Style Transfer

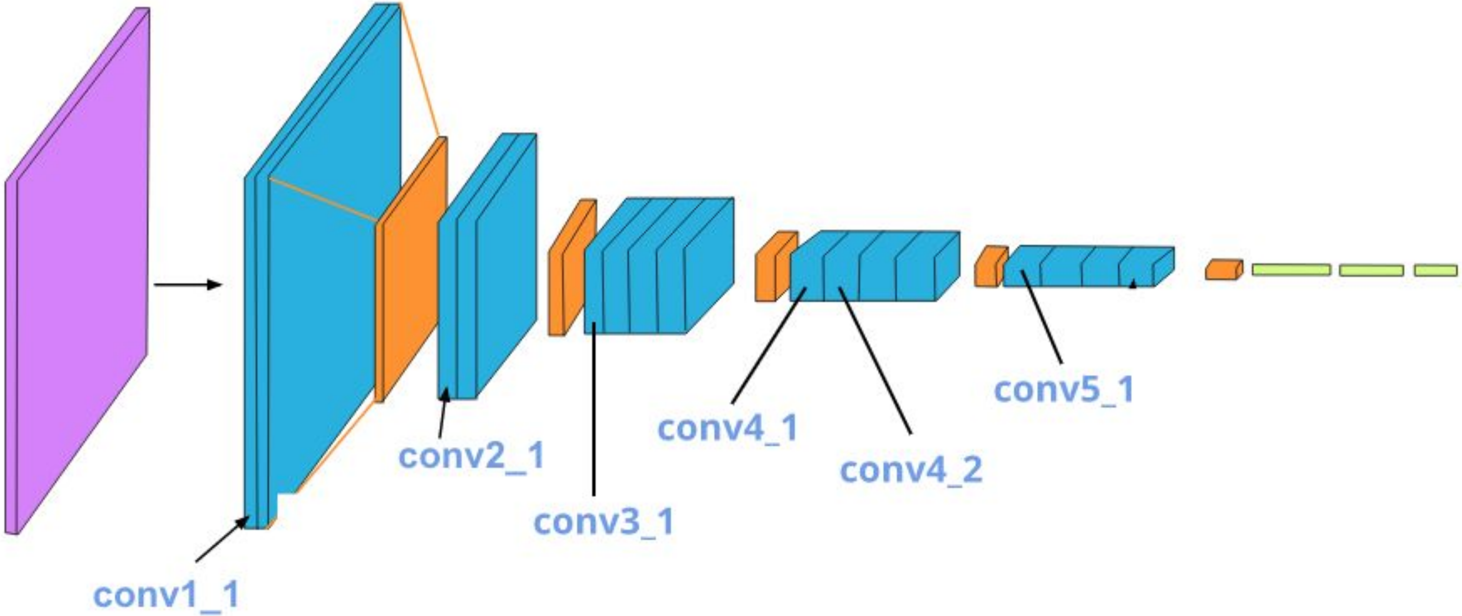
An Introduction

Eka Kurniawan

Background

Style Transfer

Introduction



VGG19 Model

Deep Learning with PyTorch. Lesson 6: Style Transfer. 2018. Udacity.

Invert Representation

This section introduces our method to compute an approximate inverse of an image representation. This is formulated as the problem of finding an image whose representation best matches the one given [34]. Formally, given a representation function $\Phi : \mathbb{R}^{H\times W\times C} \to \mathbb{R}^d$ and a representation $\Phi_0 = \Phi(\mathbf{x}_0)$ to be inverted, reconstruction finds the image $\mathbf{x} \in \mathbb{R}^{H\times W\times C}$ that minimizes the objective:

$$\mathbf{x}^* = \underset{\mathbf{x}\in\mathbb{R}^{H\times W\times C}}{\operatorname{argmin}} \ \ell(\Phi(\mathbf{x}), \Phi_0) + \lambda\mathcal{R}(\mathbf{x}) \tag{1}$$

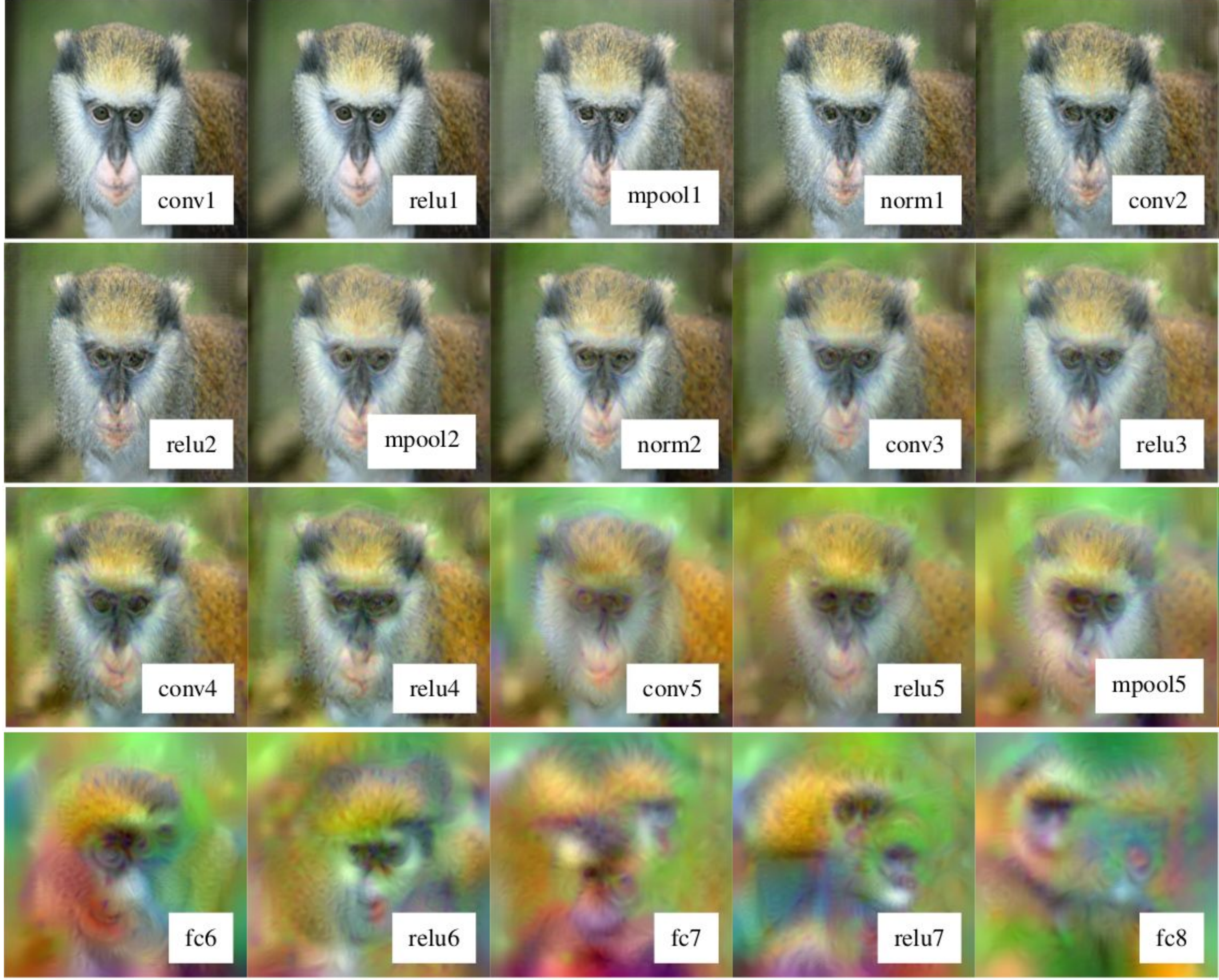
where the loss ℓ compares the image representation $\Phi(\mathbf{x})$ to the target one Φ_0 and $\mathcal{R} : \mathbb{R}^{H\times W\times C} \to \mathbb{R}$ is a regulariser capturing a *natural image prior*.

Mahendran and Vedaldi. Understanding Deep Image Representations by Inverting Them. 2014. arXiv:1412.0035.

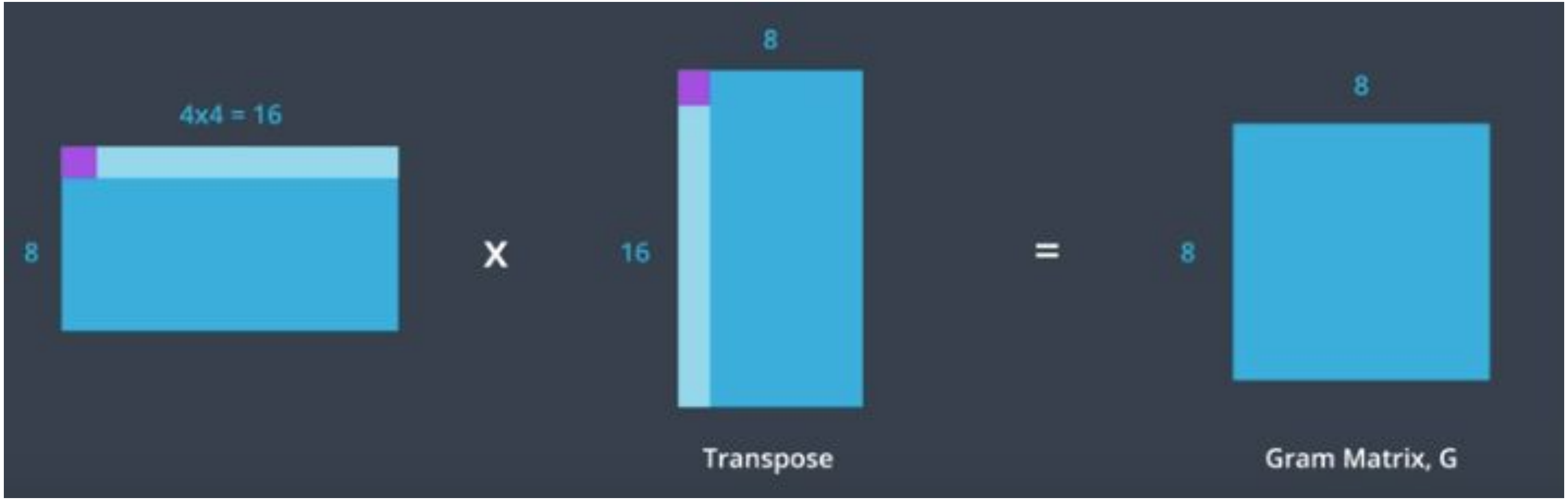
Invert Representation

Mahendran and Vedaldi. Understanding Deep Image Representations by Inverting Them. 2014. arXiv:1412.0035.

Invert Representation



Mahendran and Vedaldi. Understanding Deep Image Representations by Inverting Them. 2014. arXiv:1412.0035.



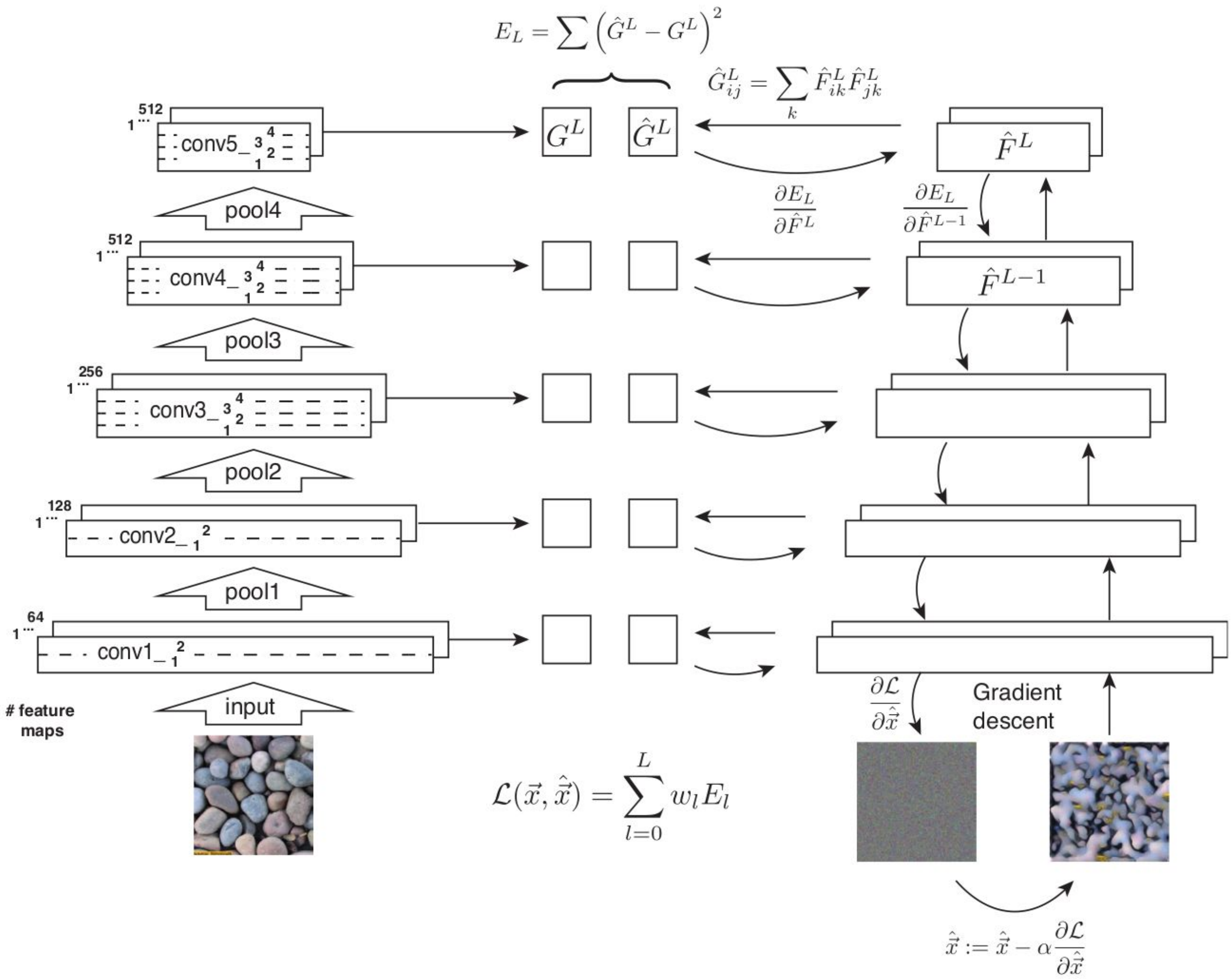
Gram Matrix

Localization Removal

to extract just the style information from the features without the information where the style is located in the image

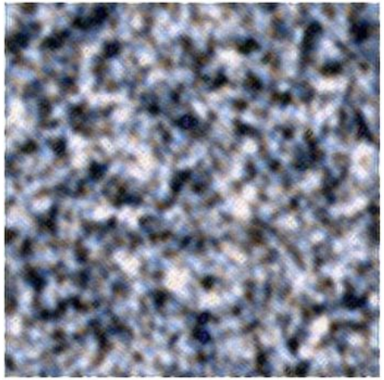
Deep Learning with PyTorch. Lesson 6: Style Transfer. 2018. Udacity.

Texture Synthesis



Gatys and Ecker and Bethge. Texture Synthesis Using Convolutional Neural Networks. 2015. arXiv:1505.07376.

Texture Synthesis



conv1_1

pool1

pool2

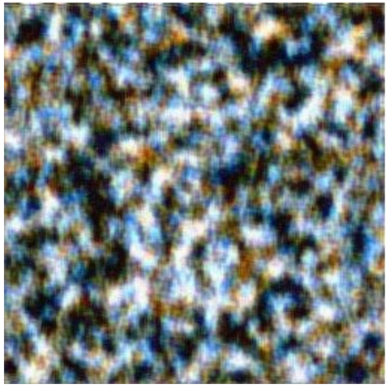
pool3

pool4

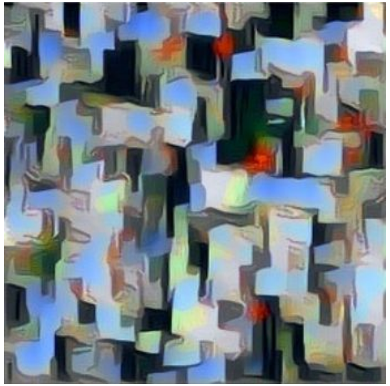
original

Gatys and Ecker and Bethge. Texture Synthesis Using Convolutional Neural Networks. 2015. arXiv:1505.07376.

Texture Synthesis



conv1_1



pool1

pool2

pool3

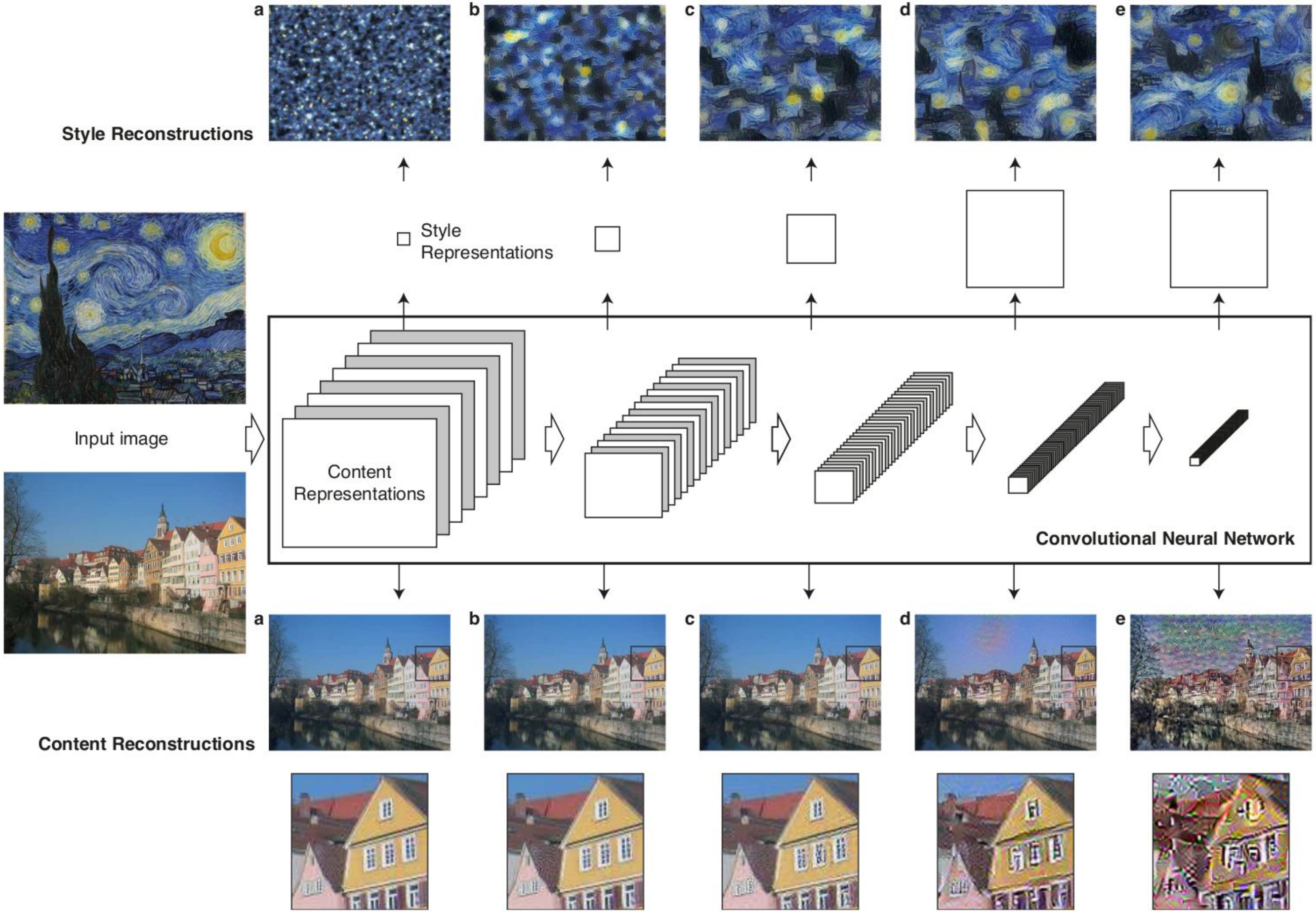
pool4

original

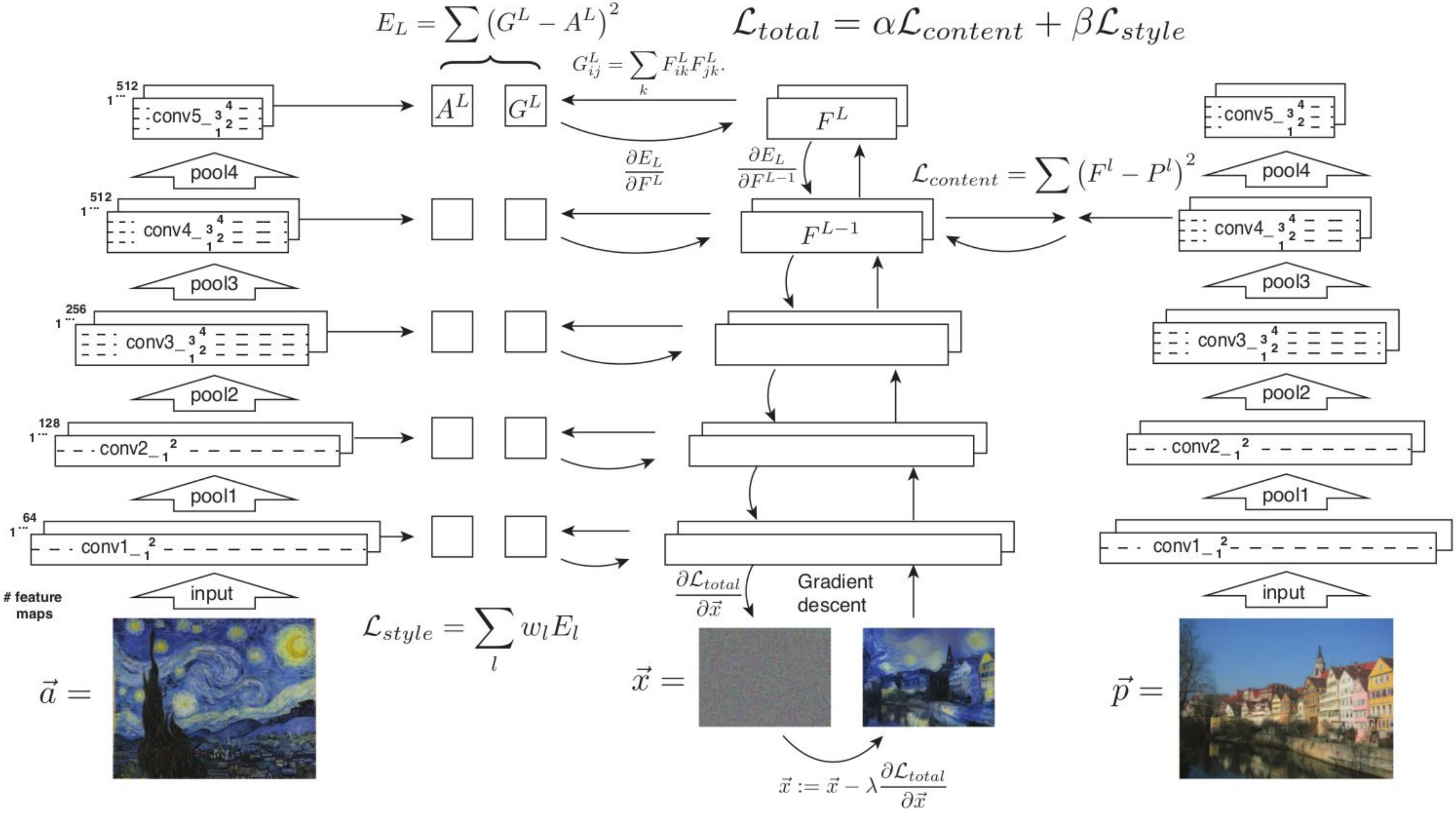
Gatys and Ecker and Bethge. Texture Synthesis Using Convolutional Neural Networks. 2015. arXiv:1505.07376.

Style Transfer

Invert Representation + Texture Synthesis (Style and Content Reconstructions)

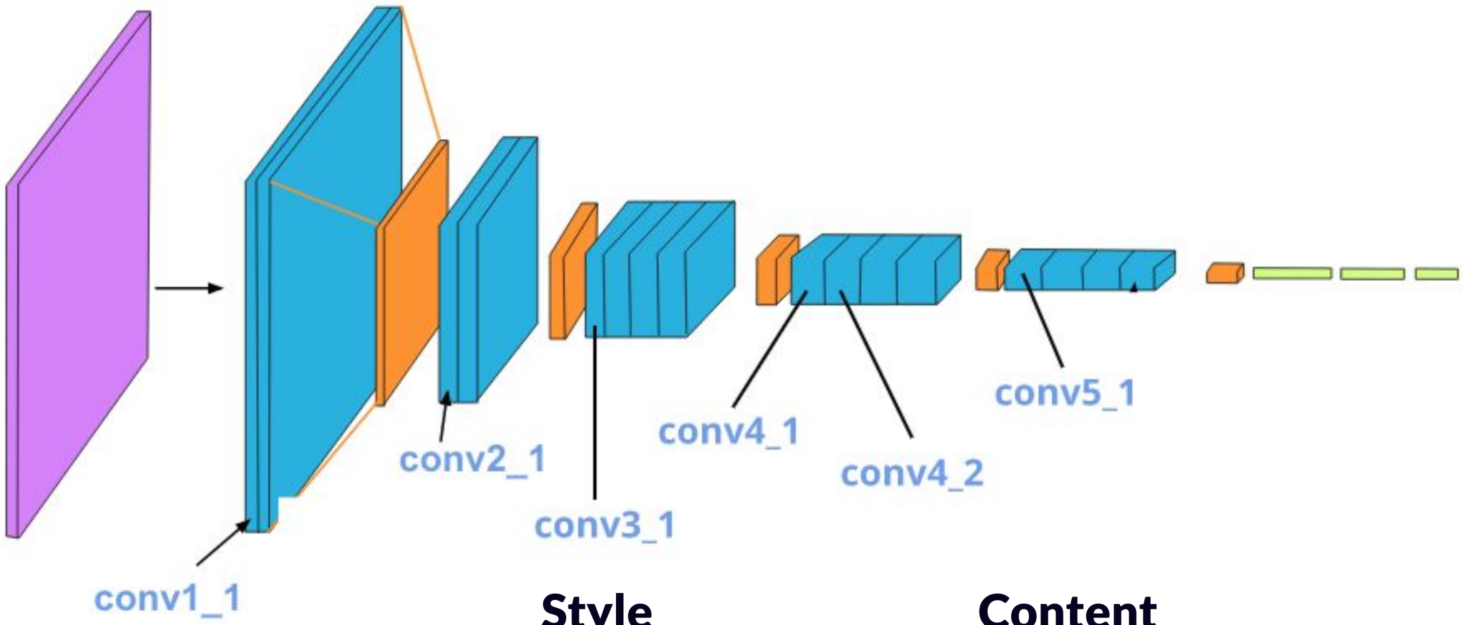


Gatys and Ecker and Bethge. Image Style Transfer Using Convolutional Neural Networks. 2016. IEEE Conference on CVPR. pp 2414-2423.



Gatys and Ecker and Bethge. Image Style Transfer Using Convolutional Neural Networks. 2016. IEEE Conference on CVPR. pp 2414-2423.

Implementation



VGG19 Model

Style

conv1_1

conv2_1

conv3_1

conv4_1

conv5_1

Content

conv4_2

Deep Learning with PyTorch. Lesson 6: Style Transfer. 2018. Udacity.

```
# get the "features" portion of VGG19 (we will not need the "classifier" portion)
vgg = models.vgg19(pretrained=True).features

# freeze all VGG parameters since we're only optimizing the target image
for param in vgg.parameters():
    param.requires_grad_(False)
```


https://github.com/ekaakurniawan/Deep-Learning-with-PyTorch/blob/master/style-transfer/Style_Transfer_Exercise.ipynb

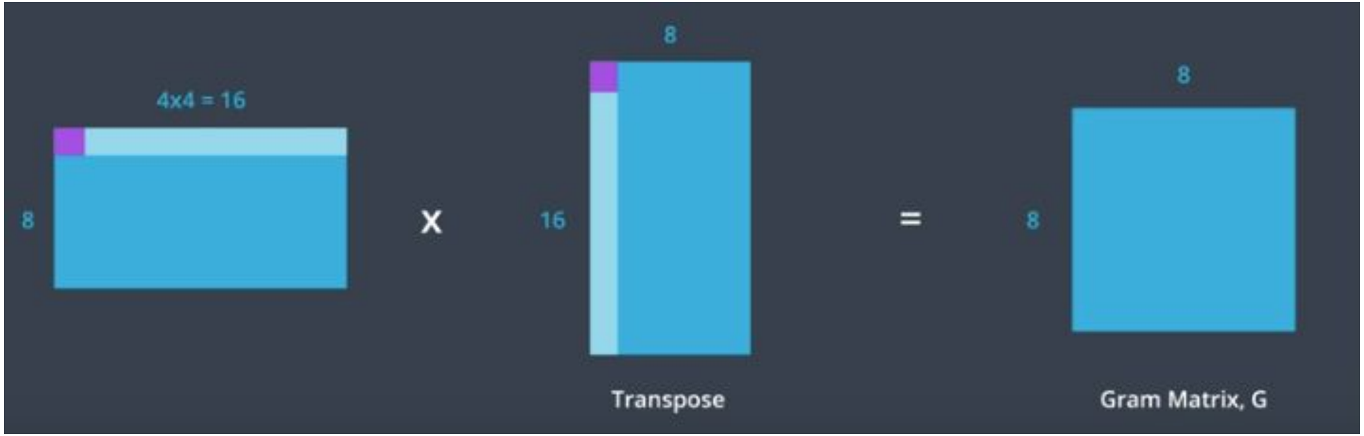
```
(0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(1): ReLU(inplace)
(2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(3): ReLU(inplace)
(4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
(5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(6): ReLU(inplace)
(7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(8): ReLU(inplace)
(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)
(12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(13): ReLU(inplace)
(14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(15): ReLU(inplace)
(16): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(17): ReLU(inplace)
(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)
(21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(22): ReLU(inplace)
(23): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(24): ReLU(inplace)
(25): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(26): ReLU(inplace)
(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)
(30): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(31): ReLU(inplace)
(32): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(33): ReLU(inplace)
(34): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(35): ReLU(inplace)
(36): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
```

https://github.com/ekaakurniawan/Deep-Learning-with-PyTorch/blob/master/style-transfer/Style_Transfer_Exercise.ipynb

```
def get_features(image, model, layers=None):
    """ Run an image forward through a model and get the features for
        a set of layers. Default layers are for VGGNet matching Gatys et al (2016)
    """

    ## TODO: Complete mapping layer names of PyTorch's VGGNet to names from the paper
    ## Need the layers for the content and style representations of an image
    if layers is None:
        layers = {'0': 'conv1_1',  # for style
                  '5': 'conv2_1',  # for style
                  '10': 'conv3_1', # for style
                  '19': 'conv4_1', # for style
                  '21': 'conv4_2', # for content
                  '28': 'conv5_1', # for style
                 }


    ## -- do not need to change the code below this line -- ##
    features = {}
    x = image
    # model._modules is a dictionary holding each module in the model
    for name, layer in model._modules.items():
        x = layer(x)
        if name in layers:
            features[layers[name]] = x

    return features
```

https://github.com/ekaakurniawan/Deep-Learning-with-PyTorch/blob/master/style-transfer/Style_Transfer_Exercise.ipynb

```python
def gram_matrix(tensor):
    """ Calculate the Gram Matrix of a given tensor
        Gram Matrix: https://en.wikipedia.org/wiki/Gramian_matrix
    """

    ## get the batch_size, depth, height, and width of the Tensor
    ## reshape it, so we're multiplying the features for each channel
    ## calculate the gram matrix
    tensor_shape = tensor.shape
    tensor_flat = tensor.view(tensor_shape[0], tensor_shape[1], -1)
    gram = torch.bmm(tensor_flat, torch.transpose(tensor_flat, 1, 2))

    return gram
```



https://github.com/ekaakurniawan/Deep-Learning-with-PyTorch/blob/master/style-transfer/Style_Transfer_Exercise.ipynb

```
# get content and style features only once before forming the target image
content_features = get_features(content, vgg)
style_features = get_features(style, vgg)

# calculate the gram matrices for each layer of our style representation
style_grams = {layer: gram_matrix(style_features[layer]) for layer in style_features}

# create a third "target" image and prep it for change
# it is a good idea to start of with the target as a copy of our *content* image
# then iteratively change its style
target = content.clone().requires_grad_(True).to(device)
```

https://github.com/ekaakurniawan/Deep-Learning-with-PyTorch/blob/master/style-transfer/Style_Transfer_Exercise.ipynb

```
# weights for each style layer
# weighting earlier layers more will result in *larger* style artifacts
# notice we are excluding `conv4_2` our content representation
style_weights = {'conv1_1': 0.9, #bar 1.,
                 'conv2_1': 0.8, #bar 0.8,
                 'conv3_1': 0.5, #bar 0.5,
                 'conv4_1': 0.3, #bar 0.3,
                 'conv5_1': 0.1} #bar 0.1}

# you may choose to leave these as is
content_weight = 1  # alpha
style_weight = 1e6 #bar 1e6  # beta
```


https://github.com/ekaakurniawan/Deep-Learning-with-PyTorch/blob/master/style-transfer/Style_Transfer_Exercise.ipynb

```
# iteration hyperparameters
optimizer = optim.Adam([target], lr=0.003)
steps = 1000 #bar 2000  # decide how many iterations to update your image (5000)
```


https://github.com/ekaakurniawan/Deep-Learning-with-PyTorch/blob/master/style-transfer/Style_Transfer_Exercise.ipynb

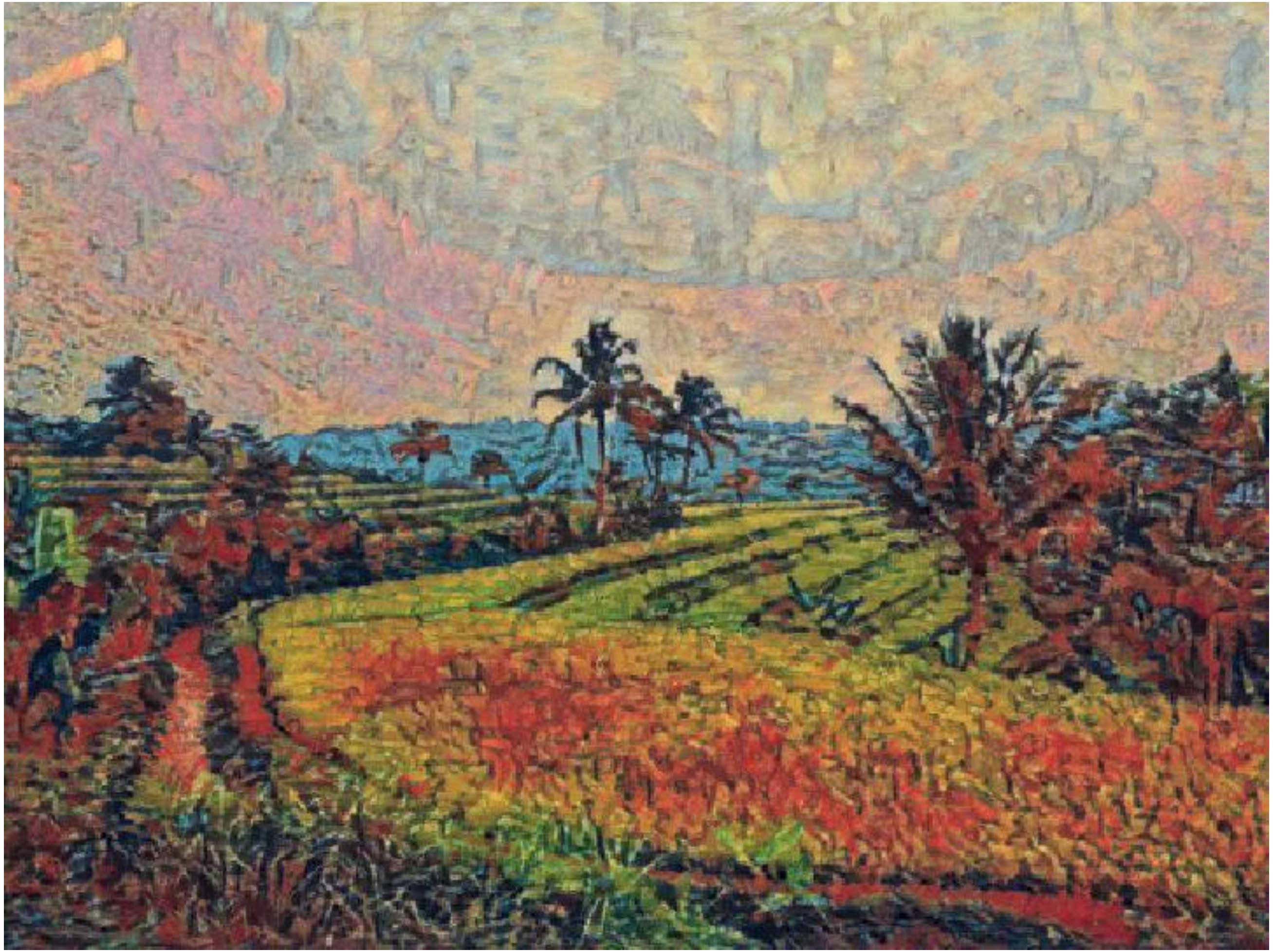
```
for ii in range(1, steps+1):

    target_features = get_features(target, vgg)
    content_loss = torch.mean((target_features['conv4_2'] - content_features['conv4_2']) ** 2)

    style_loss = 0
    # iterate through each style layer and add to the style loss
    for layer in style_weights:
        # get the "target" style representation for the layer
        target_feature = target_features[layer]
        _, d, h, w = target_feature.shape

        target_gram = gram_matrix(target_feature)

        style_gram = style_grams[layer]
        layer_style_loss = style_weights[layer] * \
                           torch.mean((target_gram - style_gram) ** 2)

        # add to the style loss
        style_loss += layer_style_loss / (d * h * w)

    total_loss = (content_weight * content_loss ) + \
                 (style_weight * style_loss)

    optimizer.zero_grad()
    total_loss.backward()
    optimizer.step()
```


https://github.com/ekaakurniawan/Deep-Learning-with-PyTorch/blob/master/style-transfer/Style_Transfer_Exercise.ipynb

Style Transfer

style

content



Bukalapak

https://github.com/ekaakurniawan/Deep-Learning-with-PyTorch/blob/master/style-transfer/Style_Transfer_Exercise.ipynb

Conclusions

01 Features that represent style of an input image can be acquired from layers at the beginning of VGG19 model architecture whereas content is from the end.

02 Gram matrix can be used to remove localization.

03 Fully connected layers also lost the localization.

04 Style transfer optimizes features instead of parameters.

Thank You

Eka Kurniawan

AI Engineer