PRACTICAL WITH CNN

- Classification using CNN
- Apply CNNs to transfer the style of one image to another

CNN example

```
class Classifier(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()
        self.features = nn.Sequential(
            # 3 input image channel, 6 output channels (meaning 6 different convolutions), 5x5 square convolution
            nn.Conv2d(3, 6, 5)
            nn.ReLU(),
            nn.MaxPool2d(2, 2),
            nn.Conv2d(6, 16, 5)

            # ... to do
            # ...

        )

        self.classifier = nn.Sequential(
            nn.Linear(16 * 5 * 5, 120),
            nn.ReLU(),
            nn.Linear(120, 84),
            nn.ReLU(),
            nn.Linear(84, 10)
        )
        print(self.features)
        print(self.classifier)


    def forward(self, input):
        x = self.features(x)
        x = x.view(x.size(0), -1)       # change the view in order to flatten the tensor
        x = self.classifier(x)
```

3 since RGB
6 images as ouput
5x5 convolution

6 since previous ouput is 6
16 images as ouput
5x5 convolution

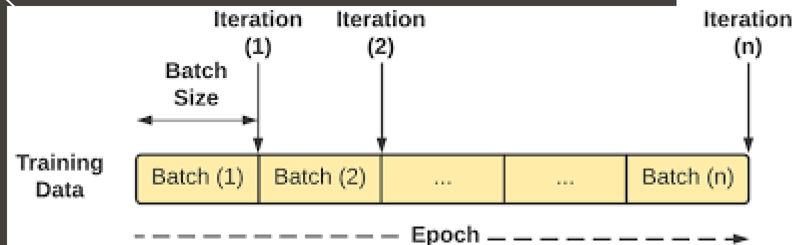
16 since previous layer
If input 32x32

- 1st conv→ 28x28
- MaxPool→ 14x14
- 2nd conv→ 10x10
- MaxPool→5x5

CNN example : optimization

```
net = Classifier()
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)

for epoch in range(2):  # loop over the dataset multiple times
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        if i % 2000 == 1999:    # print every 2000 mini-batches
            print(f'[{epoch + 1}, {i + 1:5d}] loss: {running_loss / 2000:.3f}')
            running_loss = 0.0

print('Finished Training')
```

Loop on epoch



Padding

- Padding = add borders around the input image before applying convolution.
  - Preserve image size after convolution.
  - Avoid loss of information at image edges

- Padding types
  - **Valid (or no padding)**: no borders added, reducing image size.
  - **Same (or with padding)**: borders are added so that the image size remains the same after convolution.
  - **Custom padding**: the user can specify the exact number of pixels to be added

Padding

- Image 5×5 filter 3×3
  - stride = 1   padding = 1
- Before padding

```
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
```

- After padding

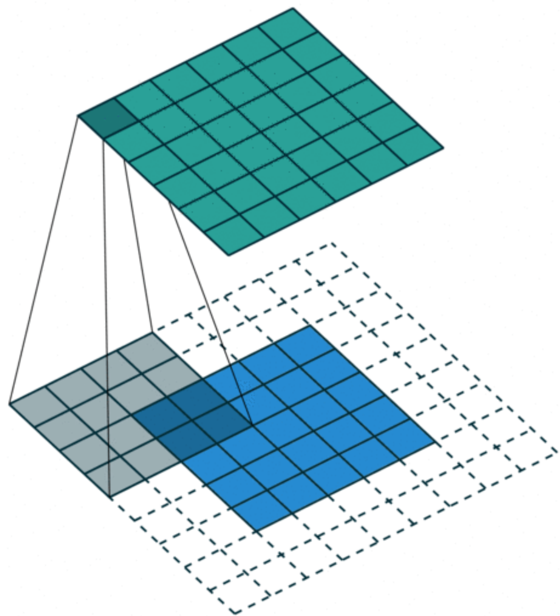
```
0 0 0 0 0 0 0
0 1 1 1 1 1 0
0 1 1 1 1 1 0
0 1 1 1 1 1 0
0 1 1 1 1 1 0
0 0 0 0 0 0 0
```

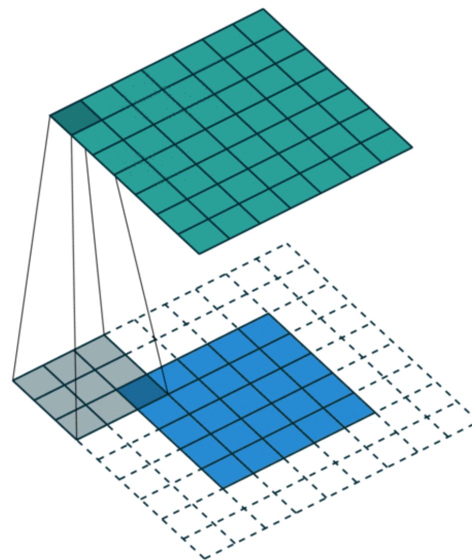
- Filter 3×3 does not reduce the image size

Padding for enlarging an image

Padding: used to enlarge an image (in a decoder, for example)

- Left a 5x5 image, padding of 2 and Conv 4x4 becomes 6x6
- Right an image 5x5, padding 2 and Conv 3x3 becomes 7x7





Input : blue   ouput :green

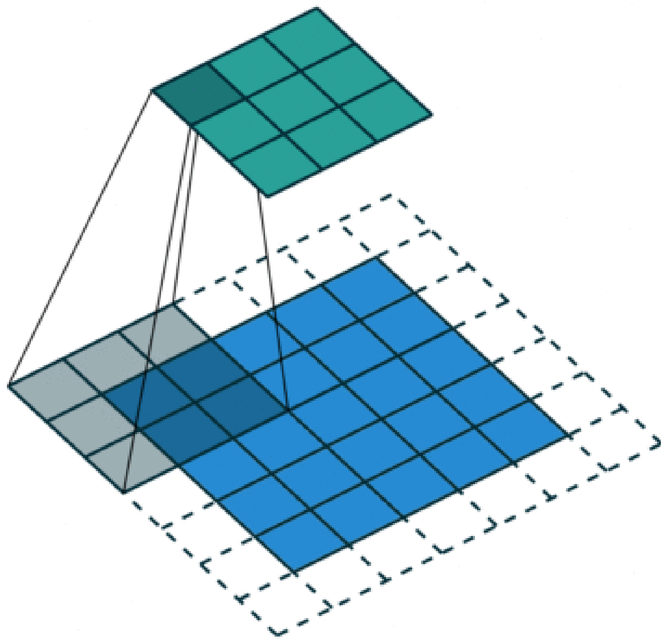
Stride (pas)

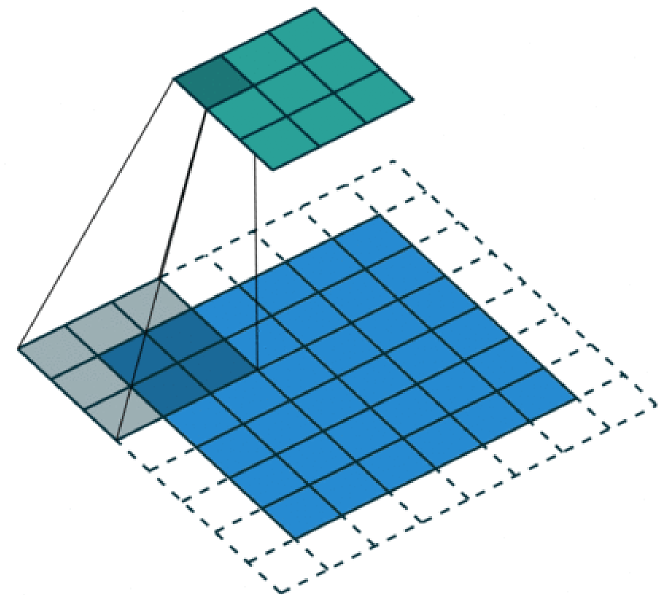
- The stride is the number of pixels between each convolution filter (displacement at each step)
  - Stride of 1: the filter moves one pixel at a time (classic case)
  - Stride of 2: the filter moves two pixels at a time, reducing the size of the output.
  - Etc.

- **In short, padding helps control the size of the image after convolution, while stride controls the distance the filter moves.**

Stride

- Left: image 5x5, stride of 1, padding of 1 → image 3x3
- Right: image 6x6, stride of 2, padding of 1 → image 3x3

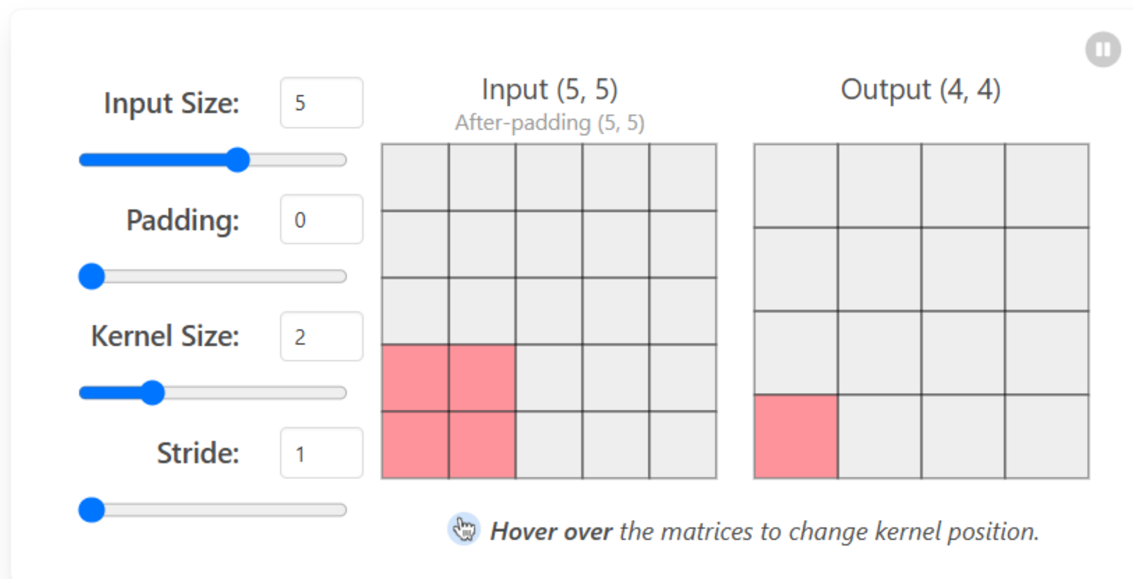




Input : blue   ouput :green

Stride, padding: test online

- https://poloclub.github.io/cnn-explainer/



STYLE TRANSFER BETWEEN IMAGES

1 Upload photo

The first picture defines the scene you would like to have painted.

2 Choose style

Choose among predefined styles or upload your own style image.

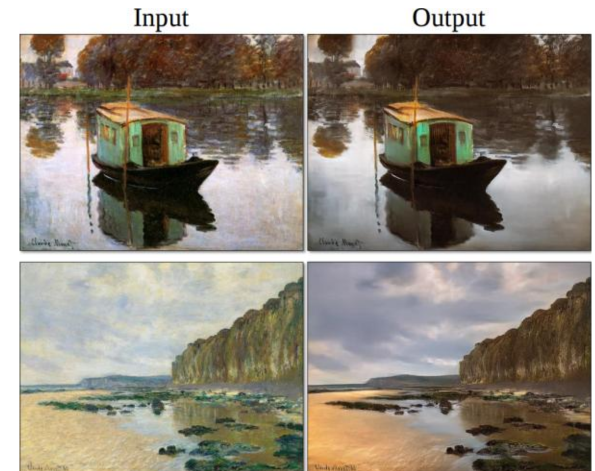
3 Submit

Our servers paint the image for you. You get an email when it's done.

Image Style Transfer Using Convolutional Neural Networks
Gatys etal . CVPR 2016

Style transfer between images

- All the qualities of a good TP
  - Using DL Frameworks
  - Optimization
  - Use networks, but without needing to train them (reasonable calculation time for a practical exercise)



Style transfer between images

- Differentiate
  - Image content: objects and their places/positions/orientations
  - Style : color and textures
- VGG19 for extracting features
  - Each convolution layer will produce a feature map
  - Optimization with two terms: Cost_Content + Cost_Style

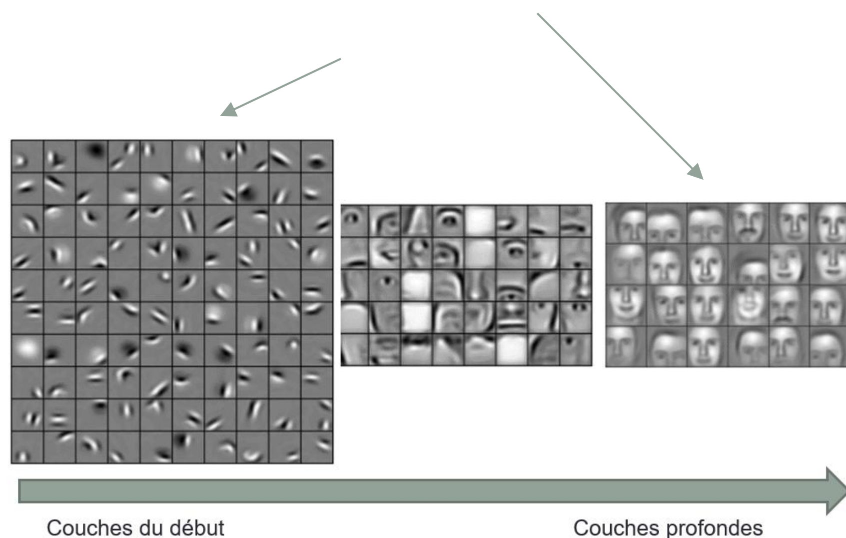
Features at different scales

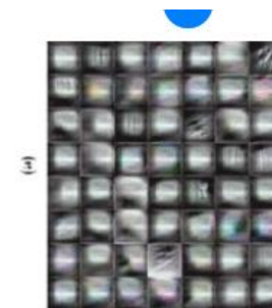
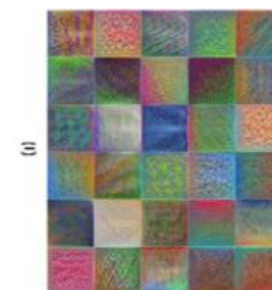
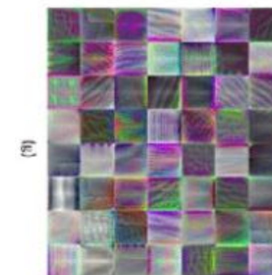
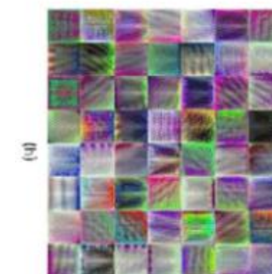
Style transfer between images

VGG19 for extracting features

- Each convolution layer will produce a feature vector of dimension
  Batch_size x N_features x Height x Weight

  - Some layers encode content (towards the bottom of the network), others encode style (towards the beginning)

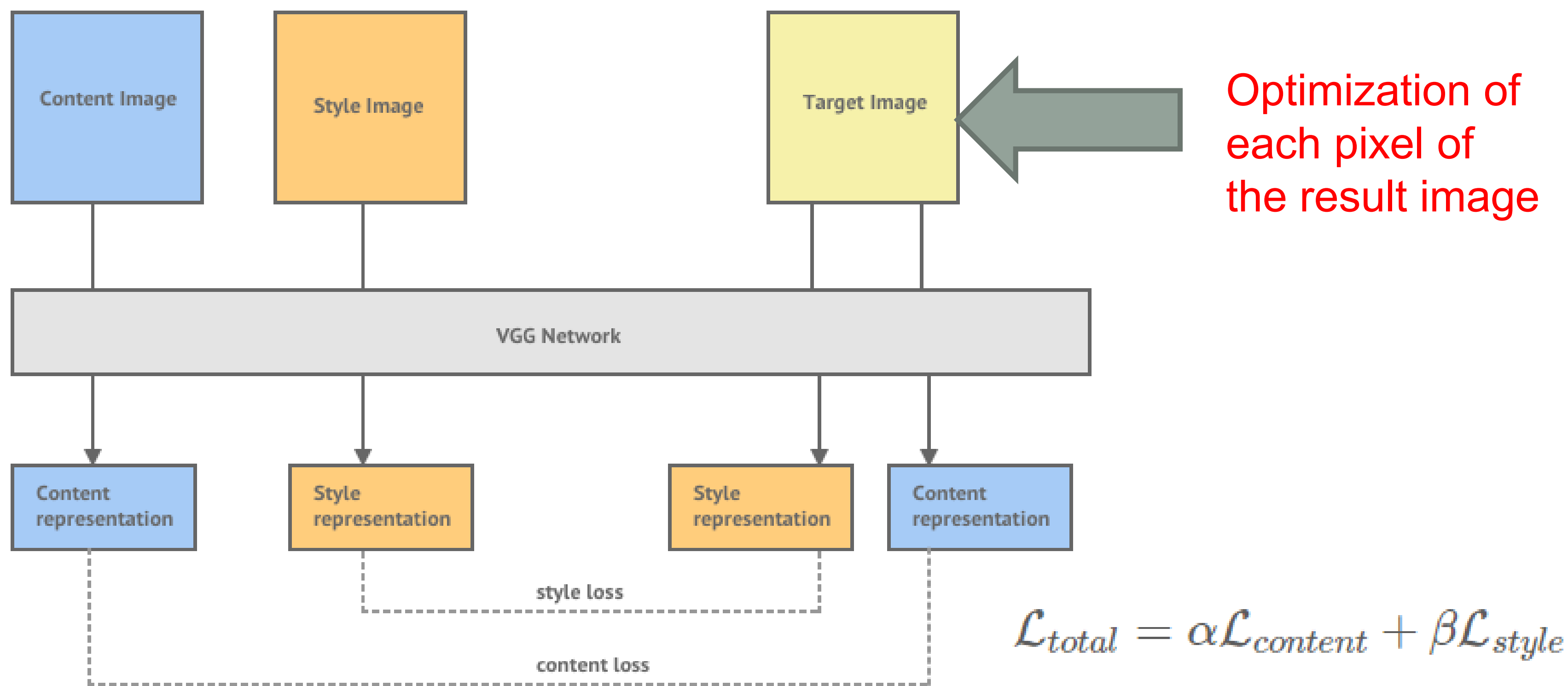


Features visualization of VGG network

Style transfer: optimization

- No network optimization
- The network (VGG) is used to produce the descriptors (features)
- Deeplearning framework (Pytorch) is used to optimize the pixels



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

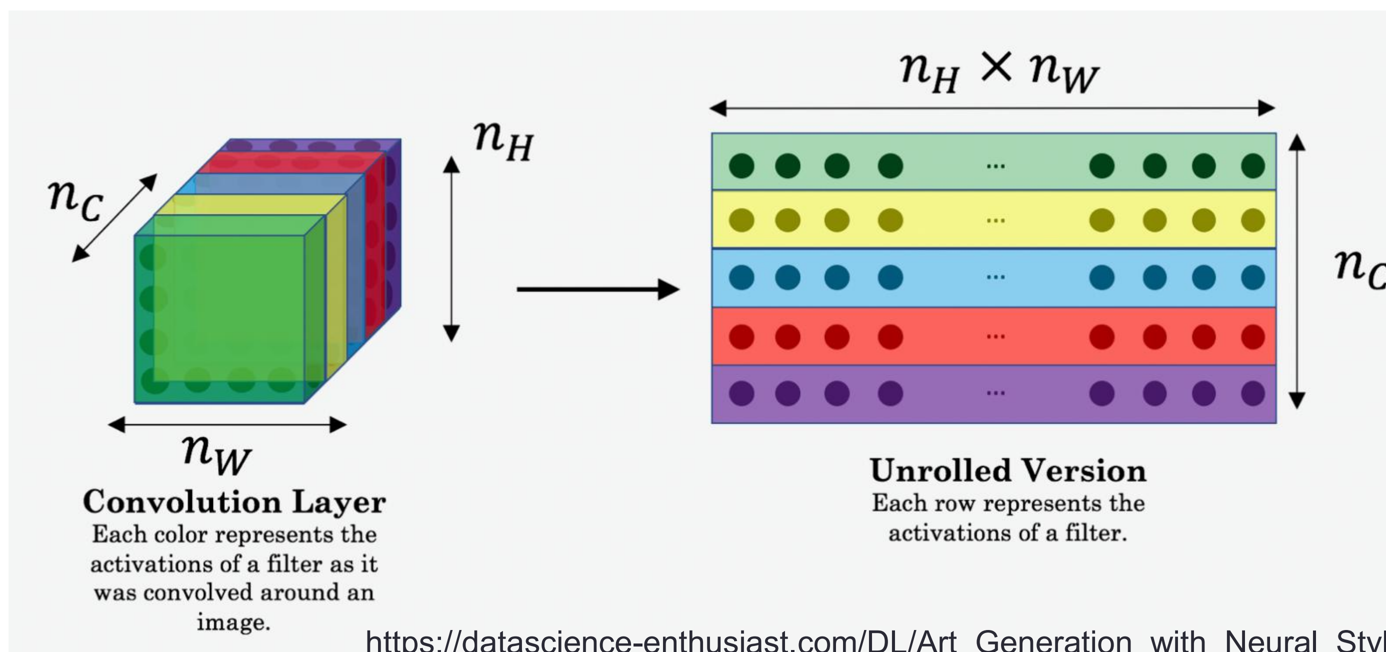
Style transfer between images

VGG19 for extracting features

- Each convolution layer will produce a feature vector of dimension

  N_features (N_c in the figure) x Height x Weight

  ➔ to flatten into N_features x N_pixels with N_pixels = n_height x n_weight



https://datascience-enthusiast.com/DL/Art_Generation_with_Neural_Style_Transfer_v2.html

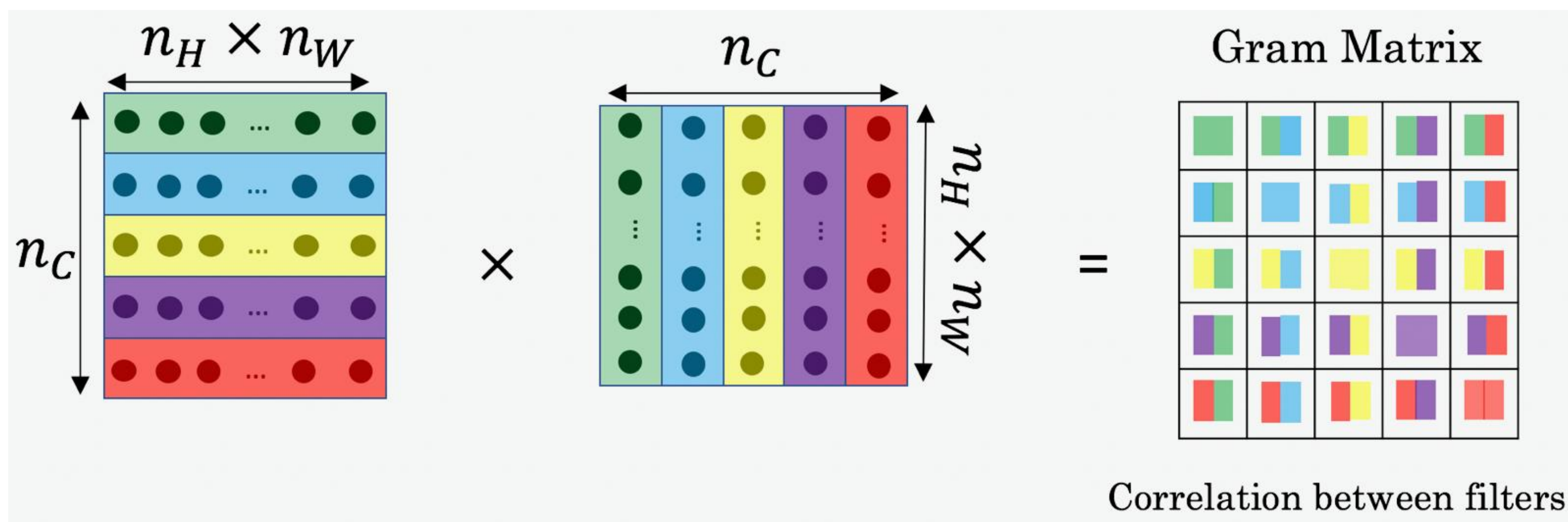
Style transfer: Gram matrix

- **Gram matrix: M x M ^{t}**
  - Dot product between all features
  - Correlation between features

- With a feature matrix F, an entry of the Gram matrix G is the scalar product between 2 features

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

Style transfer: Gram matrix



If an entry in the Gram matrix has a value close to 0, this means that the 2 *features* do not activate simultaneously (no-correlation). And vice versa, if an input has a large value, it means that the 2 *features* activate simultaneously (correlation).

We try to create an image that replicates the same activation patterns of style *features.*

https://datascience-enthusiast.com/DL/Art_Generation_with_Neural_Style_Transfer_v2.html

Style Transfer: Content Cost

- If we can construct an image that has an equivalent feature map for a given convolution level to another image. These two images will have the same content (especially for the deep layers) — but not necessarily the same texture or style.

- Given a convolution layer **l** in VGG, the content cost function is defined as the mean squared error between the feature *map* **F** of the content image **C** and the *feature map* of the generated image **Y** .

$$\mathcal{L}_{content} = \frac{1}{2} \sum_{i,j} (F_{ij}^{l} - P_{ij}^{l})^2$$

Style Transfer: Style Cost

- The style cost calculation is similar to the content cost calculation, but it is calculated from the Gram matrix instead of directly using features .

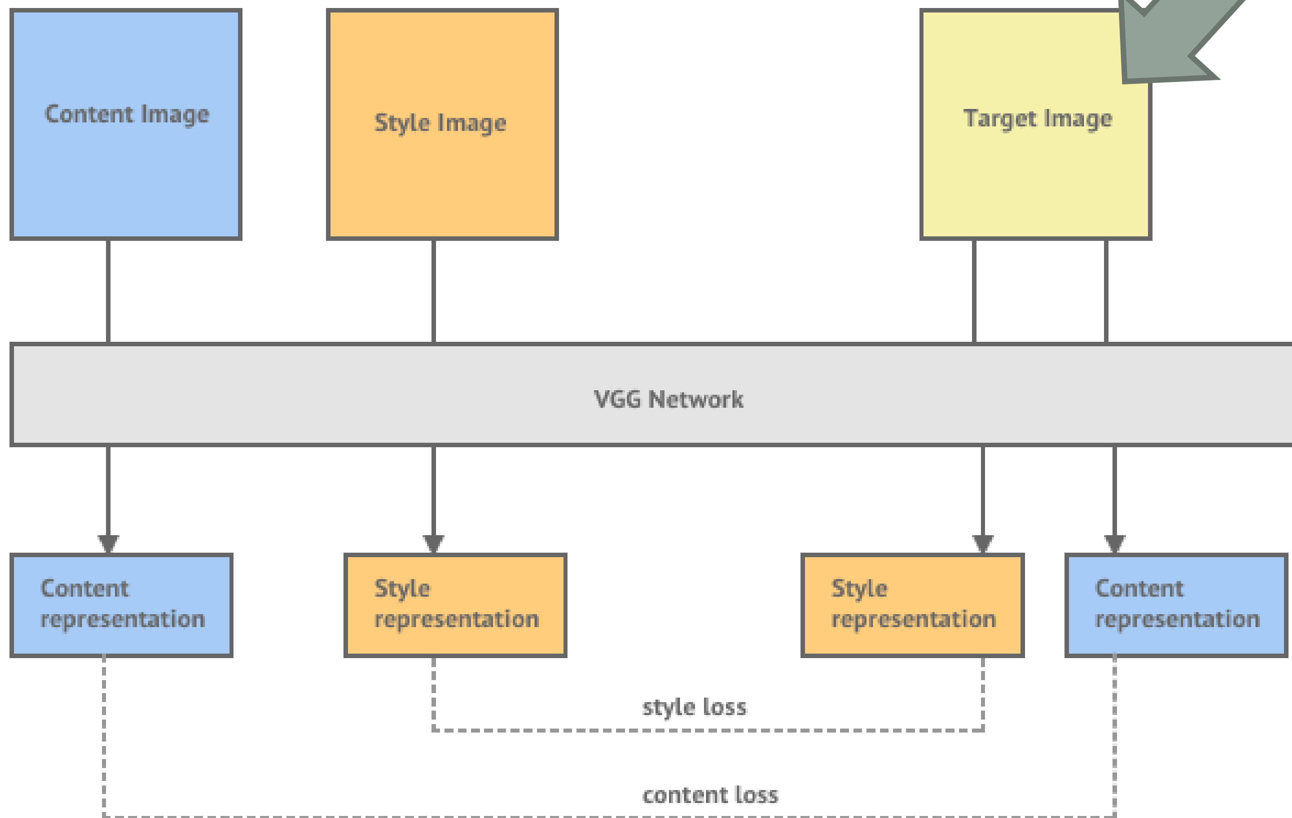
$$\mathcal{L}_{style} = \frac{1}{2}\sum_{l=0}^{L}(G_{ij}^{l} - A_{ij}^{l})^2$$

Style transfer: optimization

- The cost function to be optimized

Optimization of each pixel of the output image

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



Style transfer

content image

style image

generated image

Conclusion

- CNN was the first Deep Learning Revolution
  ➔TP
- Many approaches still to be seen…