Face Recognition

Dienstag, 21. Juli 2020 17:11

face verification vs. face recognition

1:1 1:k

One shot Learning - learning from one example to recognize again

-- not actually using CNN to give prediction
-- learn the similiarity -> degree of difference between img1 and img2

use only one image -> CNN -> softmax according to each person --> won't perfrom well because dataset is too small; furthermore, if you have new image in order to predict the new image the whole CNN have to be trained again.

$d(\text{img}1, \text{img}2)$ = degree of difference

if $d(\text{img}1, \text{img}2) \le \tau$ same ✓
$> \tau$ different

} verification

Siamese network

$f(x^{(1)}) \rightarrow$ encoding of $x^{(1)}$

$x^{(1)} \rightarrow \text{CNN} \rightarrow$ 128

$x^{(2)} \rightarrow \text{CNN} \rightarrow$ $f(x_2)$ $\rightarrow$ encoding of $x^{(2)}$

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

$\Rightarrow$ parameters of NN define an encoding of $f(x^{(1)})$

Learn parameters to:

if $x^{(i)}, x^{(j)}$ are same person $\Rightarrow \| f(x^{(i)}) - f(x^{(j)}) \|_2^2$ is small

else $\Rightarrow$ large

Loss for siamese network -- triple loss

A: anchor (example) P: positive N: negative

want: $\| f(A) - f(P) \|_2^2 \le \| f(A) - f(N) \|_2^2$

$\Rightarrow d(A, P) \le d(A, N)$

$\Rightarrow \| f(A) - f(P) \|_2^2 - \| f(A) - f(N) \|_2^2 \le 0 - \alpha$

margin

$\Rightarrow$ — — ——— ——— $+ \alpha \le 0$

given 3 images A P N

$\alpha \leq 0$

given 3 images A P N

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

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

training 10 k images of 1 k persons

After training can be applied to one short problem

How to choose A P N

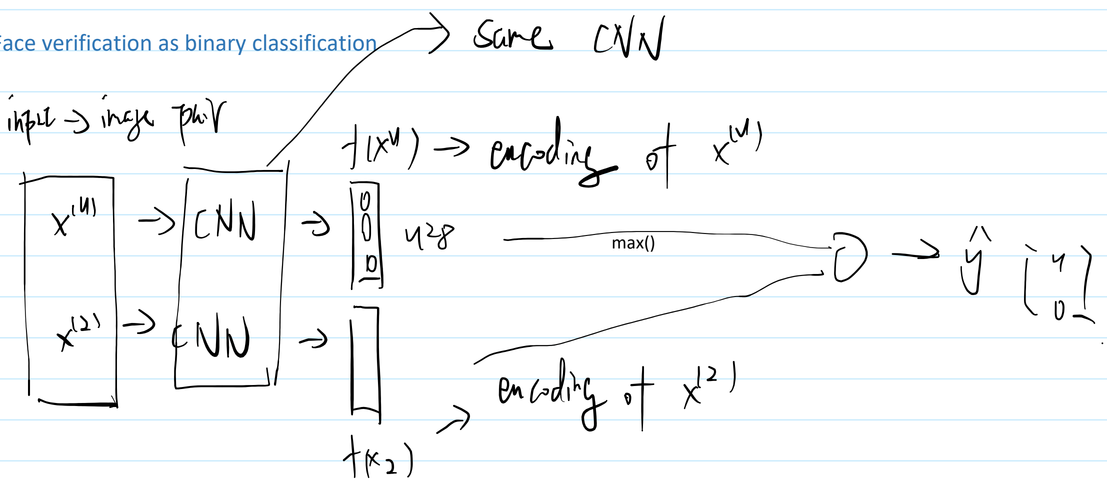
if choosed randomly, it is easy to satisfy $d(A,P) + \alpha \leq d(A,N)$ because A N distance large

-> learning algorithms will not learn much, will not perform well on one short problem

Choose triplets hard to train on $\Rightarrow$ $d(A,P) \approx d(A,N)$

Face verification as binary classification

input -> image pair



$$\hat{y} = \sigma\left(\sum_{k=1}^{128} w_i \left| f(x^{(i)})_k - f(x^{(j)})_k \right| + b\right) \Rightarrow \text{learn } w_i, b$$

k-th element

or $\dfrac{(f(x^{(i)})_k - f(x^{(j)})_k)^2}{f(x^{(i)})_k + f(x^{(j)})_k}$ -> χ^2 similarity

Training also uses image pairs and labels (1 or 0).

Neural Style Transfer

Content C + Style S -> new image G

how close are these two images' content

how close are these two images' style

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

Content C + Style S -> new image G

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

1. initiate G randomly
2. Use gradient descent to minimize J(G)

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

Content cost

use hidden layer l to compute content cost (l is usually mid-layer)
use pre trained ConvNet (e.g. VGG)

$a^{[L](C)}$ -> activation of layer on content image
$a^{[L](G)}$ -> - - - - - generated image
if $a^{[L](C)}$ $a^{[L](G)}$ similar => similar content on images

$J_{content}(C, G) = \frac{1}{2} || a^{[L](C)} - a^{[L](G)} ||^2$ => elementwise sum of square difference

Face verification and face recognition

Siamese network

Neural Style Transfer
Cost function of NST
Content Cost Function
Style Cost Function

Take away from programming assignment:

Neural Style Transfer:

1. NST uses a previously trained convolutional network, and builds on top of that. The pre-trained NN can be different, such as VGG in original paper.
2. The first step is usually load pre-trained model. The model is stored in python dictionary, the key-value represents each layer.
3. feed a image as input to the model by using: model["input"].assign(image)
4. access a specific layer when the network is running on image: sess.run(model["conv4_2"])
5. The shallower layers of a ConvNet tend to detect lower-level features such as edges and simple textures. The deeper layers tend to detect higher-level features such as more complex textures as well as object classes.
6. Make generated image G match the content of image C:
   a. Choose a "middle" activation layer a[l]
   b. Forward propagate image "C"
   c. Forward propagate image "G"
   d. Compute the "content cost"
   e. $$J_{content}(C,G) = \frac{1}{4 \times n_H \times n_W \times n_C} \sum_{\text{all entries}} (a^{(C)} - a^{(G)})^2$$
7. G(gram)i,j: correlation
   1) The result is a matrix of dimension (nC,nC) where nC is the number of filters (channels). The value G(gram)i,j measures how similar the activations of filter i are to the activations of filter j.
8. Compute the style cost on one layer
9. Combine style cost with different layers with weights:

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

10. compute style cost on each layer with weights
11. Total cost function

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

After all these, in total:

1. Create an Interactive Session
2. Load the content image (load and reshape)
3. Load the style image (load and reshape)
4. Randomly initialize the image to be generated (slightely correlated to content image, which helps to mach content image quickly)
5. Load the VGG19 model
6. Build the TensorFlow graph:
    1) Run the content image through the VGG19 model and compute the content cost:
        - Assign the content image to be the input to the VGG model.
        - Set a_C to be the tensor giving the hidden layer activation for specific layer.
        - Set a_G to be the tensor giving the hidden layer activation for the same layer. -- (assigned but not evaluated)
        - Compute the content cost using a_C and a_G,

    2) Run the style image through the VGG19 model and compute the style cost
    3) Compute the total cost, using initialized alpha and beta
    4) Define the optimizer, the learning rate, training step (optimizer.minimize(J))
    5) Initialize the TensorFlow graph, assign generated image to the model and run the train step tensor for a large number of iterations, updating the generated image at every step.

Face Detection:

1. face verification: learn a neural network that encodes a face image into a vector of 128 numbers. By comparing two such vectors, you can then determine if two pictures are of the same person. ---- "is this the claimed person?", this is a 1:1 matching problem
2. face recognition: "who is this person?" ---- this is a 1:K matching problem
3. Walk through face verification:
    1) Encoding face image into 128-D-vector
        i. Use pre-trained ConvNet (weights): input batch of m faces images (m, n_c, n_h, n_w); output encoder (m, 128)
        ii. Evaluation criteria, if an encoding is a good one then:
            1) The encodings of images of same person are similar <= threshold
            2) The encodings of images of different persons are different
        iii. Using triplet loss function to define the criteria:

$$\mathcal{J} = \sum_{i=1}^{m} \big[\underbrace{\mid\mid f(A^{(i)}) - f(P^{(i)}) \mid\mid_2^2}_{(1)} - \underbrace{\mid\mid f(A^{(i)}) - f(N^{(i)}) \mid\mid_2^2}_{(2)} + \alpha \big]_+$$

$$\|x - y\|_2^2 = \sum_{i=1}^{N} (x_i - y_i)^2$$

    2) train model to minimize the triplet loss
    3) run forward propagation to encode each image (face) and store the f(x_i) encoding in a database: key(name/id), value(128 encoding)
    4) input new image (image to be verified) in to model to run the forward propagation to encode.
    5) compute the distance between the encoding of new input image and the corresponding encoding in database

1. Walk through face recognition (takes an image as input, check if it is one of the authorized persons; and if so, who):
    1) based on previous face_verification, compute the input encoding
    2) find encoding from database that has smallest distance
        i. Initialize a temp variable with a large enough number to update, used to keep track of what is the closest encoding to the input's encoding.
        ii. Loop over the database dictionary's names and encodings.
            a) Compute the L2 distance between the target "encoding" and the current "encoding" from the database.
            b) If this distance is less than the temp, then update