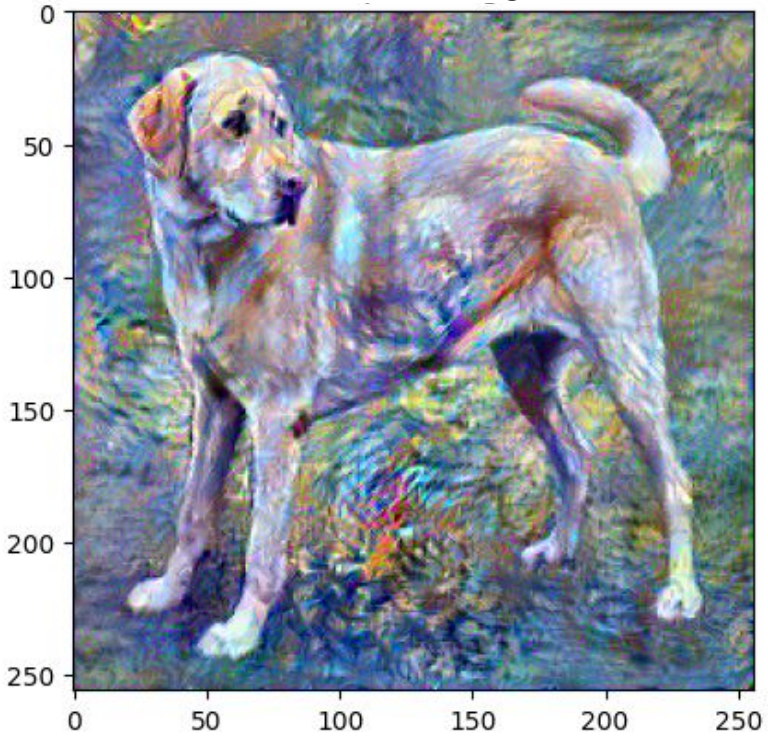


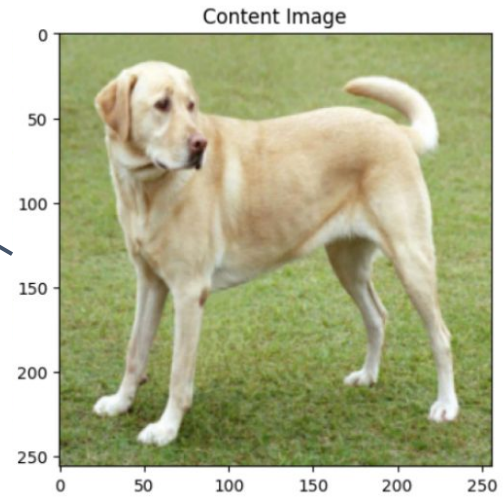




After 3000 steps

Generated image

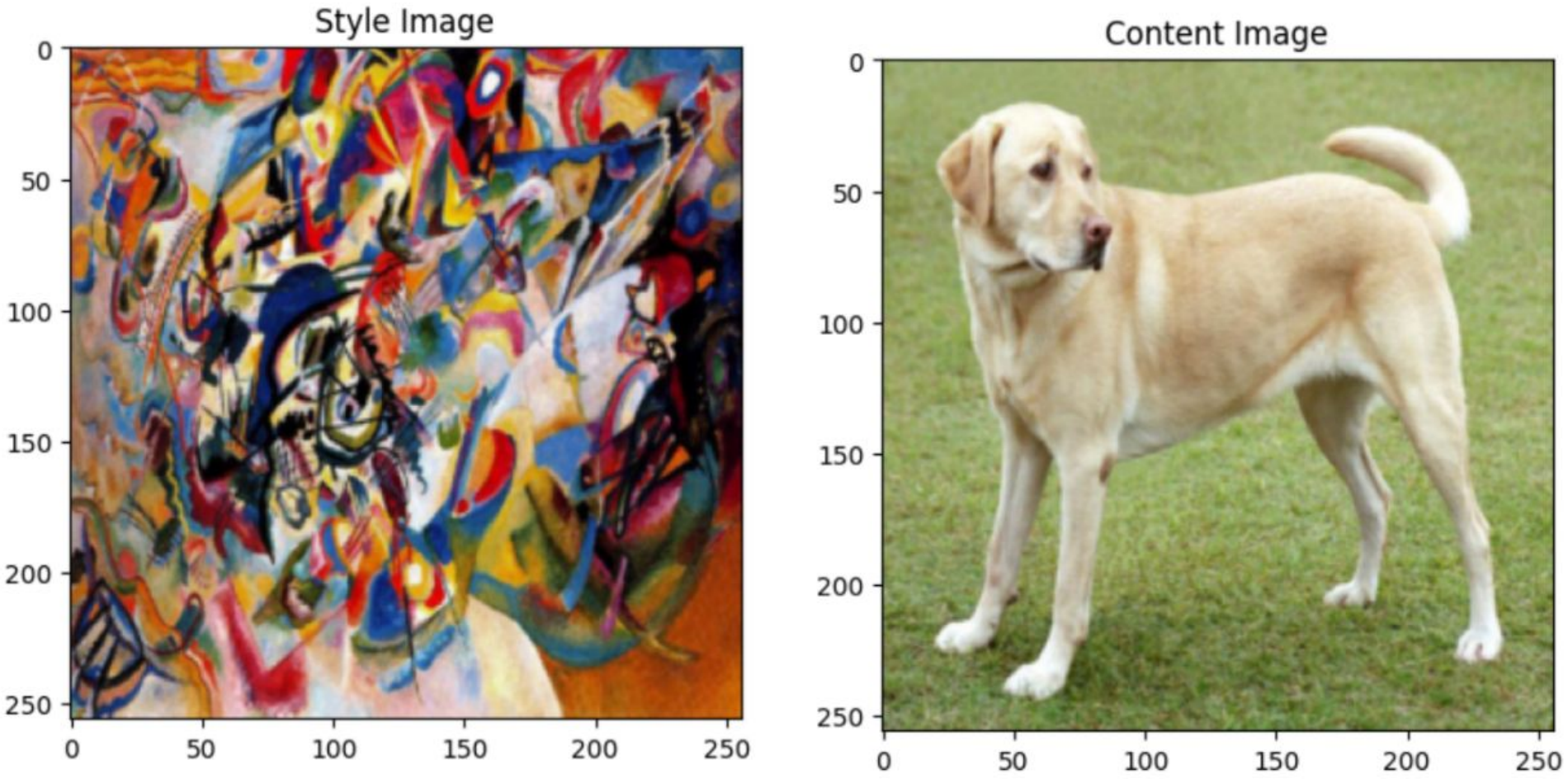




AI VIETNAM
All-in-One Course

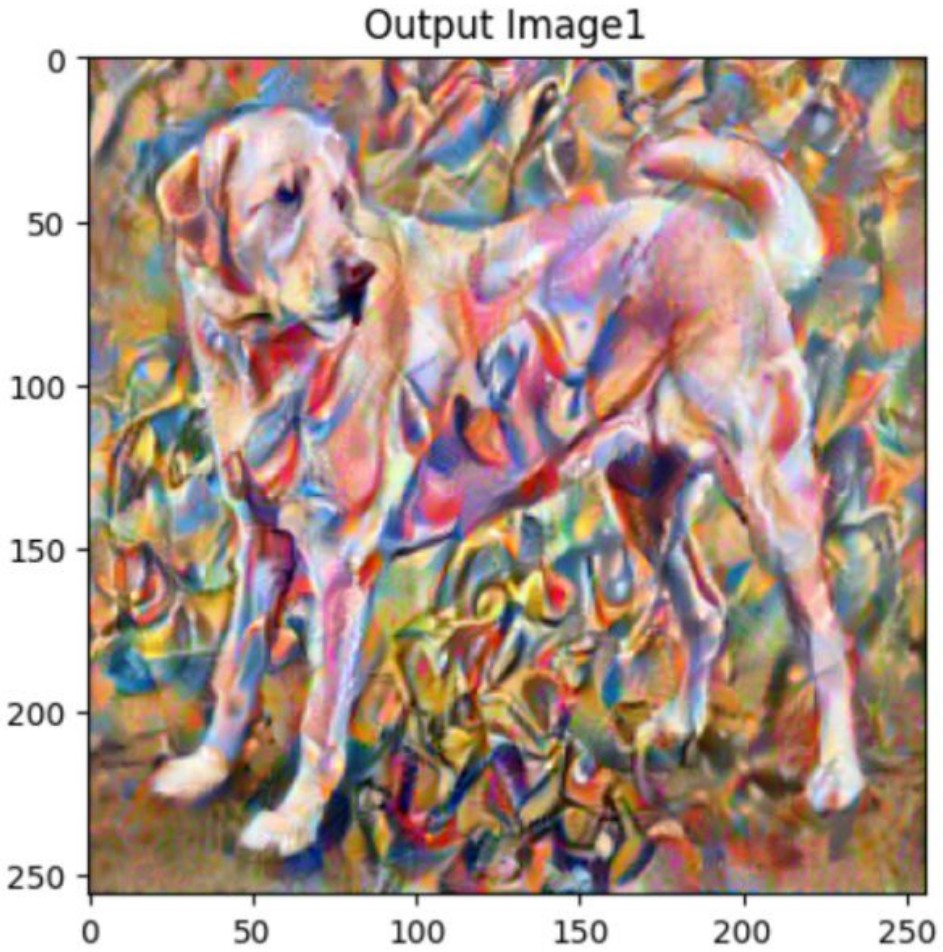
Style Transfer with Rotation + DE

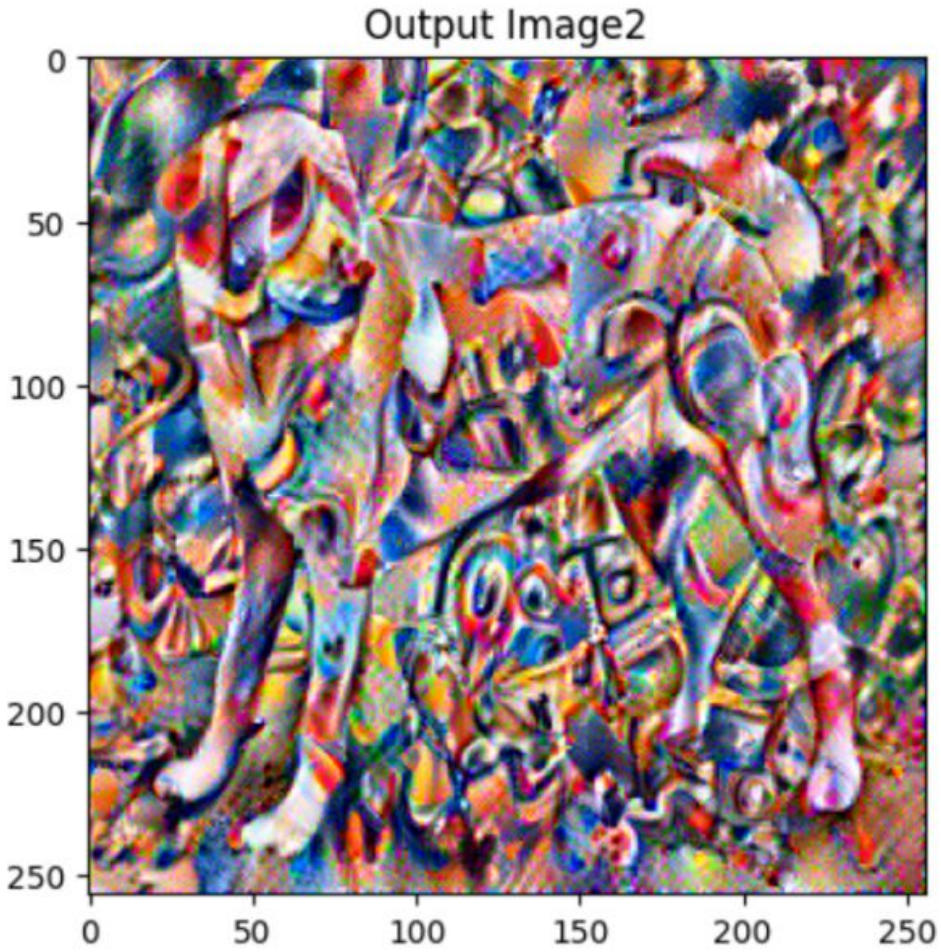
Style Transfer with Rotation + DE



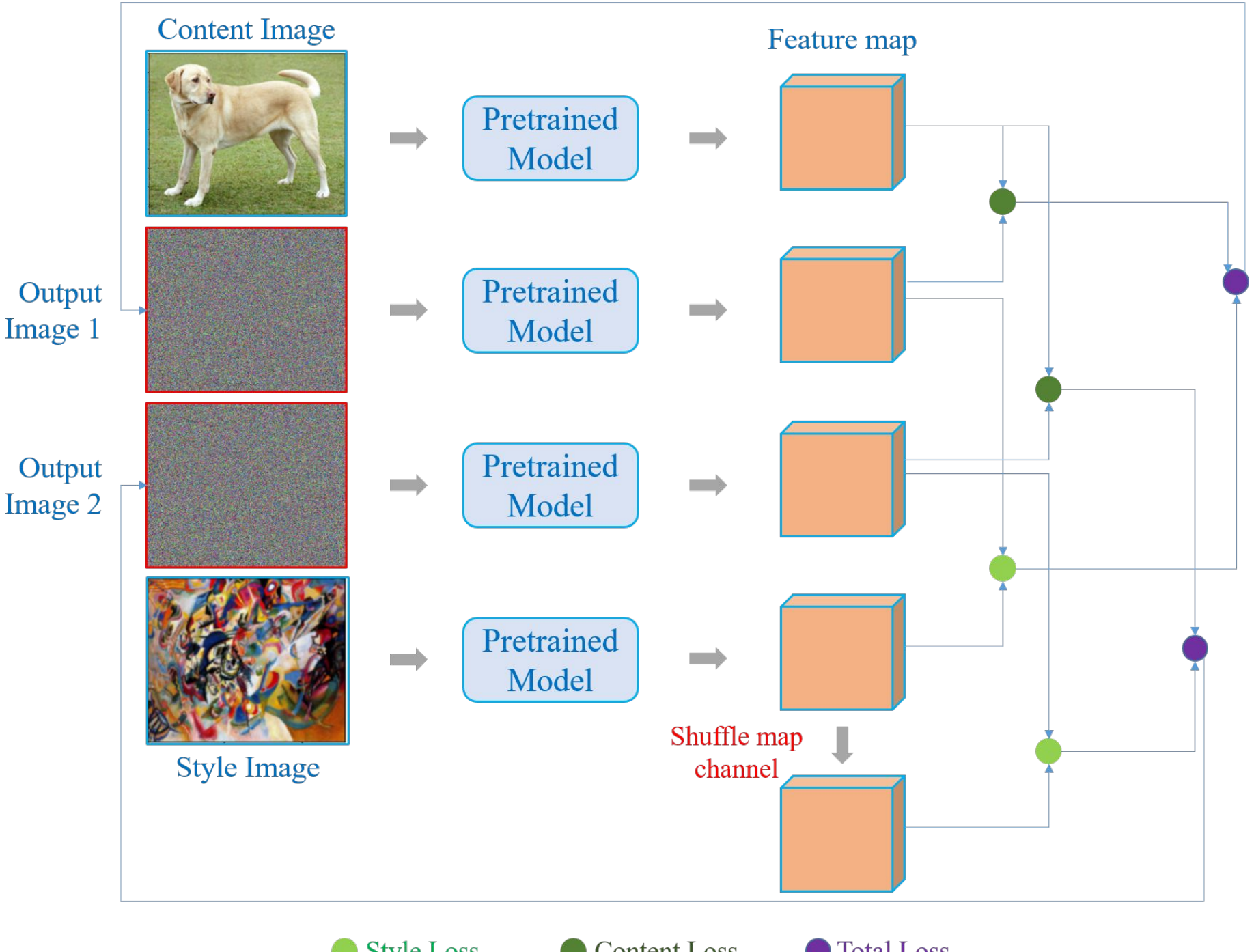
(a) Ảnh đầu vào của bài tập 2, ảnh trái là ảnh style 1, và ảnh phải là ảnh content

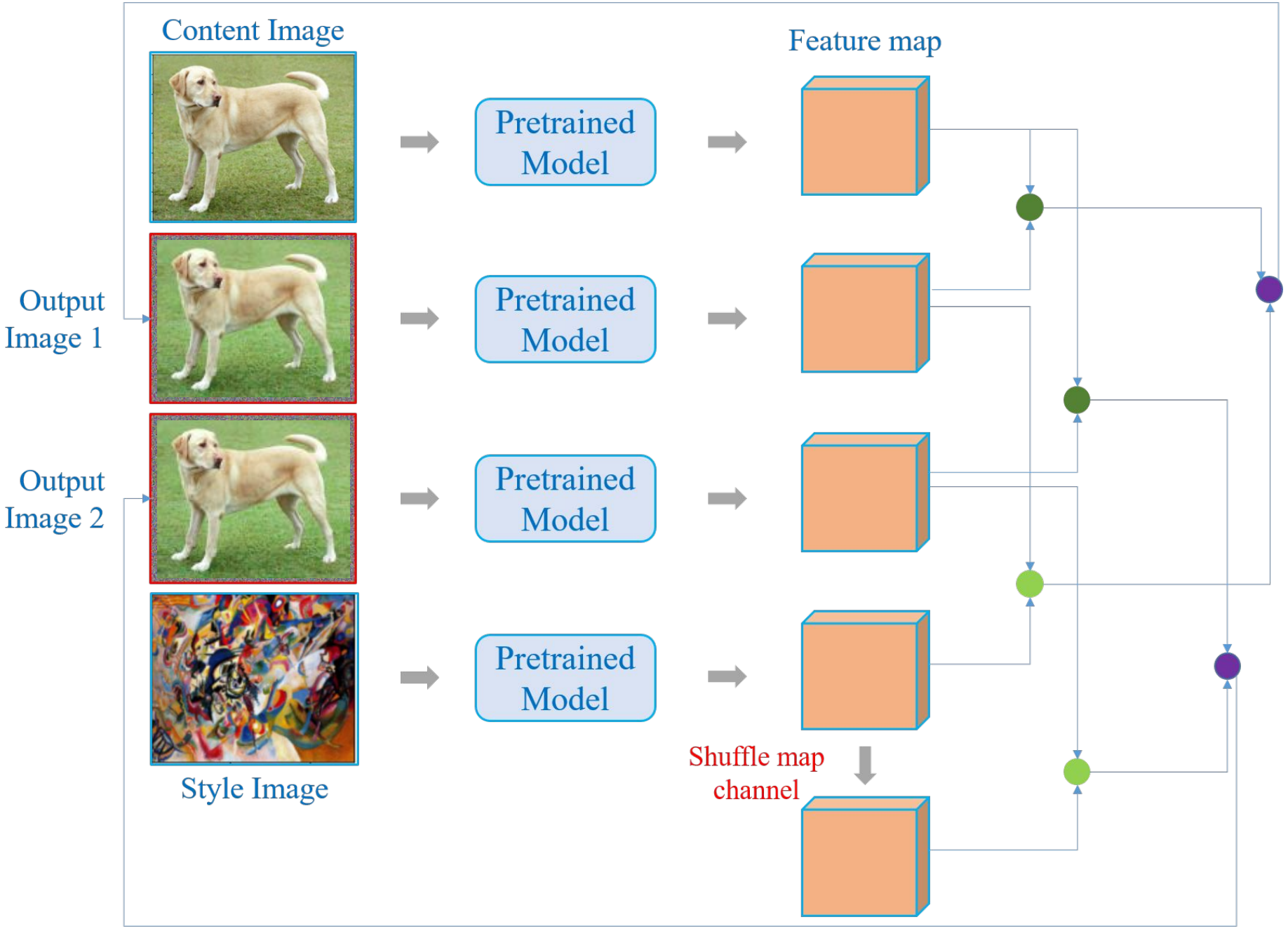
Style Transfer with Rotation + DE

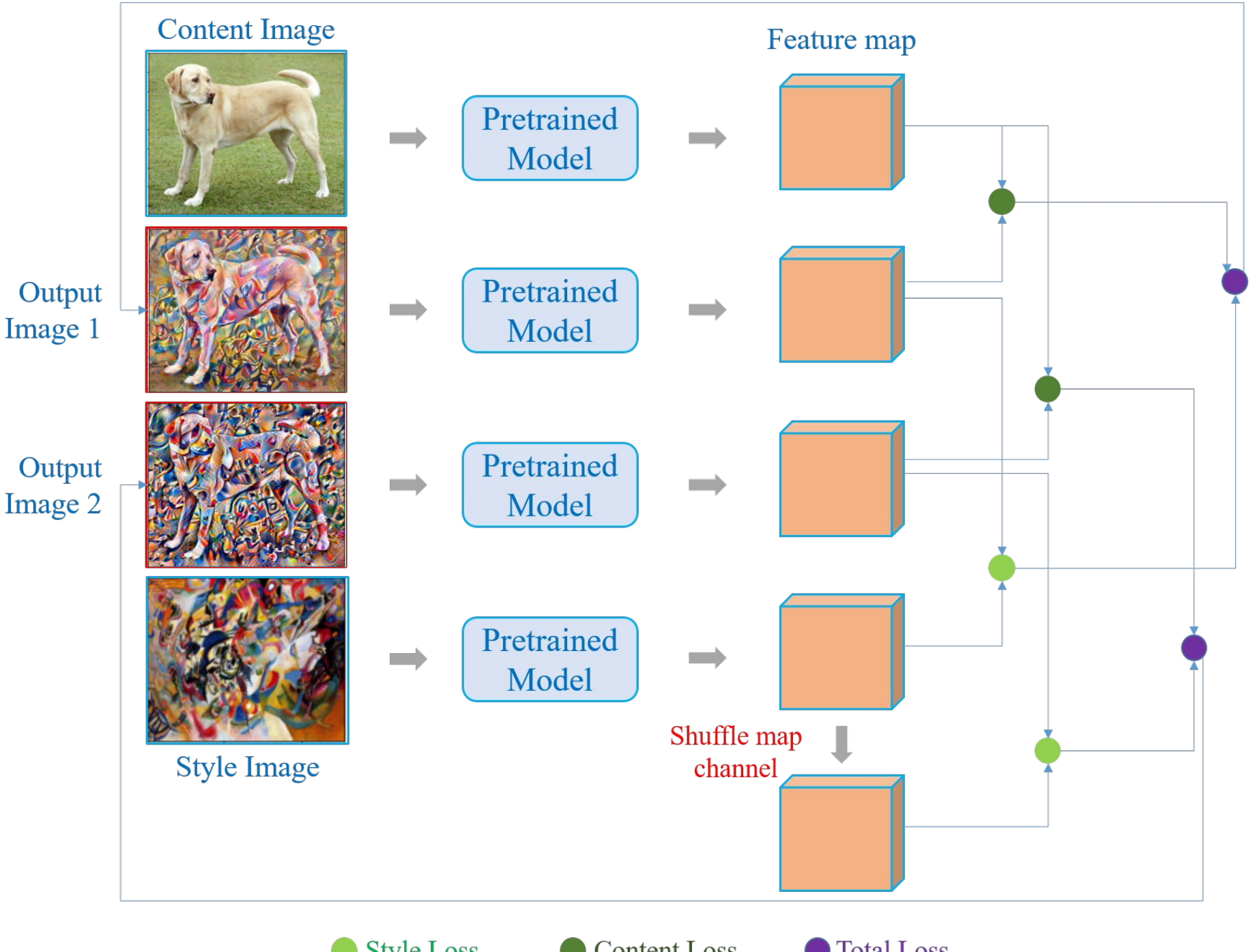




(b) Ảnh output của bài tập 2, ảnh trái là ảnh khi transfer theo style 1 và ảnh phải là ảnh khi transfer theo style 1 đã thực hiện các biến đổi rotate 90, 180 và Differential Evolution

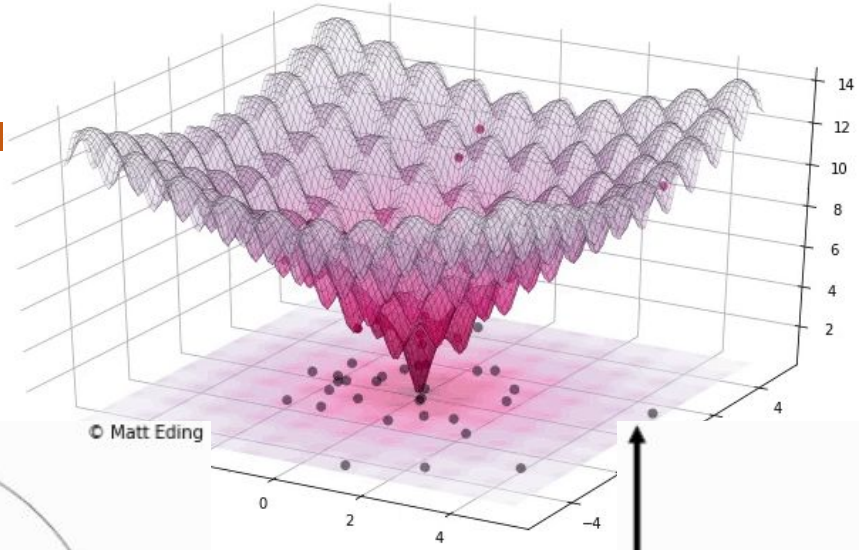

Content Image
Feature map
Pretrained Model
Pretrained Model
Pretrained Model
Pretrained Model
Output Image 1
Output Image 2
Style Image
Shuffle map channel
Style Loss
Content Loss
Total Loss

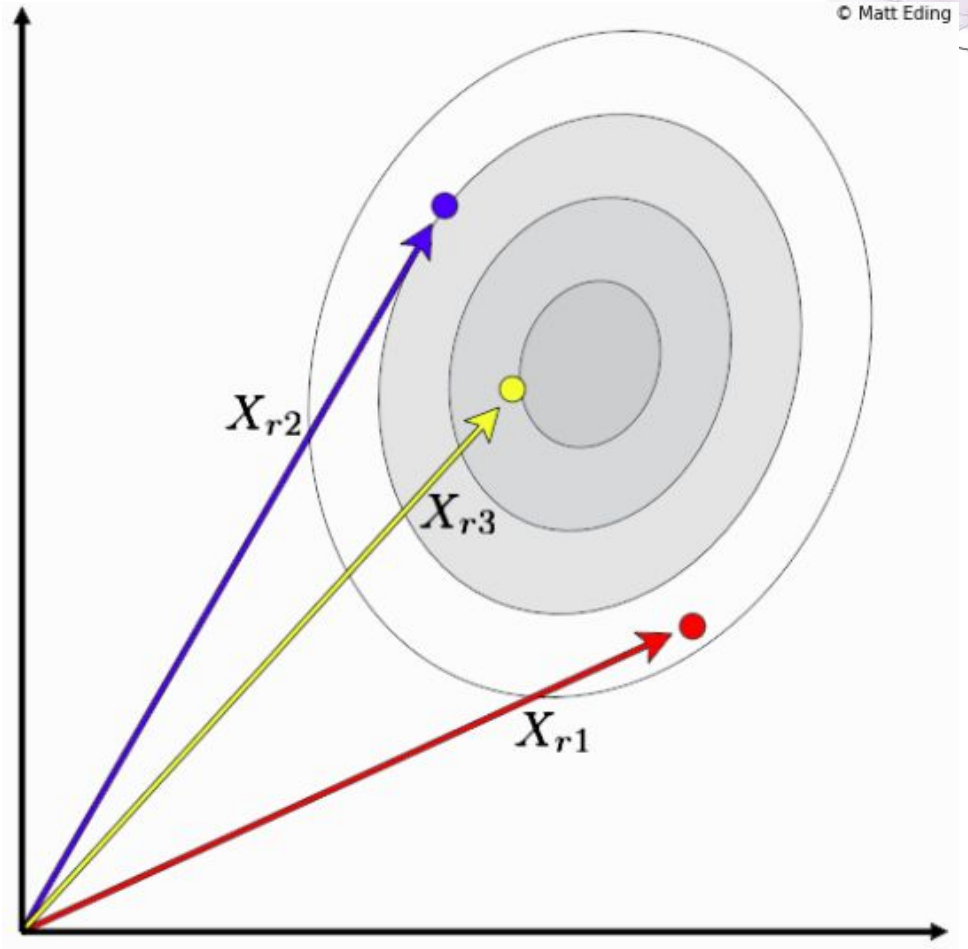

Content Image
Feature map
Pretrained Model
Pretrained Model
Pretrained Model
Pretrained Model
Output Image 1
Output Image 2
Style Image
Shuffle map channel
Style Loss
Content Loss
Total Loss

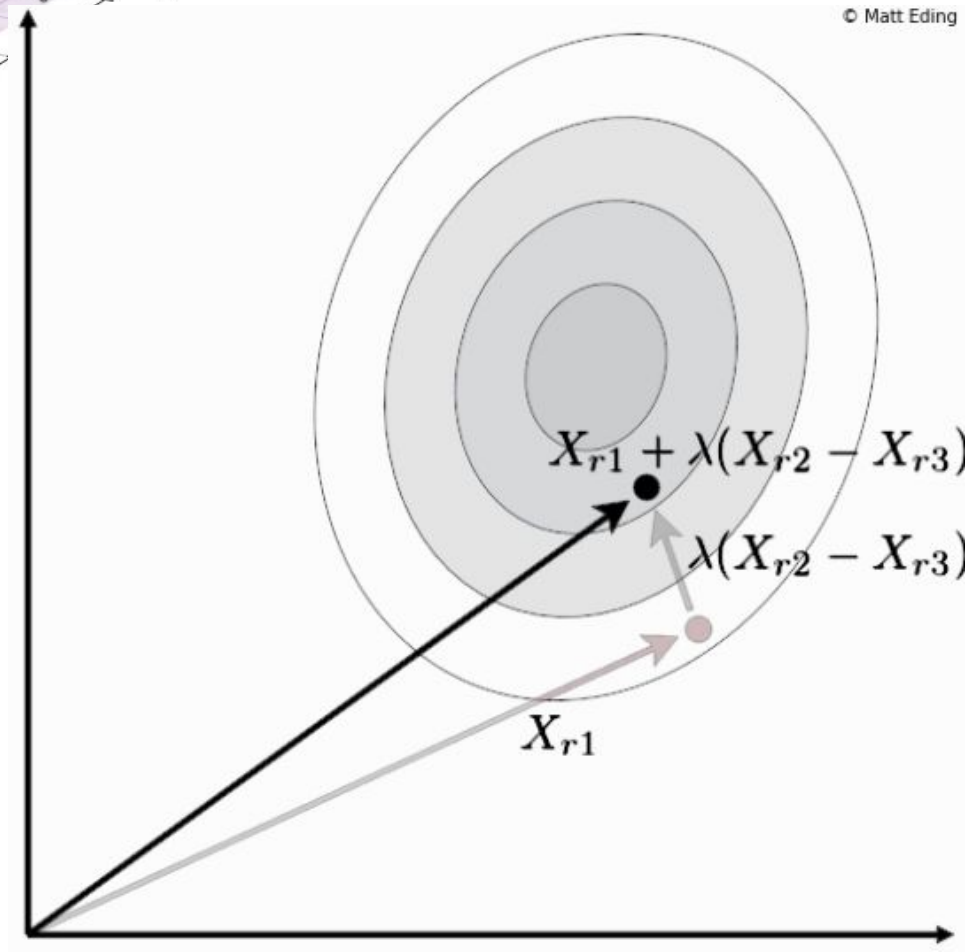

Content Image
Feature map
Pretrained Model
Pretrained Model
Pretrained Model
Pretrained Model
Output Image 1
Output Image 2
Style Image
Shuffle map channel
Style Loss
Content Loss
Total Loss

Style Tra on + DE

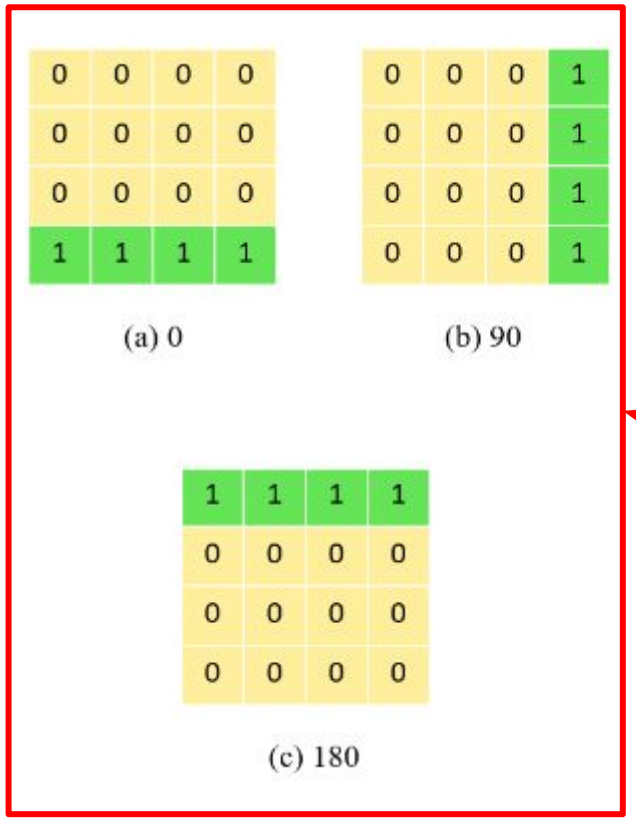
❖ Differential Evolution







Style Transfer with Rotation + DE



$$x_{new} = x + (x_{90} - x_{180})$$

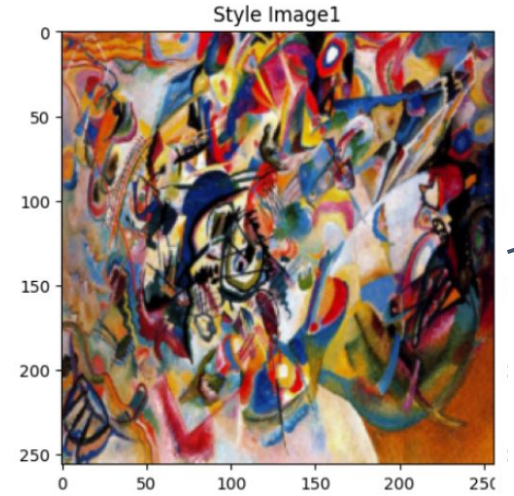
```python
def rot_style_features(style_features, style_layers):
    final_rot_style_features = {}
    for layer in style_layers:
        sf = style_features[layer].clone()
        rot90 = torch.rot90(sf.clone(), 1, (2, 3))
        rot180 = torch.rot90(rot90.clone(), 1, (2, 3))
        final_rot = sf + (rot90 - rot180)
        final_rot_style_features[layer] = final_rot
    return final_rot_style_features
```

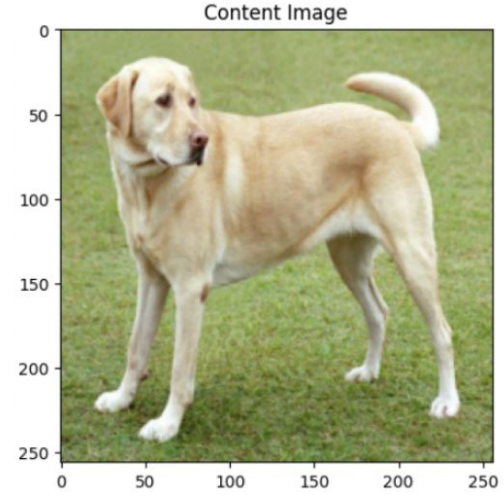
```python
content_features = get_features(VGG19_pretrained, content_img)
style_features1 = get_features(VGG19_pretrained, style_img)
final_rot_style_features = rot_style_features(style_features1, style_layers)
```

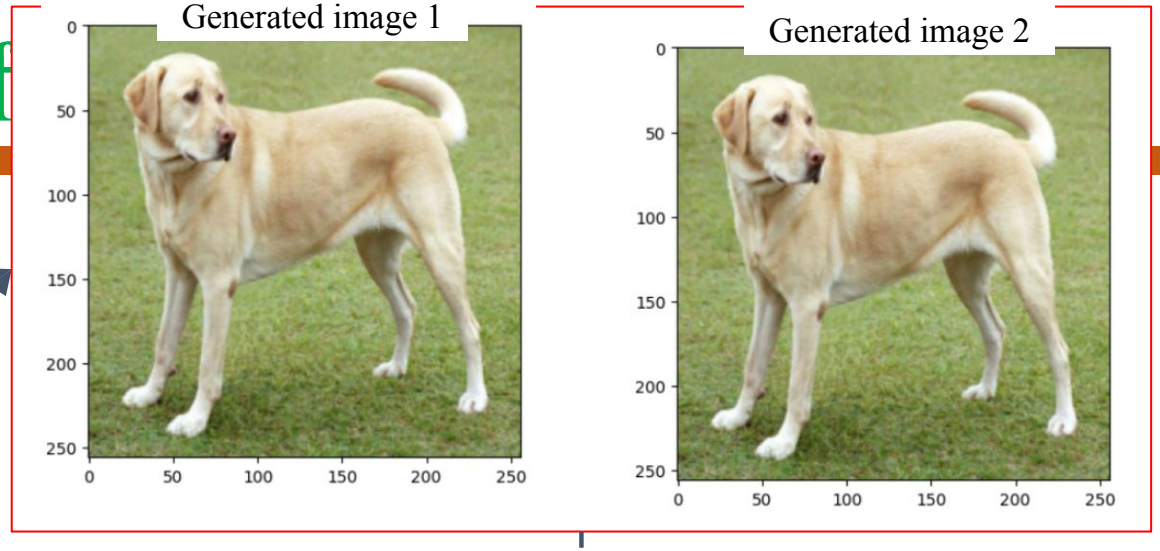
```python
import torch.optim as optim

target_img1 = content_img.clone().requires_grad_(True).to(device)
target_img2 = content_img.clone().requires_grad_(True).to(device)
optimizer1 = optim.Adam([target_img1], lr=0.02)
optimizer2 = optim.Adam([target_img2], lr=0.02)
```

```python
STEPS = 3000

for step in range(STEPS):

    total_loss1, content_loss1, style_loss1 = style_tranfer_(VGG19_pretrained, optimizer1, target_img1,
                                                             content_features, style_features1,
                                                             style_layers, content_weight, style_weight)

    total_loss2, content_loss2, style_loss2 = style_tranfer_(VGG19_pretrained, optimizer2, target_img2,
                                                             content_features, final_rot_style_features,
                                                             style_layers, content_weight, style_weight)

    if step % 100 == 99:
        print(f"Epoch [{step+1}/{STEPS}] Total loss1: {total_loss1.item():.6f} - \
              Content loss1: {content_loss1.item():.6f} - Style loss1: {style_loss1.item():.6f}")
        print(f"Epoch [{step+1}/{STEPS}] Total loss2: {total_loss2.item():.6f} - \
              Content loss1: {content_loss2.item():.6f} - Style loss2: {style_loss2.item():.6f}")

    with torch.no_grad():
        target_img1.clamp_(0, 1)
        target_img2.clamp_(0, 1)
```

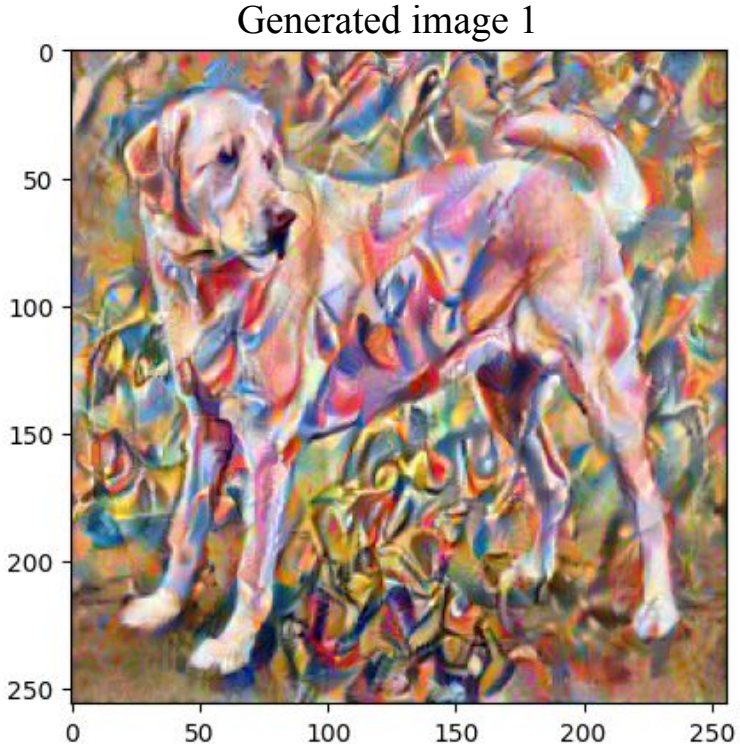
Style Transf

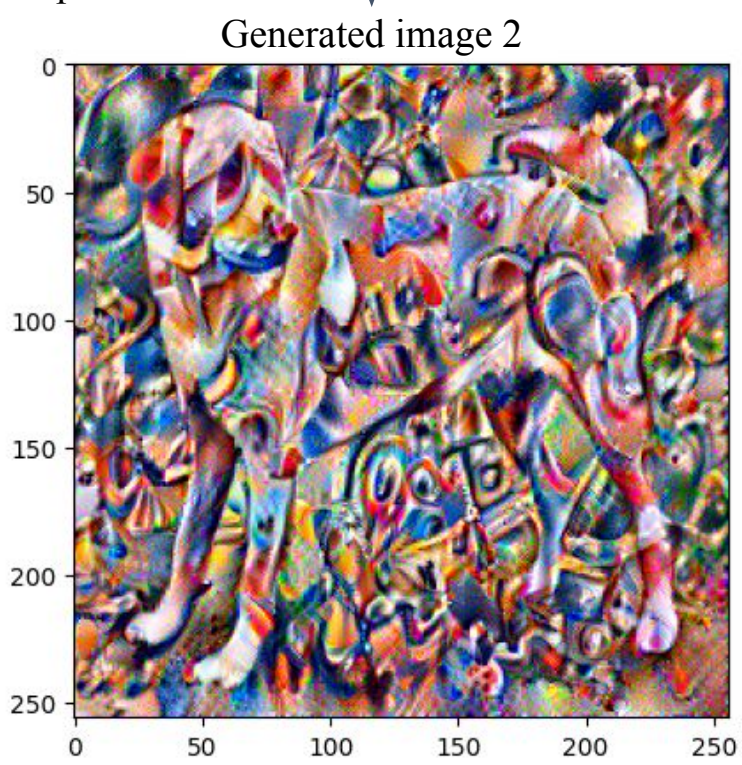






After 3000 steps





Summary

- ✔ Reviewed Style Transfer
- ✔ Reviewed fundamental necessary understanding
- ✔ Presented Style Transfer Examples
- ✔ Reviewed Milestone Papers
- ✔ Executed Style Transfer Using Two Style References
- ✔ Executed Style Transfer with Rotation + DE