Week #4 :- Special applications: Face recognition & Neural Style Transfer

Face Recognition:-

What is face recognition?

Face detection + Liveliness ⇒ Face recognition.

Liveliness:- Whether the picture is real human or not. (→ himself)

Face Verification vs Face recognition:

Verification:- 1:1 problem.

- Input: image or name/ID
- Output: Whether the input image is that of claimed person.

Recognition:-

- Has a database of K persons
- Get an Input: Image
- Output: ID if the image is any of the K persons (or "not recognized")

Let's see how can we build a face verification system and we will use that to develop face recognition.

One Shot learning :- What this means :-

- "You need to be able to recognize an image given just one single image"

→ actually what we have learned so far is that, more the training examples, more the accuracy of your algorithm will be

One approach :-

- This doesn't work because, what if the amount of the persons in your team is not fixed what if another person joins your team, do you have to train your Convnet again? Seems like, not a good idea.

Instead You would do something like :

Learning a "similarity" function

- $d(img1, img2)$ = degree of difference b/w images

So, if two images are of same person's you want it to output a small number otherwise a very large number.

Threshold

if $d(img1, img2) \leq \tau$ "Same person"

" " " $> \tau$ "Different person"

- That's how you address a verification problem

To use this for **recognition task**:-

Given the input image, you compare that ~~with~~ with the images from your data base using the $d(img1, img2)$ function that will be learned and that's how the recognition problem is solved.

→ Now if another person joins your team, you just add another picture of that new person and it just works fine.

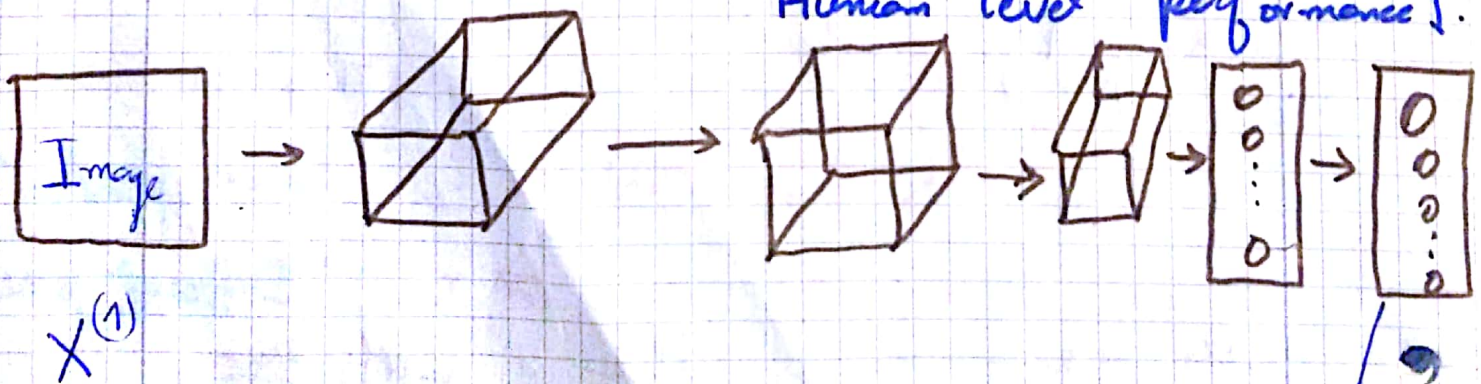
→ Let's see how you can actually train a neural network to learn this function "d"

Siamese Network:-

Job of the 'd' function is to input two images and tell you how close and far they are from being similar.

Good way to do it is through Siamese network.

[Taigman et. al., 2014. DeepFace closing the gap to Human level performance].



The way you build a face recognition is the way you want to ~~learn~~ compare two pictures:-

What you could do, is feed the second picture $X^{(2)}$ with some parameters to some neural network and get a different encoding of that picture called $f(x^{(2)})$

$$d(x^{(1)}, x^{(2)}) = \| f(x^{(1)}) - f(x^{(2)}) \|_2^2$$

- Now how do you train this Siamese network?

→ So, you want to train a N.N such that the mapping or encoding it computes results in the function 'd'.

Parameters of NN define an encoding $f(x^{(i)})$

Learn parameters so that:

- if $x^{(i)}, x^{(j)}$ are the same person, $\| f(x^{(i)}) - f(x^{(j)}) \|_2^2$ is small.
- if $x^{(i)}, x^{(j)}$ are the different persons, $\| f(x^{(i)}) - f(x^{(j)}) \|_2^2$ is large.

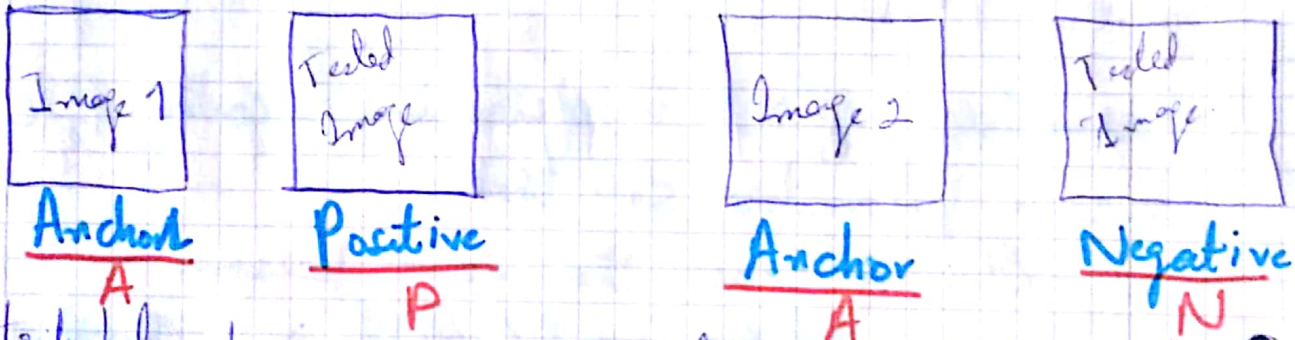
Now, how do you define an objective function for this?

Tripled Loss:- One good way to learn the parameters of the neural network is to define the objective function called "Tripled loss" and apply gradient descent on that.

[Schroff et al., 2015, FaceNet: A unified embedding for face recognition and Clustering]

To apply the tripled loss, you need to compare pairs of images

e.g. given a picture, to learn the parameters of a neural network, you have to look at several pictures at the same time.



In tripled loss, you are always going to look one anchor image and you want the distance b/w anchor and positive image (meaning same person) and the distance of anchor from the negative image (meaning different person) much further apart.

This is what gives rise the term tripled loss - you always will be looking at 3 images. anchor, positive and negative image.

You want:

$$\underbrace{\|f(A) - f(P)\|^2}_{d(A,P)} \leq \underbrace{\|f(A) - f(N)\|^2}_{d(A,N)}$$

$$\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 \leq 0 \quad \text{— (1)}$$

→ One trivial way to satisfy this is to learn everything $f(A), f(P), f(N) = 0$.

→ Another trivial way to satisfy the (1) expression is to have identical values like $f(A) = f(P) = f(N)$

So, in order to prevent from above 2 cases happening, we need to make sure that not only the expression from the left hand side is ≤ 0 but it should be quite a bit smaller than that such that:

$$\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 \leq 0 - \alpha$$

α — another hyper parameter

This prevents the network from outputting trivial solutions. Normally α is on left hand side.

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

α — Margin.

So, e.g. if $d(A,P) = 0.5$ and $d(A,N) = 0.51$ then without margin the (1) equation would be satisfied, but that's not a good criteria we want $d(A,N)$ to be 0.7 at least or loosely speaking much bigger.

So, to ensure that, we set some margin that our algorithm needs to overcome in order to give verdict that, person is same

→ Lets take the equation ② and formalize it to define the triplet loss.

Loss function :-

Given 3 images A, P, N

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

You want the blue underlined expression to be less than zero, if it is then we will get 0 in output and the output is negative, otherwise the output is positive if that is not ~~greater~~ less than or equal to zero.

Cost function :-

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

Suppose Training set : 10k pictures of 1K persons.

what you will do is, take 1.0k pictures and use it to generate / select triplets and train your algorithm by applying gradient descent.

You need pairs of (A, P), so for this algo training you need multiple pictures atleast for same persons

Choosing the triplets A, P, N

- During training if A, P, N are chosen randomly, $d(A,P) + \alpha \leq d(A,N)$ is easily satisfied because chances are that, A and N can be very different pictures.

  Choose the triplets that're "hard" to train on so that $d(A,P) \approx d(A,N)$.

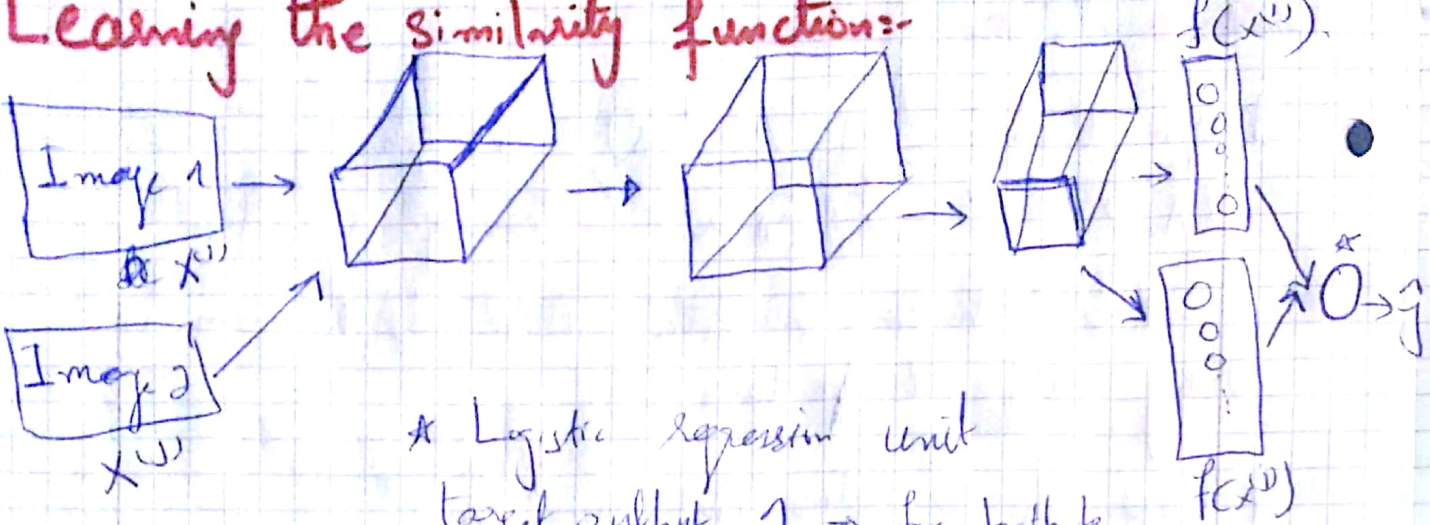
- this way algorithm will have to try harder to try to keep the thing on the left down and the right one up. So that, there is at least a margin of left or right side.

→ Choosing random triplets won't let gradient descend any more and your (2) identity will be easily satisfied. Choosing it carefully will push the gradient descend to work harder

Face Verification and Binary Classification:-

Besides using a triplet loss, there is another way to perform this task. Let's see how face recognition can be easily posed as binary classification problem.

Learning the similarity function:-



* Logistic regression unit

target output 1 ⇒ for both to be same persons

output 0 ⇒ if both of them are different

→ face recognition as binary classification problem ⇒ alternative to triplet loss

Now what does final logistic regression do?

$$\hat{y} = \sigma\left(\sum_{k=1}^{\text{total units in FC}} w_i \left| f(x^{(i)})_k - f(x^{(j)})_k \right| + b\right)$$

w_i: weight; b: bias

Compute and see whether the images are of same or different persons.

→ Other variations of blue underlined expression:-

$$\frac{\left(f(x^{(i)})_k - f(x^{(j)})_k\right)^2}{f(x^{(i)})_k + f(x^{(j)})_k}$$

χ^2 → Kai encoding

So:

Input : Pair of images : Output 0/1 whether same or different.

One computational hack is that:

- You compute embeddings of new person's image and the one already in your database and compare them.

→ You can pre-compute the embeddings of the images already in your database and then you won't have to compute it again and again.

- Face Verification: Supervised learning.

Input: pairs of images

Output: 0/1 ⇒ use backprop for learning.

Neural Style Transfer:

What is Neural Style Transfer?

- Take one image and another image of an animation or artistic style. What neural style transfer will do is that, it will allow you to generate an image which will be the 1st original image painted in style of the second one.

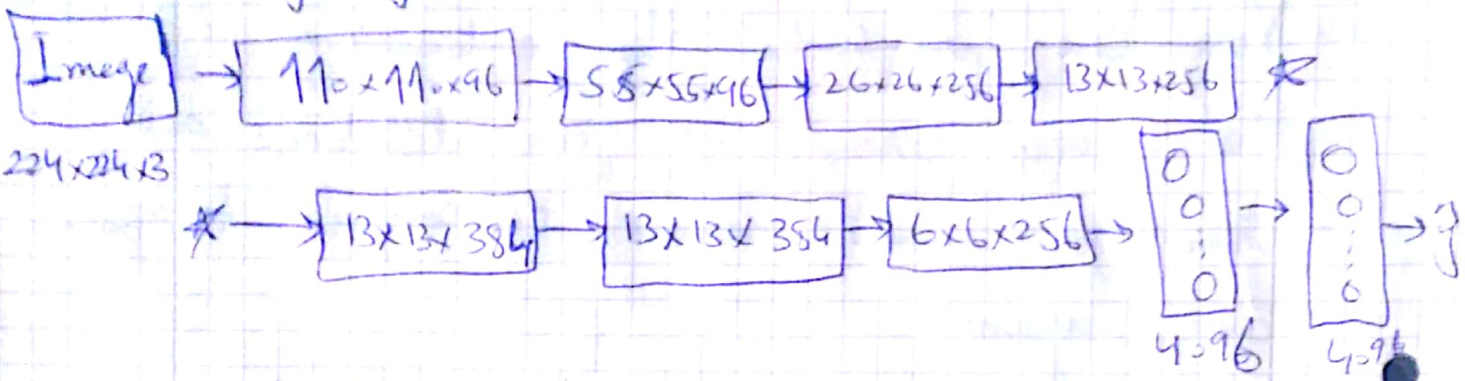
Let's define Notations

- Image to style ⇒ Content (C)
- Style Image ⇒ Style (S)
- Generated Image ⇒ (G)

So, in order to do that, let's see what are deep ConvNets actually doing?

What are deep ConvNets doing?

Let's say you have trained a ConvNet



→ We want to visualize what the hidden layers in this network are computing.

→ Let's start like this:-

Pick a unit in layer 1. Find the 9 image patches that maximize the unit's activation.

In other words:-

Pass your training set through the ConvNet and figure out, what is the image that maximizes that particular units activation.

You'll find small patches of 9 images (under consideration) that will affect the maximization of that unit.

→ Now you can pick a different hidden unit in the layer and do the same thing.

→ You'll see different patterns observed by different hidden

units. One main thing you can observe is that, layer 1 is detecting simple things such as edge or particular shade of color.

[Zeiler and Fergus., 2013. Visualizing and understanding Convolutional networks]

What if you do this for some of the deeper layers hidden in the network.

→ In deeper layer, a hidden unit will see a larger region of the image

So as a result units in the deeper layers will start detecting more complex patterns.

Now that we have somewhat proper understanding of how does CNNs work, let's use this intuition in order to build full algorithm.

- Cost function :-

To build a neural style transfer system let's define a cost function for the generated image and later you'll see that, by reducing that cost function we can generate the image we want.

Content(C) + Style (S) → Generated Image. (G).

- Apply J(G) to estimate how good is a generated image and use gradient descent to optimize it.

We will define 1 pad Cost function :-

$$J(G) = \alpha J_{content}(C, G) + \beta J_{style}(S, G)$$

Checks the similarity b/w Content of generated image and Content image

Check similarity b/w Content of generated and style image

$\alpha, \beta \Rightarrow$ weights to specify the relative weighing b/w content cost and style cost.

[Gatys et al., 2015, A neural algorithm of artistic style, Images on slide generated by Justin Johnson]

Find the generated image G

1. Initiate G randomly
   G: $100 \times 100 \times 3 \Rightarrow$ any size
2. Use gradient descent to minimize $J(G)$

$$G := G - \frac{\partial}{\partial G} J(G).$$

→ Randomly initialized image will be a random noise and with every iteration it will converge.

Now let's define Cost function

Content Cost function:-

- Overall Cost function is:-

$$J(G) = \alpha J_{content}(C, G) + \beta J_{style}(S, G)$$

- Say you use hidden layer l, to compute content cost.

→ If layer l is very earlier layer then large content of your generated image would be very similar to your content image.

→ If l is very deep layer then, it's just asking, "Well if there is a dog in your content image then make sure there is a dog somewhere in your generated image." So, in practice layer l is chosen somewhere in between.

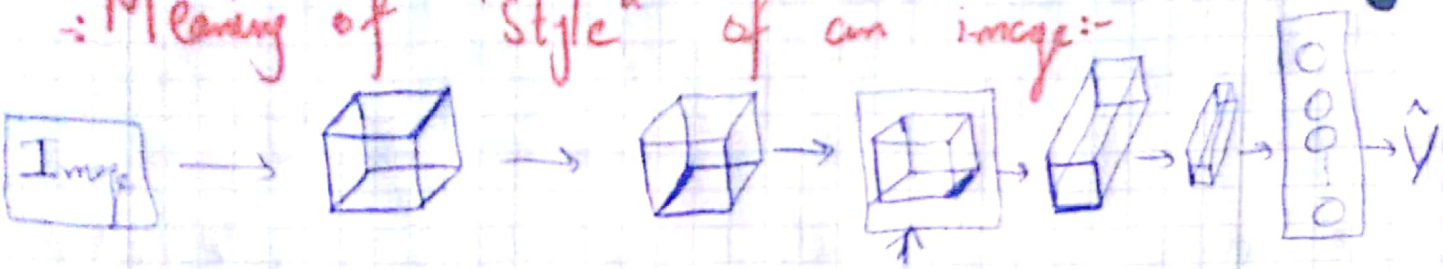
- Use pre-trained ConvNet (e.g. VGG Network)
- Let $a^{[l](C)}$ and $a^{[l](G)}$ be the activation of layer l on the images
- If $a^{[l](C)}$ and $a^{[l](G)}$ are similar, both images have similar content

$$J_{content}(C, G) = \frac{1}{2} \| a^{[l](C)} - a^{[l](G)} \|^2$$

- $J_{content}$ shows how similar are content and generated images in terms of content.
- $\frac{1}{2}$: Normalization Constant
- $\| a^{[l](C)} - a^{[l](G)} \|^2$: element wise sum of squares of differences

Style Cost function :-

: Meaning of "style" of an image :-



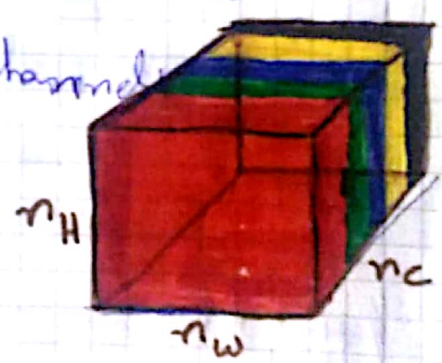
- Say you are using layer l's activation to measure "style"

What we are going to do is to:

→ Define style as Correlation b/w activation across channels in layer l.

Here is what I mean.

Suppose you take that layer's activations and we are gonna ask how correlated are the activations across different channels



To understand this phrase let's look at the channels of this block. For the sake of understanding let's say we have 5 block's layers

→ Take red channel and green channel and say how correlated are the activations in first 2 channels

What you do is look at values at different positions of these two channels and see how much correlation these two channels show.

Now, Why does this capture style?

Recall, we were trying to get an intuition about how ConvNets compute what they compute.

Let's say red channels corresponds to the neuron we are trying to figure out if there is a little vertical texture in the particular position.

and second (green) channel corresponds to the neuron trying to figure out orangish color.

Now, what does this mean for this channels to be highly correlated?

This means that, wherever part of the image will have vertical texture, it will also has orangish color.

and opposite is the case for them to be uncorrelated.

→ So, correlation tells you that, whichever of these high level texture components tend to occur or not occur together and degree of correlation tells you, how often these high level features occur together or not in different parts of the images.

If degree of correlation tells you the similarity then applying the same concept you can access how much style in generated image is correlated with the style of the style image at certain place of an image

Let's formalize intuition :-

Given image we compute **Style Matrix** which will measure all those correlation we have discussed.

Let $a^{[l]}_{i,j,k}$ = activation at (i, j, k). H, W, CH. $G^{[l]}$ is $n_c^{[l]} \times n_c^{[l]}$

$n_c \Rightarrow$ channels

So $G^{[l]}_{kk'} \Rightarrow$ Compares the correlation b/w activation of layer l b/w channel k and k'.

$k = 1, \ldots, n_c^{[l]}$

Un-normalized Cross Co-variance

$$G^{[l](S)}_{kk'} = \sum_{i}^{n_H^{[l]}} \sum_{j}^{n_W^{[l]}} a^{[l](S)}_{ijk} \, a^{[l](S)}_{ijk'}$$

$S \Rightarrow$ style

If activations $\Rightarrow$ both of them are large then $G^{[l]}_{kk'}$ will be large, if they are uncorrelated then it might be small.

$G \Rightarrow$ generated image

$$G^{[l](G)}_{kk'} = \sum_{i=1}^{n_H^{[l]}} \sum_{j=1}^{n_W^{[l]}} a^{[l](G)}_{i,j,k} \, a^{[l](G)}_{i,j,k}$$

Cost function

$$J^{[l]}_{style}(S, G) = \left\| G^{[l](S)} - G^{[l](G)} \right\|^2_F$$

$$J_{style}^{[l]}(S,G) = \frac{1}{(2 n_H^{[l]} n_W^{[l]} n_C^{[l]})^2} \sum_k \sum_{k'} \left(G_{kk'}^{[l](S)} - G_{kk'}^{[l](G)} \right)^2$$

→ Finally, it turns out that, you get more pleasing results if you use the style cost function from multiple different layers so overall style cost function can be defined as:

$$J_{style}(S,G) = \sum_l \lambda^{[l]} J_{style}^{[l]}(S,G)$$

So Overall Cost function:

$$J(G) = \alpha J_{content}(C,G) + \beta J_{style}(S,G)$$

Use gradient descent or other sophisticated algorithm on it to minimize the cost.

1D and 3D Generalizations:-

Just like we do convolutions in 2D images like

$(14 \times 14 \times 3) * [5 \times 5 \times 3] \rightarrow [10 \times 10 \times 16] * [5 \times 5 \times 16] \rightarrow [6 \times 6]$

16 filters

1D data

1	20	15	3	18	12	4	17

*

1	3	10	3	1

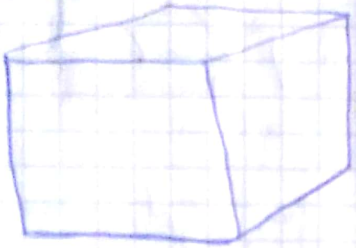
$14 \times 1 * [5 * 1] \rightarrow [10 \times 16] * [5 \times 16] \rightarrow [6 \times 32]$

6 filters

32 filters

Mostly for 1D data Recurrent NN are used.

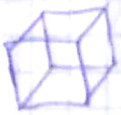
How about a 3-d data?



3D volume

14 × 14 × 14

One channel.



3D filter

5 × 5 × 5

→ 10 × 10 × 10

e.g. 3D data from CT scan

This filter detects feature from 3D data.