face recognition

-If you apply a verification system that's 99 percent accurate to a recognition task with a 100 people in your database, you now have a hundred times of chance of making a mistake and if the chance of making mistakes on each person is just one percent

-So, if you have a database of a 100 persons, and if you want an acceptable recognition error, you might actually need a verification system with maybe 99.9 or even higher accuracy

one shot learning

:for most face recognition applications you need to be able to recognize a person given just one single image, or given just one example of that person's face.(problem)

- In the one shot learning problem, you have to learn from just one example to recognize the person again.
- one approach you could try is to input the image of the person, feed it too a ConvNet. And have it output a label, y, using a softmax unit with four outputs or maybe five outputs corresponding to each of these four persons or none of the above.(to make this work, what you're going to do instead is learn a similarity function)
- similarity function: d(img1,img2):degree of difference between images

  :if degree is below criteria->"true"

siamese network

1. Focus on a vector of of 128 numbers computed by some fully connected layer that is deeper in the network.
2. feed a second picture to the same neural network with the same parameters and get a different vector of 128 numbers, which encodes this second picture.
3. Then compute distance between image 1,2 output
4. As you vary the parameters in all of the layers of the neural network, you end up with different encodings. And what you can do is use back propagation to vary all those parameters in order to make sure the conditions are satisfied.

Triplet loss

One way to learn the parameters of the neural network, so that it gives you a good encoding

for your pictures of faces, is to define and apply gradient descent on the triplet loss function.

1. compute several pair of images(same or different people)
2. what you're going to do is always look at one anchor image, and then you want to distance between the anchor and a positive image,whereas you want the anchor when pairs are compared to have a negative example for their distances to be much further apart. (always be looking at three images at a time: anchor/positive/negative)
3. compute differnece

- difference<alpha=>margin
- loss function: max(d(positive)-d(negative)+alpha,0)=> so long as this zero or less than equal to zero, the neural network doesn't care how much further negative it is.
- For the training set, you do need the dataset of multiple pictures of the same person
- Choose triplets that're hard to trian on than choosing randomly: d(positive) is close to d(negative)->gradient descent will need to work harder

Neural Style Transfer

generate new image drawn in the style of the style image

content image+style image: content painted in the style of style image

cost function: J(G)=alpha*Jcontent(C,G)+beta*Jstyle(S,G)

-initiate G(generated image) initially

-use gradient descent to minimize J(G)

Deep ConvNets

1. pick a unit in layer 1 and find the 9image patches that maximize the unit's activation=>in other words pause your training set through your neural network, and figure out what is the image that maximizes that particular unit's activation.
2. repeat this in other hidden units
3. apply this on further layers(complex objects are likely to be detected in deeper layers)

Content cost function

:how similar C&G are/ how different the two activations are

style cost function

-style of an image: we need to s define the style as the correlation between activations across different channels in this layer L activation.

-layer l's activation: measures style=>define style as correlation between activations across channels

-"to be highly correlated": have the same texture or color tint

-style matrix: contains all info of correlation