AI VIETNAM
All-in-One Course

Style Transfer

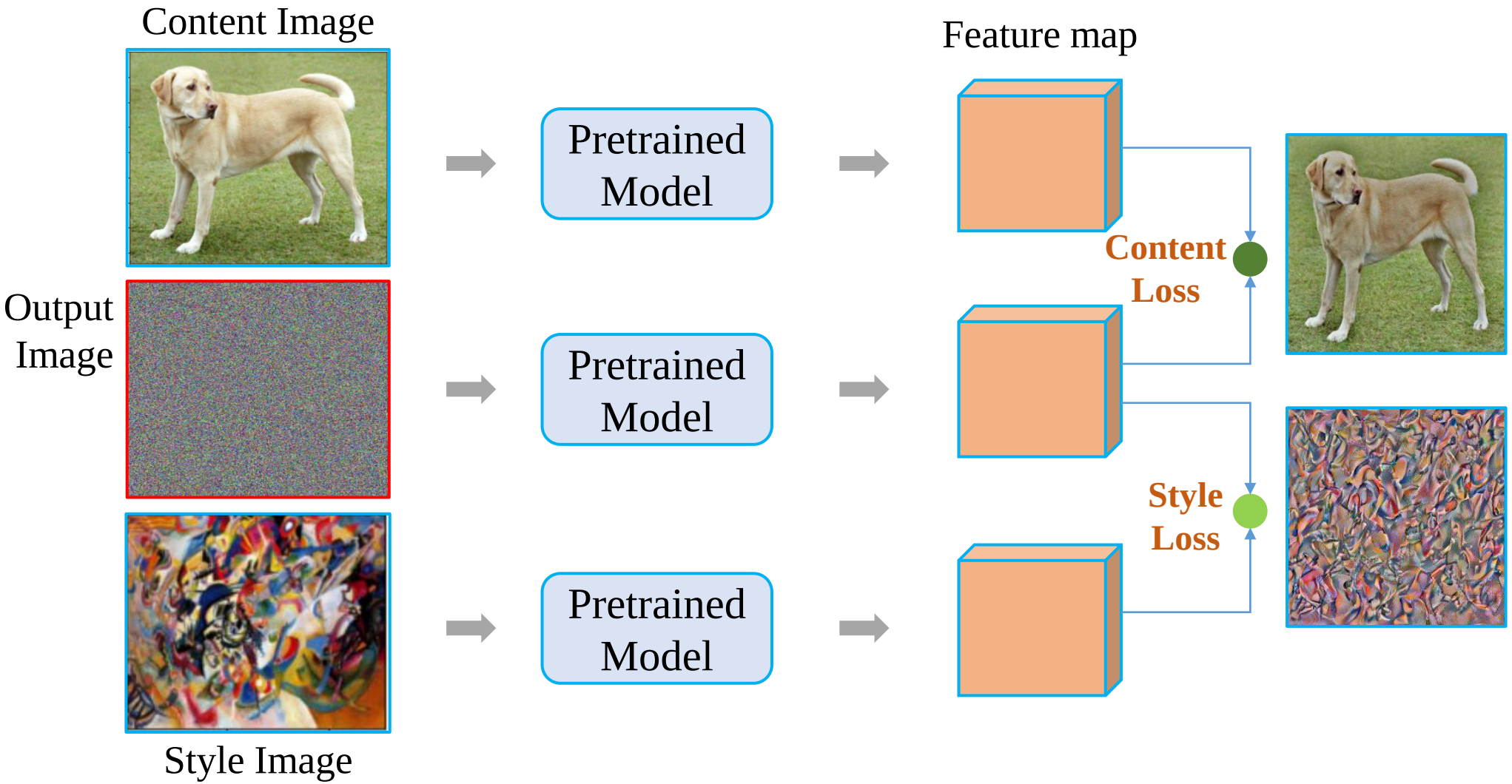
Understanding Content Loss and Style Loss

Quang-Vinh Dinh
Ph.D. in Computer Science

Year 2024

Objectives

- ✓ Understand deeply about content loss
- ✓ Understand deeply about style loss



Outline

- Introduction
- Inputs as Variables
- Content Loss
- Style Loss

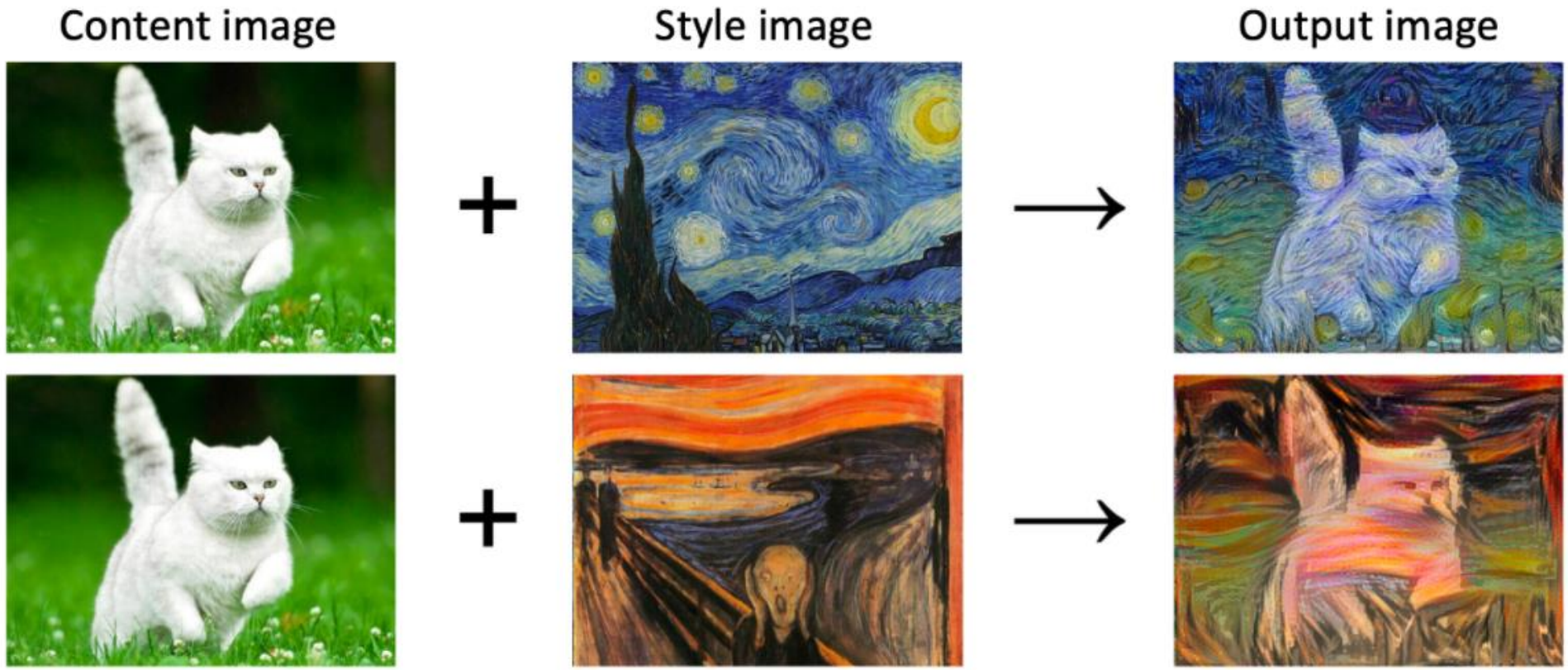
Style Transfer

❖ From different viewpoints

https://towardsdatascience.com/light-on-math-machine-learning-intuitive-guide-to-neural-style-transfer-ef88e46697ee

Style Transfer

❖ From different viewpoints



https://godatadriven.com/blog/how-to-style-transfer-your-own-images/

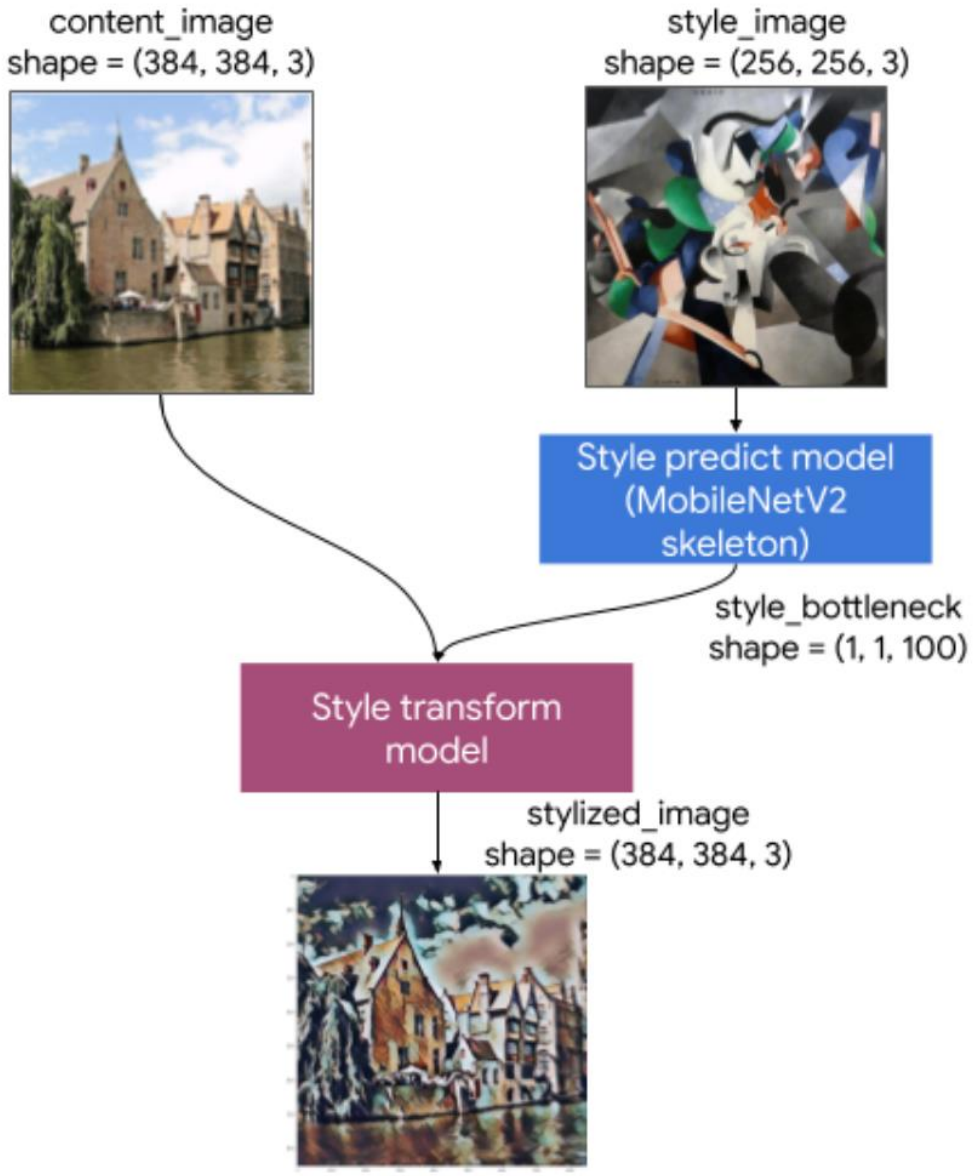
Style Transfer

❖ From different viewpoints

https://www.tensorflow.org/lite/examples/style_transfer/overview

Style Transfer

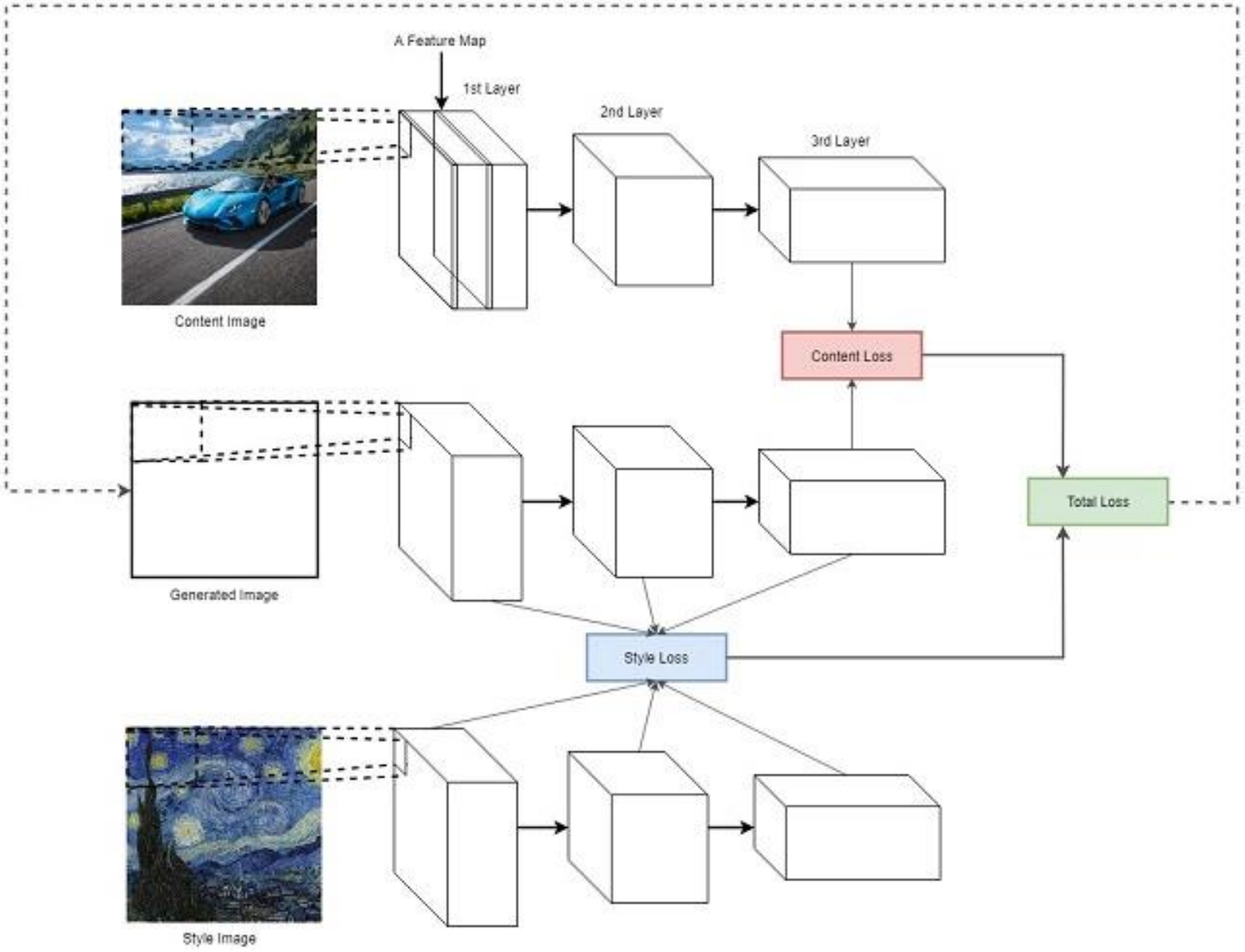
❖ From different viewpoints



https://www.tensorflow.org/lite/examples/style_transfer/overview

Style Transfer

From different viewpoints

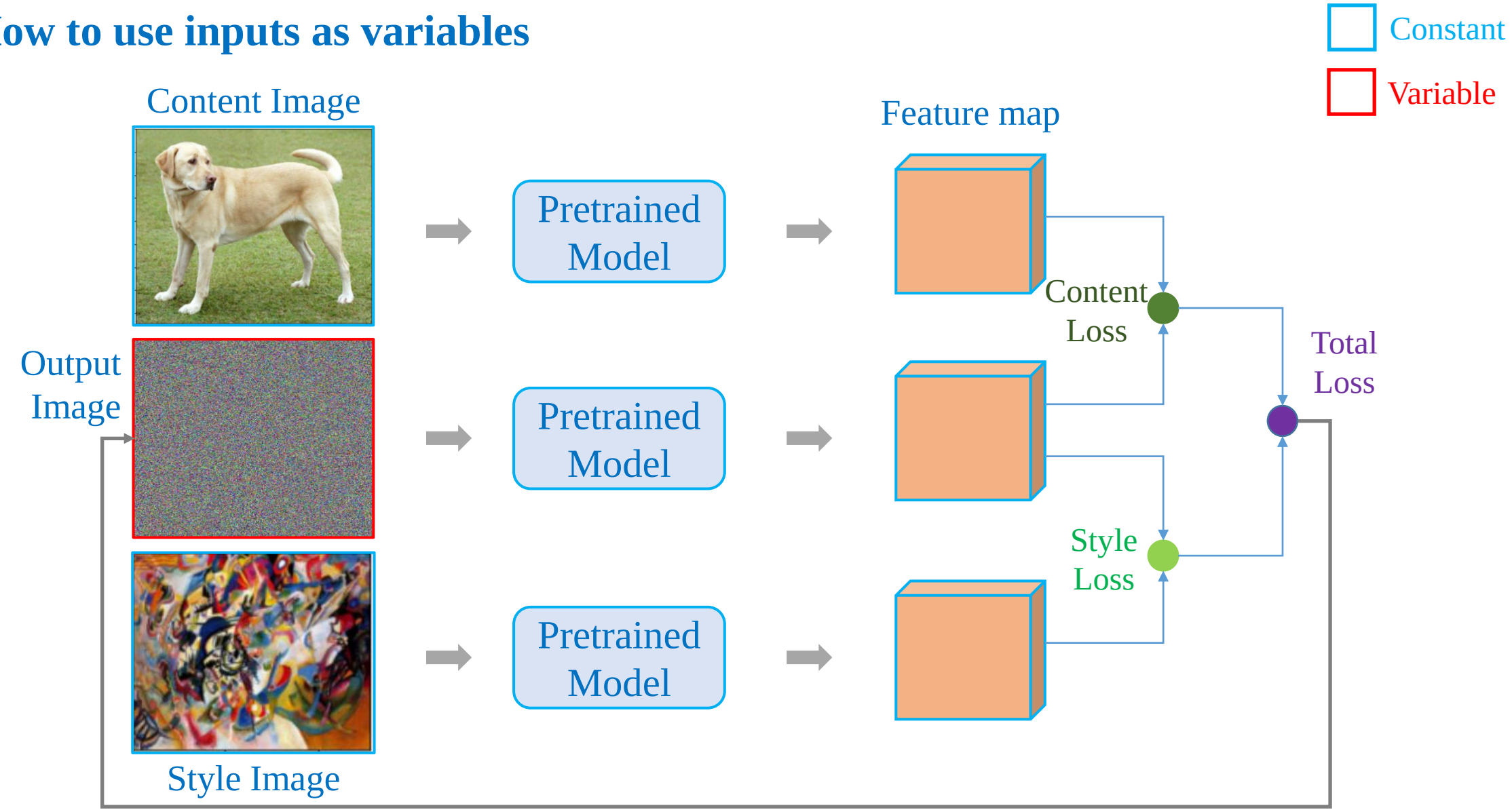


Outline

- Introduction
- Inputs as Variables
- Content Loss
- Style Loss

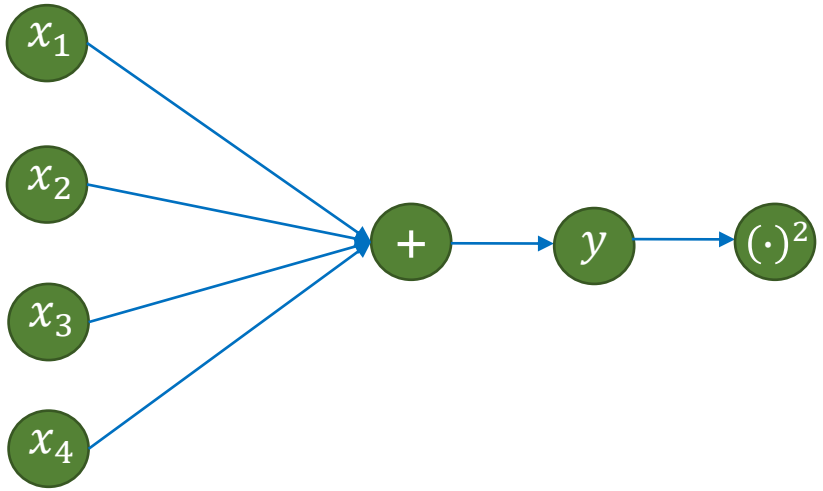
Style Transfer

❖ How to use inputs as variables



Inputs as Variables

- **Variable:** Represents a tensor whose value can be changed
- **Gradient computation**

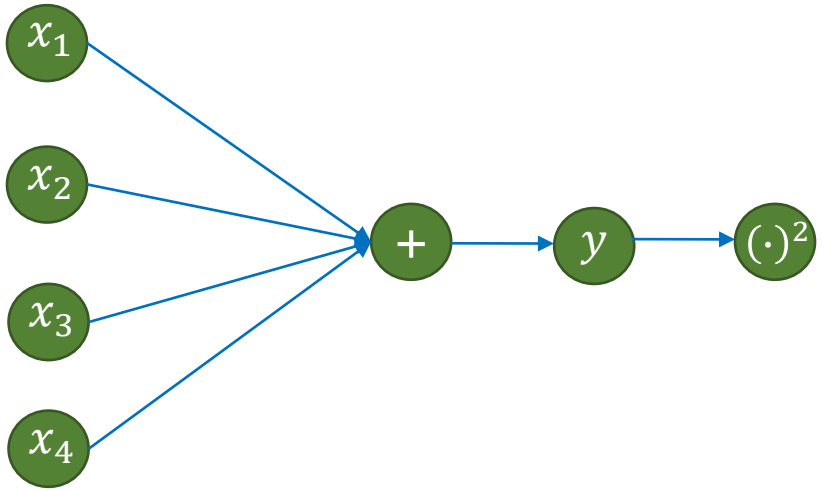



```
tensor([[1., 1.],
        [1., 1.]], requires_grad=True)
tensor([[8., 8.],
        [8., 8.]])
```

```python
import torch
from torch.autograd import Variable

x = torch.Tensor([[1.0, 1.0],
                  [1.0, 1.0]])

x = Variable(x, requires_grad=True)
y = x.sum()
z = y*y

# compute gradient
z.backward()

# print out
print(x)
print(x.grad)
```

Inputs as Variables

- Variable
- Gradient computation

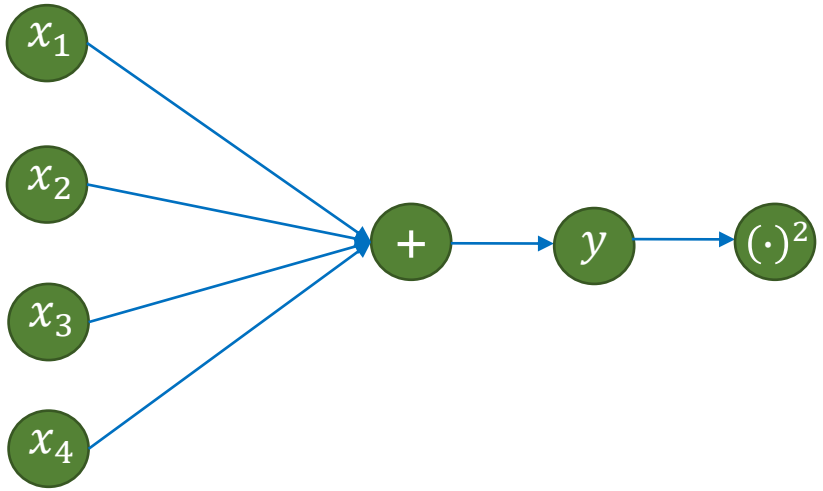



```
tensor([[0.2000, 0.2000],
        [0.2000, 0.2000]], requires_grad=True)
tensor([[8., 8.],
        [8., 8.]])
```

```python
import torch
from torch.autograd import Variable

x = torch.Tensor([[1.0, 1.0],
                  [1.0, 1.0]])

x = Variable(x, requires_grad=True)
y = x.sum()
z = y*y

# compute gradient
optimizer = optim.SGD([x], lr=0.1)
optimizer.zero_grad()
z.backward()
optimizer.step()

# print out
print(x)
print(x.grad)
```

Inputs as Variables

- Change to Variable
- Gradient computation



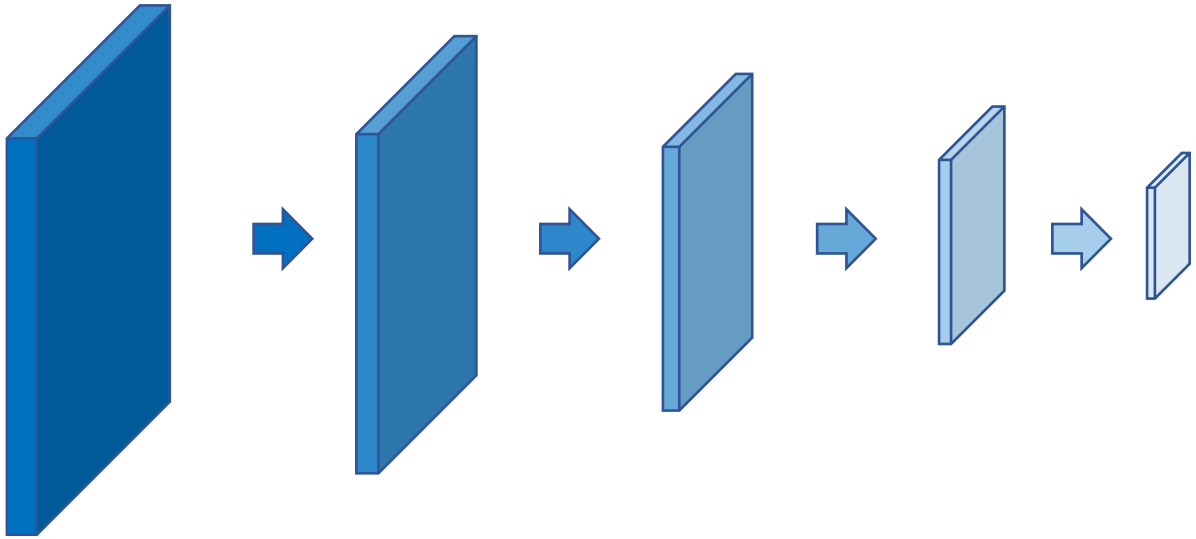

```
tensor([[0.2000, 0.2000],
        [0.2000, 0.2000]], requires_grad=True)
tensor([[8., 8.],
        [8., 8.]])
```

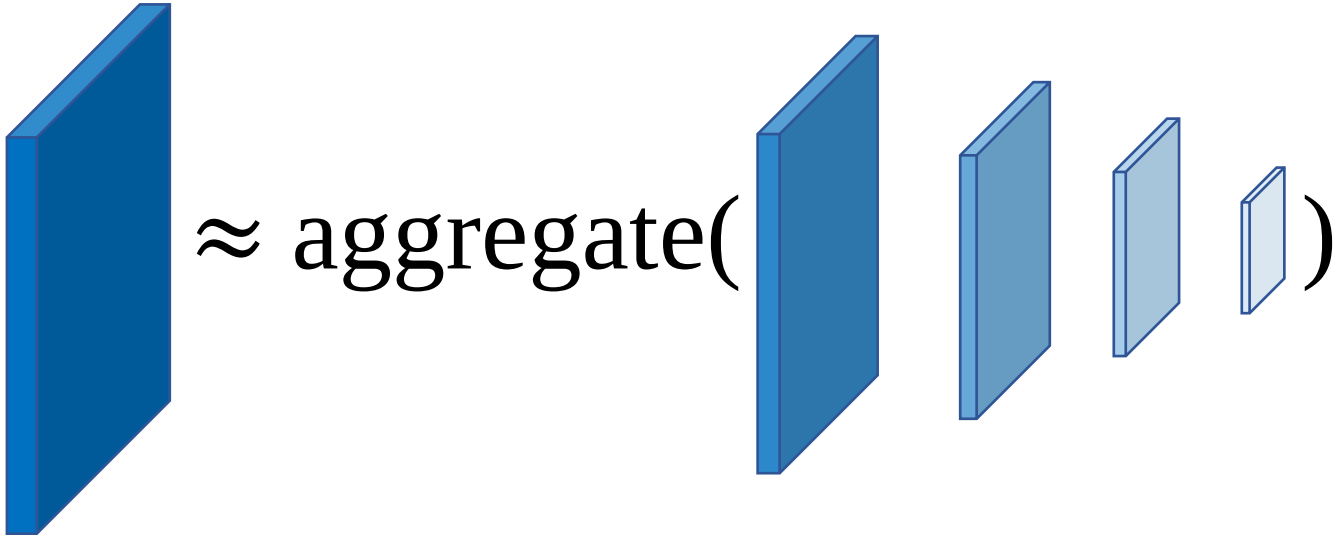
```python
import torch
import torch.optim as optim

x = torch.Tensor([[1.0, 1.0],
                  [1.0, 1.0]])
x = x.requires_grad_(True)

y = x.sum()
z = y*y

# compute gradient
optimizer = optim.SGD([x], lr=0.1)
optimizer.zero_grad()
z.backward()
optimizer.step()

# print out
print(x)
print(x.grad)
```

Outline

- Introduction
- Inputs as Variables
- Content Loss
- Style Loss

Local and Global Similarity





https://en.wikipedia.org/wiki/Twin

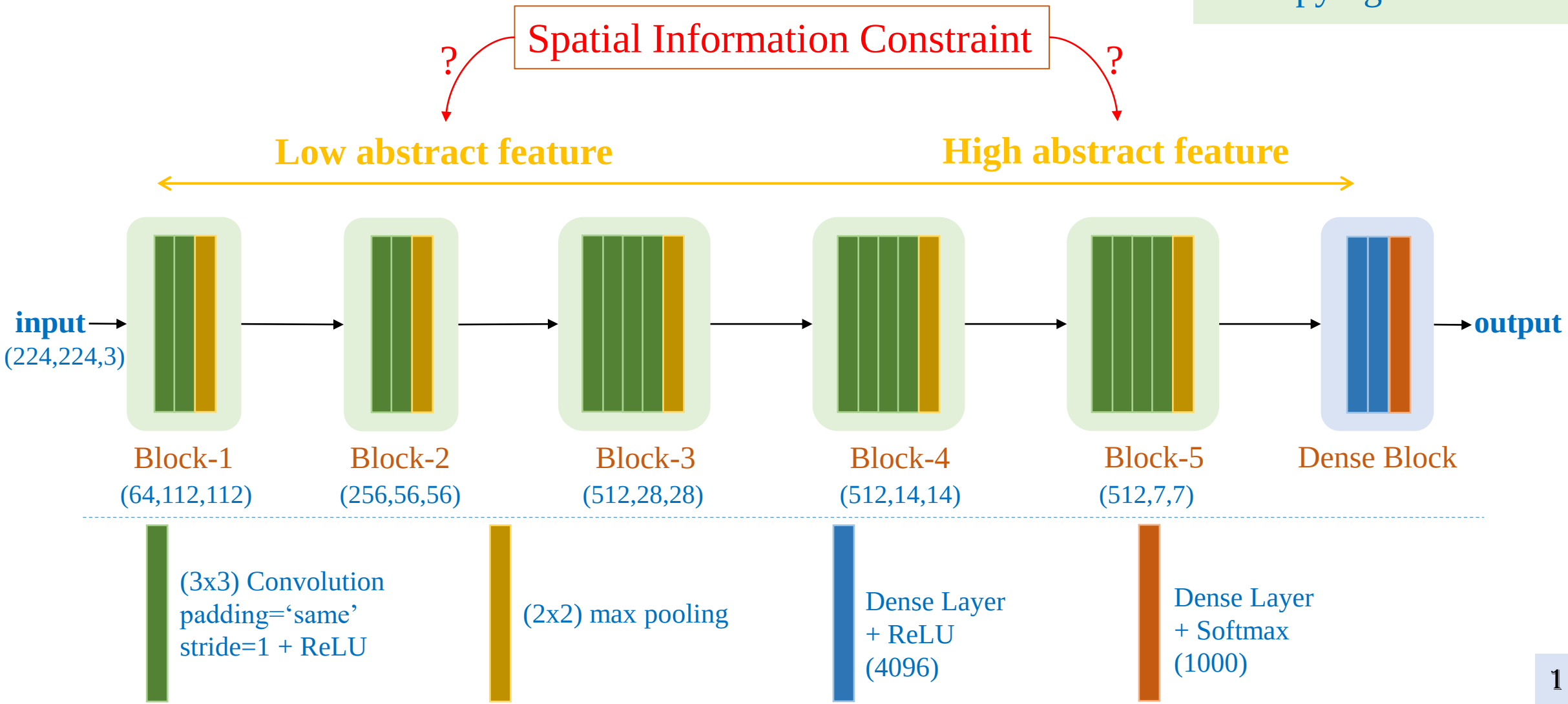
Style Transfer

Local and Global

Detection Problem

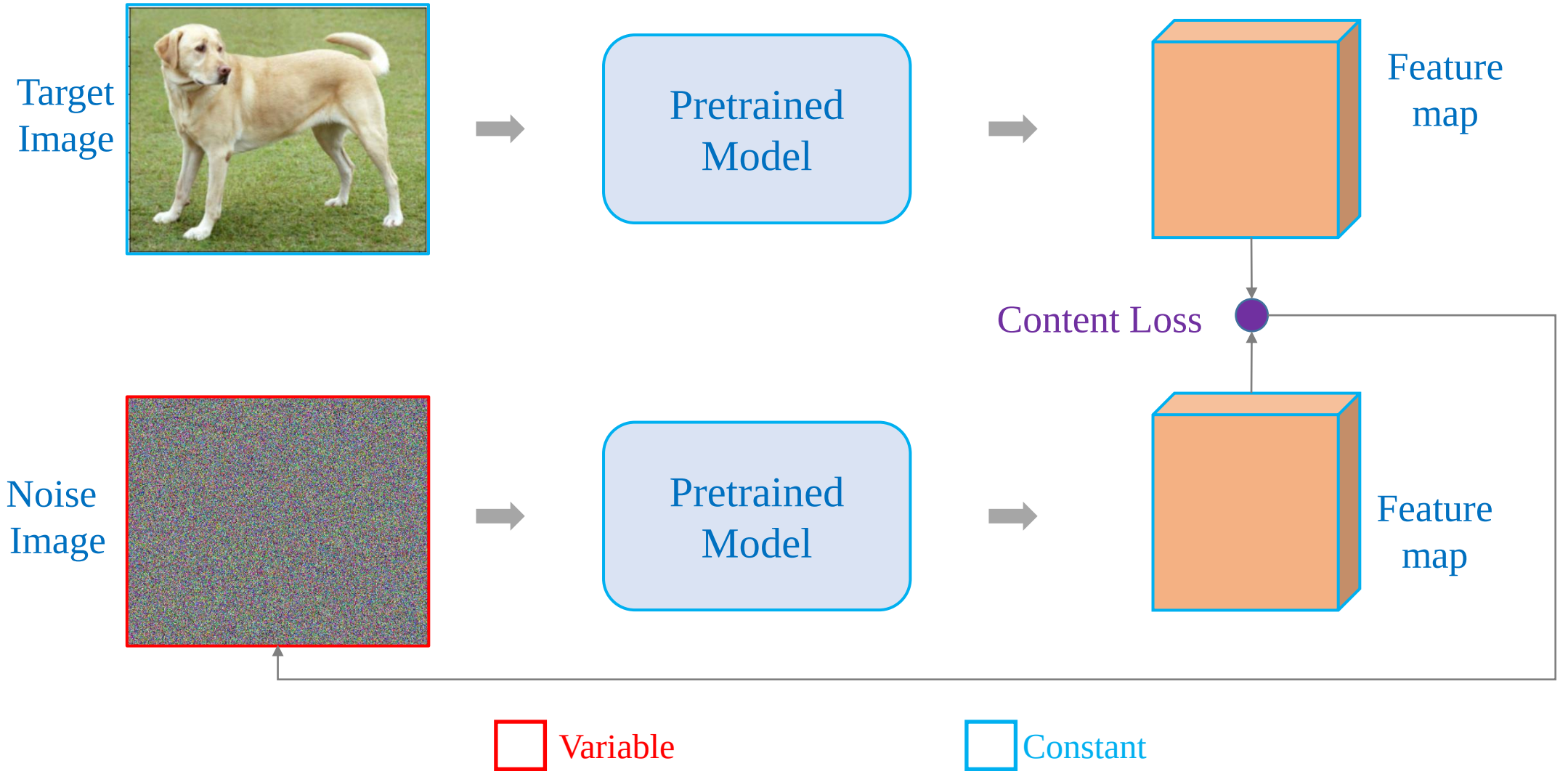
Copying Problem

❖ Content Loss: Using a pretrained model (VGG19)



Style Transfer

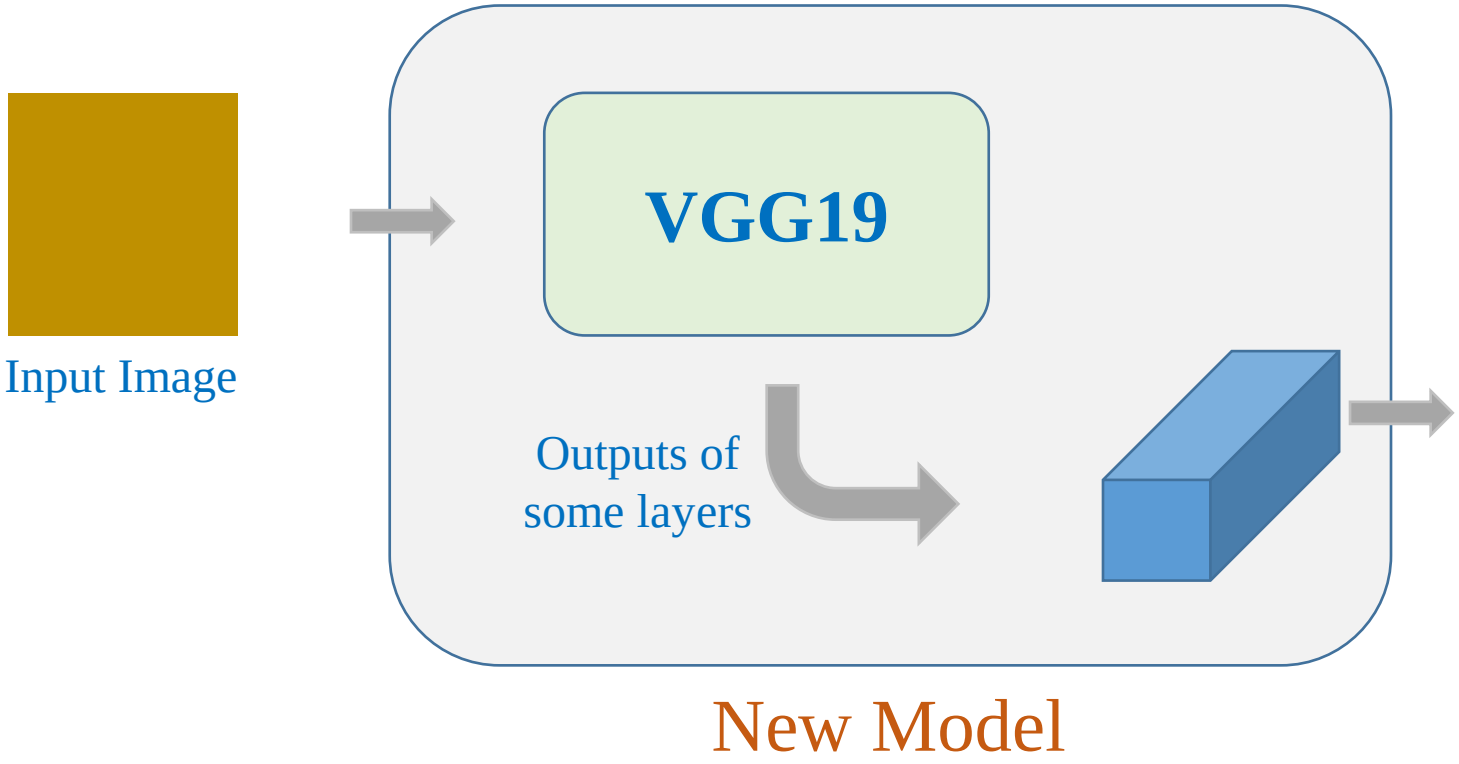
❖ Content Loss



Style Transfer

❖ Content Loss

❖ Create a model from some specific layers



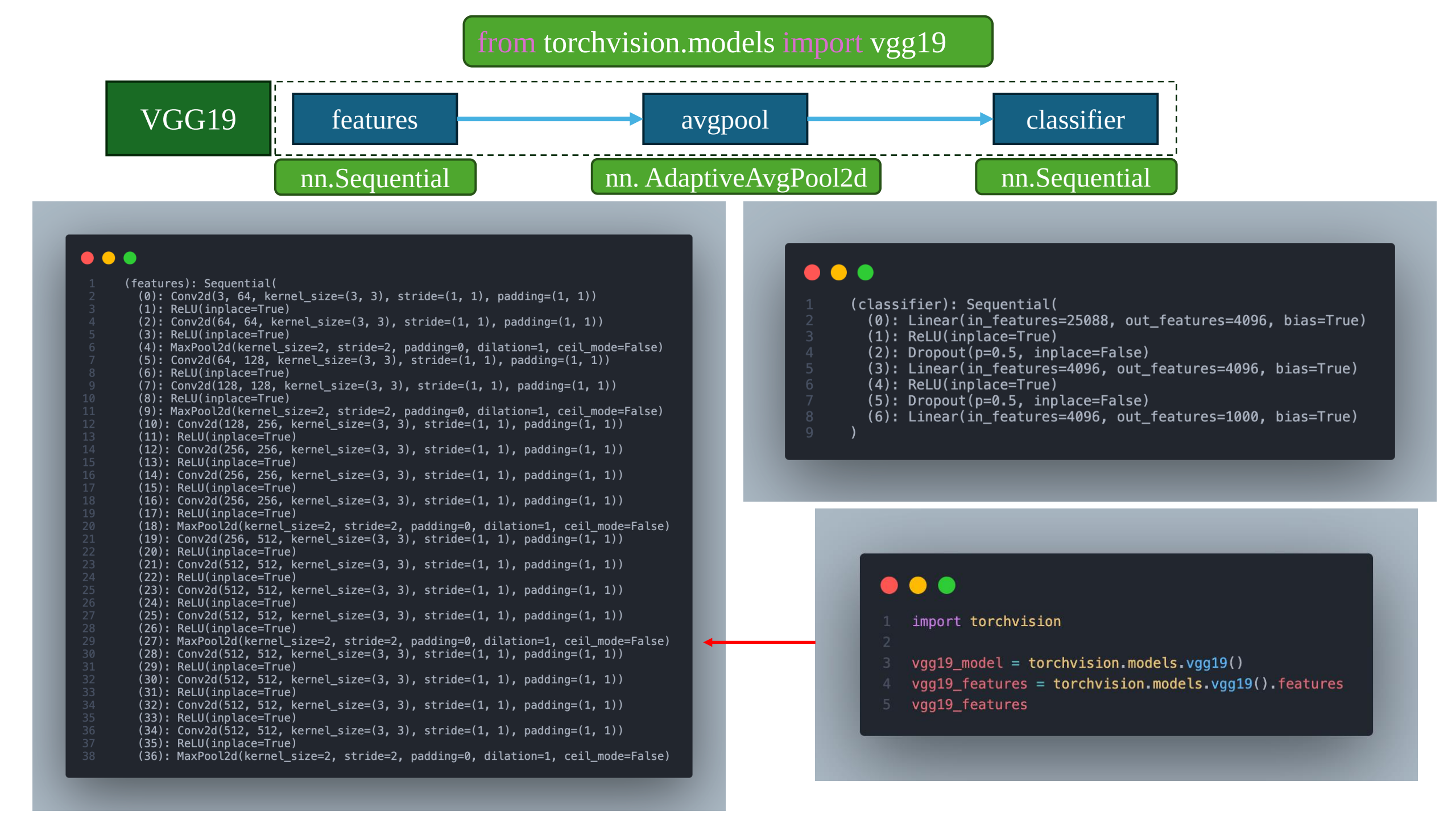
nn.Sequential

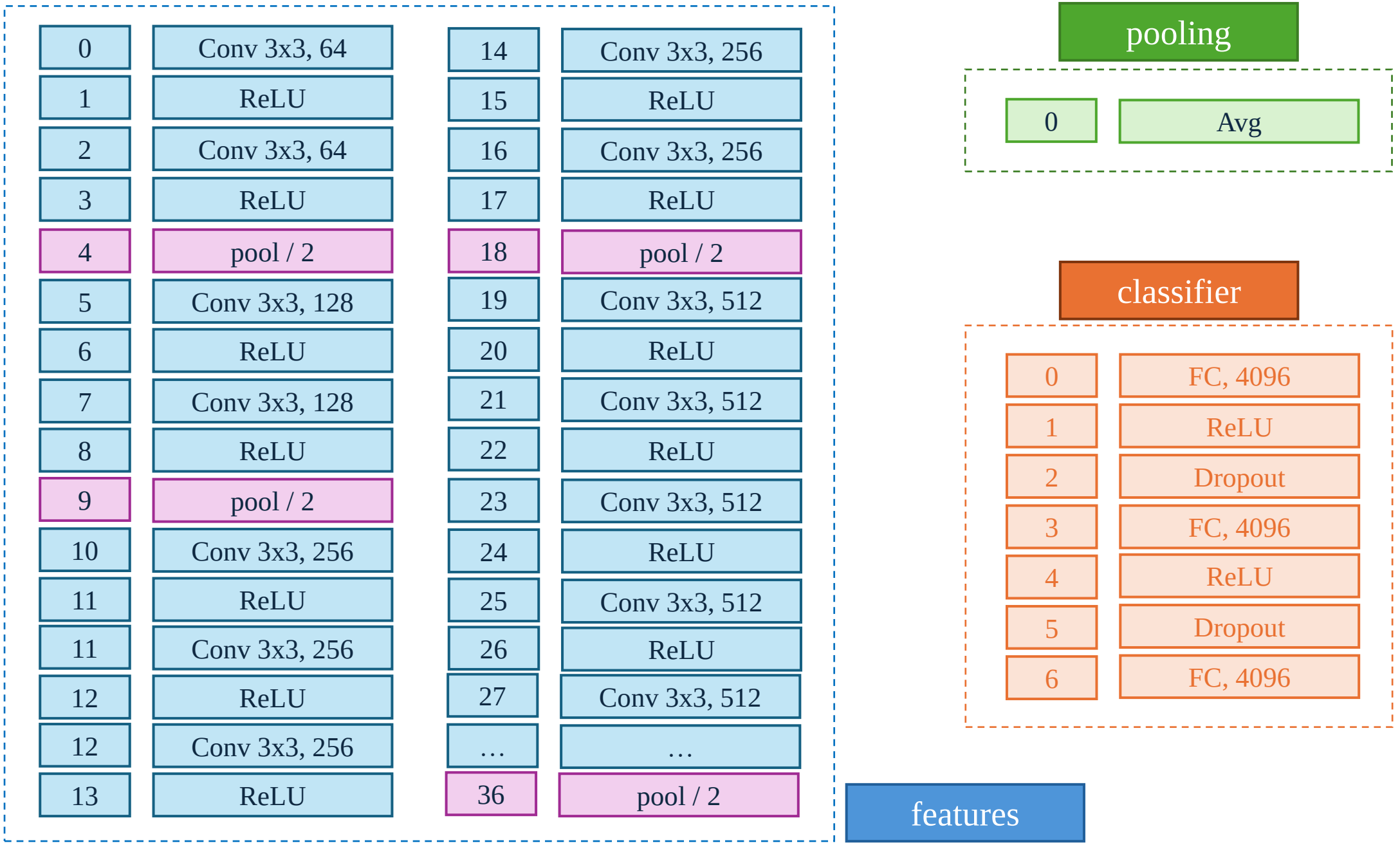
```python
import torch.nn as nn
model = nn.Sequential(
    nn.Conv2d(1, 20, 5),
    nn.ReLU(),
    nn.Conv2d(20, 64, 5),
    nn.ReLU()
)

>> Sequential(
>>     (0): Conv2d(1, 20, kernel_size=(5, 5), stride=(1, 1))
>>     (1): ReLU()
>>     (2): Conv2d(20, 64, kernel_size=(5, 5), stride=(1, 1))
>>     (3): ReLU()
>> )
```

```python
import torch.nn as nn
from typing import OrderedDict

model = nn.Sequential(OrderedDict([
          ('conv1', nn.Conv2d(1,20,5)),
          ('relu1', nn.ReLU()),
          ('conv2', nn.Conv2d(20,64,5)),
          ('relu2', nn.ReLU())
        ]))

>> Sequential(
>>     (conv1): Conv2d(1, 20, kernel_size=(5, 5), stride=(1, 1))
>>     (relu1): ReLU()
>>     (conv2): Conv2d(20, 64, kernel_size=(5, 5), stride=(1, 1))
>>     (relu2): ReLU()
>> )
```

```python
for name, layer in model._modules.items():
    print(f"Layer name : {name} - Module: {layer}")

>> Layer name : 0 - Module: Conv2d(1, 20, kernel_size=(5, 5), stride=(1, 1))
>> Layer name : 1 - Module: ReLU()
>> Layer name : 2 - Module: Conv2d(20, 64, kernel_size=(5, 5), stride=(1, 1))
>> Layer name : 3 - Module: ReLU()
```

```python
for name, layer in model._modules.items():
    print(f"Layer name : {name} - Module: {layer}")

>> Layer name : conv1 - Module: Conv2d(1, 20, kernel_size=(5, 5), stride=(1, 1))
>> Layer name : relu1 - Module: ReLU()
>> Layer name : conv2 - Module: Conv2d(20, 64, kernel_size=(5, 5), stride=(1, 1))
>> Layer name : relu2 - Module: ReLU()
```


from torchvision.models import vgg19
VGG19
features
avgpool
classifier
nn.Sequential
nn. AdaptiveAvgPool2d
nn.Sequential
(features): Sequential(
(0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(1): ReLU(inplace=True)
(2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(3): ReLU(inplace=True)
(4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
(5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(6): ReLU(inplace=True)
(7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(8): ReLU(inplace=True)
(9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
(10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(11): ReLU(inplace=True)
(12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(13): ReLU(inplace=True)
(14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(15): ReLU(inplace=True)
(16): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(17): ReLU(inplace=True)
(18): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
(19): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(20): ReLU(inplace=True)
(21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(22): ReLU(inplace=True)
(23): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(24): ReLU(inplace=True)
(25): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(26): ReLU(inplace=True)
(27): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
(28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(29): ReLU(inplace=True)
(30): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(31): ReLU(inplace=True)
(32): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(33): ReLU(inplace=True)
(34): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(35): ReLU(inplace=True)
(36): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
(classifier): Sequential(
(0): Linear(in_features=25088, out_features=4096, bias=True)
(1): ReLU(inplace=True)
(2): Dropout(p=0.5, inplace=False)
(3): Linear(in_features=4096, out_features=4096, bias=True)
(4): ReLU(inplace=True)
(5): Dropout(p=0.5, inplace=False)
(6): Linear(in_features=4096, out_features=1000, bias=True)
)
import torchvision
vgg19_model = torchvision.models.vgg19()
vgg19_features = torchvision.models.vgg19().features
vgg19_features

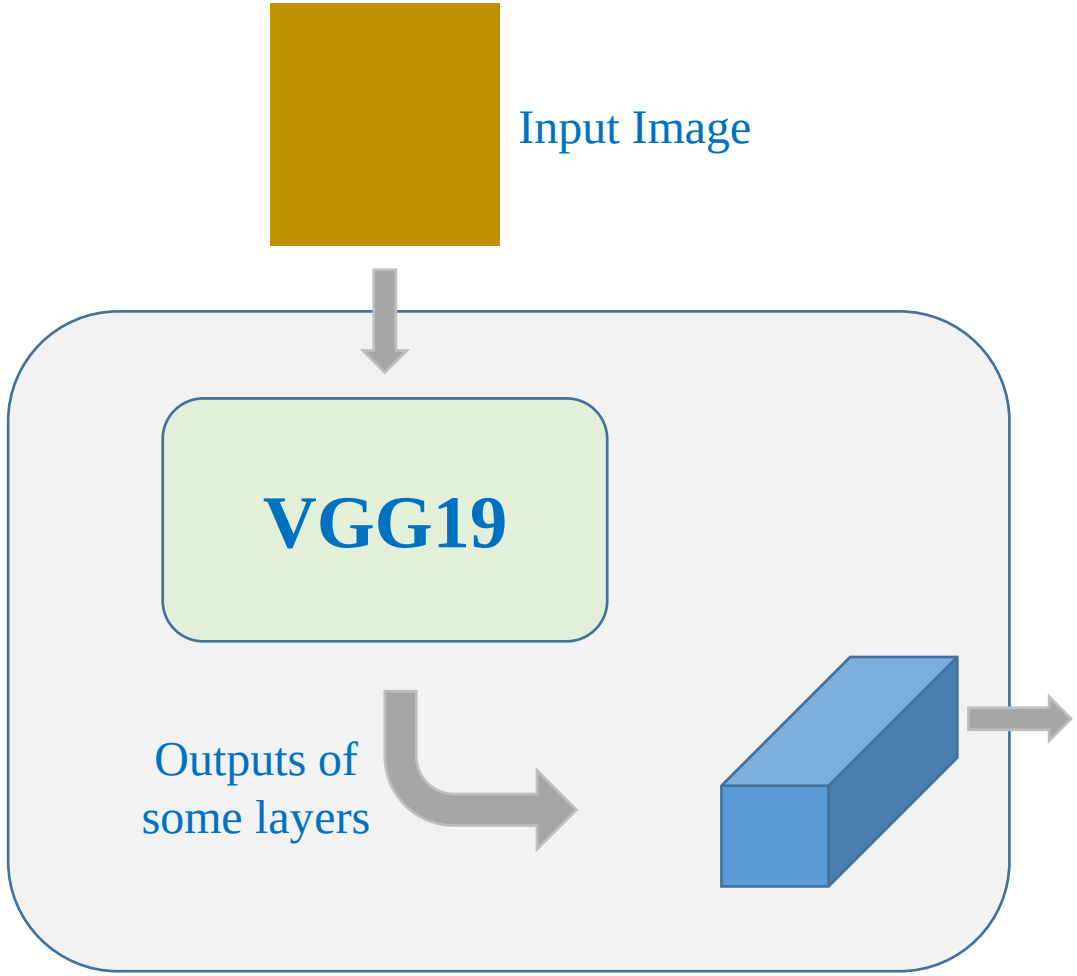

0 Conv 3x3, 64
1 ReLU
2 Conv 3x3, 64
3 ReLU
4 pool / 2
5 Conv 3x3, 128
6 ReLU
7 Conv 3x3, 128
8 ReLU
9 pool / 2
10 Conv 3x3, 256
11 ReLU
11 Conv 3x3, 256
12 ReLU
12 Conv 3x3, 256
13 ReLU
14 Conv 3x3, 256
15 ReLU
16 Conv 3x3, 256
17 ReLU
18 pool / 2
19 Conv 3x3, 512
20 ReLU
21 Conv 3x3, 512
22 ReLU
23 Conv 3x3, 512
24 ReLU
25 Conv 3x3, 512
26 ReLU
27 Conv 3x3, 512
… …
36 pool / 2
features
pooling
0 Avg
classifier
0 FC, 4096
1 ReLU
2 Dropout
3 FC, 4096
4 ReLU
5 Dropout
6 FC, 4096

Style Transfer

❖ Content Loss: Demo

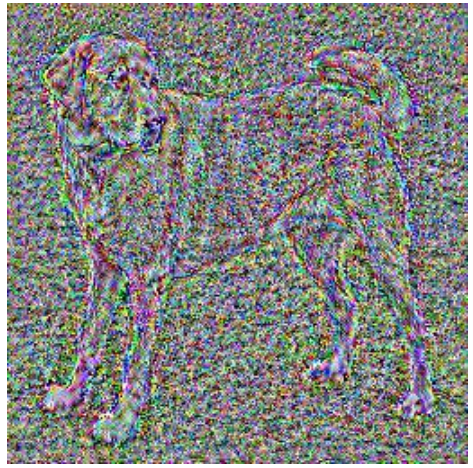
❖ Some utility functions

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

imsize = 256
img_transforms = transforms.Compose([
    transforms.Resize((imsize, imsize)),
    transforms.ToTensor()
])

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

content_img = image_loader("content_img.jpg")
```

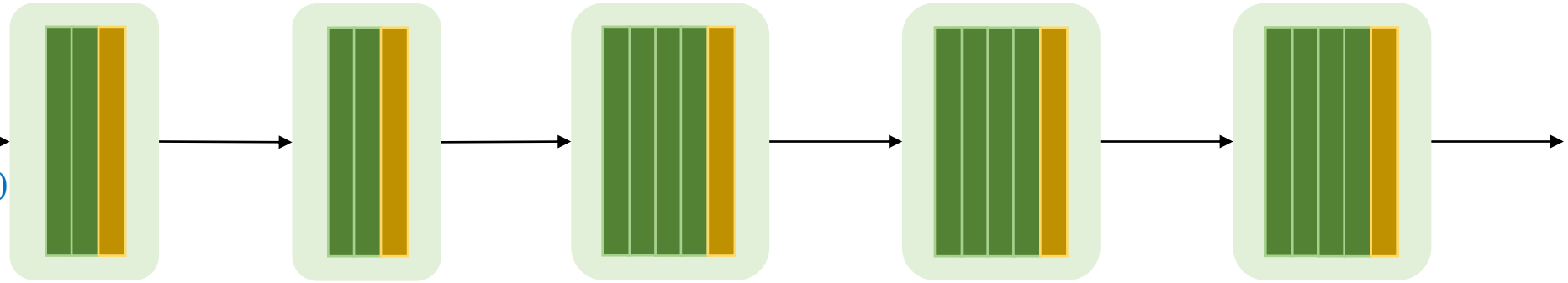


❖ Content Loss

❖ Using different features



input
(224,224,3)



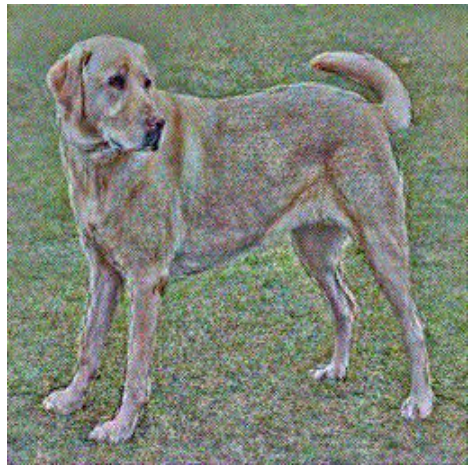
Block-1
(64,112,112)

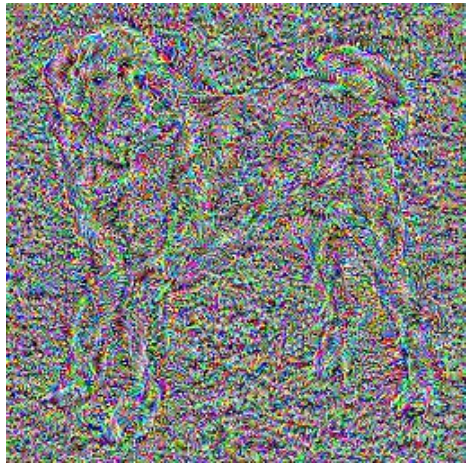
Block-2
(256,56,56)

Block-3
(512,28,28)

Block-4
(512,14,14)

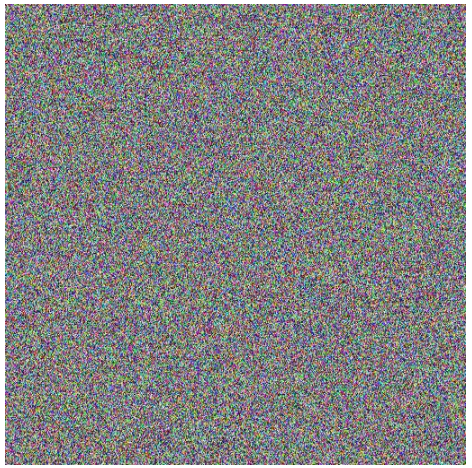
Block-5
(512,7,7)

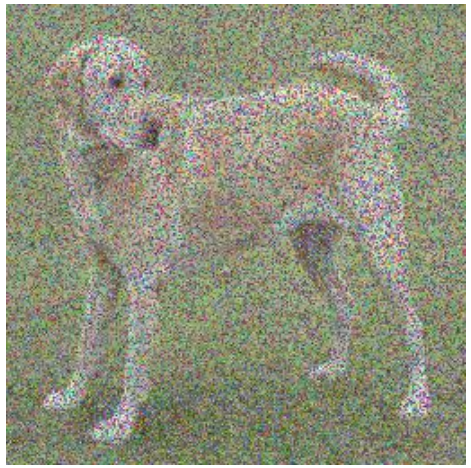




Style Transfer

❖ Content Loss: Using the Content Image


Initial Image


Epoch 11


Epoch 21


Epoch 31


Epoch 41

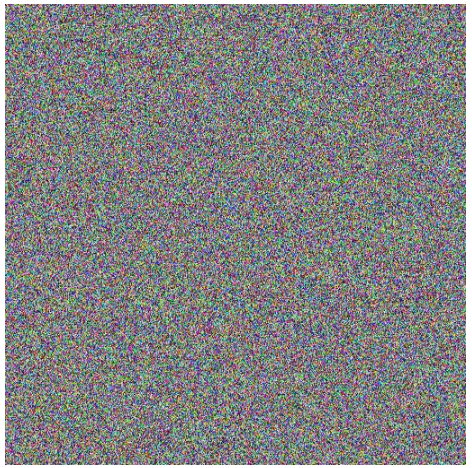

Epoch 71


Epoch 91


Target Image

Style Transfer

❖ Content Loss: Using Conv1


Initial Image


Epoch 11


Epoch 21

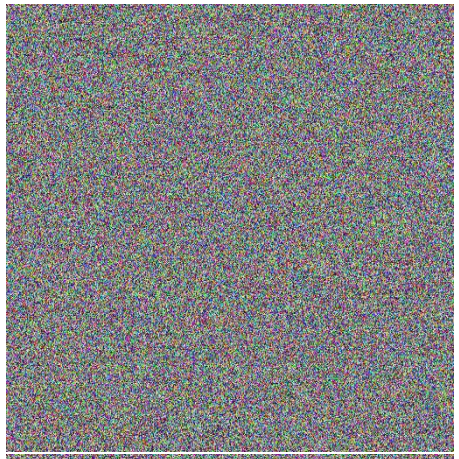
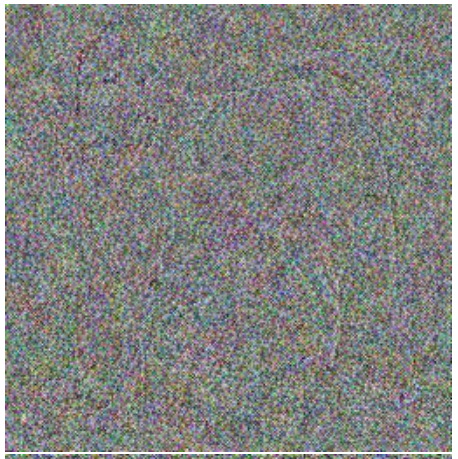
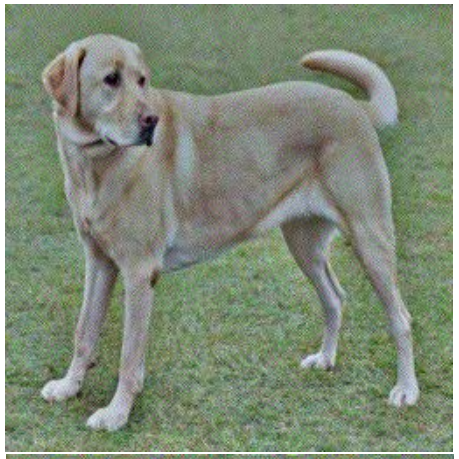
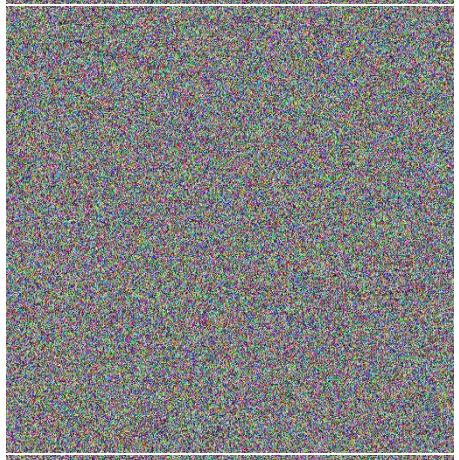
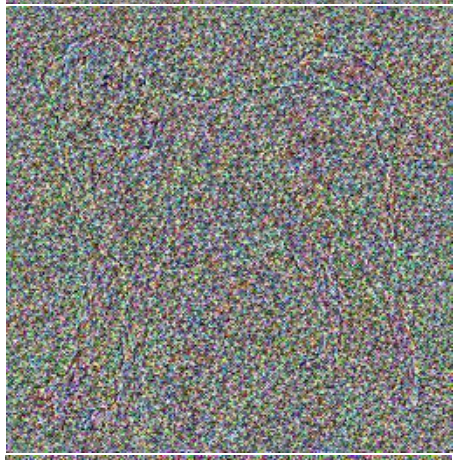
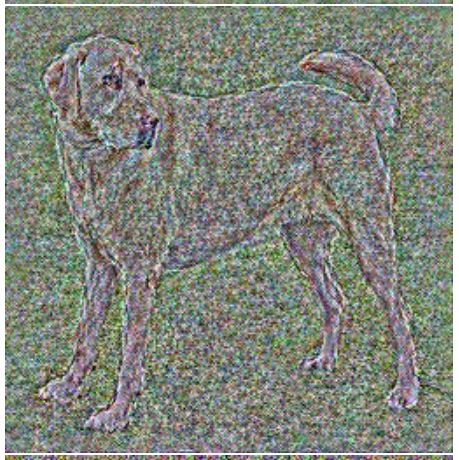
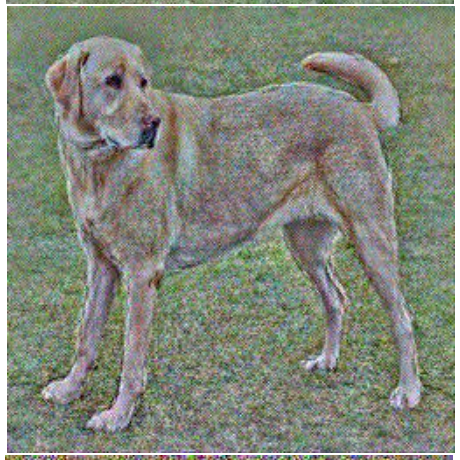
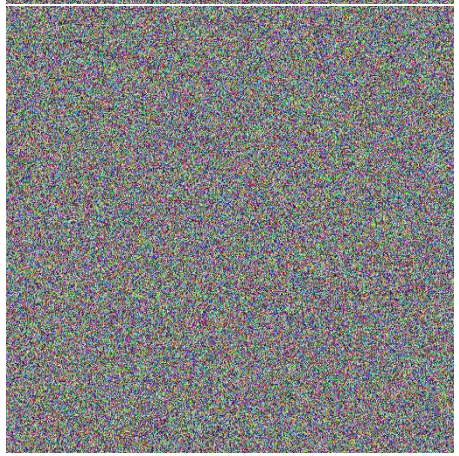
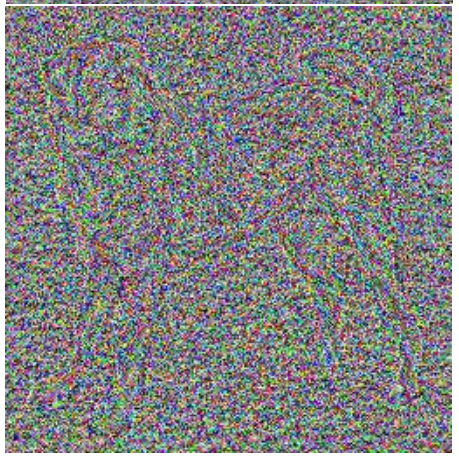
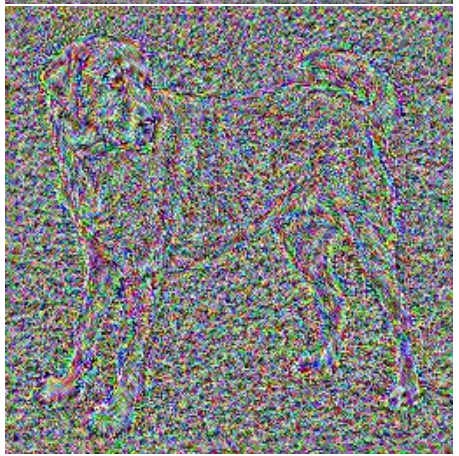
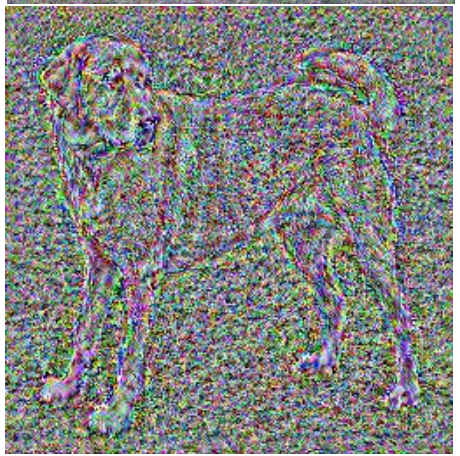

Epoch 31


Epoch 41


Epoch 71

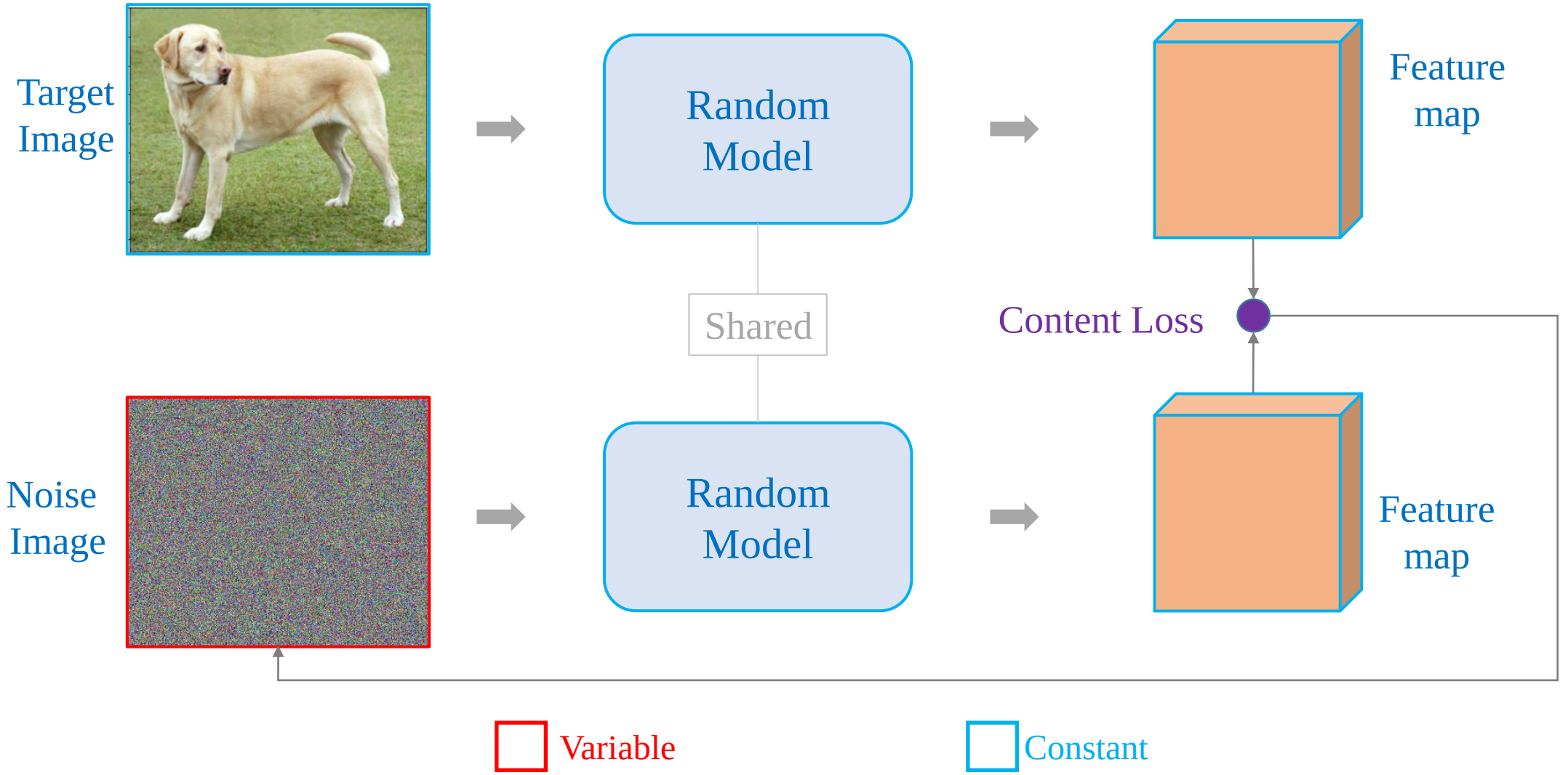

Epoch 91


Target Image

	Initial Image	Epoch 11	Epoch 41	Epoch 91
Using Conv2				
Using Conv3				
Using Conv4				

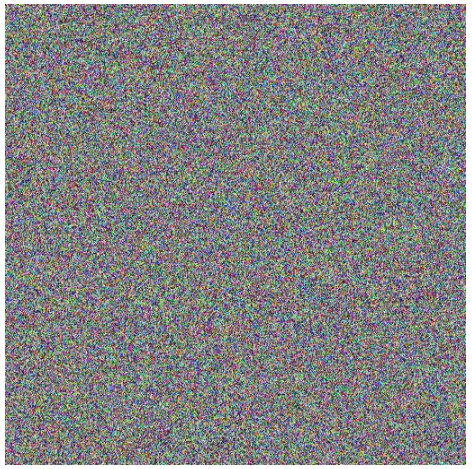
Style Transfer

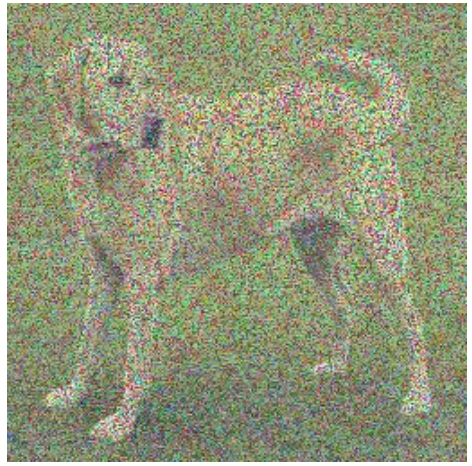
❖ Content Loss: Using RandomNet



Style Transfer

❖ **Content Loss: Using RandomNet and the conv1 features (Why?)**


Initial Image


Epoch 11


Epoch 21


Epoch 31


Epoch 41

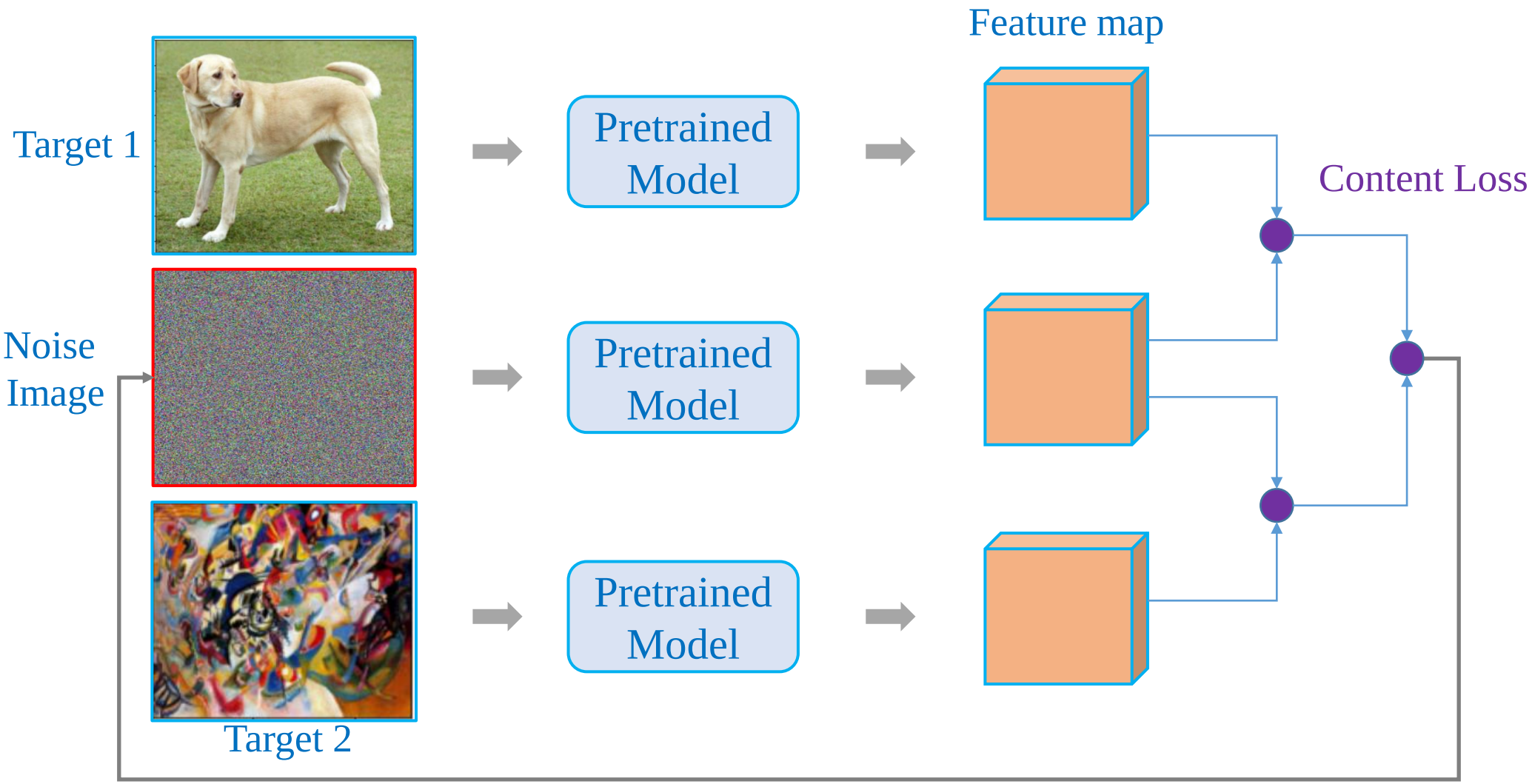

Epoch 71


Epoch 91


Target Image

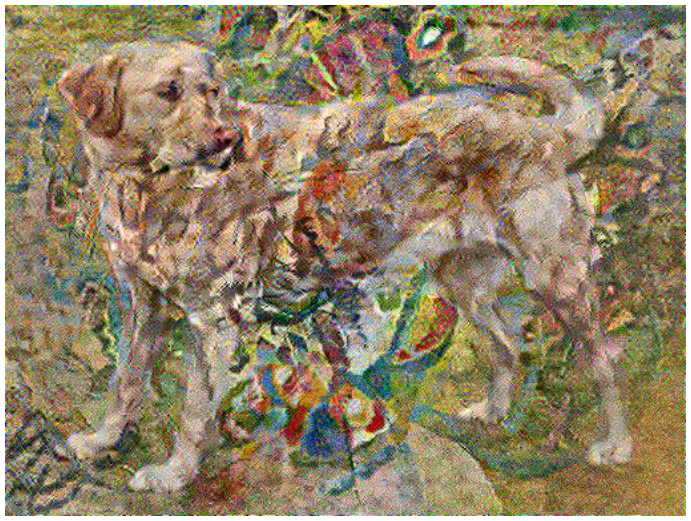
Style Transfer

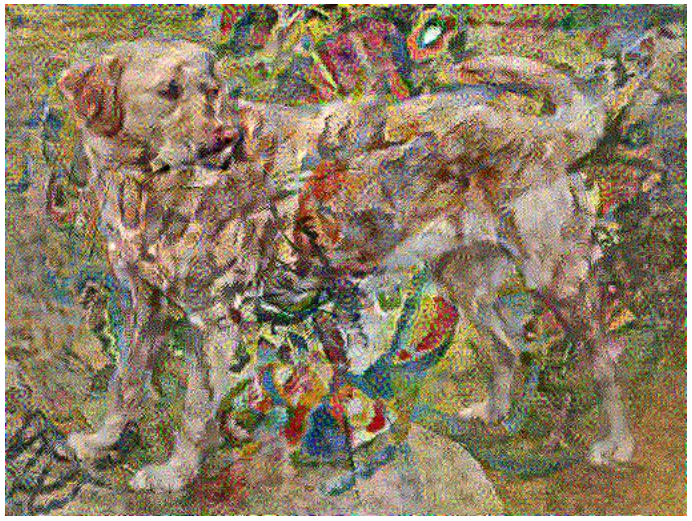
❖ Content Loss: Using Two Target



Style Transfer

❖ Content Loss: Using Two Target





Outline

- Introduction
- Inputs as Variables
- Content Loss
- Style Loss

Vector Operations

Hadamard product

$$\vec{v} = \begin{bmatrix} v_1 \\ \dots \\ v_n \end{bmatrix} \qquad \vec{u} = \begin{bmatrix} u_1 \\ \dots \\ u_n \end{bmatrix}$$

$$\vec{v} \odot \vec{u} = \begin{bmatrix} v_1 \\ \dots \\ v_n \end{bmatrix} \odot \begin{bmatrix} u_1 \\ \dots \\ u_n \end{bmatrix} = \begin{bmatrix} v_1 \times u_1 \\ \dots \\ v_n \times u_n \end{bmatrix}$$

```python
def Hadamard_product(vector1, vector2):
    '''
    Compute Hadamard product between two vectors
    Output is a vector
    '''
    return [v1*v2 for v1, v2 in zip(vector1, vector2)]

# test case
vector1 = [1, 2]
vector2 = [3, 4]

output = Hadamard_product(vector1, vector2)
print(output)
```

```
[3, 8]
```

Vector Operations

Dot product

$$\vec{v} = \begin{bmatrix} v_1 \\ \dots \\ v_n \end{bmatrix} \qquad \vec{u} = \begin{bmatrix} u_1 \\ \dots \\ u_n \end{bmatrix}$$

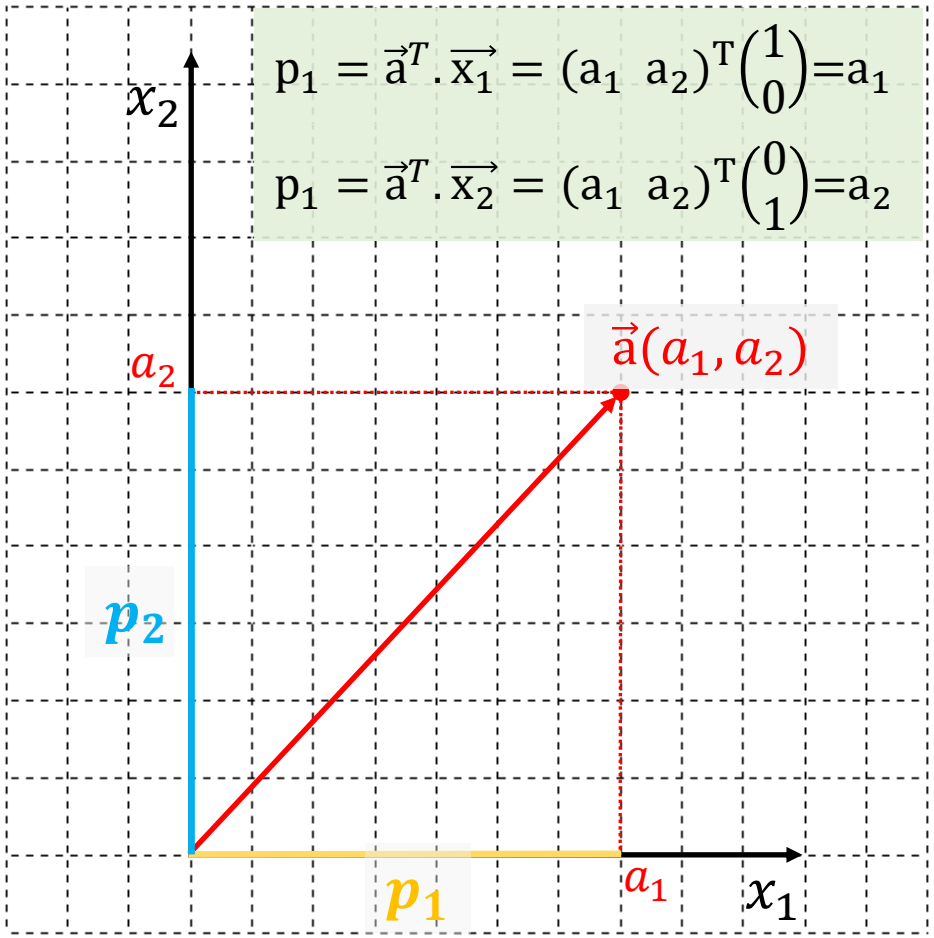
$$\vec{v} \cdot \vec{u} = v_1 \times u_1 + \dots + v_n \times u_n$$

```python
def dot_product(vector1, vector2):
    '''
    Compute dot product between two vectors
    Output is a floating-point number
    '''

    return sum([v1*v2 for v1, v2 in zip(vector1, vector2)])

# test case
vector1 = [1, 2, 3]
vector2 = [2, 3, 4]

ouptut = dot_product(vector1, vector2)
print(ouptut)
```

```
20
```

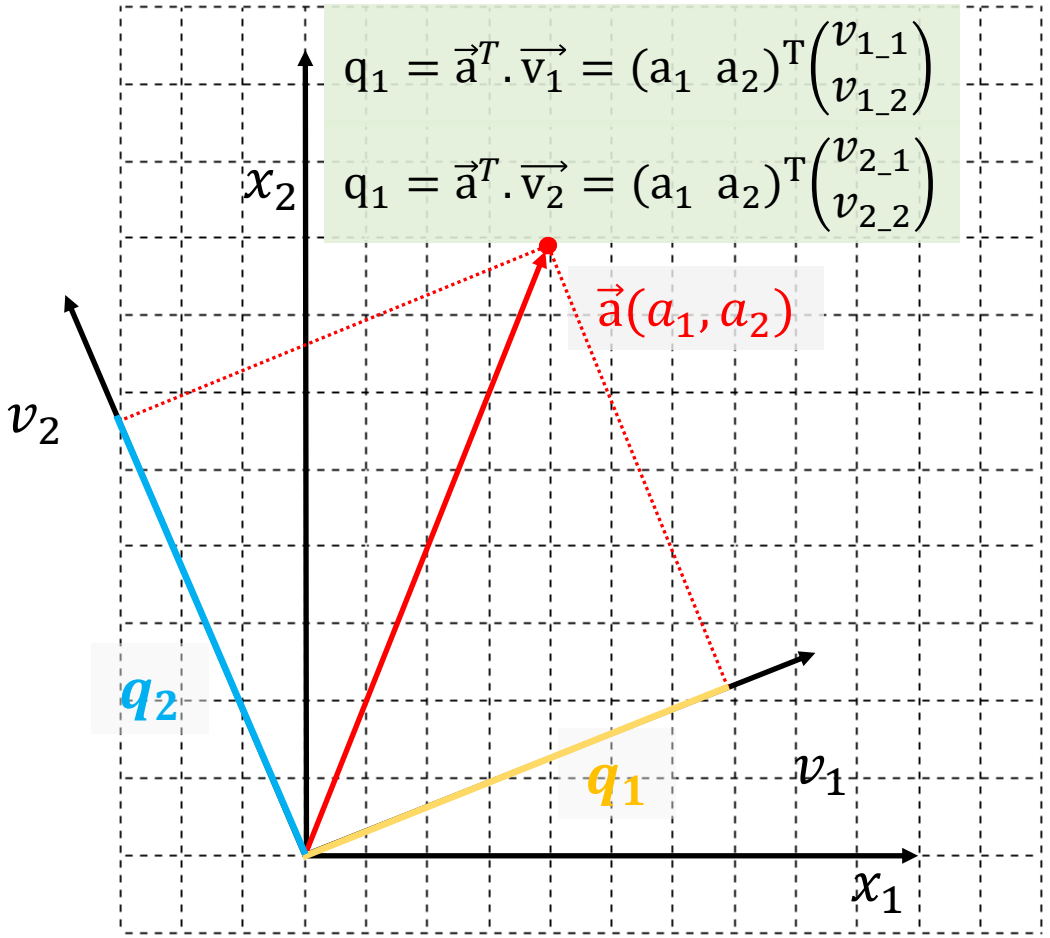
Dot Product

$\overrightarrow{x_1} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ $\overrightarrow{x_2} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$



Tìm độ dài hình chiếu của $\vec{a}$ lên $\overrightarrow{x_1}$ và $\overrightarrow{x_2}$

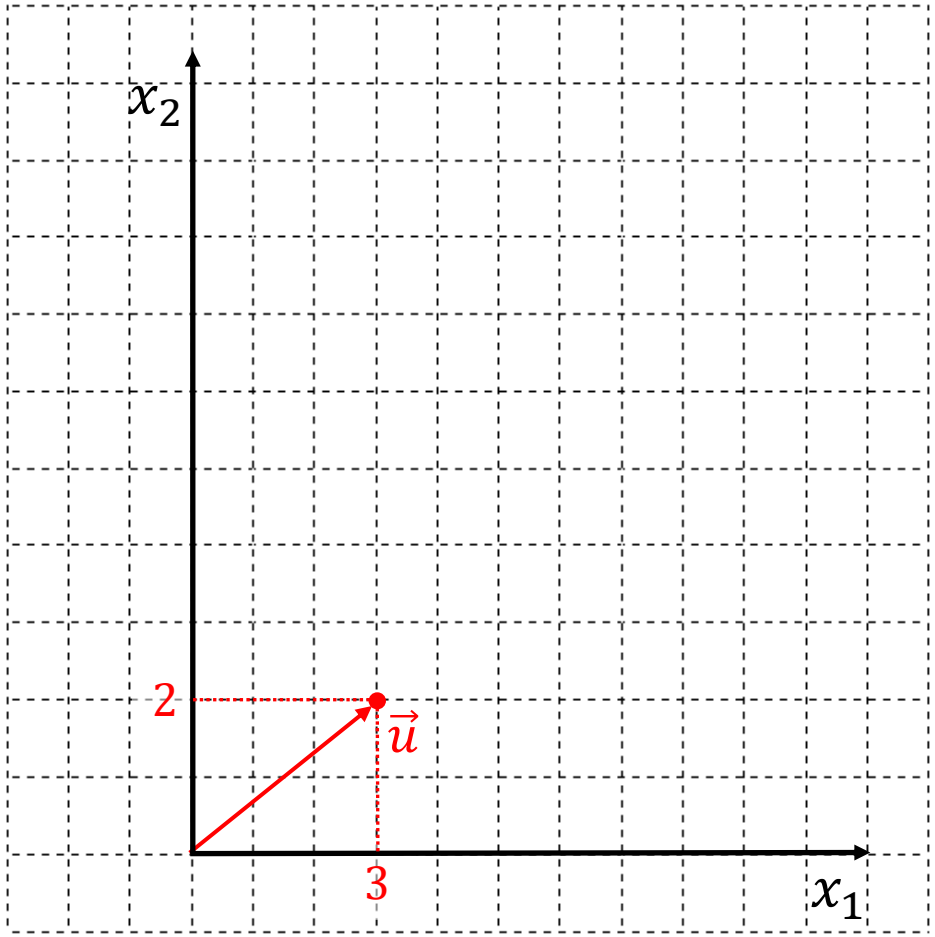
$\overrightarrow{v_1} = \begin{pmatrix} v_{1\_1} \\ v_{1\_2} \end{pmatrix}$ $\overrightarrow{v_2} = \begin{pmatrix} v_{2\_1} \\ v_{2\_2} \end{pmatrix}$



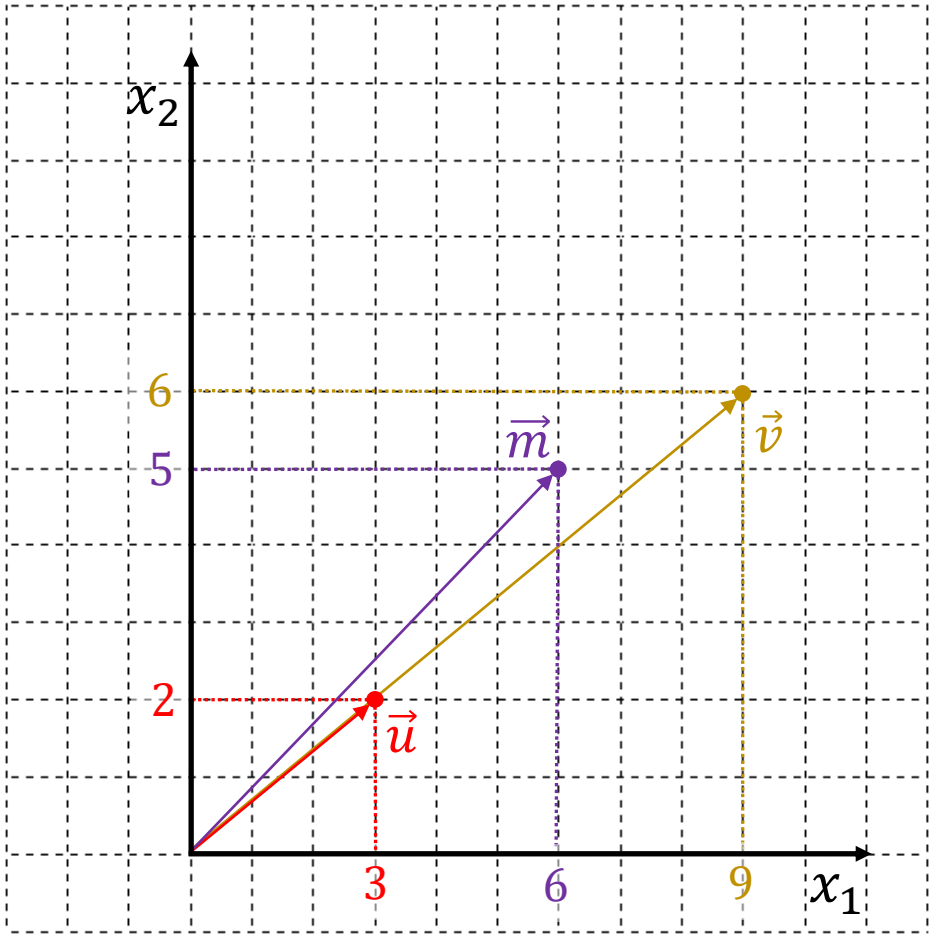
Tìm độ dài hình chiếu của $\vec{a}$ lên $\overrightarrow{v_1}$ và $\overrightarrow{v_2}$

Dot Product

$$\vec{u} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 3 \\ 2 \end{pmatrix}$$

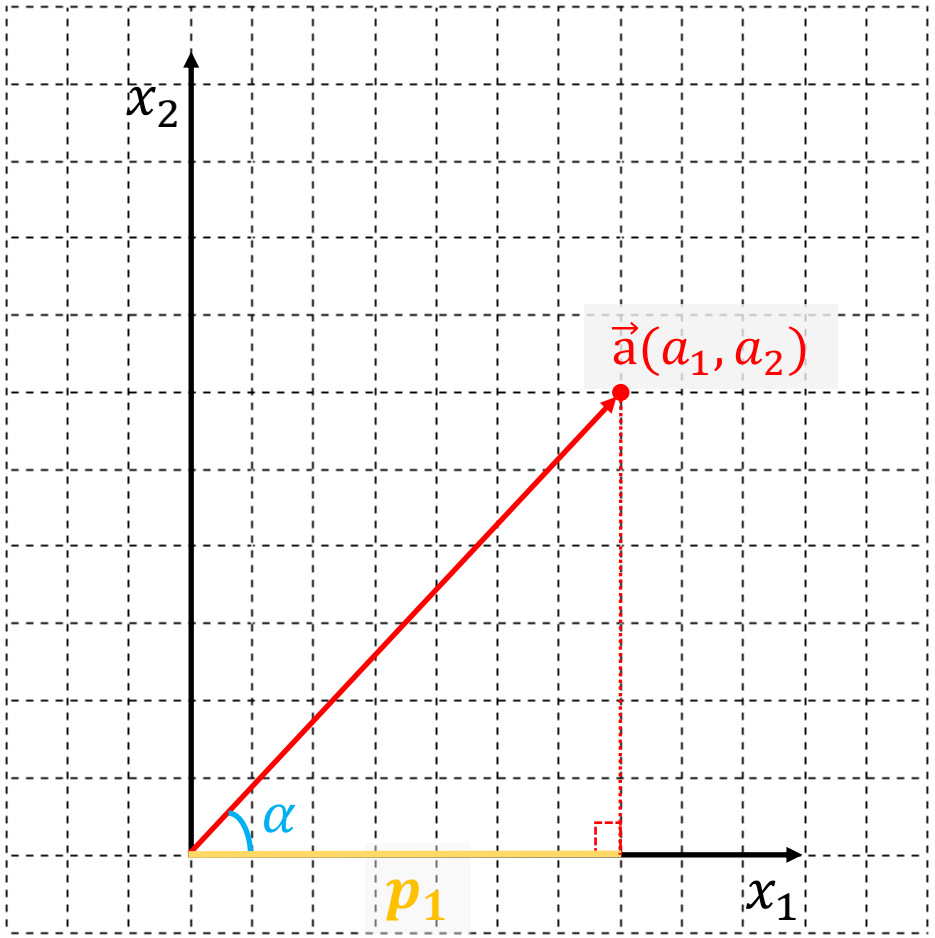


$$\vec{u} = \begin{pmatrix} 3 \\ 2 \end{pmatrix} \quad \vec{v} = 3 * \vec{u} = \begin{pmatrix} 9 \\ 6 \end{pmatrix} \quad \vec{m} = 3 + \vec{u} = \begin{pmatrix} 6 \\ 5 \end{pmatrix}$$

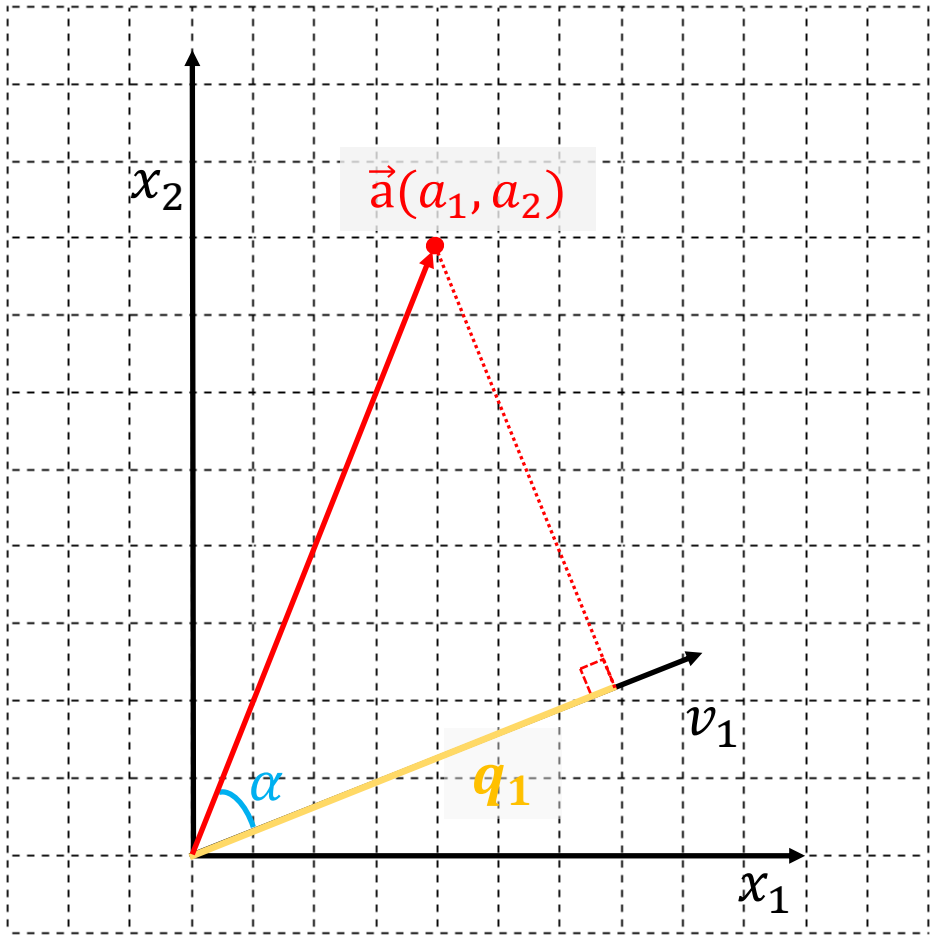


Dot Product

$$\overrightarrow{x_1} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \overrightarrow{x_2} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$



$$\overrightarrow{v_1} = \begin{pmatrix} v_{1\_1} \\ v_{1\_2} \end{pmatrix} \quad \overrightarrow{v_2} = \begin{pmatrix} v_{2\_1} \\ v_{2\_2} \end{pmatrix}$$



Dot Product

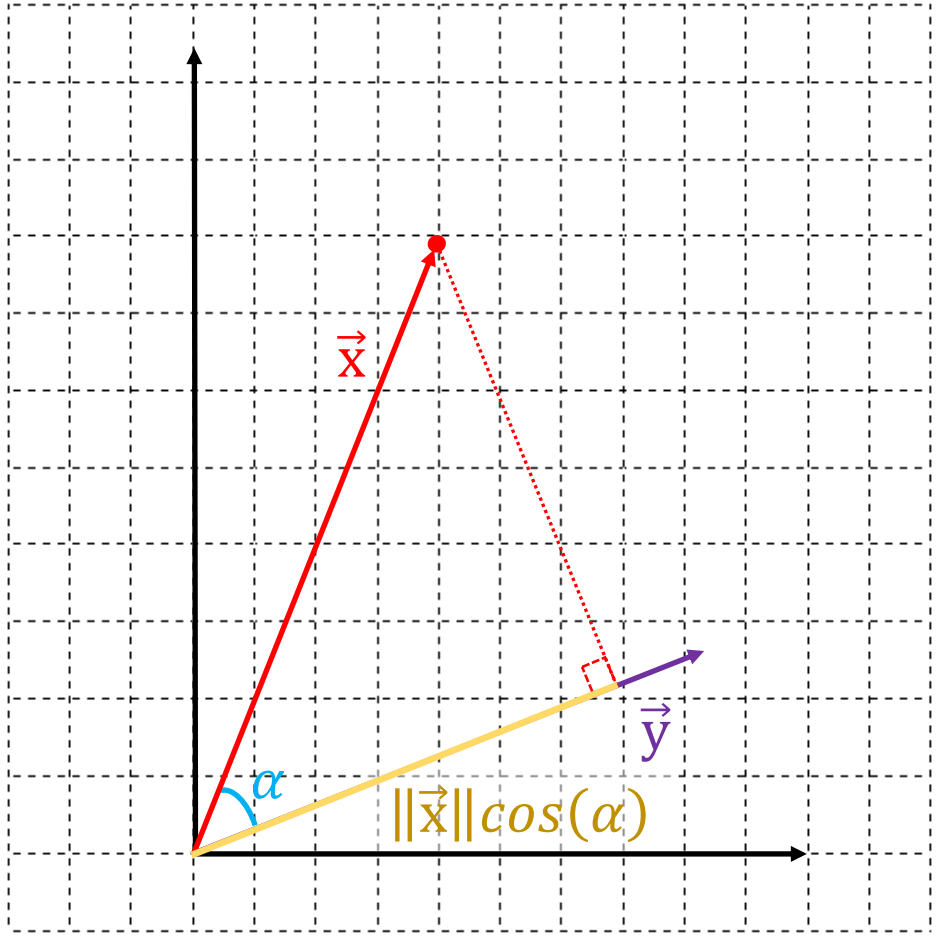
❖ Definition

Algebra	Geometry
$\vec{x}.\vec{y} = \sum_{1}^{n} x_i y_i$	$\vec{x}.\vec{y} = \|\vec{x}\|\|\vec{y}\|cos(\alpha)$

$$\begin{bmatrix}4 & 0 & 3 & 1\end{bmatrix} \cdot \begin{bmatrix}4\\0\\3\\1\end{bmatrix} = 26$$

$$\begin{bmatrix}3 & 0 & 4 & 1\end{bmatrix} \cdot \begin{bmatrix}3\\0\\4\\1\end{bmatrix} = 26$$

Position-invariant measure



Dot Product

❖ Definition

Algebra

$$\vec{x}.\vec{y} = \sum_{1}^{n} x_i y_i$$

Geometry

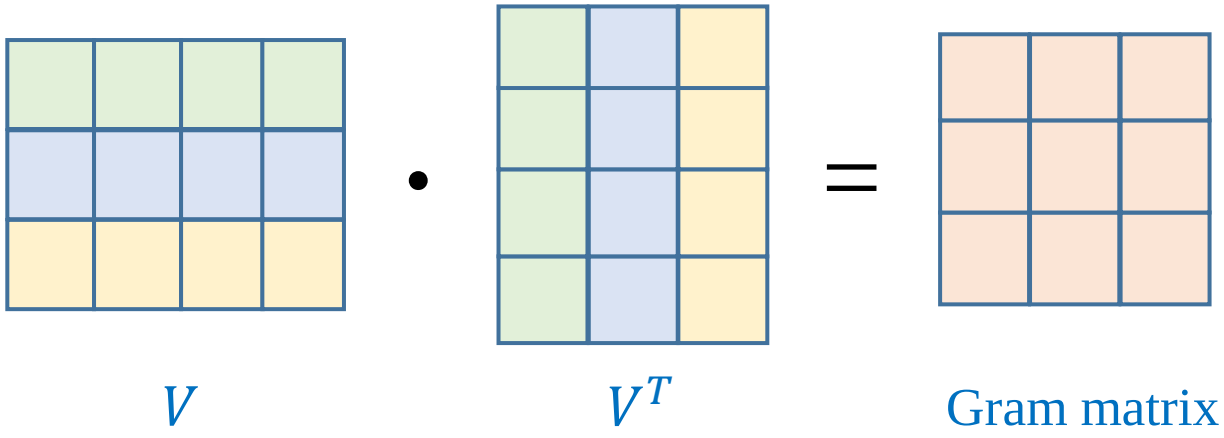
$$\vec{x}.\vec{y} = \|\vec{x}\|\|\vec{y}\| cos(\alpha)$$

```python
import numpy as np

u = np.array([0, 5])
v = np.array([3, 3])

dproduct_1 = np.dot(u, v)
dproduct_2 = np.dot(v, u)

print(dproduct_1)
print(dproduct_2)
```

```python
# geometry

norm_u = np.linalg.norm(u)
norm_v = np.linalg.norm(v)

alpha = 3.14/4 # 45 degree
dproduct_3 = np.cos()*norm_u*norm_v
print(dproduct_3)
```

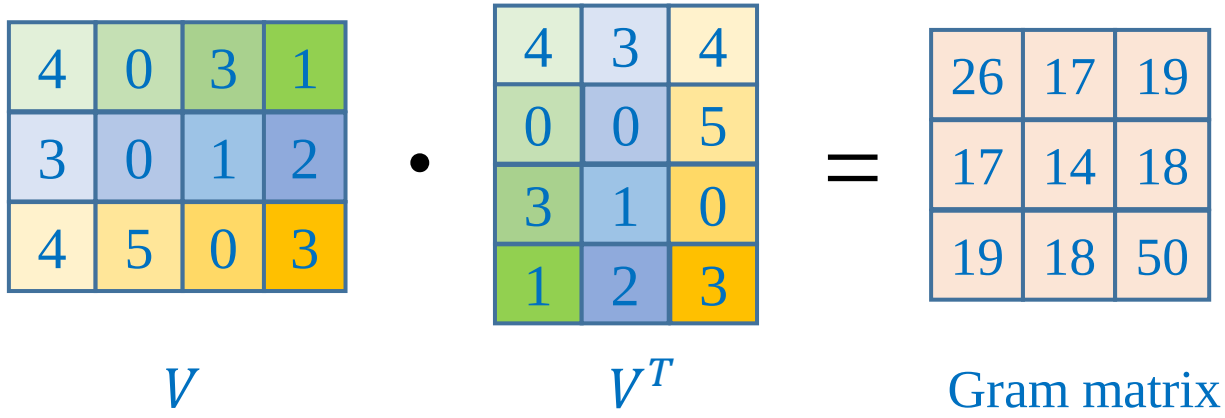
Gram Matrix

$$G_{ij} = \sum_k V_{ik} V_{jk}$$



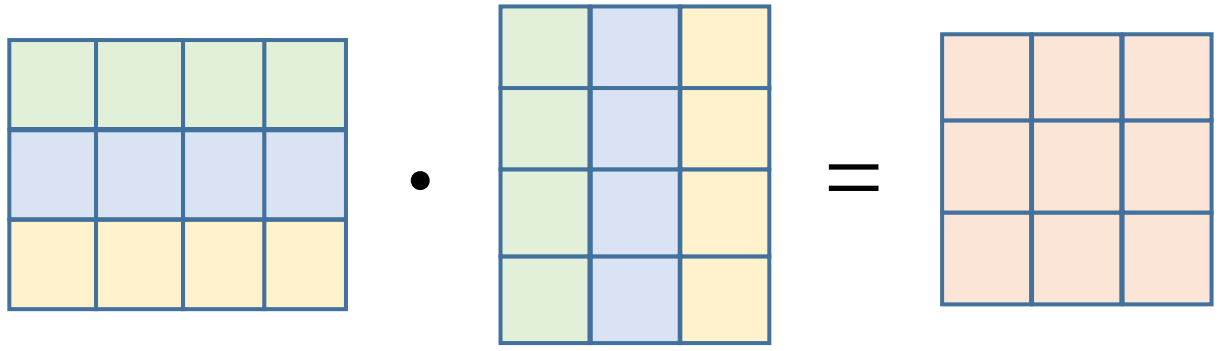

```
import torch

x = torch.Tensor([[4, 0, 3, 1],
                  [3, 0, 1, 2],
                  [4, 5, 0, 3]])
G = torch.mm(x, x.t())
print(G)
```

```
tensor([[26., 17., 19.],
        [17., 14., 18.],
        [19., 18., 50.]])
```



Gram Matrix

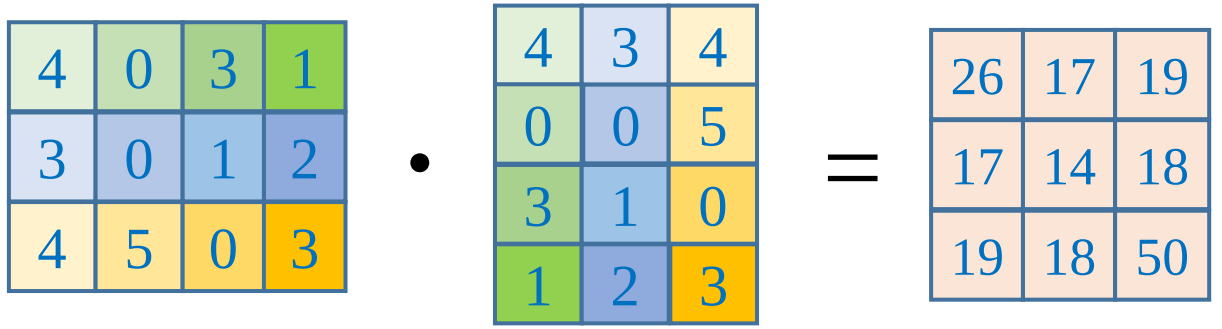
$$G_{ij} = \sum_{k} V_{ik} V_{jk}$$



V V^T Gram matrix

```
import torch

x = torch.Tensor([[4, 0, 3, 1],
                  [3, 0, 1, 2],
                  [4, 5, 0, 3]])
G = torch.einsum('ij,jk->ik', x, x.t())
print(G)
```

```
tensor([[26., 17., 19.],
        [17., 14., 18.],
        [19., 18., 50.]])
```



V V^T Gram matrix

Detailed discussion on the Einsum scheduled for Friday

Gram Matrix

$$G_{ij} = \sum_k V_{ik} V_{jk}$$

4	0	3	1
3	0	1	2
4	5	0	3

V

$\bullet$

4	3	4
0	0	5
3	1	0
1	2	3

V^T

$=$

26	17	19
17	14	18
19	18	50

Gram matrix

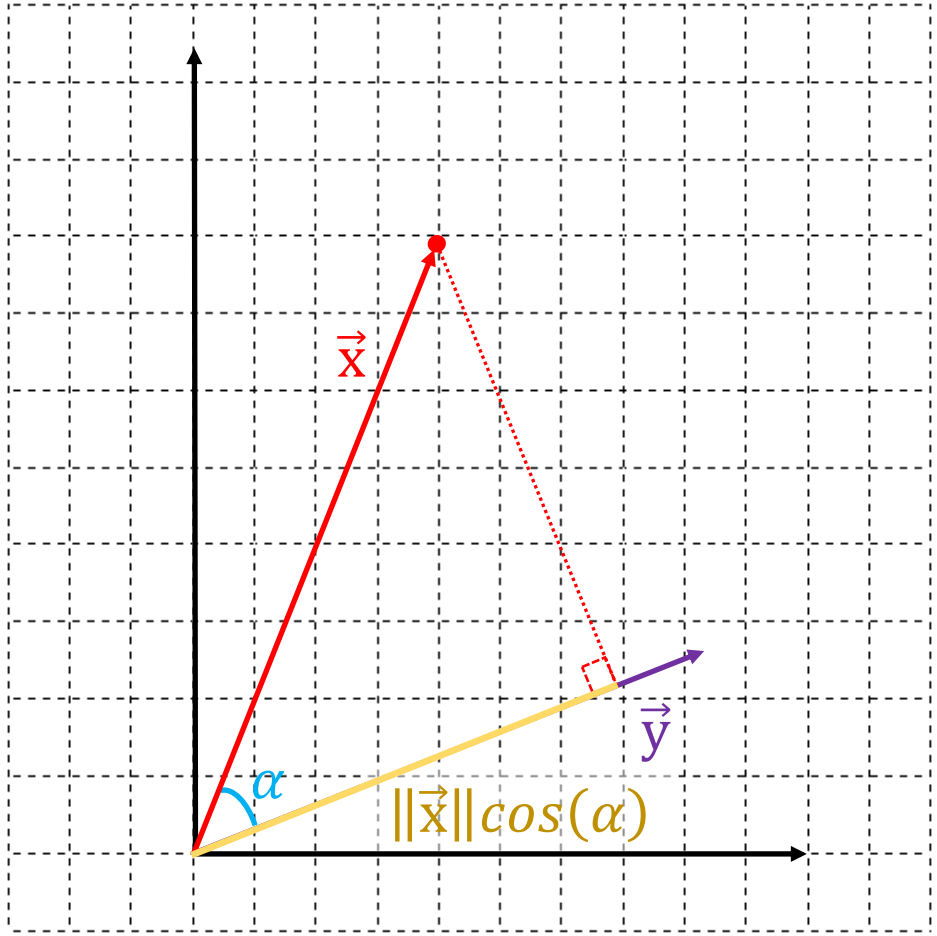
Inner Product

Algebra	Geometry
$\vec{x}.\vec{y} = \sum_1^n x_i y_i$	$\vec{x}.\vec{y} = \|\vec{x}\|\|\vec{y}\|cos(\alpha)$

4	0	3	1	$\bullet$	4	0	3	1	=	26

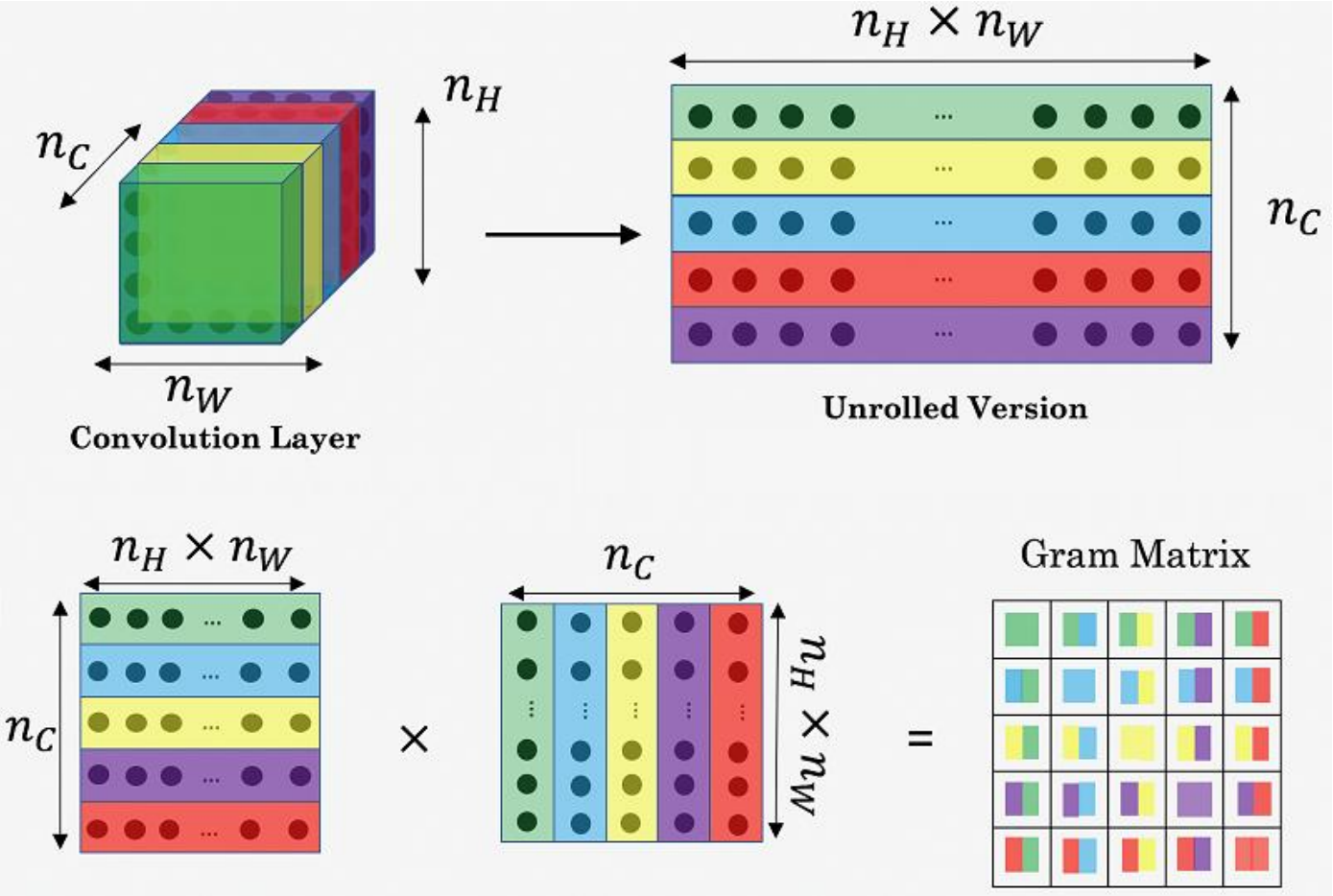
3	0	4	1	$\bullet$	3	0	4	1	=	26

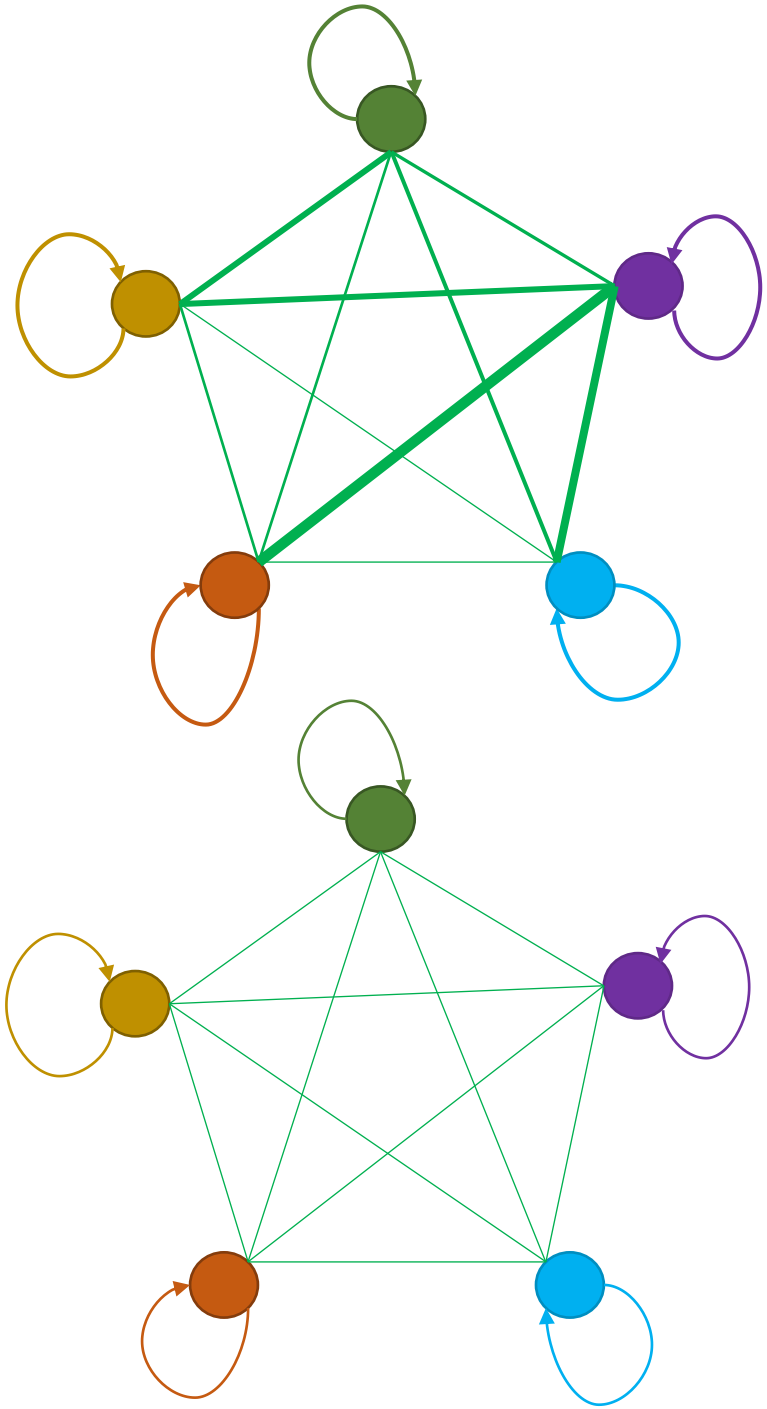
Position-invariant measure



Style Transfer

❖ Gram matrix

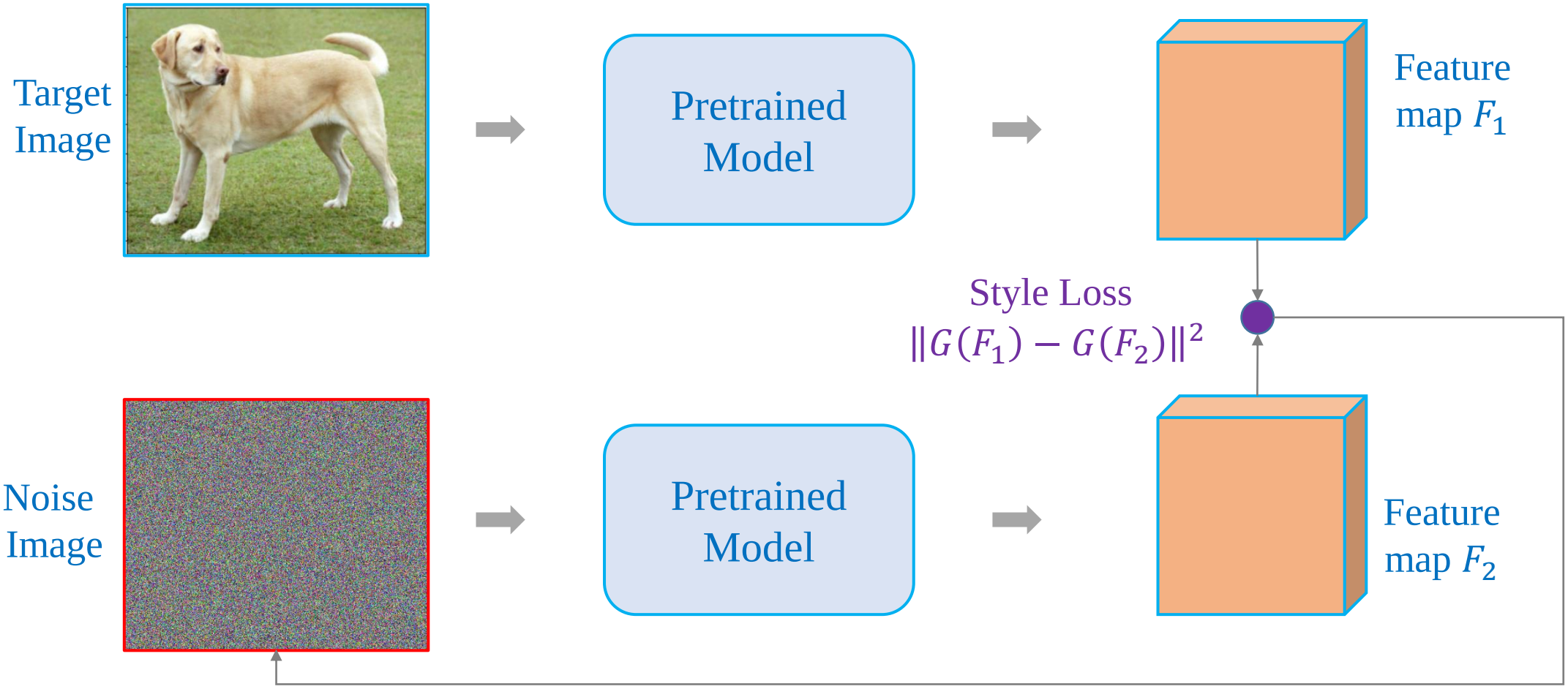




https://pytorchtaipei.github.io/articles/PyTorchTP-Style-Transfer/

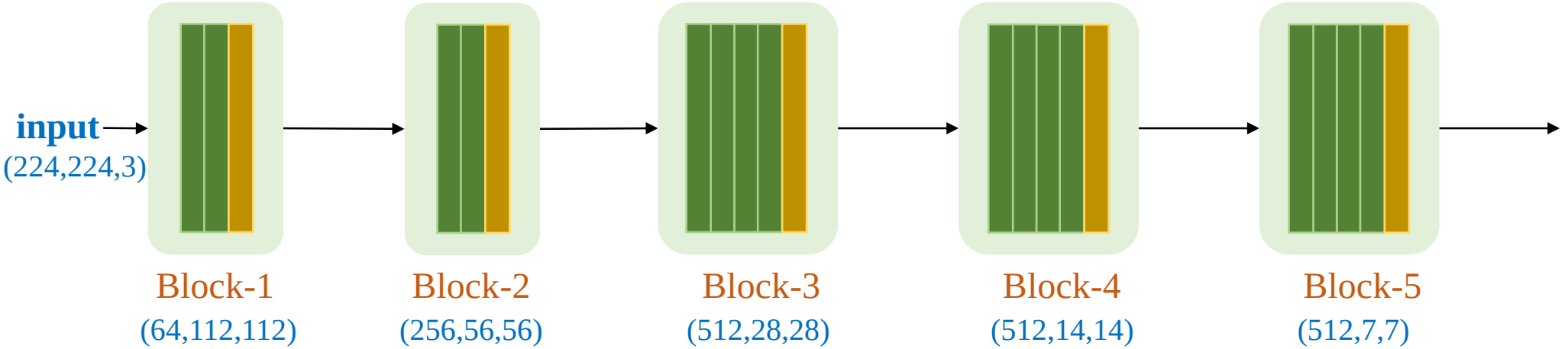
Style Transfer

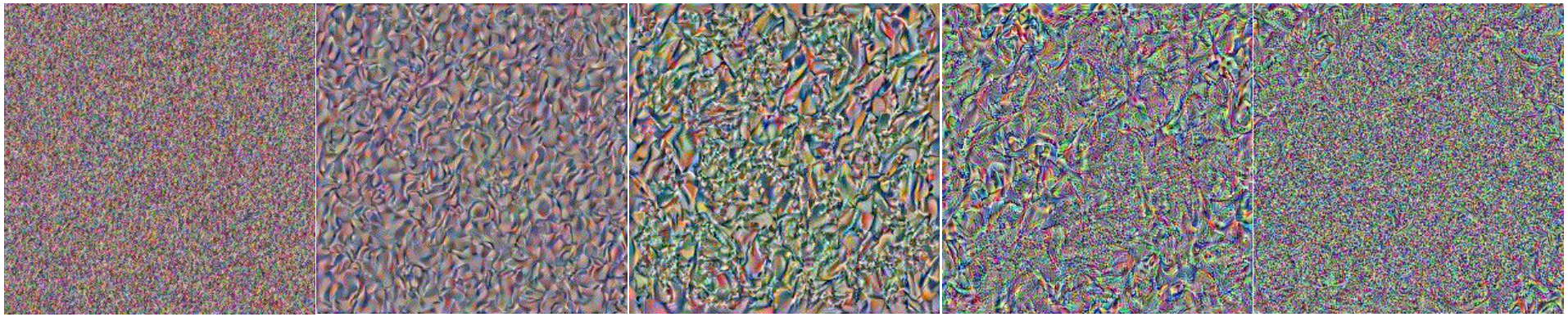
❖ Style Loss



Style Transfer

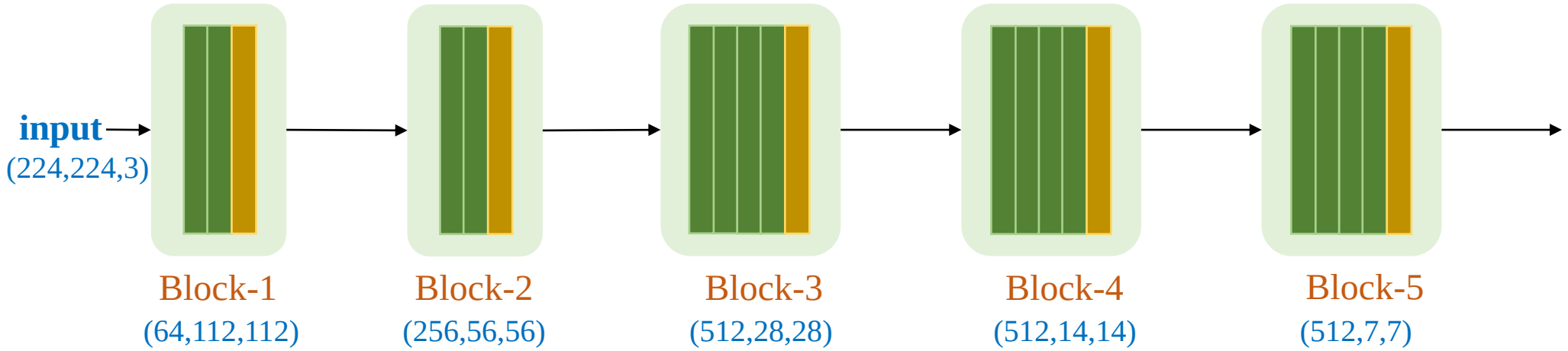
❖ Style Loss: Use which features?

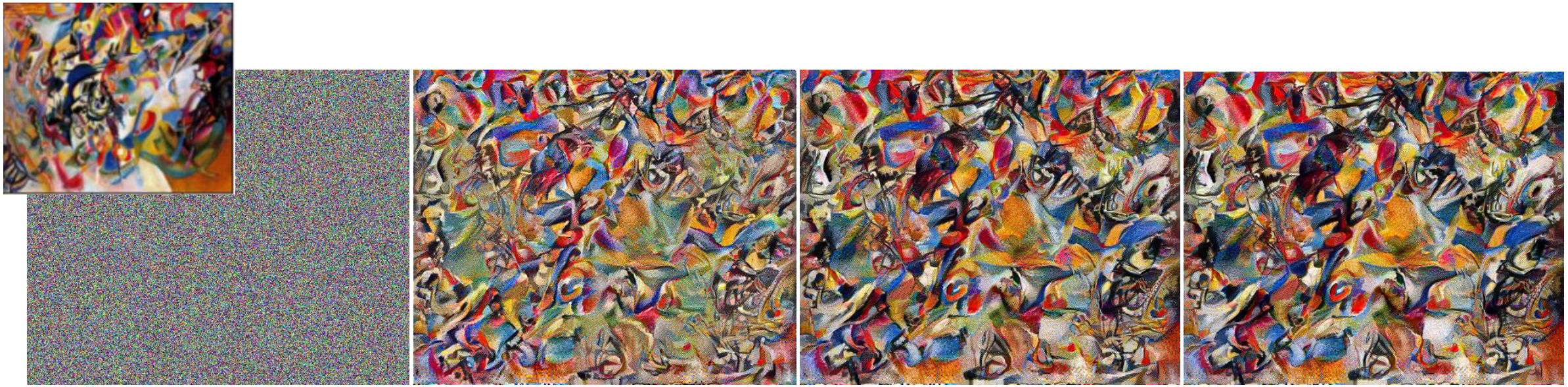




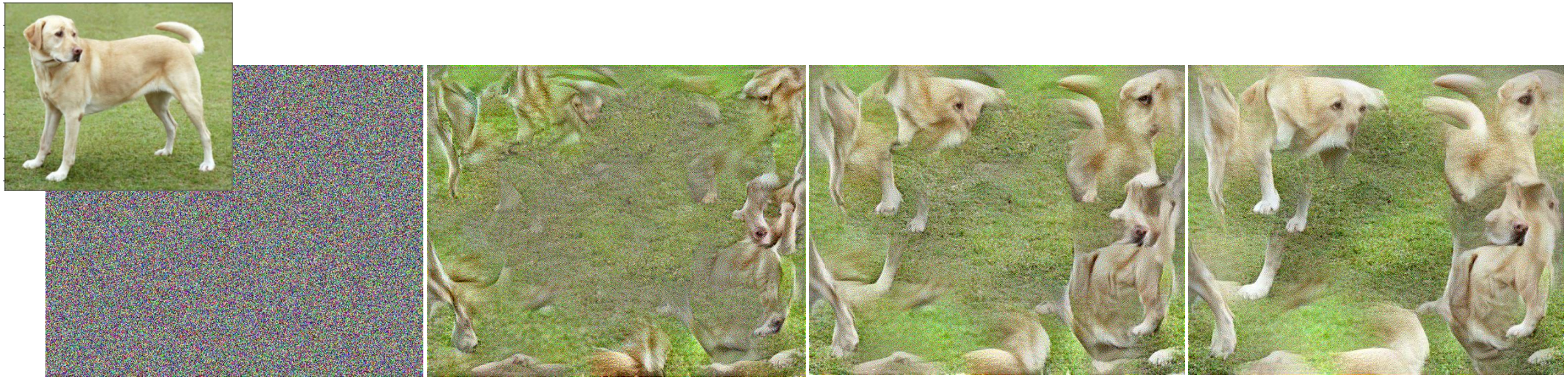
Style Transfer

❖ Style Loss: Use which features?





Initial Image | Epoch 10 | Epoch 40 | Epoch 90



Initial Image | Epoch 10 | Epoch 40 | Epoch 90


QUIZ TIME

Outline

- Introduction
- Inputs as Variables
- Content Loss
- Style Loss

Summary

- ✓ Studied about content loss
- ✓ Studied about style loss

- ✓ Studying about the total loss
- ✓ Extend to some variants