MACQUARIE University SYDNEY · AUSTRALIA

DeepDream & Neural Style Transfer

Peizhen Li

March 18, 2024

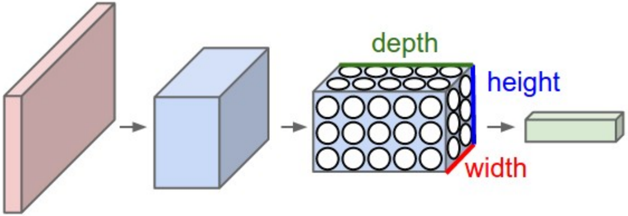
Outline

- Revisit Convolutional Neural Networks
- DeepDream
- Neural Style Transfer

Revisit Convolutional Neural Networks

A ConvNet is made up of Layers. Every Layer has a simple API: It transforms an input 3D volume to an output 3D volume with some differentiable function that may or may not have parameters

- Convolutional layer
- Pooling layer
- Fully connected layer



VGG Network Recap

Published in the paper titled Very Deep Convolutional Networks for Large-Scale Image Recognition

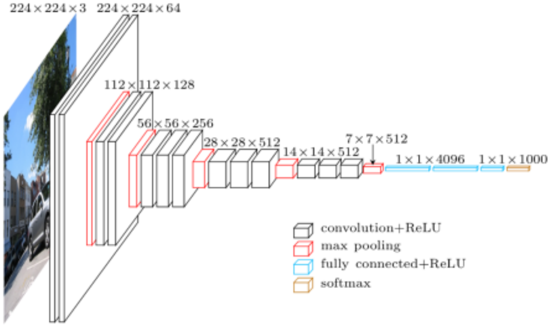


Figure: VGG16: a total of 16 weight layers.

VGG Network Recap

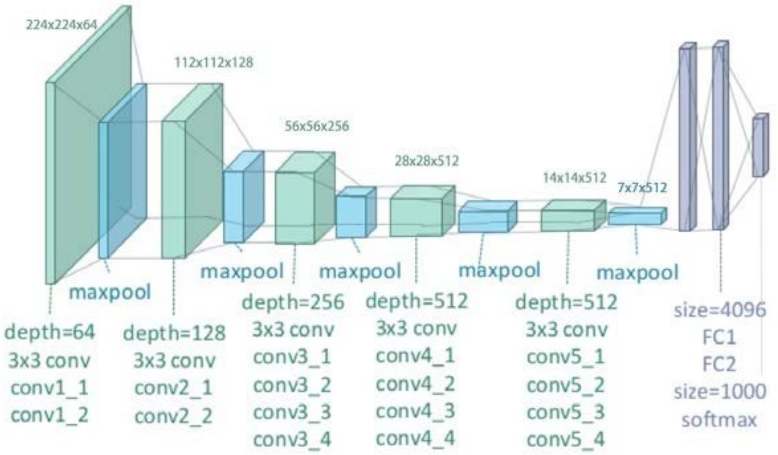


Figure: VGG19: A total of 19 weight layers.

Outline

- Revisit Convolutional Neural Networks
- DeepDream
- Neural Style Transfer

DeepDream

- What is it? An algorithm that visualizes the patterns learned by a neural network
- Over-interprets and enhances the patterns it sees in an image

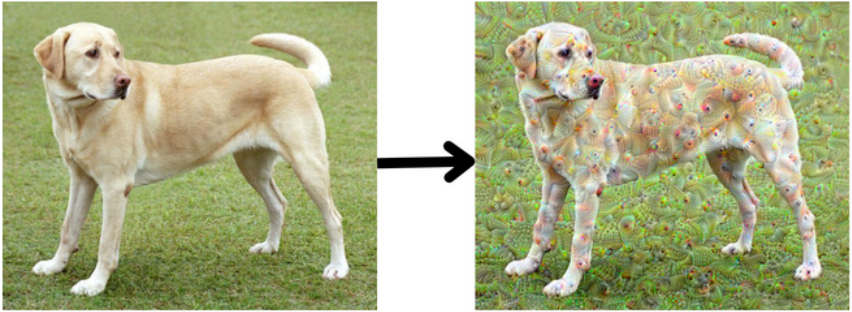


Figure: You can make a neural network “dream” and enhance the surreal patterns it sees in an image

Intuition Behind: Going Deeper into Neural Networks

Whatever you see there, I want more of it!

- Recall canonical ANN training procedures (example)
- What exactly goes on at each layer?
- Amplify certain neuron's activation

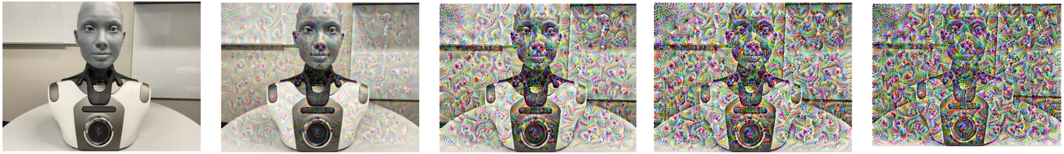
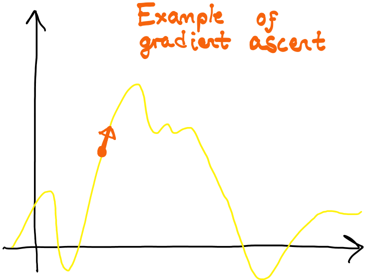


Figure: Dream on Ameca at different steps: 100, 500, 1000, 2000

How Does the DeepDream Algorithm Work

- Built upon a pre-trained convolutional neural network
- Select target layers and compute activations of the chosen layers
- Calculate the gradient of activations w.r.t the input image
- Update the image using gradient ascent



DeepDream Implementation

- Prepare for the feature extraction

```
# Prepare the feature extraction model
base_model = tf.keras.applications.InceptionV3(include_top=False, weights='imagenet')
# maximize the activations of these layers
names = ('mixed3', 'mixed5')
layers = [base_model.get_layer(_layer_name).output for _layer_name in names]
# create the feature extraction model
dream_model = tf.keras.Model(inputs=base_model.inputs, outputs=layers)
```

- Calculate loss: sum of the normalized activations of chosen layers
- Calculate gradients of the loss w.r.t the image, and add them to the original image

More Examples

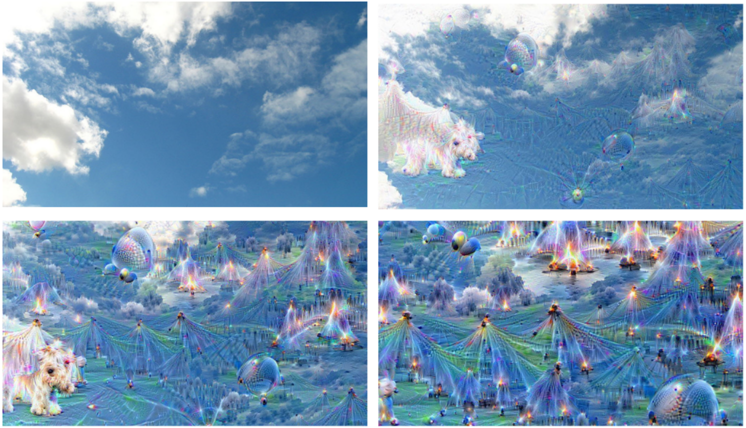


Figure: The image on the top-left is the input image. Ask the GoogleNet to amplify the features recognized by the the inception_5a/3x3 layer. Results after 10, 15, 20 iterations are demonstrated in the last 3 images.

Outline

- Revisit Convolutional Neural Networks
- DeepDream
- Neural Style Transfer

Neural Style Transfer

- What is it?
- Separate and recombine content and style of natural images [1] (how?)

[1] Image Style Transfer Using Convolutional Neural Networks

Content Representation

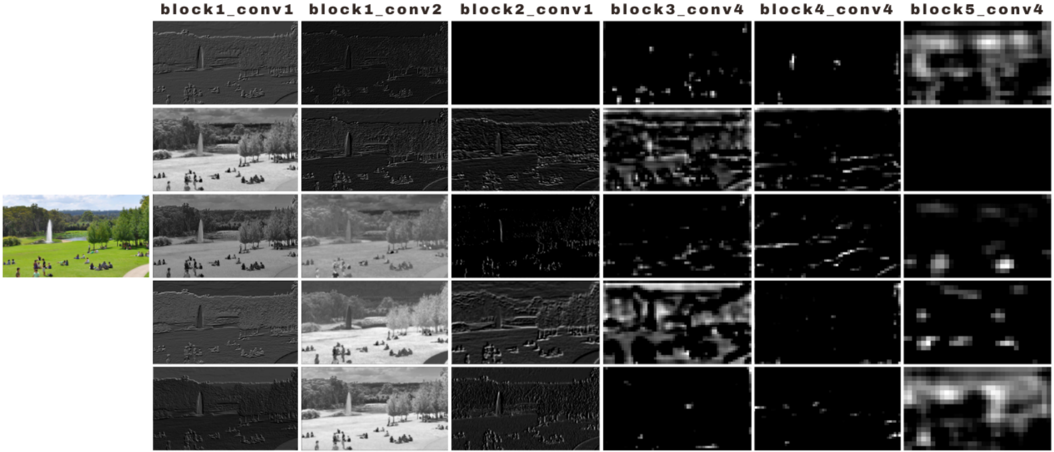


Figure: Visualization of feature maps at certain layers in VGG19.

Content Representation

- Content representation: feature responses in higher layers of the network

$$F^l, P^l \in \mathcal{R}^{N_l \times M_l}$$

$\vec{x}$ $\vec{p}$

Figure: Left: $\vec{x}$, image to update. Right: $\vec{p}$, content image.

Content Reconstruction

- Generate an image that minimizes the content loss

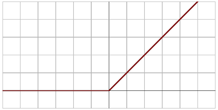
$$\mathcal{L}_{\text{content}}(\vec{p}, \vec{x}, l) = \frac{1}{2} \sum_{i,j} (F_{ij}^{l} - P_{ij}^{l})^2 \tag{1}$$

- Compute derivative w.r.t. the activations in layer l

$$\frac{\partial \mathcal{L}_{\text{content}}}{\partial F_{ij}^{l}} = \begin{cases} (F^{l} - P^{l})_{ij} & F_{ij}^{l} > 0 \\ 0 & F_{ij}^{l} < 0 \end{cases} \tag{2}$$

How to Optimize? Gradient Descent

- Gradient explanation: F^l is the feature representation at layer l with ReLU activation



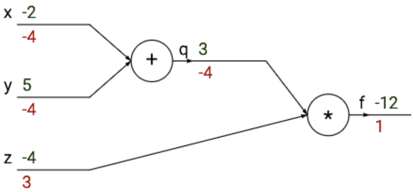
$$\mathrm{ReLU}(x) = \begin{cases} x & x > 0 \\ 0 & x \leq 0 \end{cases} = \max(0, x), \quad \mathrm{ReLU}^{'}(x) = \begin{cases} 1 & x > 0 \\ 0 & x < 0 \end{cases}$$

- Efficient gradient computation: chain rule and backpropagation

$$q = x + y, \quad f = q * z$$
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$

Computational Graph

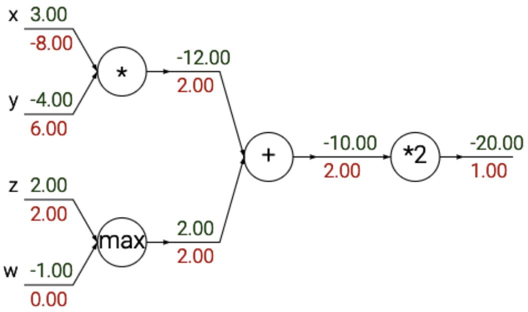
- Gates communicating to each other through gradient signal




```
# set some inputs
x, y, z = -2, 5, -4
# perform the forward pass
q = x + y  # q becomes 3
f = q * z  # f becomes -12
# perform the backward pass (backpropagation) in reverse order:
# first backprop through f = q * z
dfdz = q  # df/dz = q, so gradient on z becomes 3
dfdq = z  # df/dq = z, so gradient on q becomes -4
dqdx = 1.0
dqdy = 1.0
# now backprop through q = x + y
dfdx = dfdq * dqdx  # The multiplication here is the chain rule!
dfdy = dfdq * dqdy
```

Backpropagation: An Intuitive View

- Sum operation distributes gradients equally to all its inputs
- Max operation routes the gradient to the higher input
- Multiply gate takes the input activations, swaps them and multiplies by its gradients



Content Reconstruction in Practice

- Define your model, loss function, select an optimizer and train
- Tensorflow will take care of the rest (no need to write the backprop on your own)

```
# optmizer
opt = tf.keras.optimizers.Adam(learning_rate=0.02, beta_1=0.99, epsilon=1e-1)
# loss function
def content_only_loss(outputs: dict, target_features: dict):
    return tf.add_n([tf.reduce_mean((outputs[name]-target_features[name])**2) for name in outputs.keys()])

@tf.function()
def train_step(model, image_to_gen, target_features, loss_func, optimizer):
    with tf.GradientTape() as tape:
        outputs = model(image_to_gen)
        loss = loss_func(outputs, target_features)
    grad = tape.gradient(loss, image_to_gen)
    optimizer.apply_gradients([(grad, image_to_gen)])
    image_to_gen.assign(clip_0_1(image_to_gen))
```

Content Reconstructions

Figure: Reconstruct the content from different layers: conv1_2, conv2_2, conv3_2, conv4_2, conv5_2.

Style Representation

- Style representation: feature correlations between different filter responses

$$
\begin{aligned}
G^l, A^l &\in \mathcal{R}^{N_l \times N_l} \\
G^l_{ij} &= \sum_k F^l_{ik} F^l_{jk}
\end{aligned}
\tag{3}
$$

$\vec{x}$ $\vec{a}$

Figure: Left: $\vec{x}$, image to update. Right: $\vec{a}$, style image.

Gram Matrix in Practice

- Hermitian matrix of inner products in an inner product space
- G^l_{ij} is the inner product between the vectorised feature maps i and j in layer l

$$G^l_{ij} = \sum_k F^l_{ik} F^l_{jk}$$

- Matrix formulation

$$F^l = \begin{pmatrix} F^l_{11} & \cdots & F^l_{1M_l} \\ \vdots & \ddots & \vdots \\ F^l_{N_l 1} & \cdots & F^l_{N_l M_l} \end{pmatrix}, \quad (F^l)^T = \begin{pmatrix} F^l_{11} & \cdots & F^l_{1N_l} \\ \vdots & \ddots & \vdots \\ F^l_{M_l 1} & \cdots & F^l_{M_l N_l} \end{pmatrix}$$

$$G^l = F^l (F^l)^T \in \mathcal{R}^{N_l \times N_l}$$

Gram Matrix in Practice

Two implementations:

- version1: reshape and matmul

```
def gram_matrix_plain(input_tensor):
    input_tensor = input_tensor[0]  # (H, W, C) , C is the channel size or #filters
    input_tensor = tf.transpose(input_tensor, perm: (2, 0, 1))
    vectorized_fea = tf.reshape(input_tensor, shape: (tf.shape(input_tensor)[0], -1))
    return tf.matmul(vectorized_fea, tf.transpose(vectorized_fea))
```

- version2: einsum

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

Style Reconstruction

- Generate an image that minimizes the style loss

$$\mathcal{L}_{\text{style}}(\vec{a}, \vec{x}) = \sum_{l=0}^{L} w_l E_l \tag{4}$$

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{ij} (G_{ij}^l - A_{ij}^l)^2 \tag{5}$$

- Compute derivative w.r.t. the activations in layer l

$$\frac{\partial E_l}{\partial F_{ij}^l} = \begin{cases} \frac{1}{N_l^2 M_l^2} ((F^l)^T (G^l - A^l))_{ij} & F_{ij}^l > 0 \\ 0 & F_{ij}^l < 0 \end{cases} \tag{6}$$

Style Reconstructions

Figure: Reconstruct the style from a style representation built on different subsets of CNN layers: {conv1_1}, {conv1_1, conv2_1}, {conv1_1, conv2_1, conv3_1}, {conv1_1, conv2_1, conv3_1, conv4_1}, {conv1_1, conv2_1, conv3_1, conv4_1, conv5_1}.

Style Transfer

- Jointly minimize the distance of feature representations of a white noise image from
  - the content representation of the photograph in one layer and
  - the style representation of the painting defined on a number of layers

$$\mathcal{L}_{\text{total}}(\vec{p}, \vec{a}, \vec{x}) = \alpha \mathcal{L}_{\text{content}}(\vec{p}, \vec{x}) + \beta \mathcal{L}_{\text{style}}(\vec{a}, \vec{x}) \tag{7}$$

Style Transfer

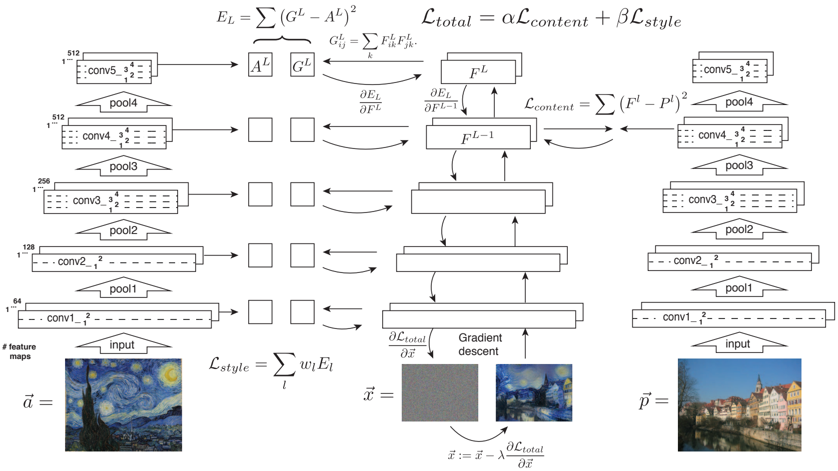


Figure: Overview of the style transfer algorithm.

Neural Style Transfer Implementation

- Define our model for both content and style feature extraction
- Loss function and optimizer
- Wrapping up

```
@tf.function()
def train_step(model, image_to_gen, loss_func, optimizer,  **kwargs):
    with tf.GradientTape() as tape:
        outputs = model(image_to_gen)
        loss = loss_func(outputs, **kwargs)
    grad = tape.gradient(loss, image_to_gen)
    optimizer.apply_gradients([(grad, image_to_gen)])
    image_to_gen.assign(clip_0_1(image_to_gen))
```

More Examples

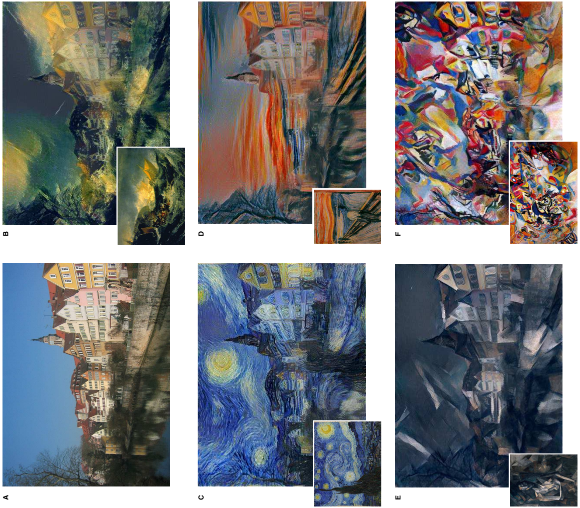


Figure: **A** Neckarfront in Tubingen, Germany. **B** *The Shipwreck of the Minotaur.* **C** *The Starry Night* **D** *Der Schrei.* **E** *Femme nue assise.* **F** *Composition VII.*

Trade-off Between Content and Style Matching

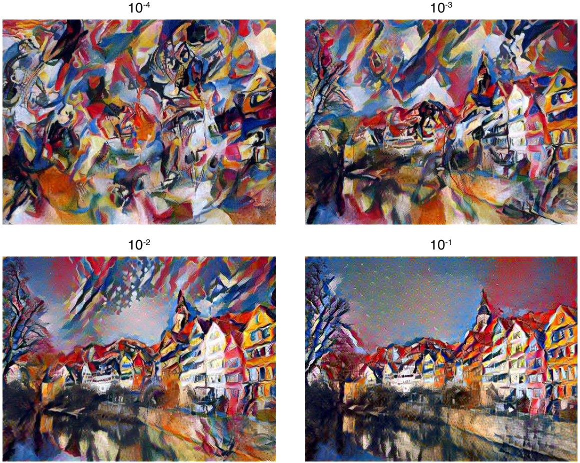


Figure: Relative weighting of matching content and style of the respective source images.

Effect of Matching Different Layers in CNN

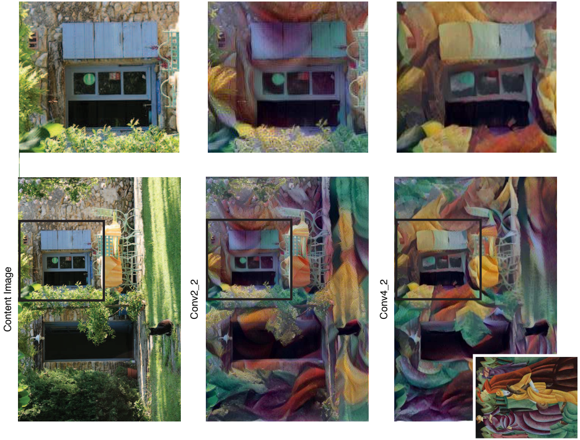


Figure: Matching the content representation on layer 'conv2_2' preserves much of the fine structure of the original image. While the content is displayed in the style of the painting when matching the content representation on layer 'conv4_2'.

Initialisation of the Gradient Descent

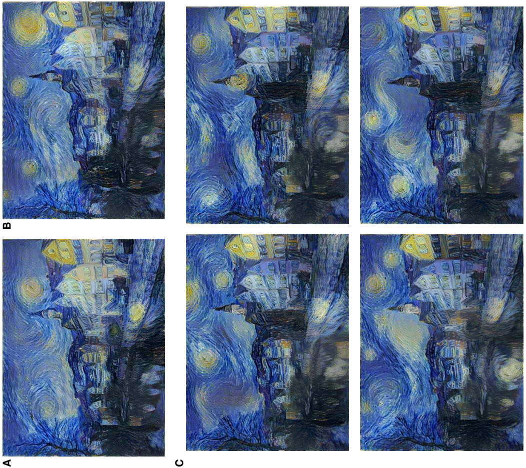


Figure: **A** Initialisation from the content image. **B** Initialisation from the style image. **C** Four samples of images initialisation from different white noise images. ($\alpha/\beta = 1 \times 10^{-3}$ for all images)

FYI

- DeepDream vs Neural Style Transfer
  - generative
  - feature map
  - gradient w.r.t the input image
  - ...
- References:
  - Convolutional Neural Networks for Visual Recognition
  - Inceptionism: Going Deeper Into Neural Networks
  - Image Style Transfer Using Convolutional Neural Networks
  - Very Deep Convolutional Networks for Large-Scale Image Recognition
- Code is available here

Thank you very much!
Q&A

peizhen.li1@students.mq.edu.au