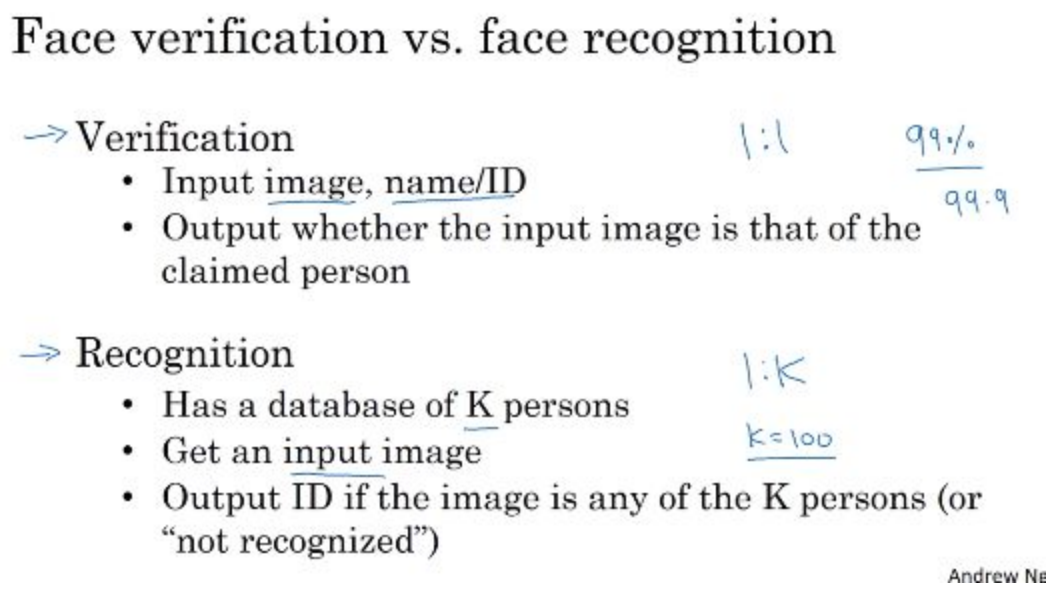
Andrew Ng's Convolutional Neural Networks

Week 4: Face Recognition & Neural Style Transfer

What is face recognition?

- We started by watching a video from Baidu where they use a face recognition system to grant you access to a building (instead of using a card id). Now we will start looking at some of the terminology used in face recognition.
- We often talk about face verification and face recognition. Face verification happens when we are given an input image and a name or id and our job is to output whether or not the image is that of the person. Face recognition is a more complex case: we have a database of K persons, we get an input image and then we output a positive id if the image belongs to any of the K persons in the database. Face verification is a 1:1 problem and face recognition is a 1:K problem

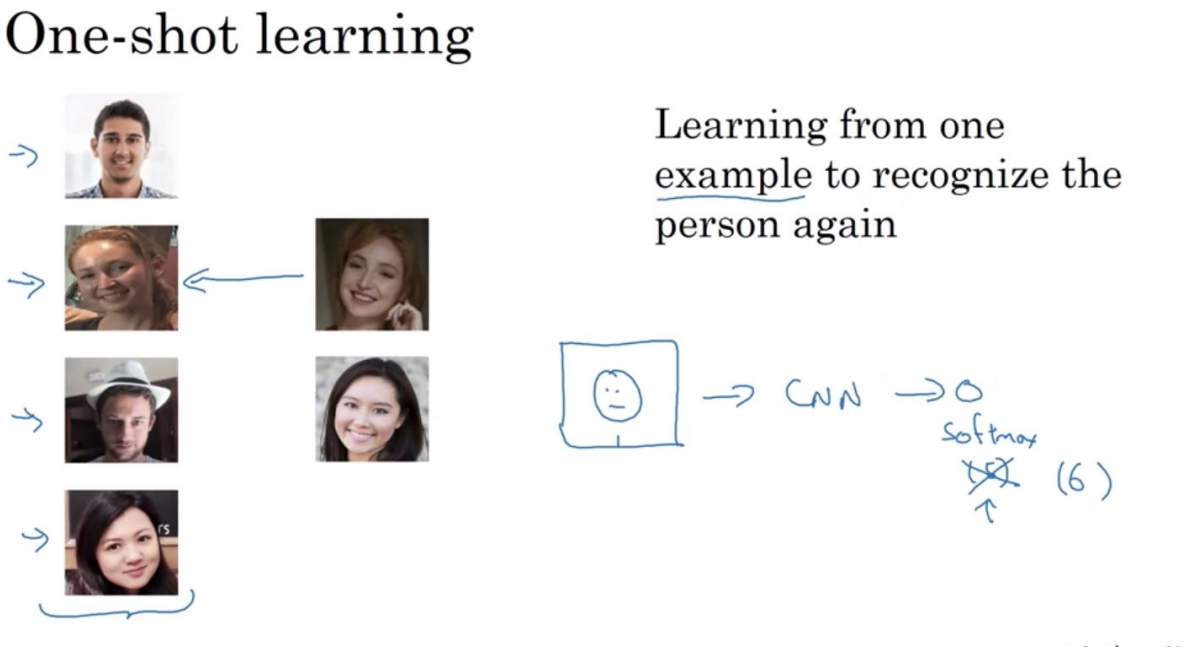


We'll first build a face verification system and, if the accuracy is high enough, we will use it as one of the building blocks in a face recognition system.

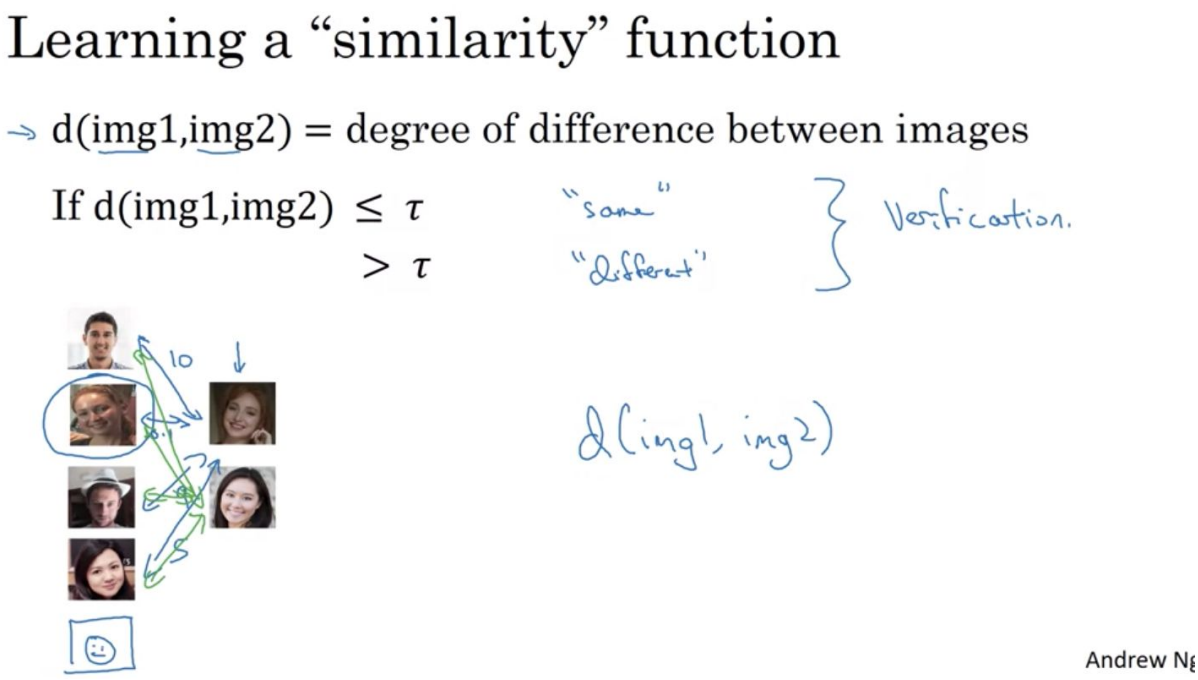
One Shot Learning

- One problem face recognition faces is solving the one-shot problem, which is learning from one example to recognize a person. This is a problem because we traditionally use a lot of data to train a neural network. An example of such a problem would be opening a turnstile by just looking at a picture of a person. If we recognize the person the turnstile opens; if not, it doesn't open. One approach to solve the problem would be to feed the

image to a convolutional neural network that has, in this case, a softmax activation with 5 outputs (1 for each employee and 1 for nobody). However, this is not a good approach because we have to retrain the CNN every time a new employee joins the company. So what do we do instead? We learn the similarity function.



- What we want to do is to have our CNN learn the similarity function which is defined as follows: $given\ d(img1,\ img2) = degree\ of\ difference\ between\ images.$

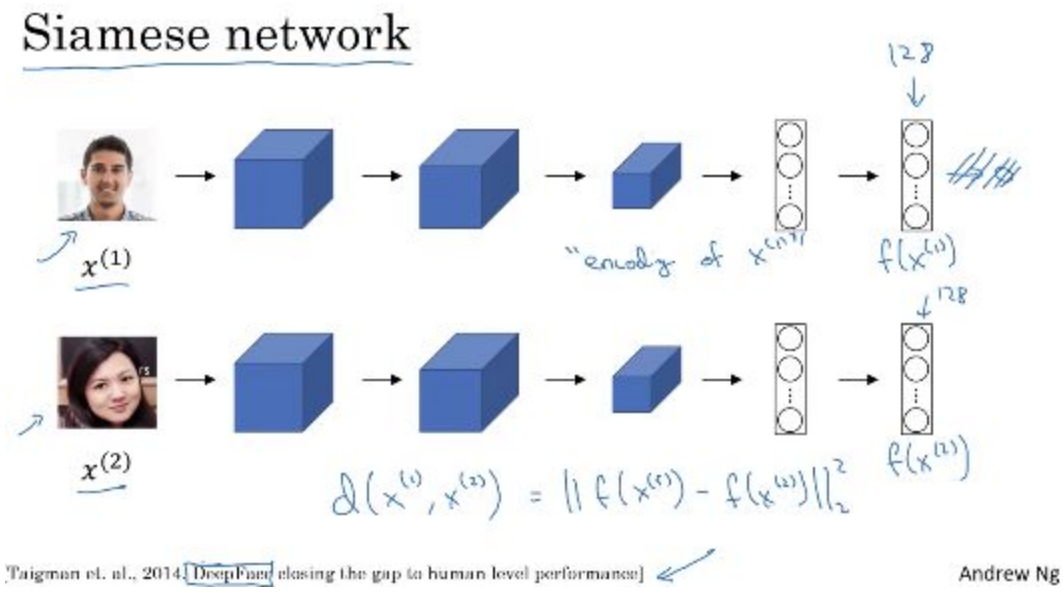


Andrew Ng

To apply this function to verification, we do the following: we compare the 2 images and we learn *d*, we then compare *d* to threshold Tao. $If\ d(img1,\ img2) \leq \tau$ then these 2 images are the same person . $If\ d(img1,\ img2) > \tau$ then these 2 images are not of the same person. To apply it to recognition, we do the same process but for every image we have . We then have a list of values and the lowest one, if there is one, is the one that tells us who this person is. If the person is not in our database, we hopefully have just a list of large *d* values and we know then she's not a person we trust. In the next session, we are going to learn how to train a neural network to learn the similarity function.

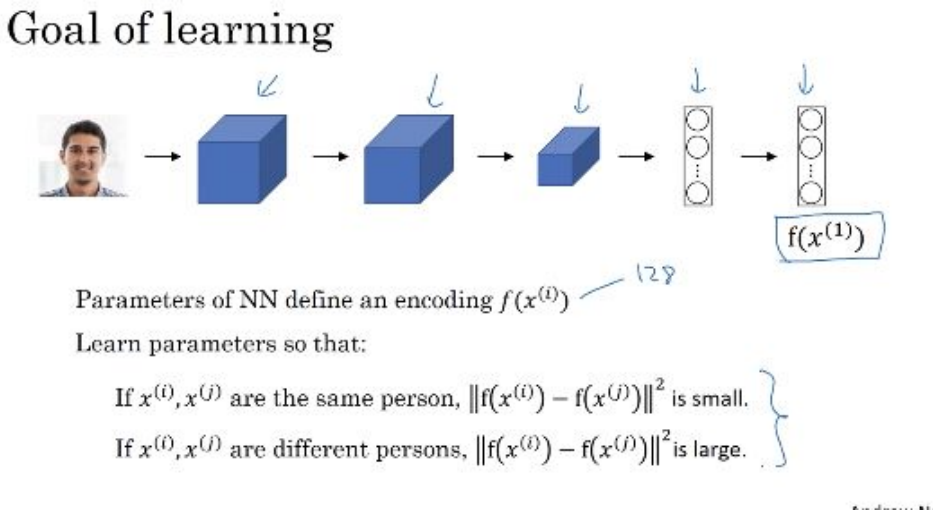
Siamese Network

- A Siamese Network is a good way to apply the similarity function we just learned. We are used to seeing convnets represented as below. What we do now is take an image, which we will call $x^{(1)}$ and input it into our conv net. At the end of the day we will have a feature vector, in this case 128-long, which we will reference as $f(x^{(1)})$. We do the same with another image, $x^{(2)}$, and the same conv net with the same parameters and we end up with another vector which we will call $f(x^{(2)})$. Given this, $d(x^{(1)},\ x^{(2)}) = || f(x^{(1)} - f(x^{(2)}) ||$. This idea, running 2 different images on the same conv net and comparing them is called a Siamese Network architecture.



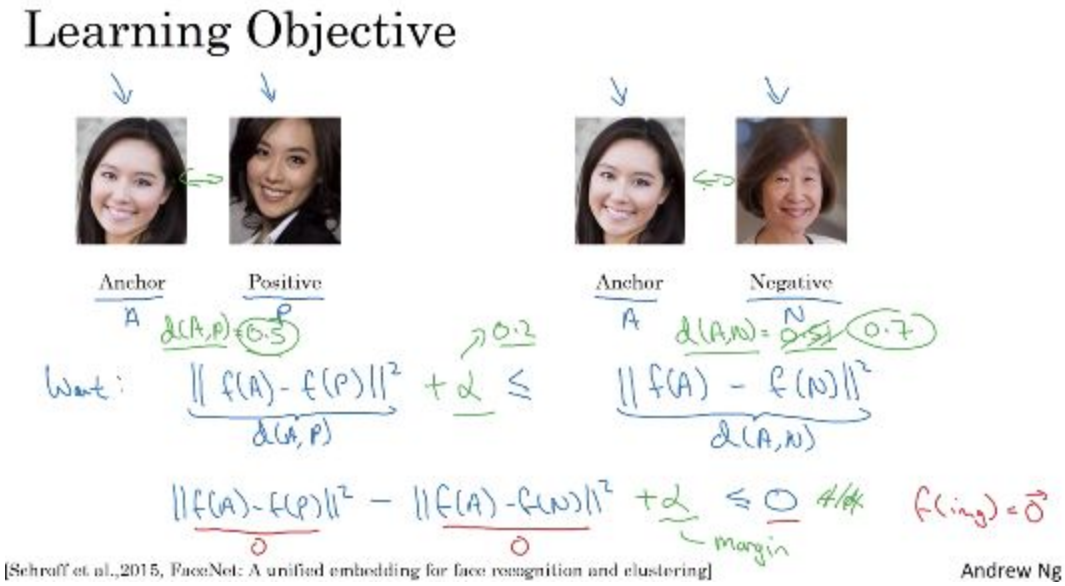
- The goal of learning is to train the neural network so that the encoding that it computes gives us a function that tells us if 2 pictures are of the same person. More formally, we are trying to learn parameters so that:
    - If $x^{(i)}$ and $x^{(j)}$ are of the same person, $|| f(x^{(i)} - f(x^{(j)}) ||^2$ is small.
    - If $x^{(i)}$ and $x^{(j)}$ are of different persons, $|| f(x^{(i)} - f(x^{(j)}) ||^2$ is large.

  As we vary the parameters in the conv net layers, we end up with different encodings and we can use backprop and vary the parameters to ensure these conditions are satisfied.



Triple Loss Function

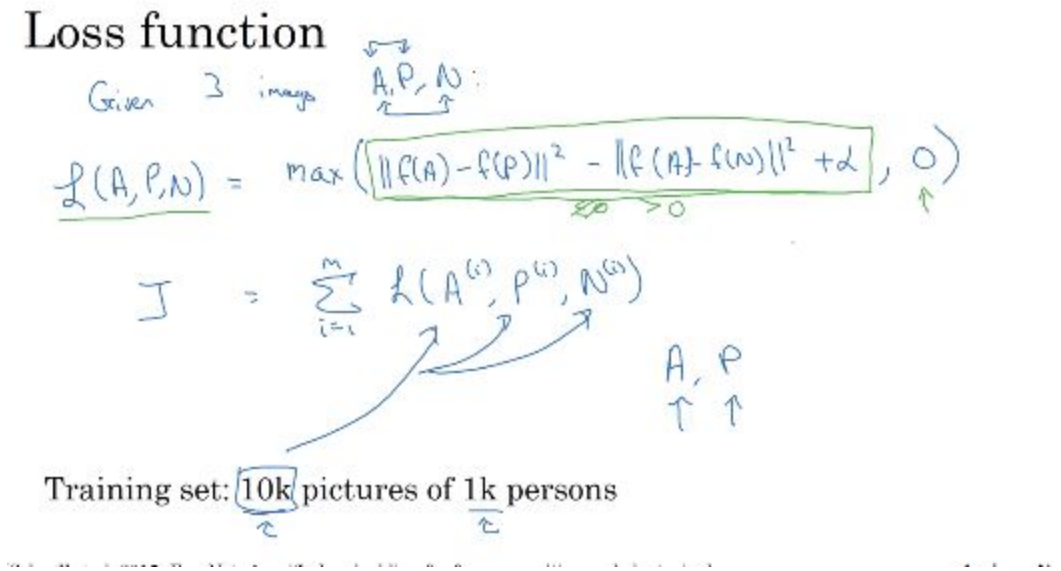
- How do we define an objective function to make sure the neural net learns to do what we discussed in our last video? We use the triplet loss function. To apply the triplet loss we need to compare pairs of images. We have an anchor A picture that we will compare against another 2 pictures: a positive P image (meaning the anchor and the positive image are pics of the same person) and a negative N image (meaning the anchor and the negative image are not pics of the same person). We call it the triplet loss function because are always comparing 3 pics (A, P & N). Furthermore, we can define $d(A,P) = \|f(A) - f(P)\|^2$ and $d(A,N) = \|f(A) - f(N)\|^2$. We also know we want $d(A,P) \leq d(A,N)$. We can rewrite the equation as $\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 \leq 0$.



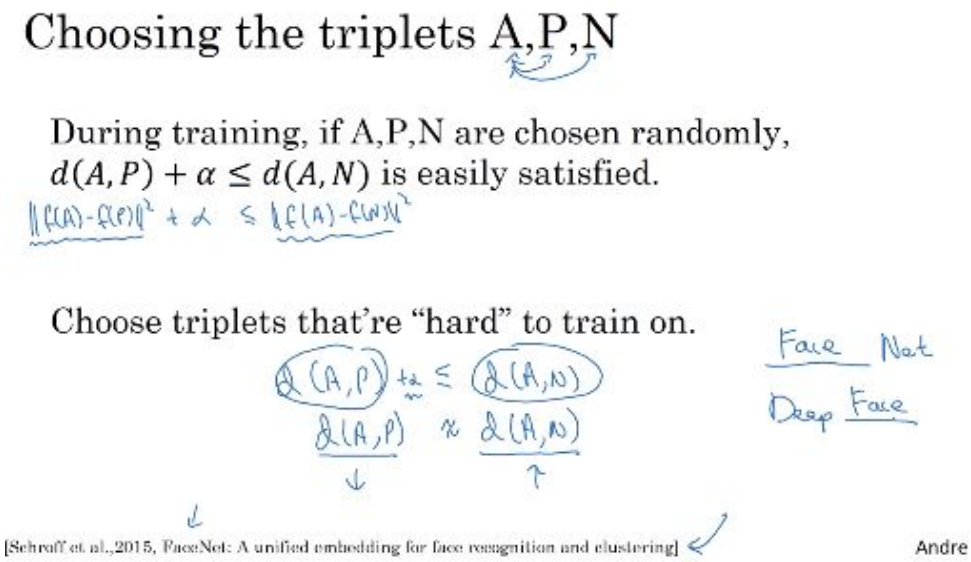
The problem with the equation above is that it can be satisfied if all the numbers are zeros or identical (say, all 2's). To prevent our neural network to do this we add a margin

α so our equation is then $\|f(A)-f(P)\|^2 - \|f(A)-f(N)\|^2 + \alpha \leq 0$. What α does is to push the 2 parts of the equation apart from each other and that's what we want. It either makes us make $\|f(A)-f(N)\|^2$ larger or make $\|f(A)-f(P)\|^2$ smaller.

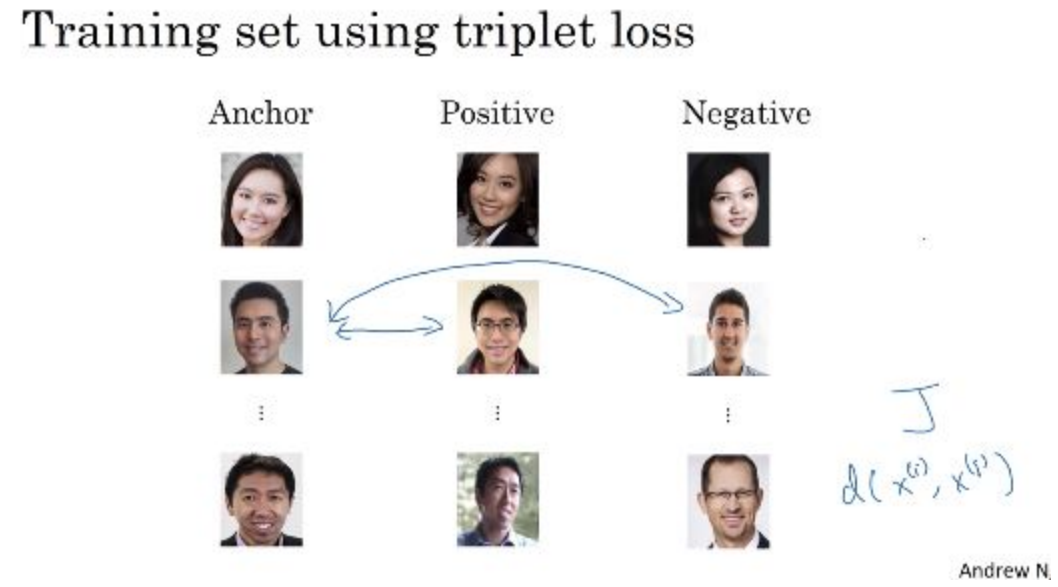
- Now we formally define the triplet loss function. Given 3 images A, P & N, the loss $L(A,P,N) = max\|f(A)-f(P)\|^2 - \|f(A)-f(N)\|^2 + \alpha, 0)$. What this does is that we are trying to make the loss function 0. If the equation on the left is <= 0 then we pick the zero and the loss function = 0. If the equation >= 0 we then have a positive loss function. The cost function is $J = \sum_{i=1}^{m} L(A^{(i)}, P^{(i)}, N^{(i)})$. Notice that we need a training set that has multiple pictures of the same person. If we have a training set of 10K pictures of 1K persons we form triplets using the training set to train the learning algorithm using gradient descent to minimize the cost function .



- How do we select images from the training set to form these triplets? We don't want to choose A, P & N randomly because then our condition below would be easily satisfied. Instead, we want triplets that satisfy the condition $d(A,P) + \alpha \leq d(A,N)$. In particular we want triplets that are hard to train; triplets where $d(A,P)$ and $d(A,N)$ are very similar because then our learning algorithm has work to do: gradient descent has to work to either make $d(A,P)$ smaller or $d(A,N)$ bigger.



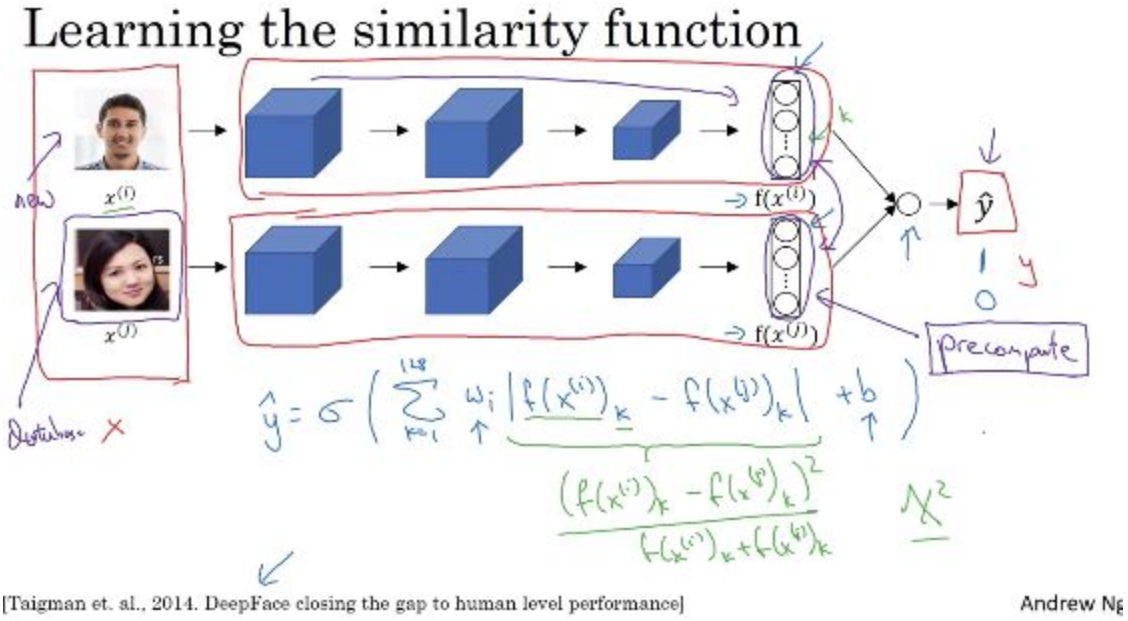
- Summarizing, what we do is map our training set to triplets as shown below and then apply gradient descent to the cost function *J* to minimize it. That will have the effect of propagating all the parameters of the neural network so it can learn an encoding so that the distance between 2 images of the same person is small and the distance between 2 images of different persons is big.



Face Verification and Binary Classification

- An alternative to the triplet loss function is to treat face verification as binary classification. In this case, we have the siamese network below that results in 2 encodings, perhaps 128-long, and have those encodings be the input to a logistic regression unit that makes a prediction (1 if the 2 images are of the same person, 0

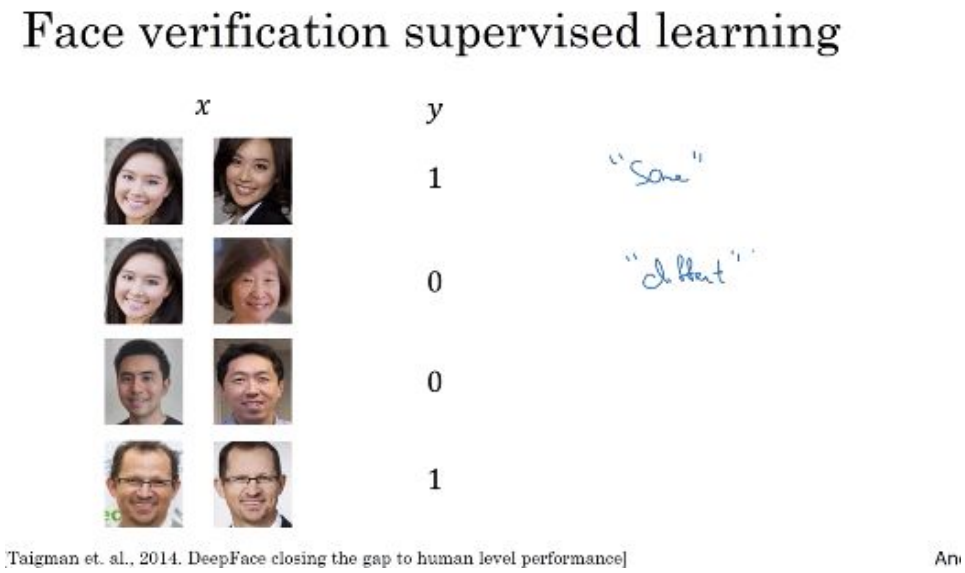
otherwise). What does the logistic regression unit do?
$yhat = \sigma(\sum_{k=1}^{128} w_i \, |f(x^{(i)})_k - f(x^{(j)})_k| \, + b)$ where $f(x^{(i)})_k - f(x^{(j)})_k$ is the element-wise difference of the 2 encodings and that difference is the feature of the logistic regression equation. We then add w and b to complete the normal logistic regression equation. Another formula to use would be chi square where $\chi^2 = (f(x^{(i)})_k - f(x^{(j)})_k)^2 / f(x^{(i)})_k + f(x^{(j)})_k$



[Taigman et. al., 2014. DeepFace closing the gap to human level performance] Andrew Ng

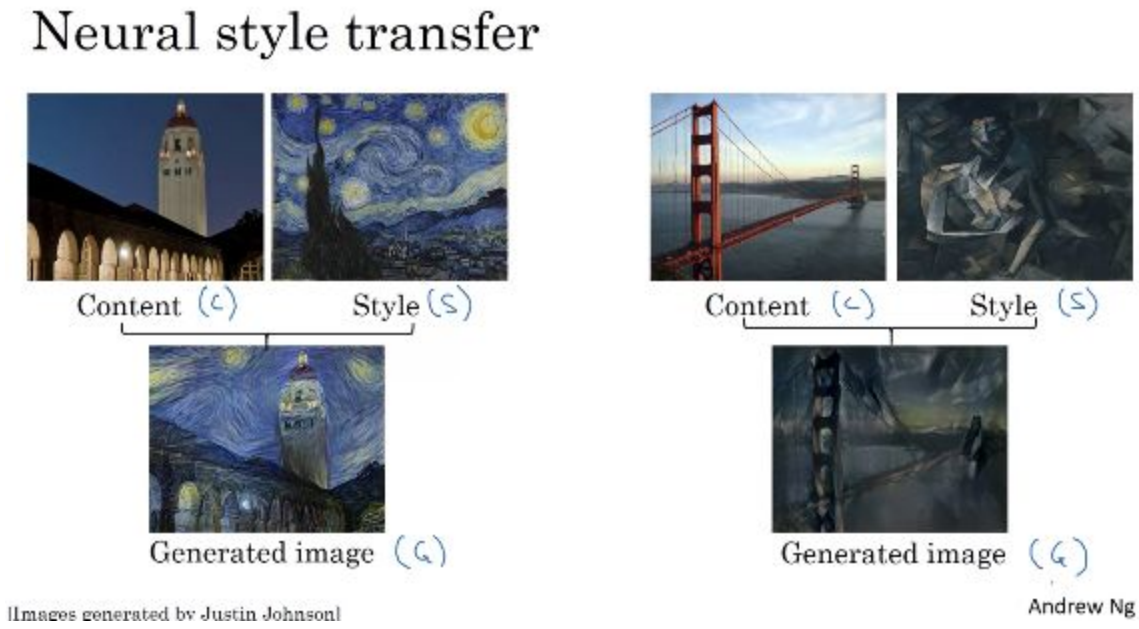
Lastly, we have a computational trick: if we have the image on top be the employee that needs to be verified what we can do is precompute the encoding of the image below and we just compute the encoding of the image of the employee that just walked in the door. This will help us to save significant computation time.

- To summarize: to treat face verification as supervised learning what we do is create a training set of image pairs where the input is either 1 (same person) or 0 (different person) and we use different pairs to train the neural network using back propagation.



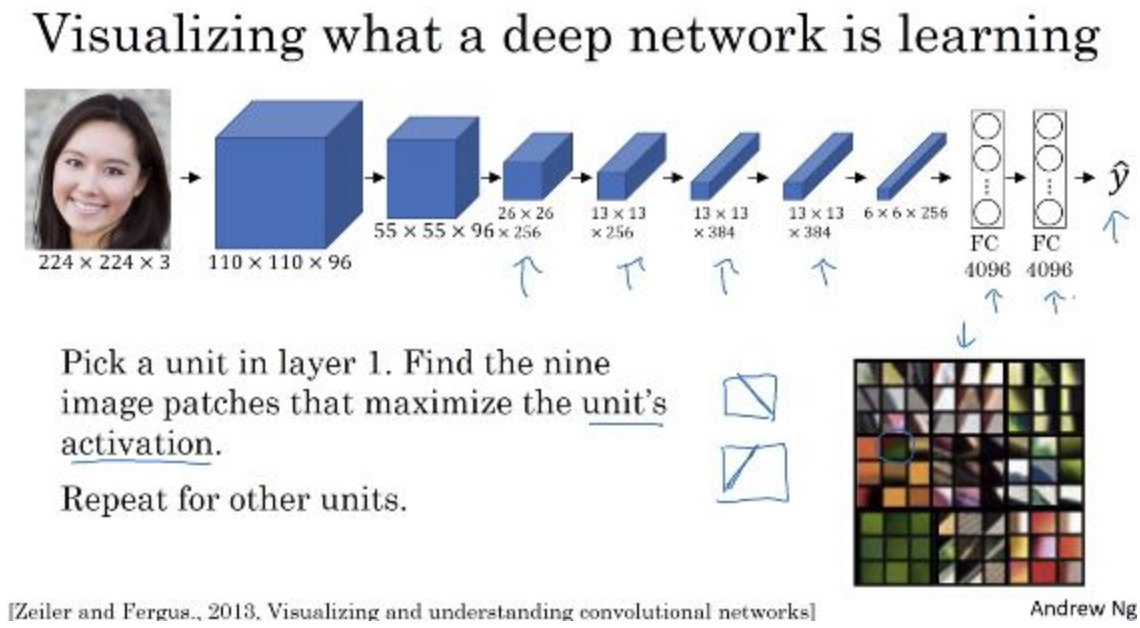
What is Neural Style Transfer?

- We can learn what is neural style transfer with a few examples. We take an image C and a style S and applying S to C we end up with a generated image G as the ones shown below. What we will learn in the next few videos is how we can generate these images ourselves.

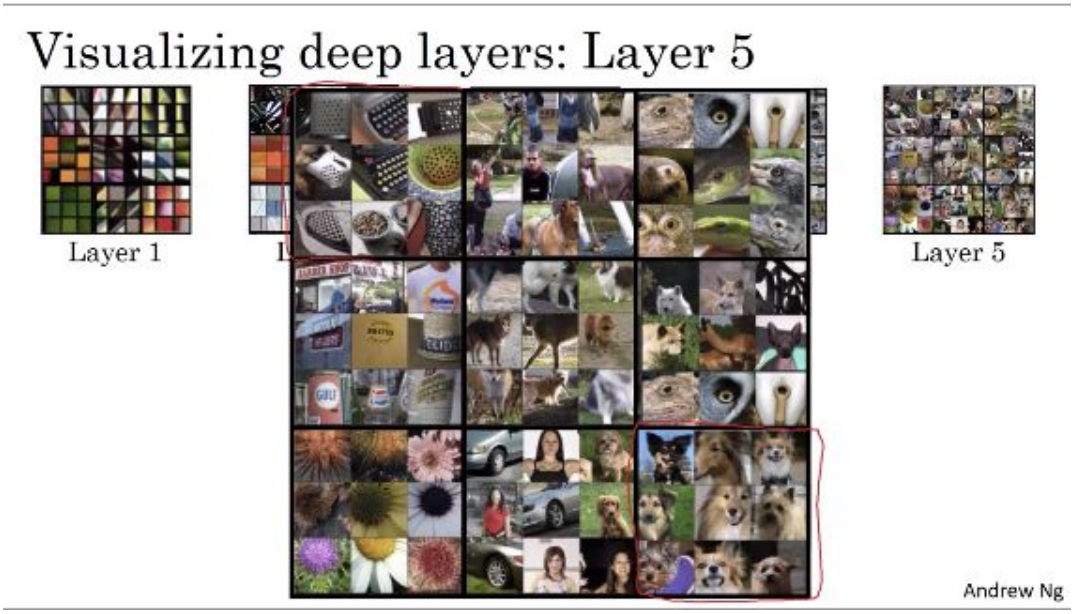


What are Deep ConvNets Learning?

- What are the deep layers of a ConvNet learning? Let's start with understanding what an 'early' layer's unit is learning. In the convnet below we pick a unit from layer 1 and we parse our training set to find the nine images that maximize that unit's activation. In this 'early' layer the unit is looking at a small image patch. The first unit seems to care about edges to the right. We then repeat the same process with other units and we end up with something like the image patches on the bottom right. Each unit cares about different things; the second one, for example, seems to care about bright/white-ish edges to the left.

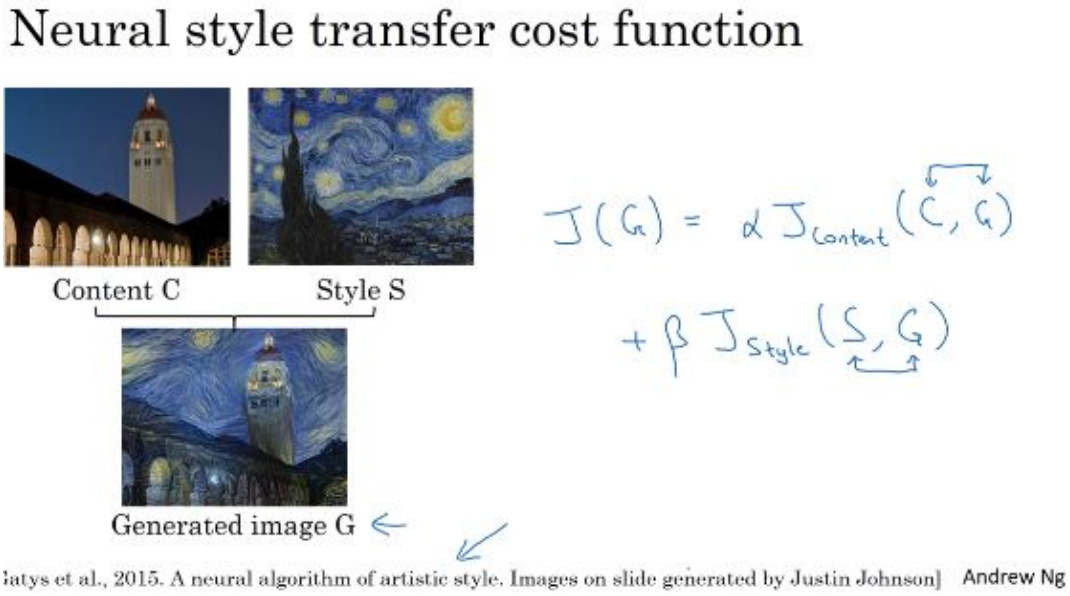


- What about deeper layers in the network? Deeper layers detect more complex patterns. For example, layer 5's activation units seem to be specialized in the following: bottom right corner seems to be a dog detector, the 4th unit (first on the 2nd row) seems to be detecting text and the 7th unit (bottom left corner) seems to be detecting flowers. The intuition is that deeper layers detect more complex patterns than shallow layers of the network.



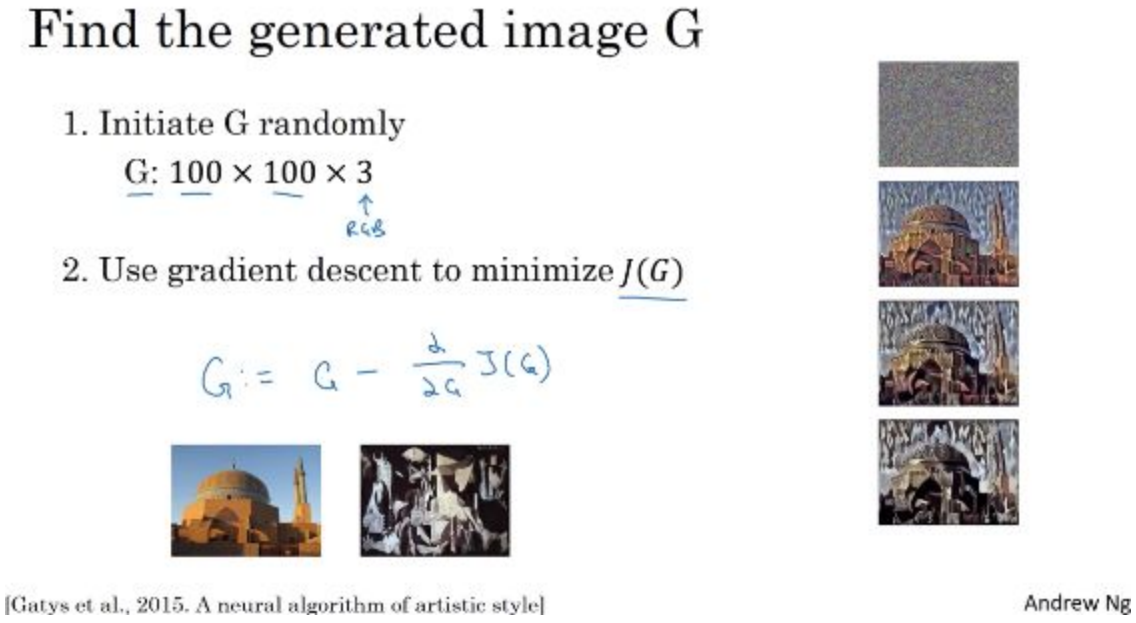
Cost Function

- Now we define a cost function for the generated image of a neural style transfer system. The cost function $J(G) = \alpha * J_{content}(C, G) + \beta * J_{style}(S, G)$. We will use gradient descent to minimize this cost function.



- The way the algorithm would run is as follows: you will initiate G randomly, say G: 100x100x3 (a RGB image) and we get an image perhaps like the one in the top right corner below. We then use gradient descent to minimize the cost function J(G) that we just defined so $G := G - dJ(G)/dG$ and we slowly get images like the ones below until

we finally get an image that looks like the combination of the style of one and content of the other.



Content Cost Function

- Our neural style transfer algorithm cost function has a content cost function and a style cost function. Let's define both starting with the content component. Say we use a hidden layer l to to compute the content cost (in practice, this is a middle layer, not too shallow, not too deep). We then use a pre-trained ConvNet (perhaps a VGG network). We then let $a^{[l](C)}$ *and* $a^{[l](G)}$ be the activation of layer l for the images. If $a^{[l](C)}$ *and* $a^{[l](G)}$ are similar, both images have similar content. We then define $J_{content}(C, G) = 1/2 * ||a^{[l](C)} - a^{[l](G)}||^2$. So this cost function is just the element-wise sum of the squares of the differences between the 2 activations.

Content cost function

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

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

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

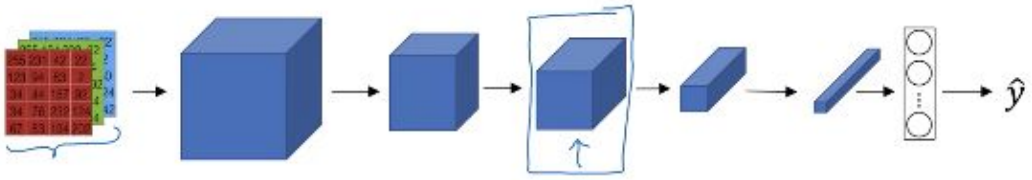
Gatys et al., 2015. A neural algorithm of artistic style]

Andrew Ng

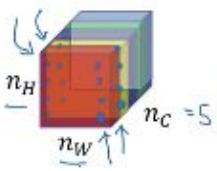
Style Cost Function

- Last time we defined the content component of the cost function. This time let's take a look at the style component of the same function. First let's agree on what is the meaning of the 'style' of an image. We start by selecting a layer l from the convnet and that layer's activation will be used to measure 'style'. We define style as the correlation between activations across channels.

Meaning of the "style" of an image



Say you are using layer l's activation to measure "style."
Define style as correlation between activations across channels.



How correlated are the activations across different channels?

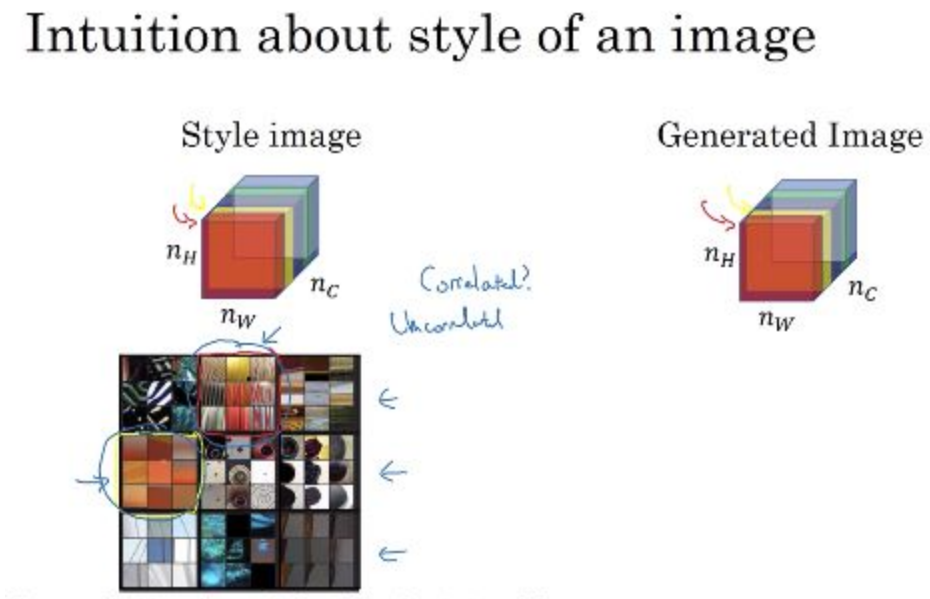
Gatys et al., 2015. A neural algorithm of artistic style]

Andrew Ng

What do we mean by correlation? Let's look at the image on the bottom left above. This activation unit has 5 channels but for now let's focus on the first 2, the red and yellow. What we do is take say, the activations on the lower right corner for both channels and

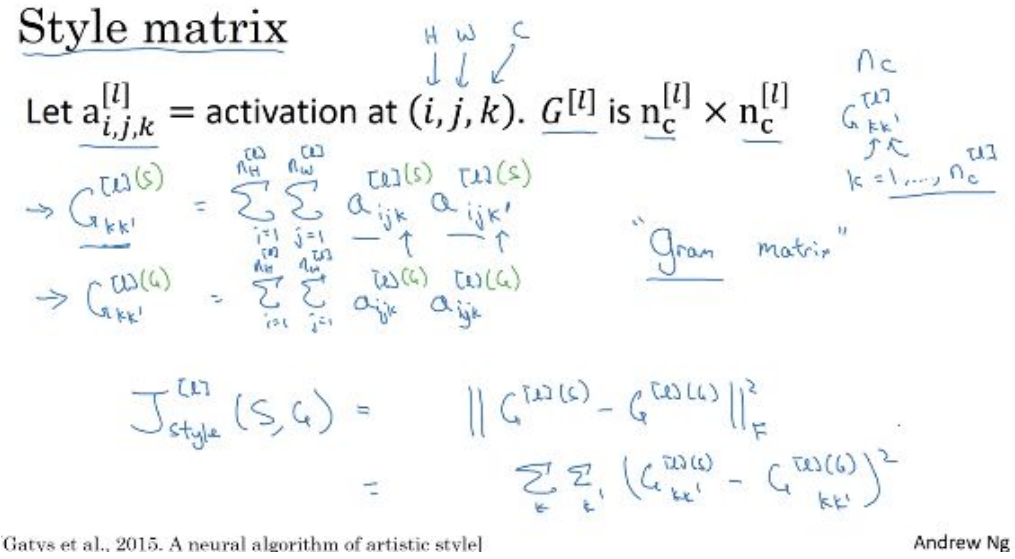
now we have 2 numbers. We do the same for the other activations of the same channels and see how correlated the number pairs are.

- Why does this correlation capture style? Let's look at an example below. Assume the red channel in the style image corresponds to the 2nd neuron (vertical texture) and the yellow channel (orange tint) correspond to the 4th neuron below. We can say they are correlated if whenever there is a vertical texture it has an orange tint. Conversely, they are uncorrelated if the orange tint does not appear when the vertical texture is present. This degree of correlation gives us a measure of style and it is what we will use for the generated image. This correlation tells us a measure of how similar the style of the generated image is to the style of the original image.

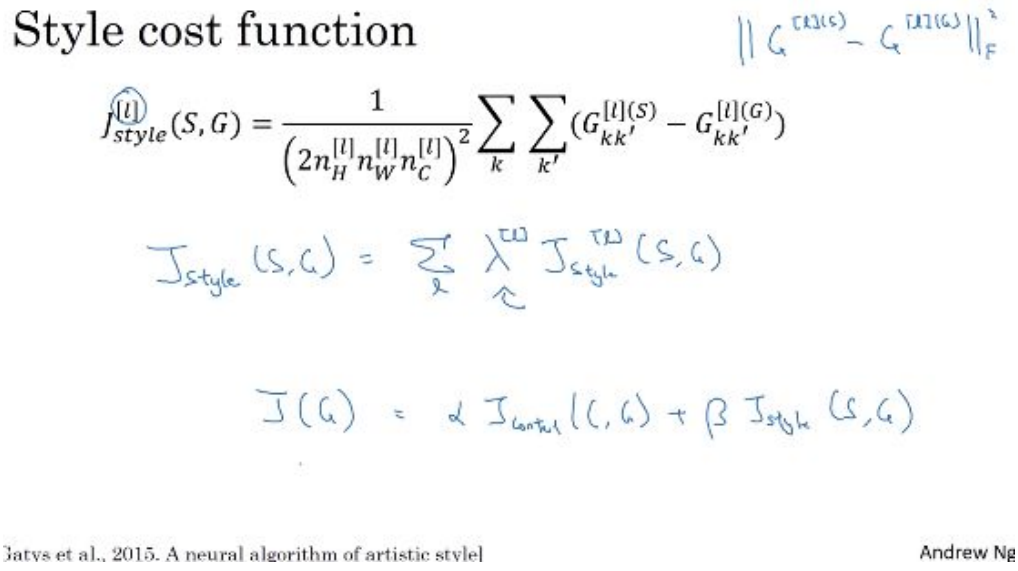


[Gatys et al., 2015. A neural algorithm of artistic style]

- Let's formalize the intuition above. We define a style matrix as follows: first we let $a^{[l]}_{i,j,k} = activation\ at\ (i,\ j,\ k).\ G^{[l]}\ is\ of\ dimensions\ n_c^{[l]}\ x\ n_c^{[l]}$. $G^{[l]}_{kk'}$ will measure how correlated are the activations in channel k with the activations in channel k prime. We can them define $G^{[l](S)}_{kk'} = \sum_{i=1}^{n_h^{[l]}} \sum_{j=1}^{n_w^{[l]}} a^{[l](S]}_{ijk} * a^{[l](S]}_{ijk'}$. Similarly we define $G^{[l](G)}_{kk'} = \sum_{i=1}^{n_h^{[l]}} \sum_{j=1}^{n_w^{[l]}} a^{[l](G]}_{ijk} * a^{[l](G]}_{ijk'}$ .These two matrices capture the style of the image S and the style of the image G . The style cost function is defined as $J^{[l]}_{style}(S, G) = || G^{[l](S)} - G^{[l](G)}||^2$ which is just $\sum_k \sum_{k'} (G^{[l](S)}_{kk'} - G^{[l](G)}_{kk'})^2$.



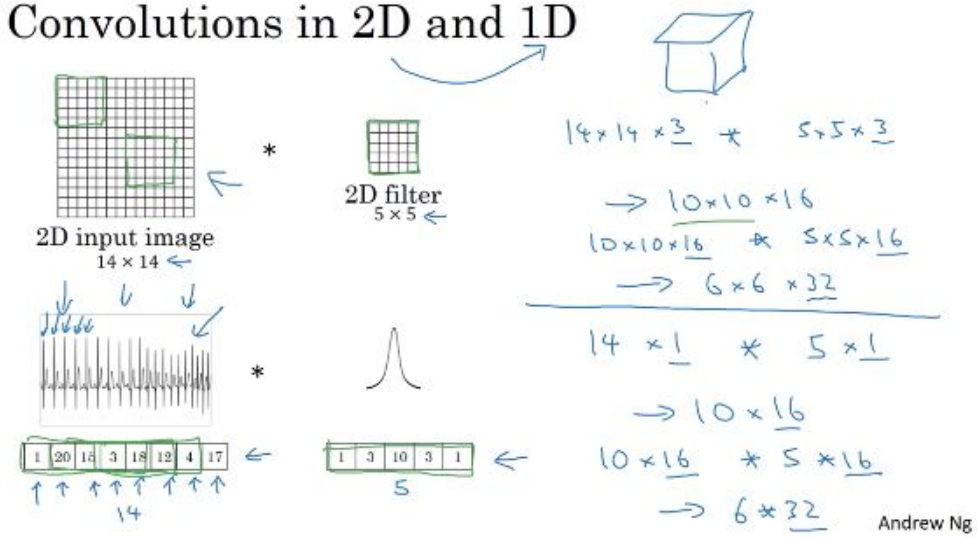
- To finish off, below is the style cost function for layer l. It turns out that we get better results if we use the style function from multiple layers. Therefore, the overall style cost function is the sum of over all the different layers of the style function for that layer; this is written as $\sum_l \lambda^{[l]} * J^{[l]}_{style}(S,\ G)$. With this we are now ready to define the overall cost function as $J(G) = \alpha * J_{content}(C,\ G) + \beta * J_{style}(S,\ G)$.



1D and 3d Generalizations

- Many of the ideas we have learned also apply to 1D and 3D cases. Let's see how they apply to a 1D case. Below is a 2D example where we have a 14x14 input image and we apply a 5x5 filter and end up with 10x10x16 volume. We also have a 1D example: the ekg

data from someone's heart. We can convolve a 5-long filter and we get an output of 10. If we have 16 filters we then end up with a 10x16 volume. We can also use recurrent neural networks for 1D problems but some people also use conv nets.



- Now let's take a look at a 3D example. Some examples of 3D data would be cat scans or movie scenes. In this case we have a 3D input volume, say 14x14x14. We can convolve it with a 3D filter of dimensions 5x5x5 and we end up with a 10x10x10 output. Could be 10x10x10x1 if we have only 1 channel or it could be 10x10x10x16 if we have 16 channels. We can then apply a 5x5x5x16 filter to our previous output and we end up with a 6x6x632 output.