Neural network

Convolution neural network

Khiem Nguyen

Email	khiem.nguyen@glasgow.ac.uk
MS Teams	khiem.nguyen@glasgow.ac.uk
Whatsapp	+44 7729 532071 (Emergency only)

May 18, 2025

Table of Contents

1. Transfer learning
2. Siamese network and application in face recognition
3. Neural style transfer

Transfer learning: Problem statement

Quick remark

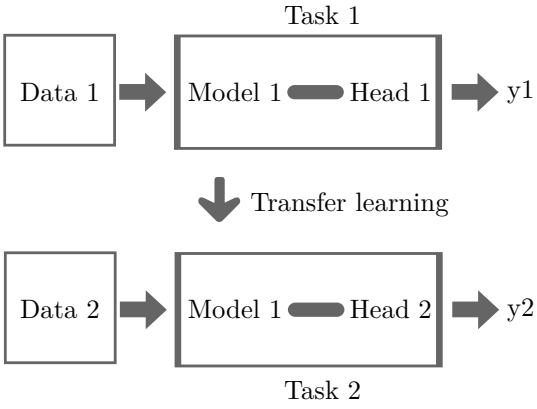
- You should see code demonstration uploaded on Moodle some time ago.
- We use the example presented in the uploaded Jupyter Notebook.

Problem statement

- We are given a very small dataset of bees and ants.
- Train a neural network (convolution neural network) to perform the classification task on this small dataset

Key issue *very small dataset*

Transfer learning: General idea

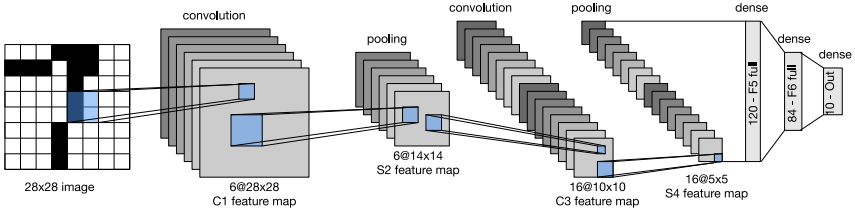


Task 1 – Model 1 + Head 1: classify many objects (perhaps ≥ 1000 objects)

$\downarrow$

Task 2 – Model 1 + Head 2: classify just bees and ants

Transfer learning: Model & Head

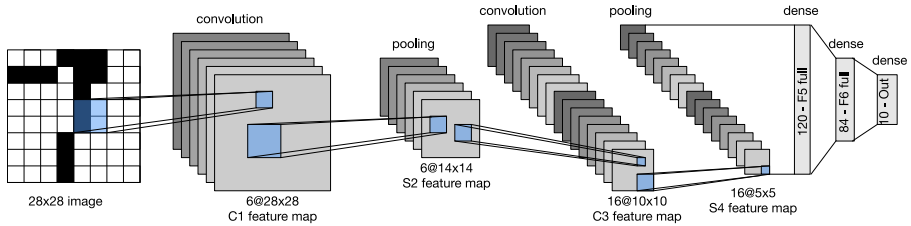


- In general a model refers to everything in the neural network.
- For now, we refer to convolution layers, pooling layers and several layers before the fully connected layer as '**model**'.
- The last or the few last fully connected layers as '**head**'.

Key point

- You want to change a small part, and keep the rest, of a very deep convolution neural network to fit your problem statement.
- The 'head' and the 'main body' can be defined by the practitioner in a custom fashion.

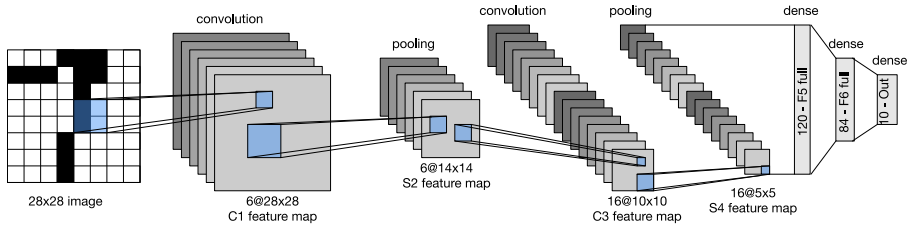
Transfer learning: Model & Head



What would we do in our “bee-vs-ant” classification problem?

Various ways – (Reminder for myself: Lecturer must draw on this slide):

Transfer learning: Model & Head

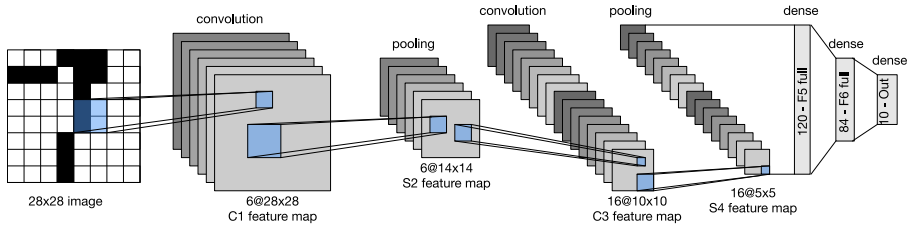


What would we do in our “bee-vs-ant” classification problem?

Various ways – (Reminder for myself: Lecturer must draw on this slide):

- Replace the last fully connected layer by a fully connected layer with only 1 output (probability that $y = 1$).

Transfer learning: Model & Head

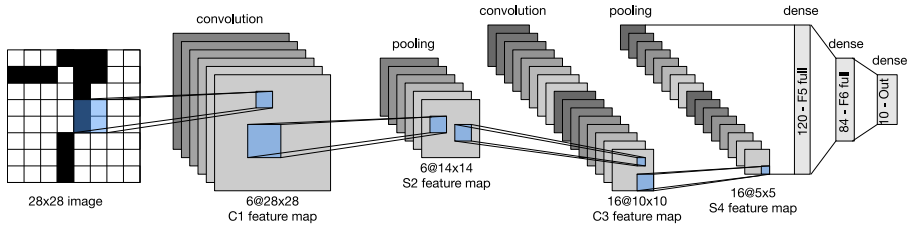


What would we do in our “bee-vs-ant” classification problem?

Various ways – (Reminder for myself: Lecturer must draw on this slide):

- Replace the last fully connected layer by a fully connected layer with only 1 output (probability that $y = 1$).
- Replace the last fully connected layer and even two fully connected layers before it using different numbers of input and output features.

Transfer learning: Model & Head



What would we do in our “bee-vs-ant” classification problem?

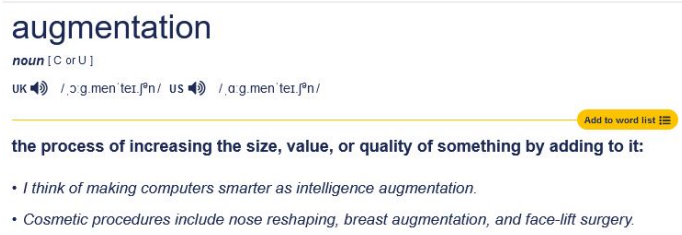
Various ways – (Reminder for myself: Lecturer must draw on this slide):

- Replace the last fully connected layer by a fully connected layer with only 1 output (probability that $y = 1$).
- Replace the last fully connected layer and even two fully connected layers before it using different numbers of input and output features.
- Replace and and add more layers if we want.

Data augmentation

- What is data augmentation?

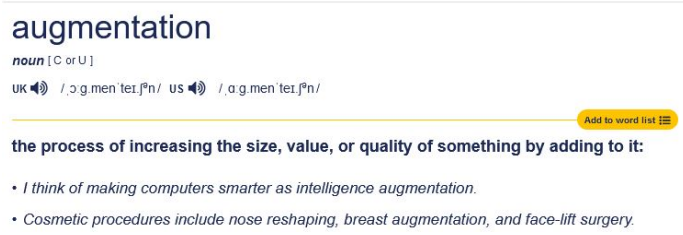
Oxford dictionary



Data augmentation

- What is data augmentation?

Oxford dictionary



augmentation

noun [C or U]

UK /ˌɔːg.menˈteɪ.ʃᵊn/ US /ˌɑːg.menˈteɪ.ʃᵊn/

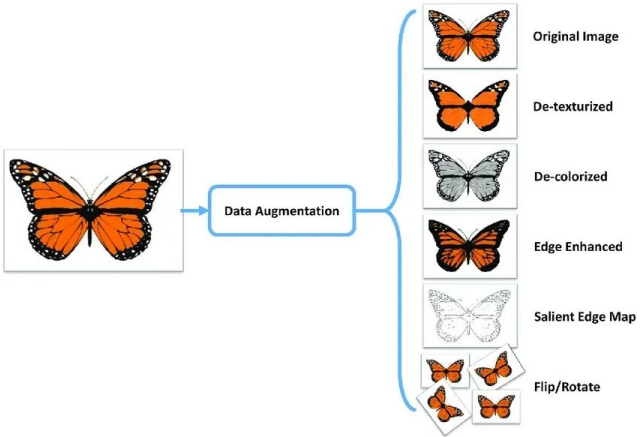
Add to word list

the process of increasing the size, value, or quality of something by adding to it:

- *I think of making computers smarter as intelligence augmentation.*
- *Cosmetic procedures include nose reshaping, breast augmentation, and face-lift surgery.*

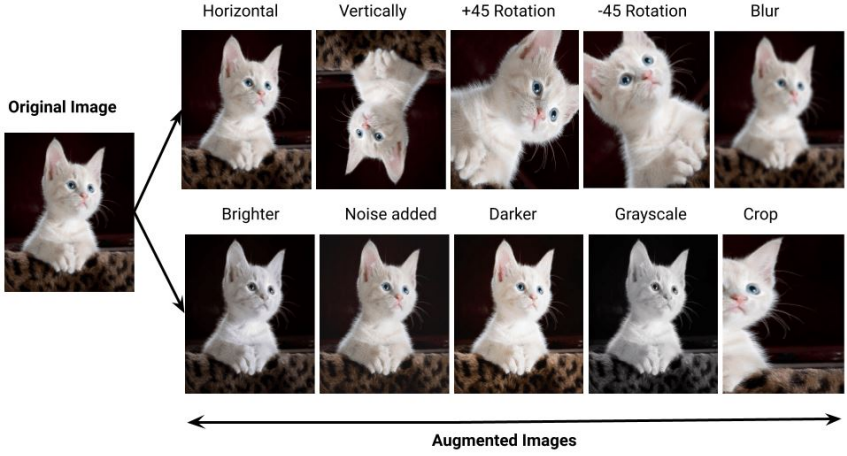
- Why do we want data augmentation?
  - We have a small dataset of images.
  - We want to generate more data from the original set of images.
  - The more data the model is trained on, the better the generalization the model can reach.

Data augmentation: Quick examples



Data augmentation: Butterfly example

Data augmentation: Quick examples



Data augmentation: Cat example

Data augmentation in PyTorch

Getting started with transforms v2

• NOTE

Try on Colab or go to the end to download the full example code.

This example illustrates all of what you need to know to get started with the new `torchvision.transforms.v2` API. We'll cover simple tasks like image classification, and more advanced ones like object detection / segmentation.

First, a bit of setup

```
from pathlib import Path
import torch
import matplotlib.pyplot as plt
plt.rcParams["savefig.bbox"] = 'tight'

from torchvision.transforms import v2
from torchvision.io import decode_image

torch.manual_seed(1)

# If you're trying to run that on Colab, you can download the assets and the
# helpers from https://github.com/pytorch/vision/tree/main/gallery/
from helpers import plot
img = decode_image(str(Path('../assets') / 'astronaut.jpg'))
print(f"{type(img) = }, {img.dtype = }, {img.shape = }")
```

Note that the most updated version is transformation v2. Click on Me!

Data augmentation in PyTorch

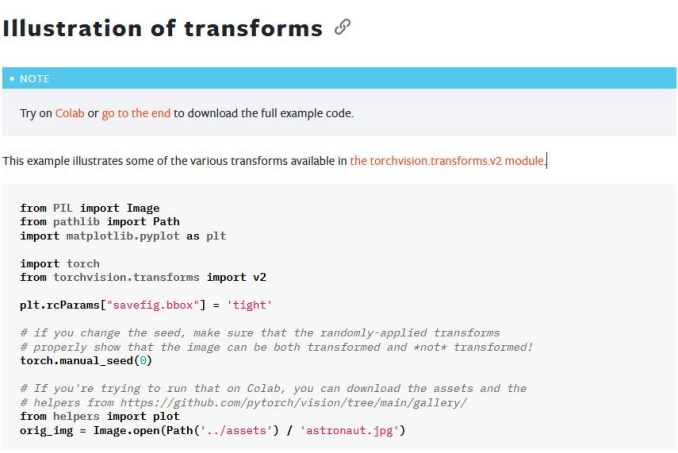


Illustration of transforms

• NOTE

Try on Colab or go to the end to download the full example code.

This example illustrates some of the various transforms available in the torchvision.transforms.v2 module.

```
from PIL import Image
from pathlib import Path
import matplotlib.pyplot as plt

import torch
from torchvision.transforms import v2

plt.rcParams["savefig.bbox"] = 'tight'

# if you change the seed, make sure that the randomly-applied transforms
# properly show that the image can be both transformed and *not* transformed!
torch.manual_seed(0)

# If you're trying to run that on Colab, you can download the assets and the
# helpers from https://github.com/pytorch/vision/tree/main/gallery/
from helpers import plot
orig_img = Image.open(Path('../assets') / 'astronaut.jpg')
```

Personally, I'd like to start from here to learn it quickly. Click on Me!

Data augmentation in PyTorch

➪ *Key messages*:

- ➢ Too many transformations to learn.
- ➢ Only need to understand the mechanisms.
- ➢ Learn a few useful and frequently transformations.
- ➢ Look up transformations when needed before writing your own code to do that.

➪ *Some useful transformations* (bullets contain links to PyTorch documentation, just click on them!)

- ☆ Normalize
- ☆ Randrom Crop
- ☆ Random Resized Crop
- ☆ Random Horizontal Flip
- ☆ Random Vertical Flip

Transformation examples using `torchvision.transforms.v2`

Original image

Transformation examples using `torchvision.transforms.v2`

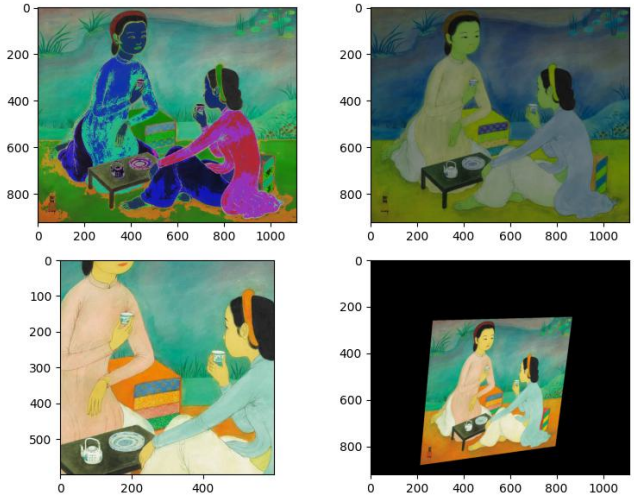
- Python code for making transformation using `torchvision.transforms.v2`

```
from torchvision.transforms import v2
file_path = './figures/painting-vietnamese-ladies.jpg'
original_image = Image.open(file_path)

transformer_list = [v2.RandomSolarize(threshold=192.0),
                    v2.ColorJitter(brightness=.5, hue=.3),
                    v2.CenterCrop(size=600),
                    v2.RandomPerspective(distortion_scale=0.6, p=1.0)]
plt.figure(figsize=(8, 6))
for j in range(len(transformer_list)):
    plt.subplot(2, 2, j+1)
    transformed_image = transformer_list[j](original_image)
    plt.imshow(transformed_image)
plt.tight_layout()
```

- For more fun, please visit Illustrations of transforms and of course our Moodle.

Transformation examples using `torchvision.transforms.v2`



Random Solarized (top left) – Color Jitter (top right)
Center Crop (bottom left) – Random Perspective (bottom right)

Table of Contents

1 Transfer learning

2 Siamese network and application in face recognition

3 Neural style transfer

Face Verification versus Face Recognition

- **Face Verification**
  - Input: Image, name/ID
  - Output: Whether the input image is that of the claimed person
- **Face Recognition**
  - Has a database of K people
  - Get an input image
  - Output: whether the image is any of the K persons (or "*not recognized*")

One-shot learning

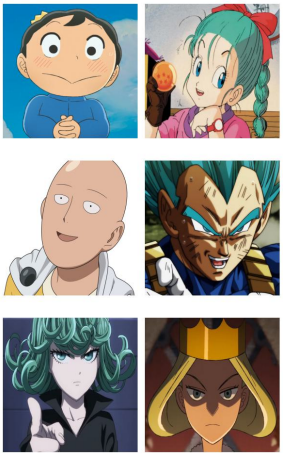


Figure: Database

Yes: Vegeta

No: A clown?

Learning from one example to recognize the person again

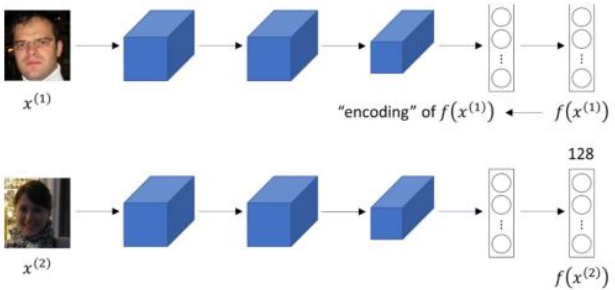
Learning a “similiartiy” function

$$d(\text{image}(1), \text{image}(2)) = \text{degree of difference between two images}$$

$$\begin{cases} \text{if } d(\text{image}(1), \text{image}(2)) \leq \tau & \rightarrow \quad \text{same} \\ \text{if } d(\text{image}(1), \text{image}(2)) > \tau & \rightarrow \quad \text{different} \end{cases}$$

Figure: Database

Siamese Network



Reference 1[1] – Reference 2[2]

➪ Distance function

$$d(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}) = \|\mathbf{f}(\mathbf{x}^{(1)}) - \mathbf{f}(\mathbf{x}^{(2)})\|^2 = \sum_{i=1}^{n_f} \left[f_i(\mathbf{x}^{(1)}) - f_i(\mathbf{x}^{(2)}) \right]^2$$

[1] Learning a similarity metric discriminatively, with application to face verification
[2] DeepFace: Closing the Gap to Human-Level Performance in Face Verification

Siamese Network

Goal of learning

- Parameters of neural network define the encoding $\mathbf{f}(\mathbf{x}^{(i)})$
- Learn parameters so that:
  - If $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$ are the same person (same identity), then $\|d(\mathbf{f}(\mathbf{x}^{(i)}), \mathbf{f}(\mathbf{x}^{(j)}))\|^2$ is small.
  - If $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$ are different persons (different identities), then $\|d(\mathbf{f}(\mathbf{x}^{(i)}), \mathbf{f}(\mathbf{x}^{(j)}))\|^2$ is large.

Learning Objective

(a) Anchor (A)

(b) Positive (P)

(a) Anchor (A)

(b) Negative (N)

➪ One training example has the form (A, P, N).

Learning Objective

(a) Anchor (A)

(b) Positive (P)

(a) Anchor (A)

(b) Negative (N)

- One training example has the form (A, P, N).
- We want distance function satisfying

$$\underbrace{\|\mathbf{f}(A) - \mathbf{f}(P)\|^2}_{d(A,P)} \leq \underbrace{\|\mathbf{f}(A) - \mathbf{f}(N)\|^2}_{d(A,N)} \text{ easily satisfied by setting } d(A, P) = d(A, N)$$

Learning Objective

(a) Anchor (A)

(b) Positive (P)

(a) Anchor (A)

(b) Negative (N)

- One training example has the form (A, P, N).
- We want distance function satisfying

$$\underbrace{\|\mathbf{f}(A) - \mathbf{f}(P)\|^2}_{d(A,P)} \leq \underbrace{\|\mathbf{f}(A) - \mathbf{f}(N)\|^2}_{d(A,N)} \text{ easily satisfied by setting } d(A, P) = d(A, N)$$

- We update the condition with a positive margin $\alpha > 0$

$$\|\mathbf{f}(A) - \mathbf{f}(P)\|^2 - \|\mathbf{f}(A) - \mathbf{f}(N)\|^2 + \alpha \leq 0$$

Triplet loss function

- Given 3 images: Anchor (A), Positive (P), Negative (N) put into one tuple (A, P, N)

[3] FaceNet: A Unified Embedding for Face Recognition and Clustering

Triplet loss function

- Given 3 images: Anchor (A), Positive (P), Negative (N) put into one tuple (A, P, N)
- Triplet loss function for one training example – Reference[3]:

$$\mathcal{L}(A, P, N) = \max\left(\|\mathbf{f}(A) - \mathbf{f}(P)\|^2 - \|\mathbf{f}(A) - \mathbf{f}(N)\|^2 + \alpha, 0\right)$$

[3] FaceNet: A Unified Embedding for Face Recognition and Clustering

Triplet loss function

- Given 3 images: Anchor (A), Positive (P), Negative (N) put into one tuple (A, P, N)
- Triplet loss function for one training example – Reference[3]:

$$\mathcal{L}(A, P, N) = \max\left(\|\mathbf{f}(A) - \mathbf{f}(P)\|^2 - \|\mathbf{f}(A) - \mathbf{f}(N)\|^2 + \alpha, 0\right)$$

- Triplet loss function for a batch of training examples

$$\mathcal{L}(A, P, N) = \sum_{i=1}^{m} \mathcal{L}\left(A^{(i)}, P^{(i)}, N^{(i)}\right)$$

[3] FaceNet: A Unified Embedding for Face Recognition and Clustering

Triplet loss function

- Given 3 images: Anchor (A), Positive (P), Negative (N) put into one tuple (A, P, N)
- Triplet loss function for one training example – Reference[3]:

$$\mathcal{L}(A, P, N) = \max\left(\|\mathbf{f}(A) - \mathbf{f}(P)\|^2 - \|\mathbf{f}(A) - \mathbf{f}(N)\|^2 + \alpha, 0\right)$$

- Triplet loss function for a batch of training examples

$$\mathcal{L}(A, P, N) = \sum_{i=1}^{m} \mathcal{L}\left(A^{(i)}, P^{(i)}, N^{(i)}\right)$$

- Training set: For 1k persons, we may need around 5k – 10k pictures to form dataset of (A, P, N) triplets.

[3] FaceNet: A Unified Embedding for Face Recognition and Clustering

Triplet loss function

- Triplet Loss in PyTorch: `TripletMarginLoss` – Click on Me!
  - `TripletMarginLoss` receives the encodings $\mathbf{f}(A), \mathbf{f}(P), \mathbf{f}(N)$ (not the images A, P, N themselves) as inputs.
  - `TripletMarginLoss` allows p-norm (default $p = 2$ means Euclidean norm):

$$\|\mathbf{f}\|_p = \left(\sum_i^{n_f} f_i^p \right)^{1/p}$$

- For shallow convolution neural network, have a look at Reference[4].

[4] Learning local feature descriptors with triplets and shallow convolutional neural networks

Choosing the triplets (A, P, N)

- During training, if A, P, N are chosen randomly, it may happen that $d(A, P) + \alpha \leq d(A, N)$ is easily satisfied.

Choosing the triplets (A, P, N)

- During training, if A, P, N are chosen randomly, it may happen that $d(A, P) + \alpha \leq d(A, N)$ is easily satisfied.
- Choose triplets that are "hard" to train on. Repeat

$$d(A, P) + \alpha \leq d(A, N)$$

Choosing the triplets (A, P, N)

- During training, if A, P, N are chosen randomly, it may happen that $d(A, P) + \alpha \leq d(A, N)$ is easily satisfied.
- Choose triplets that are "hard" to train on. Repeat

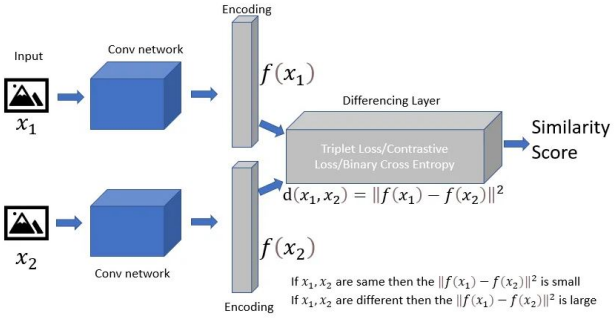
$$d(A, P) + \alpha \leq d(A, N)$$

- If we choose (A, P, N) so that $d(A, P)$ is quite close to $d(A, N)$, the model will try harder to learn the encodings $\mathbf{f} = \mathbf{f}(\mathbf{x})$ better and better. In other words, the encoding for each person will be more characteristic for that person.

Training set using triplet loss

Anchor	Positive	Negative
		
		
		

Learning the similiarty function



A different way to measure similarity score → Use binary cross entropy

$$\widehat{y} = \sigma\left(\sum_{k=1}^{n_f} w_i |f_k(\mathbf{x})^{(i)} - f_k(\mathbf{x}^{(j)}) + b\right)$$

Dataset: pair $\mathbf{X}^{(a)} = (\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$ with label $y^{(a)} = 1$ if $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$ are the same person,

with label $y^{(a)} = 0$ if $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$ are two different persons.

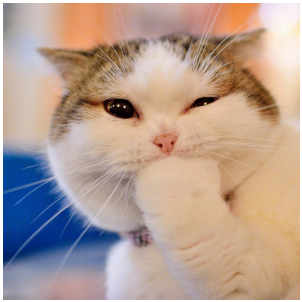
Table of Contents

1 Transfer learning

2 Siamese network and application in face recognition

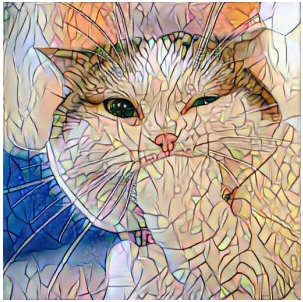
3 Neural style transfer

What is neural style transfer



(a) Content Image

(b) Style Image



(c) Generated Image

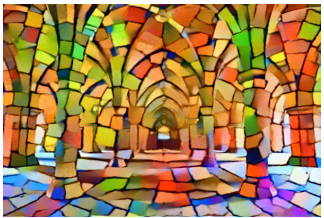
Content Image + Style Image $\longrightarrow$ Generated Image

Reference. Click on Me!

What is neural style transfer

(a) Content Image

(b) Style Image

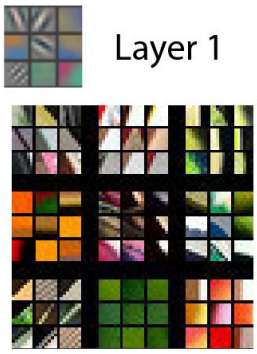


(c) Generated Image

Content Image + Style Image $\longrightarrow$ Generated Image

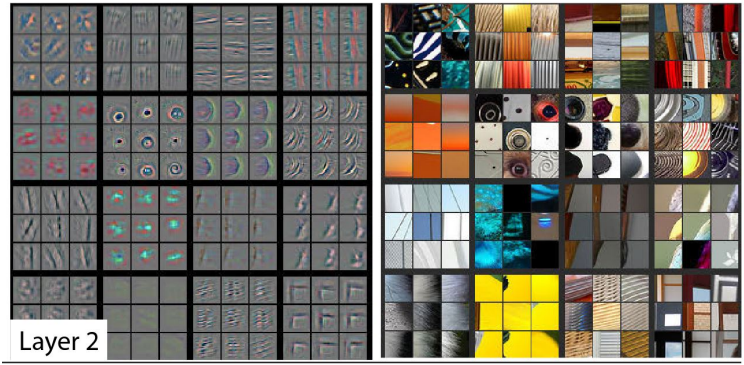
Reference. Click on Me!

What are deep ConvNets learning?



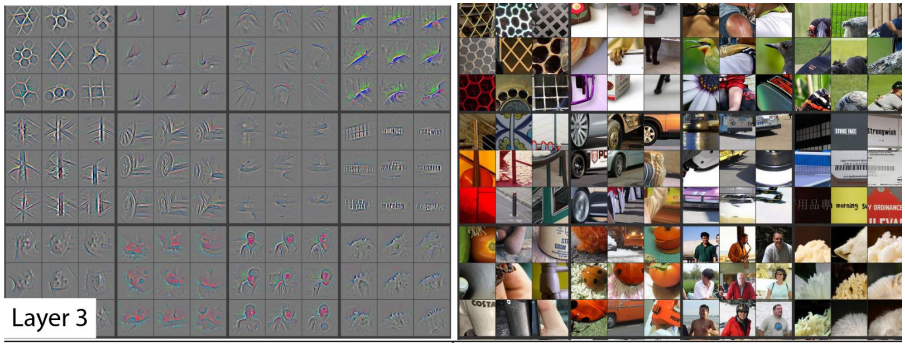
For layers 2-5 the authors showed the top 9 activations in a random subset of feature maps across the validation data, projected down to pixel space using our “deconvolutional” network approach. Reference. The authors studied the famous AlexNet at the time.

What are deep ConvNets learning?



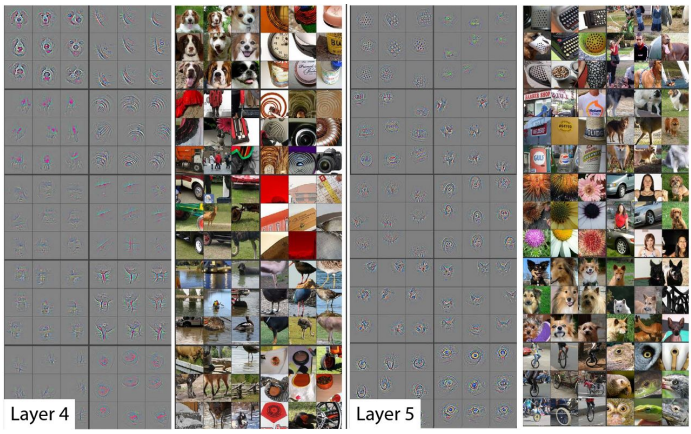
For layers 2-5 the authors showed the top 9 activations in a random subset of feature maps across the validation data, projected down to pixel space using our “deconvolutional” network approach. Reference. The authors studied the famous AlexNet at the time.

What are deep ConvNets learning?



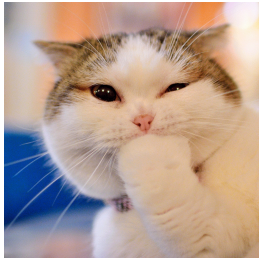
For layers 2-5 the authors showed the top 9 activations in a random subset of feature maps across the validation data, projected down to pixel space using our “deconvolutional” network approach. Reference. The authors studied the famous AlexNet at the time.

What are deep ConvNets learning?



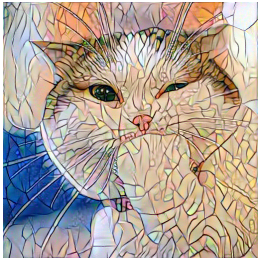
For layers 2-5 the authors showed the top 9 activations in a random subset of feature maps across the validation data, projected down to pixel space using our “deconvolutional” network approach. Reference. The authors studied the famous AlexNet at the time.

Neural style transfer loss function



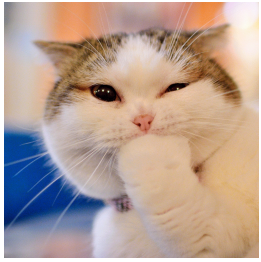
(a) Content C

(b) Style S

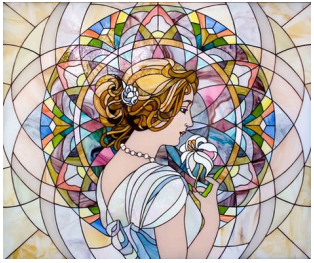


(c) Generated G

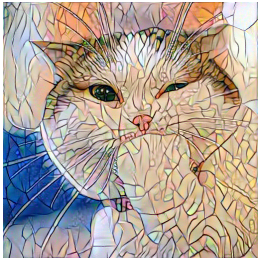
Neural style transfer loss function



(a) Content C



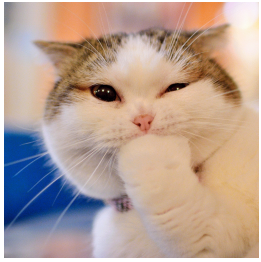
(b) Style S



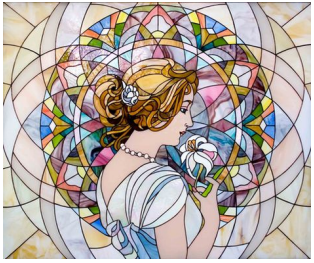
(c) Generated G

$$\mathcal{L}_{\text{content}}(C, G) +$$

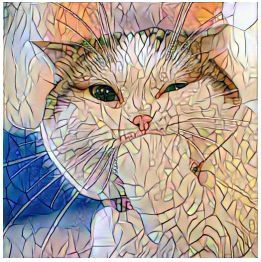
Neural style transfer loss function



(a) Content C



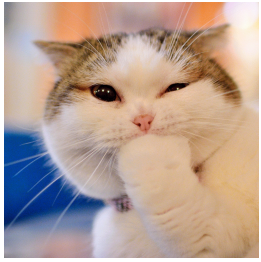
(b) Style S



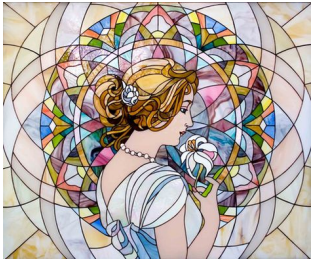
(c) Generated G

$$\mathcal{L}_{\text{content}}(C, G) + \quad \mathcal{L}_{\text{style}}(S, G)$$

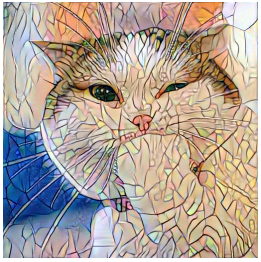
Neural style transfer loss function



(a) Content C



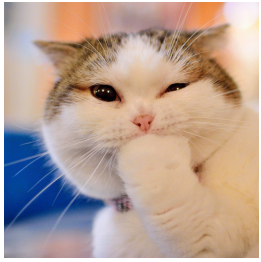
(b) Style S



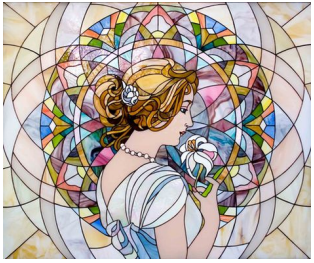
(c) Generated G

$$\mathcal{L}(G) = \quad \mathcal{L}_{\text{content}}(C, G) + \quad \mathcal{L}_{\text{style}}(S, G)$$

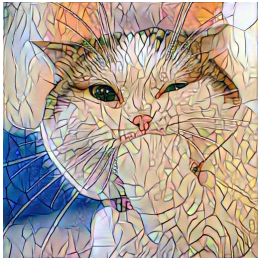
Neural style transfer loss function



(a) Content C



(b) Style S



(c) Generated G

$$\mathcal{L}(G) = \alpha\,\mathcal{L}_{\text{content}}(C, G) + \beta\,\mathcal{L}_{\text{style}}(S, G)$$

Find the generated image G

➪ *Generate the neural-style transfer image G*

1. Initialize G randomly
   For example: $G : 256 \times 256 \times 3$
2. Use optimization method to minimize $\mathcal{L}(G)$

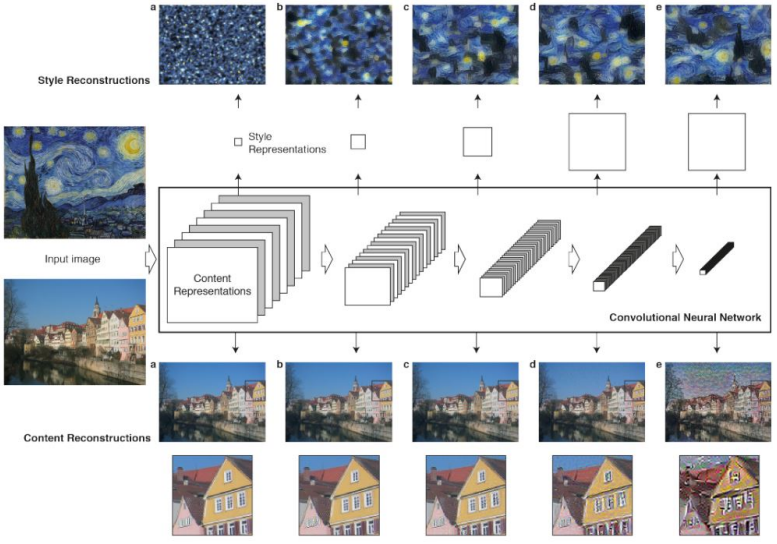
Find the generated image G

➪ *Generate the neural-style transfer image G*

1. Initialize G randomly
   For example: $G : 256 \times 256 \times 3$
2. Use optimization method to minimize $\mathcal{L}(G)$

➪ Example of training process:

Style and Content Reconstruction



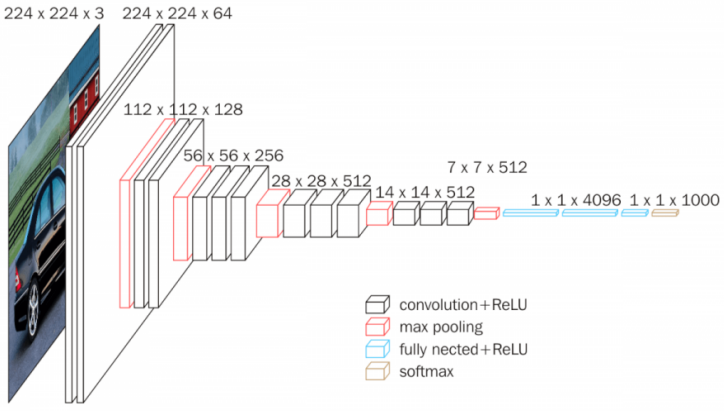
Content loss function

$$\mathcal{L}(G) = \alpha \mathcal{L}_{\text{content}}(C, G) + \beta \mathcal{L}_{\text{style}}(S, G)$$

- Say, e.g., we use hidden layer l to compute content loss
- Use pre-trained ConvNet, .e.g, VGG network (Reference – VGGs on PyTorch)
- Let $\mathbf{a}^{[l](C)}$ and $\mathbf{a}^{[l](G)}$ be the activation of layer l on the images
- If $\mathbf{a}^{[l](C)}$ and $\mathbf{a}^{[l](G)}$ are similar, both images have similar content

$$\mathcal{L}_{\text{content}}(C, G) = \frac{1}{2} \| \mathbf{a}^{[l](C)} - \mathbf{a}^{[l](G)} \|^2$$

Content loss function



$$\mathcal{L}_{\text{content}}(C, G) = \frac{1}{2}\|\mathbf{a}^{[l](C)} - \mathbf{a}^{[l](G)}\|^2$$

Intuition about style of an image

Are the channels in one layer correlated/uncorrelated?

Are the feature maps in one layer correlated/uncorrelated?

Style matrix

- Let $a^{[l]}_{i,j,k}$ denote activation at (i, j, k) in the order (Channel, Height, Width). Gram matrix:

$$G^{[l]}_{rs} = \sum_{i=1}^{n^{[l]}_H} \sum_{j=1}^{n^{[l]}_W} a^{[l]}_{rij} a^{[l]}_{sij}$$

- Gram matrices for style image and generated image:

$$G^{[l](S)}_{rs} = \sum_{i=1}^{n^{[l]}_H} \sum_{j=1}^{n^{[l]}_W} a^{[l](S)}_{rij} a^{[l](S)}_{sij} \qquad G^{[l](G)}_{rs} = \sum_{i=1}^{n^{[l]}_H} \sum_{j=1}^{n^{[l]}_W} a^{[l](G)}_{rij} a^{[l](G)}_{sij}$$

- Style loss function for layer $[l]$

$$\mathcal{L}^{[l]}_{\text{style}}(S, G) = \left\| \mathbf{G}^{[l](S)} - \mathbf{G}^{[l](G)} \right\|^2 = \sum_r \sum_s \left(G^{[l](S)}_{rs} - G^{[l](G}_{rs} \right)^2$$

Style loss function

- Style loss function for layer $[l]$
  According to the paper, the authors made normalization:

$$\mathcal{L}_{\text{style}}^{[l]}(S, G) = \frac{1}{2n_H^{[l]} n_W^{[l]} n_C^{[l]}} \sum_r \sum_s \left(G_{rs}^{[l](S)} - G_{rs}^{[l](G)} \right)^2$$

$$\mathcal{L}_{\text{style}}(S, G) = \sum_l \lambda^{[l]} \mathcal{L}_{\text{style}}^{[l]}(S, G)$$

  where $\lambda^{[l]}$ are user-defined weights for computing the style loss (Repeat $[l]$ refers to the l^{th} layer).

- Further notes:
  - My implementation does not need those normalization and also weight coefficients $\lambda^{[l]}$.
  - These constants pretty much go to the effect of the weights α and β in the total loss function

$$\mathcal{L}^G = \alpha \mathcal{L}_{\text{content}}(C, G) + \beta \mathcal{L}_{\text{style}}(S, G)$$