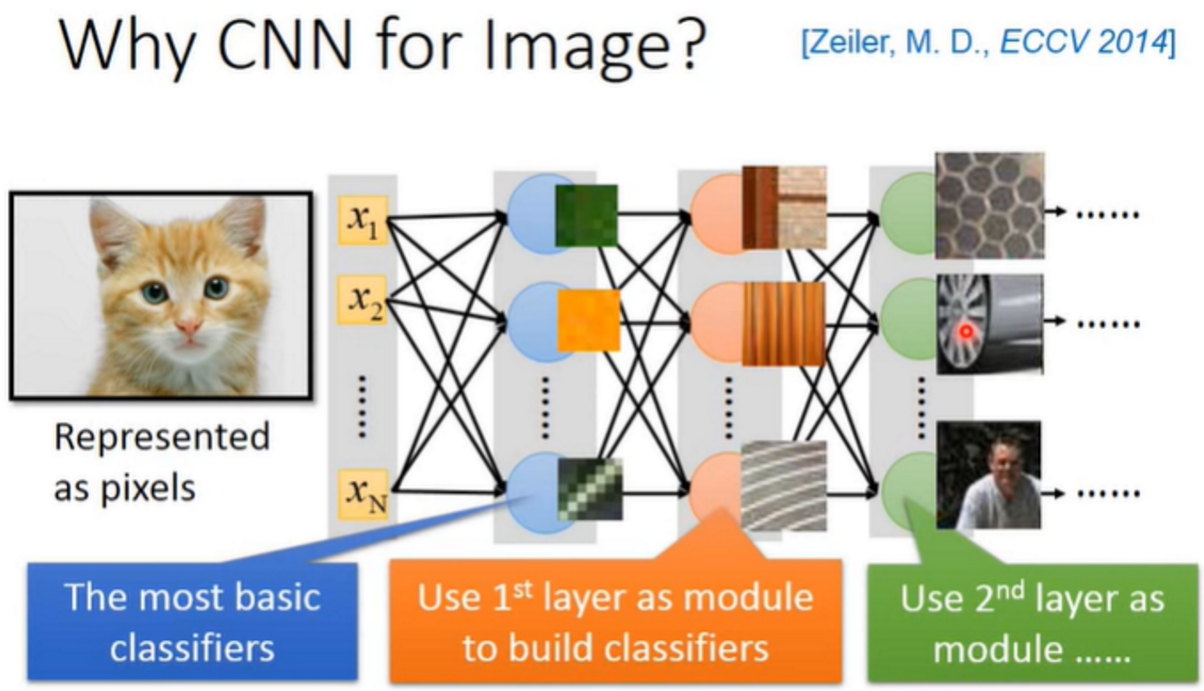
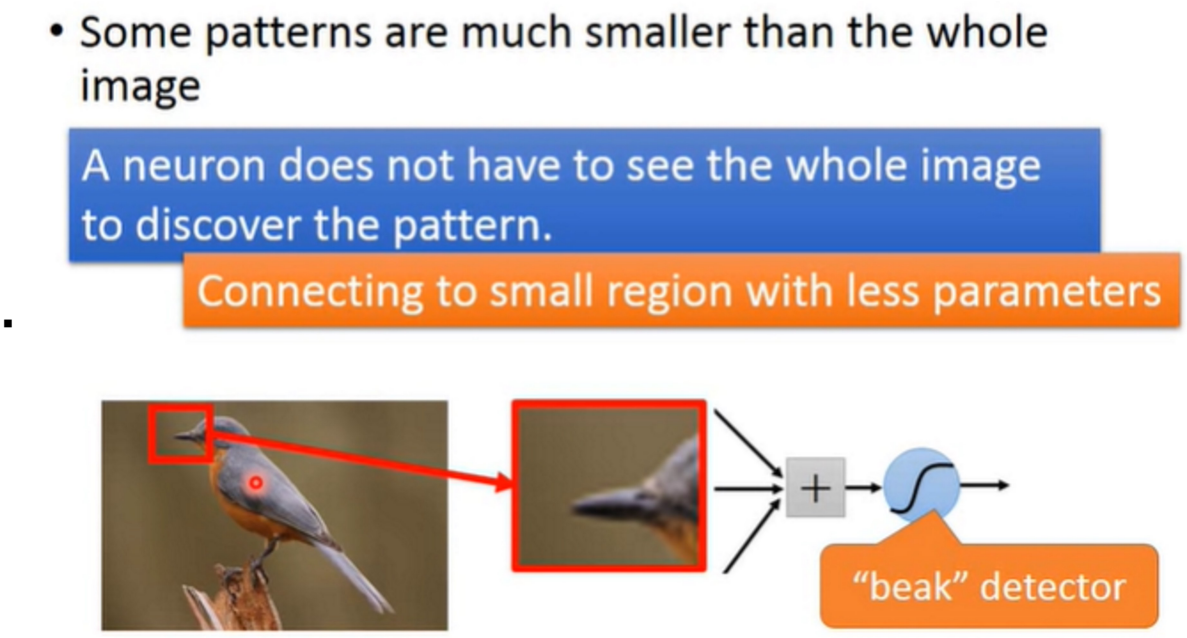
ML10台大李宏毅

Saturday, March 28, 2020 3:11 PM

[ML Lecture 10: Convolutional Neural Network]

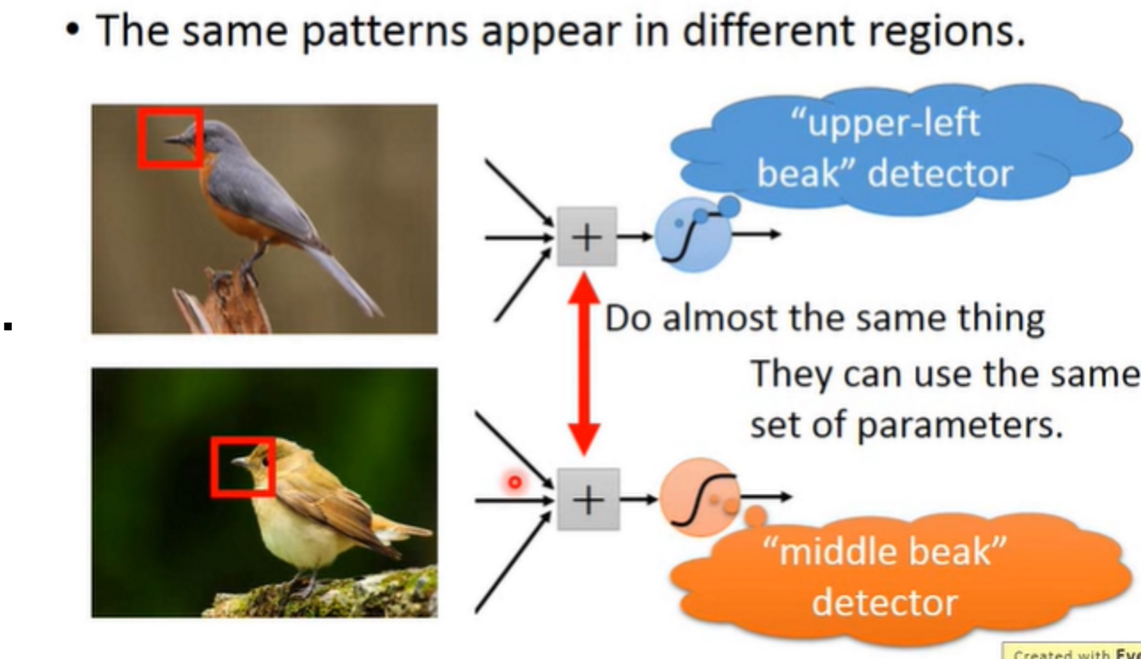


- Fully connected network:
  - image size: 100 * 100
    - pixel vector: 100 * 100 *3 = 30,000
    - hidden layer neurons: 1000
    - weight size: 30,000 * 1000 = 30000000
    - ==> too much
- we might not need the full connected network: Some patterns are much smaller than the whole image
  - e.g. is beak exist in image? is bird claw exist? Is bird wing exist? Is bird tail exist?
  - So when we are using CNN to train the recognization model, each neurons has their own duty in detecting different pattern from image. After certain step of training, each neurons are pretty good in detector certain features, the deeper layer wil combine those small feature and to discover a larger pattern… so no and so forth..



- ==> In this case, in order to detect the existance of "beak", we know a neuron only need to connect to a small region of image, instead of whole image.

- Beak Detector:
  - Pre-trained model ==> Had trained parameter for detecting certain pattern/features. So, when we using it, we can derectly add them to our model, instead to waste tons of time in training!
  - For example, there are two image, we want to detect whether beak exit in these image. We see the location of peak are different, but they can be detected with same set of parameters!



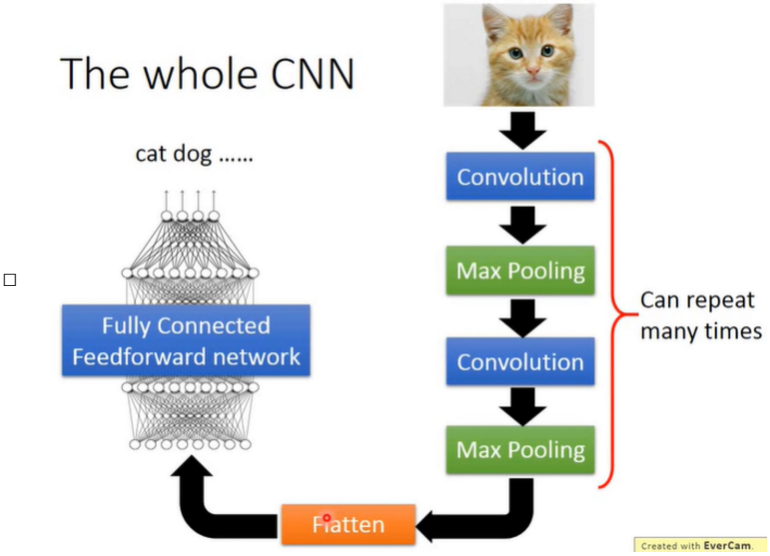
- Subsampling: Subsampling the pixels will not change the object
  - We can subsample the pixels in order to make the image size smaller, so less parameters for the network to process the image

- Subsampling the pixels will not change the object

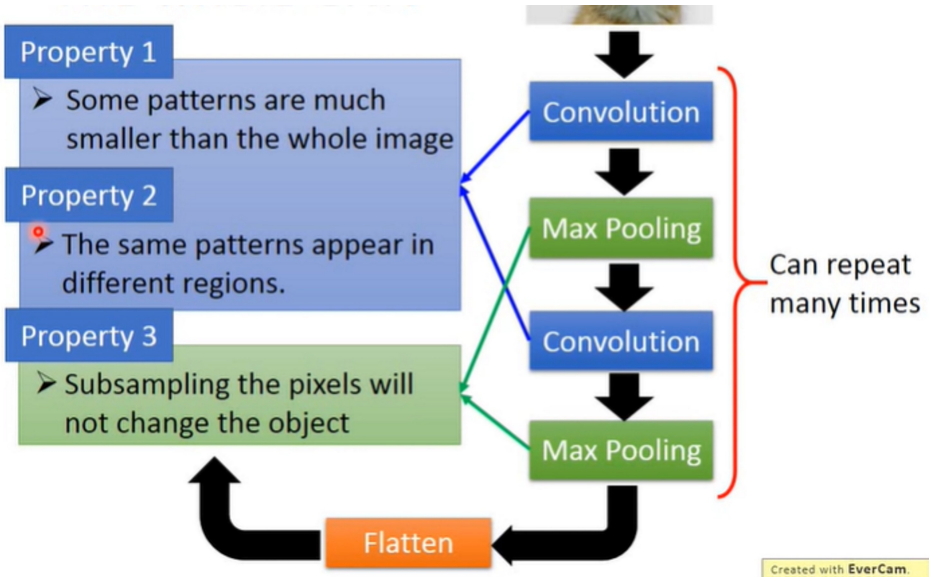
We can subsample the pixels to make image smaller

Less parameters for the network to process the image

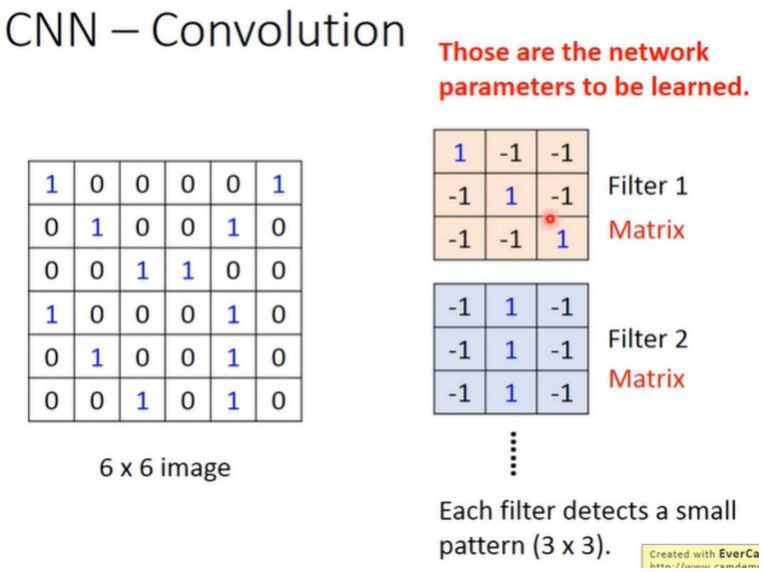
- The Whole CNN structures:
  - Input a image ==> Conv layer ==> Max Pooling ==> Conv ==> Max pooling …. ==> Flatten ==> Fully connected Feedforward network
    - This process(Conv layer ==> Max pooling) can repeat many times, the exactly number time of process is the CNN structuer you need to decided beforehand



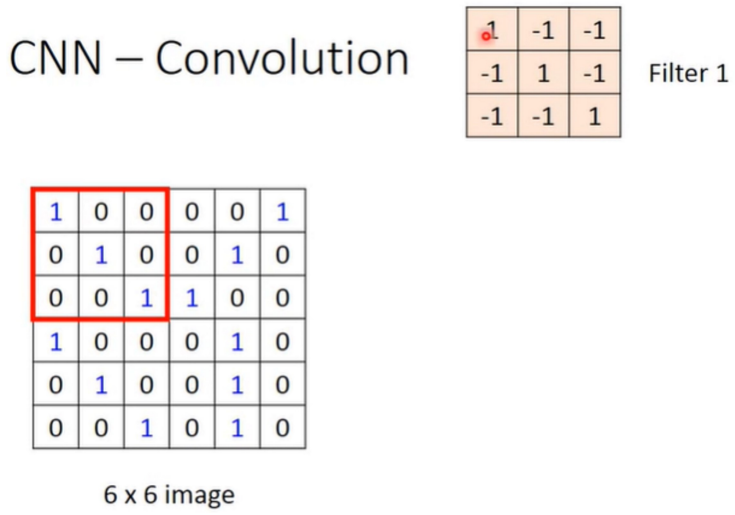
  - Properties:
    - Property 1: Some patterns are much smaller than the whole image
    - Property 2: The same patterns appear in different regions
    - Property 3: Subsampling the pixels will not change the object
  - The first two property are take cared by Conv layer, and the last one is take cared by Max pooling



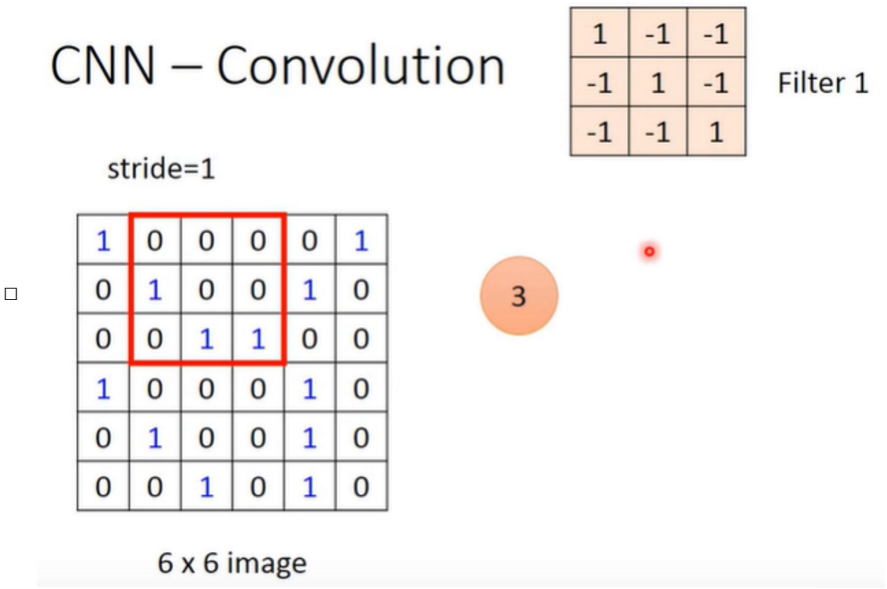
- CNN--Convolution
  - Let's say we have image matrix, a 6*6 black-white image, each pixel has two value, 1--black, 0 -- white
  - In Conv layer, there are a set of filter (Can also think of as a set of matrix). The exactly value for those filter are the network parameters to be learned, like some training process.
  - If the filter has size 3*3, then it's detecting a 3*3 pattern. It would looking for the whole image in a region of 3*3 size of image

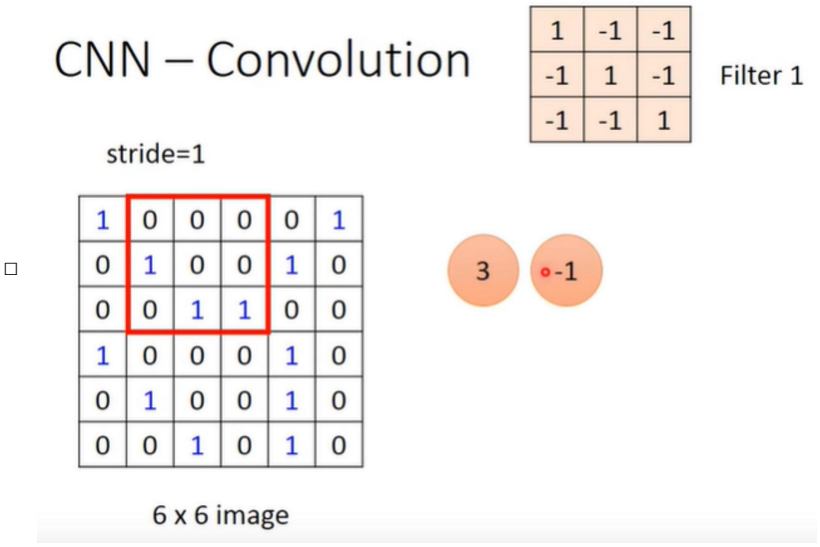


  - First step: Put your filter in the [0,0] position in image --> Take the dot product of there of matrix

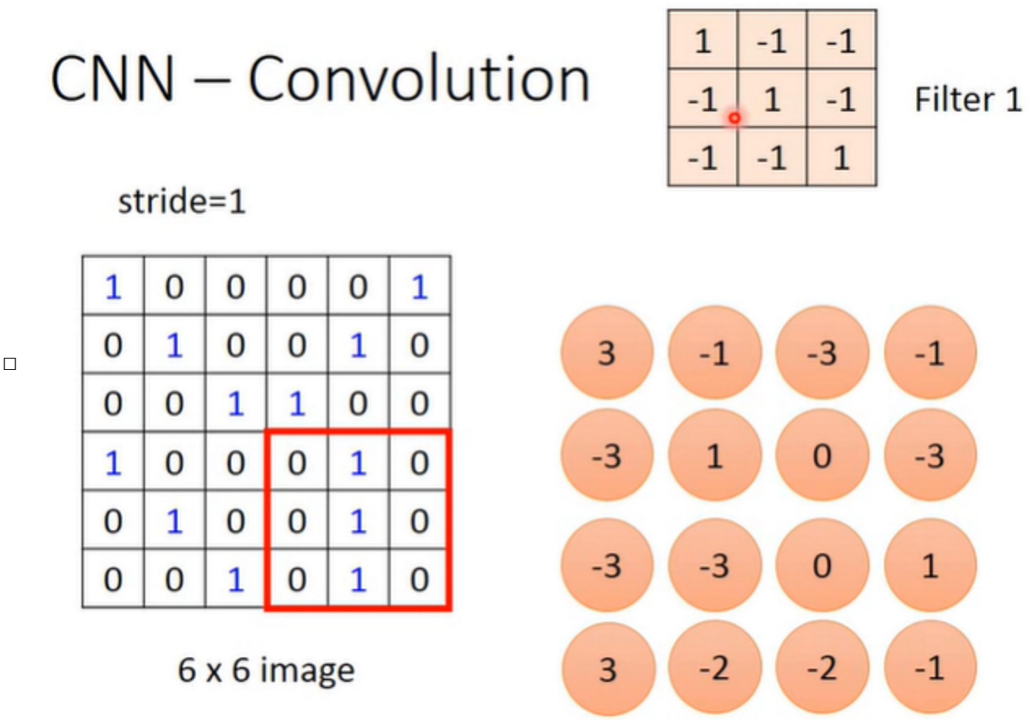


- Step 2: Move to the filter to next position. The distance we need to move called **stride** (It's a parameter you need to decide), For example
  - If stride == 1:

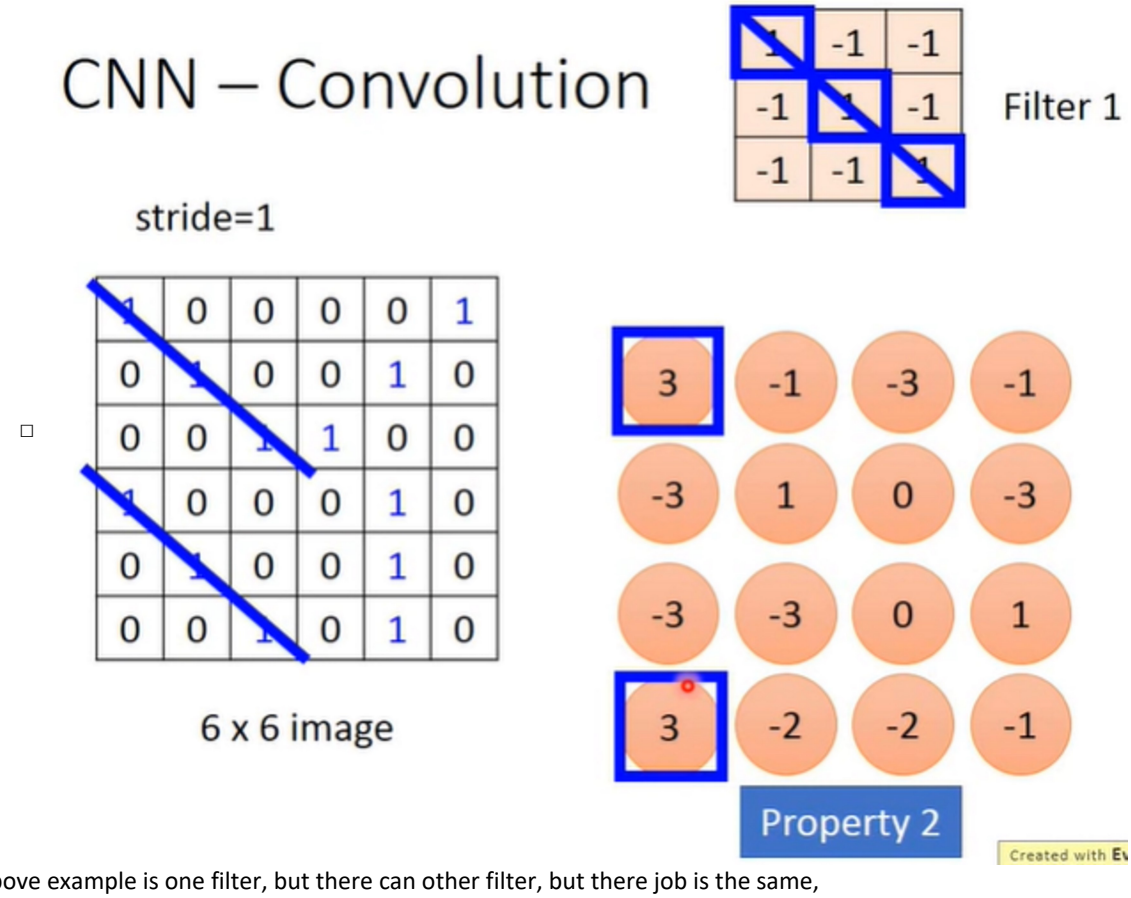




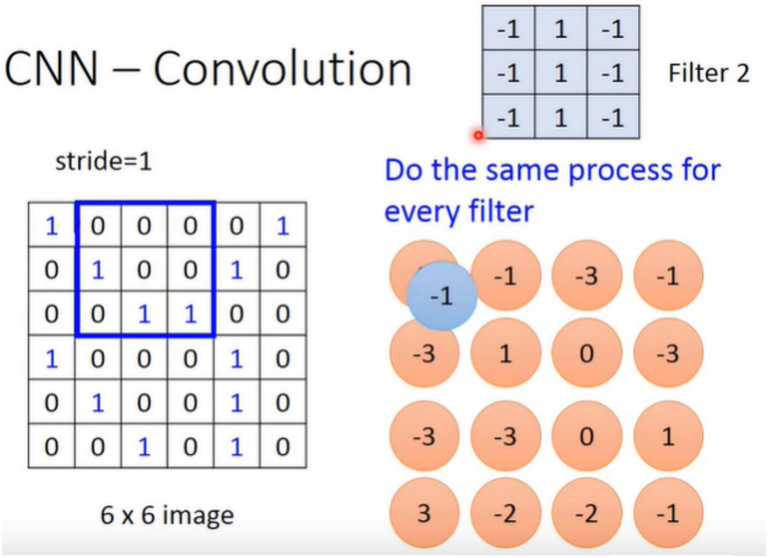
  - ....

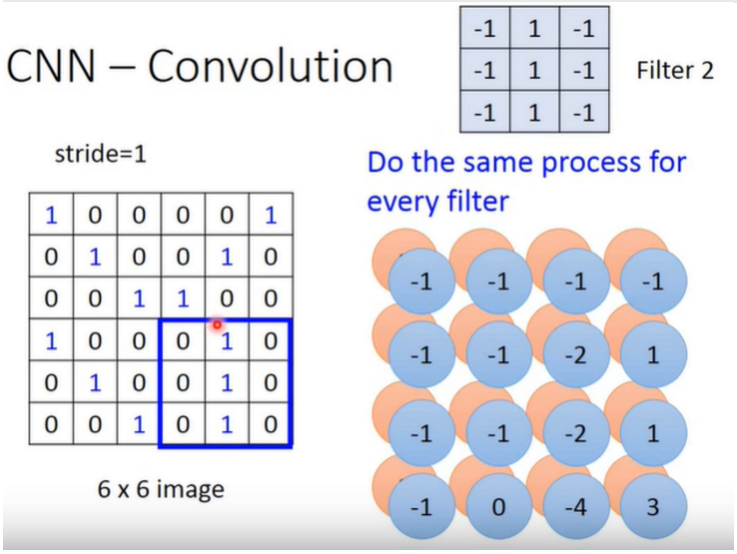


- Overall, based on the value of filter, we can see the job of this filter is detect this pattern:
  - We notice this patten appeared twice in this image, and this face reflected to our feature map(two maximum value--3)



- The above example is one filter, but there can other filter, but there job is the same,

- 

- 

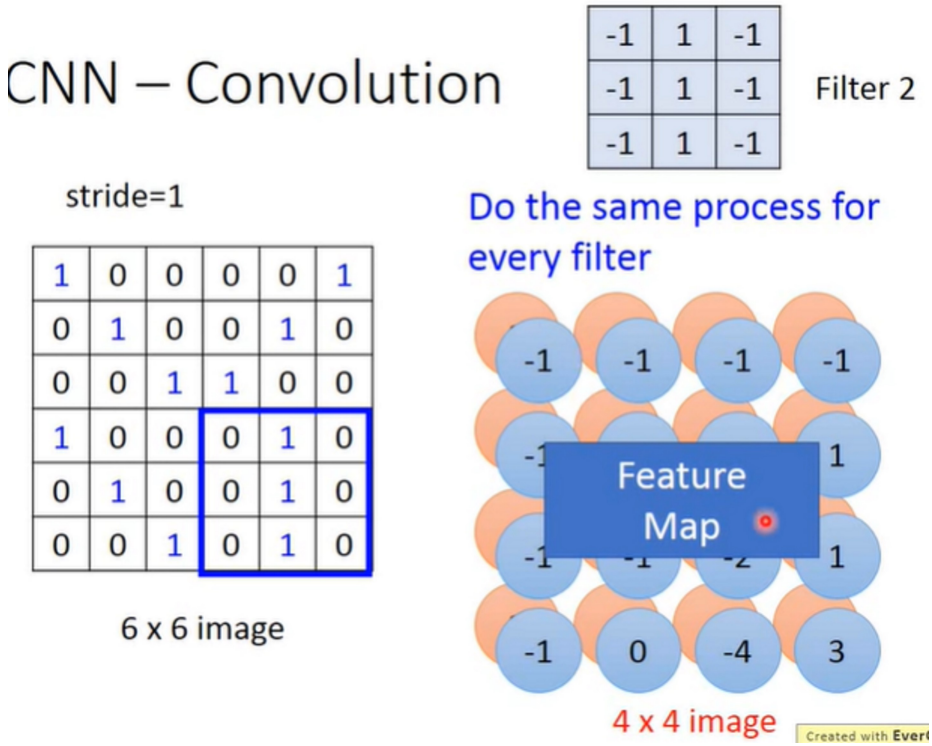
- So, now you have two feature map after above process. Those all are in the size of 4*4, there is a formulate to compute the size of this feature map:
  - Without padding: (N - F)/stride + 1
  - With padding: (N-F + 2*P)/S + 1

Common settings:

K = (powers of 2, e.g. 32, 64, 128, 512)
- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0

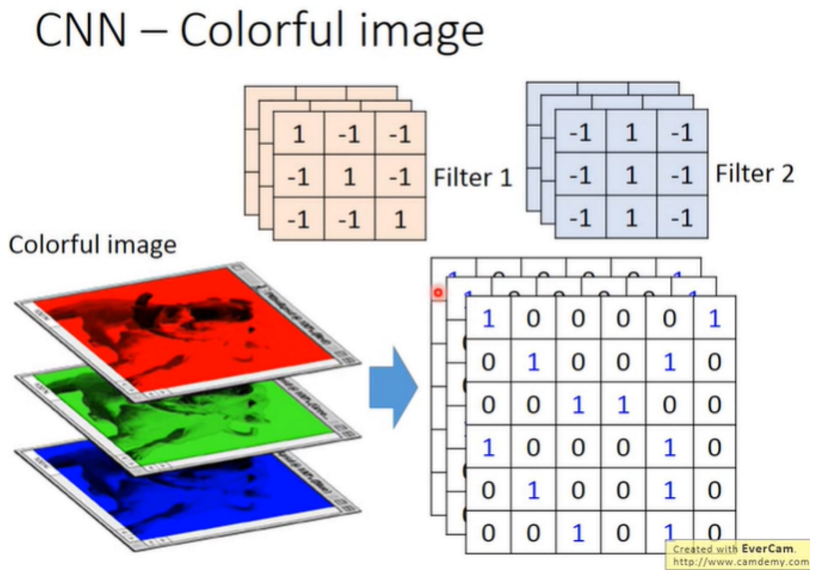
- **Summary**. To summarize, the Conv Layer:
  - Accepts a volume of size $W_1 \times H_1 \times D_1$
  - Requires four hyperparameters:
    - Number of filters K,
    - their spatial extent F,
    - the stride S,
    - the amount of zero padding P.
  - Produces a volume of size $W_2 \times H_2 \times D_2$ where:
    - $W_2 = (W_1 - F + 2P)/S + 1$
    - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
    - $D_2 = K$
  - With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
  - In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

  - If you have 100 filter, you will get 100 of Featuer map



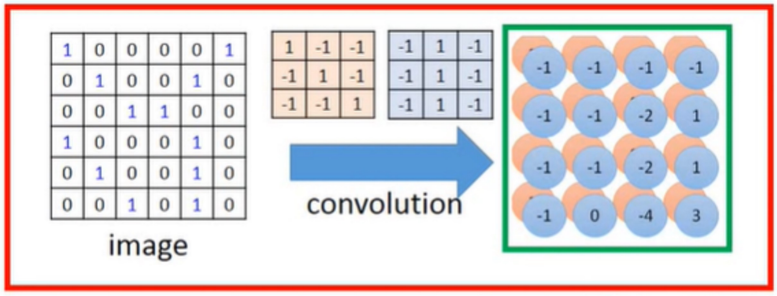
- There is one problem: Right now, all the filter size is the same, but in reality, the size of pattern can be different, e.g. some bird has larger beak, and some smaller. ==>if you know, you can do normalization.
- However, CNN don't have the ability to take care it, since they have different scale. ==> In ImageNet, there is a paper, …. suggest you can do some scaling, rotation, or zooming before through into CNN, and this can produce a better performance.

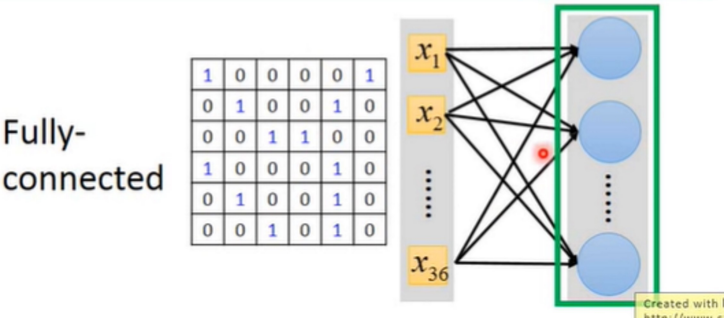
- Colorful image
  - Previous is B-W image, color image is the same. It has 3 dimension/channel (RGB). What different is your matrix is not a matrix, but a cubic (or tensor): Such as 3*3*3



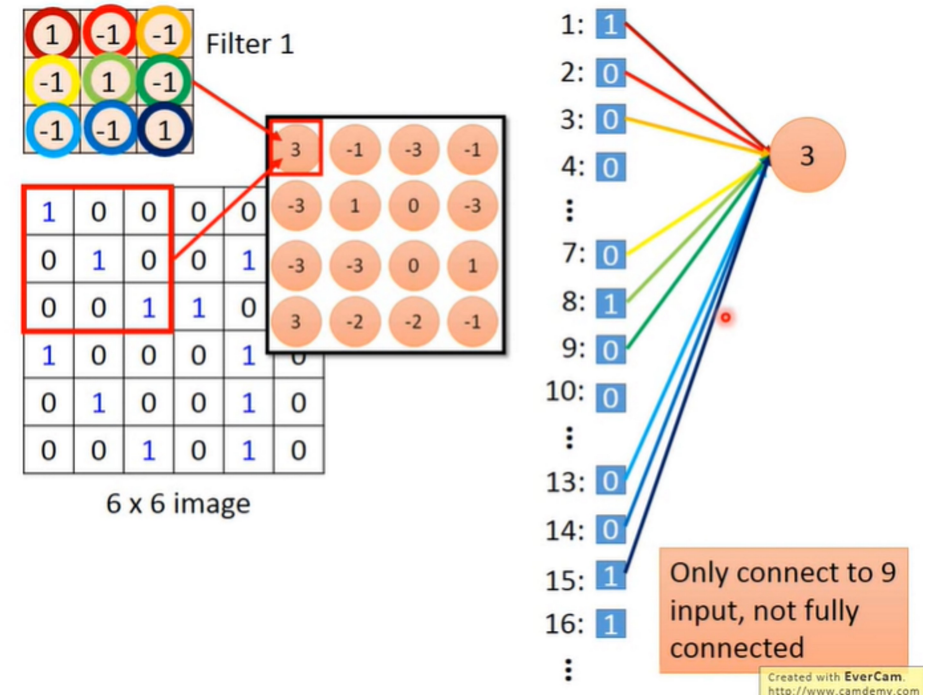
- Conv Vs Fully connected
  - In short: Conv Layer is a special case of Fully connected layer, just loss same weight/connection

Convolution v.s. Fully Connected

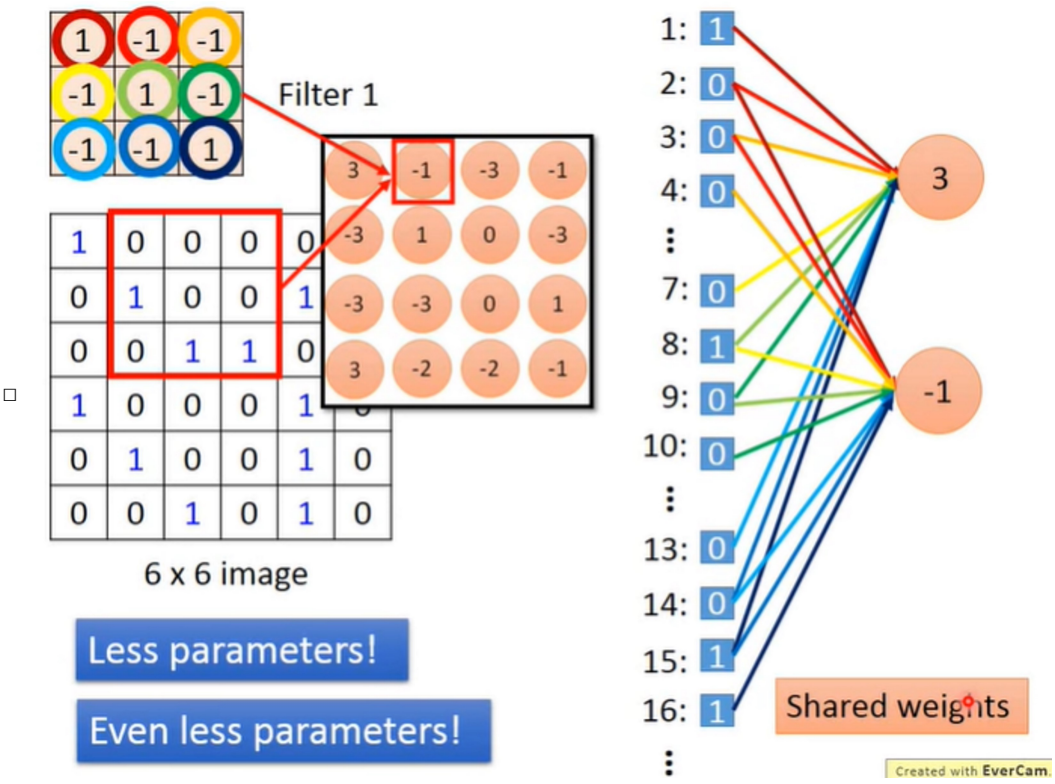




- What is in the left hand is a matrix, in the right hand, it's a long and thin vector (just different view). e.g. So a 6*6 matrix ==> 36 * 1 vector
  - The first filter in this vector is: 1,2,3,7,8, 9, 13, 14, 15
  - It's only connected to 9 input, not fully connected.
  - To detect a pattern, we don't need to look over the whole image, so less parameters!

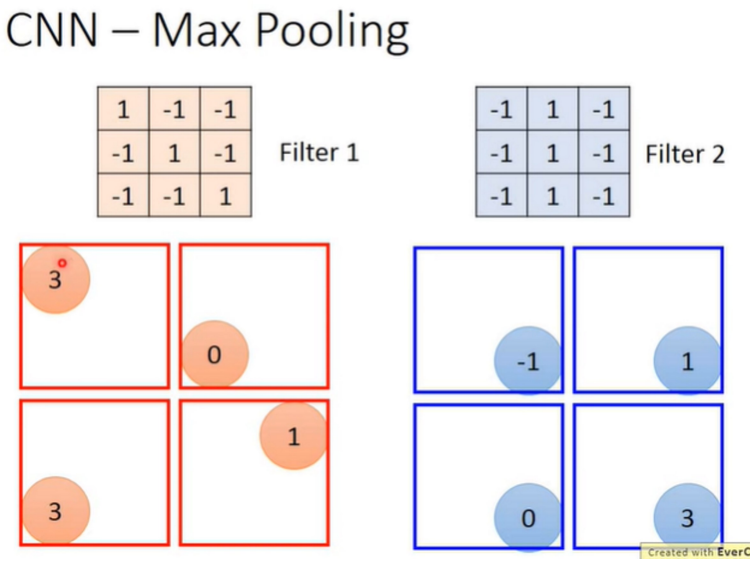


  - Weight sharing: same color means they using the same weight ==> So even less parameters:

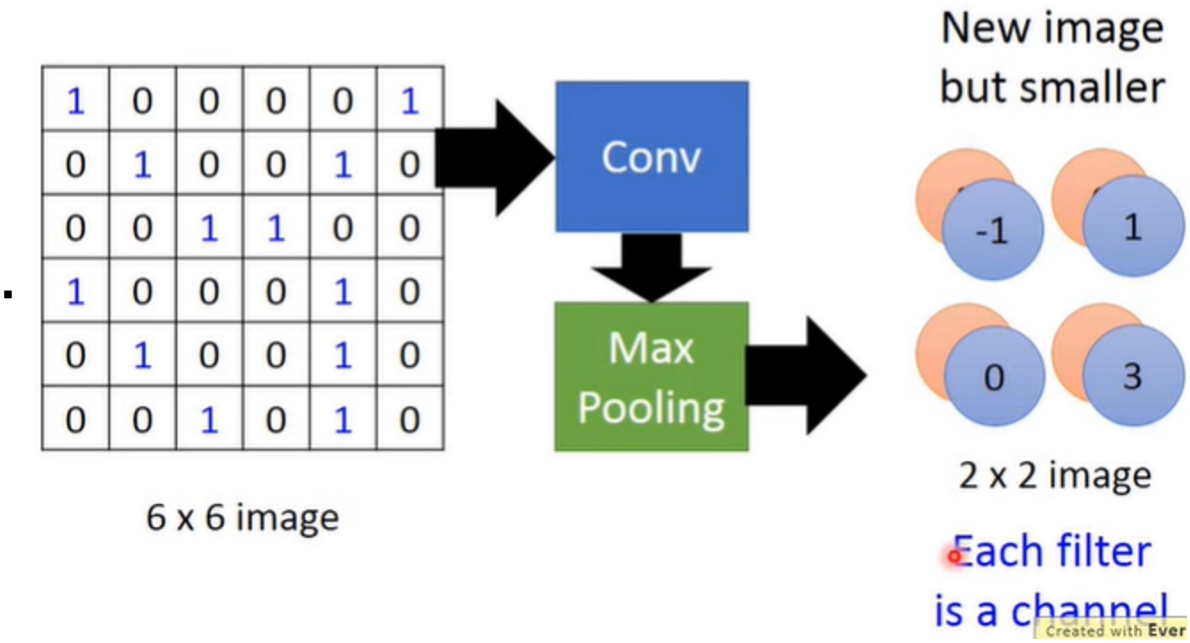


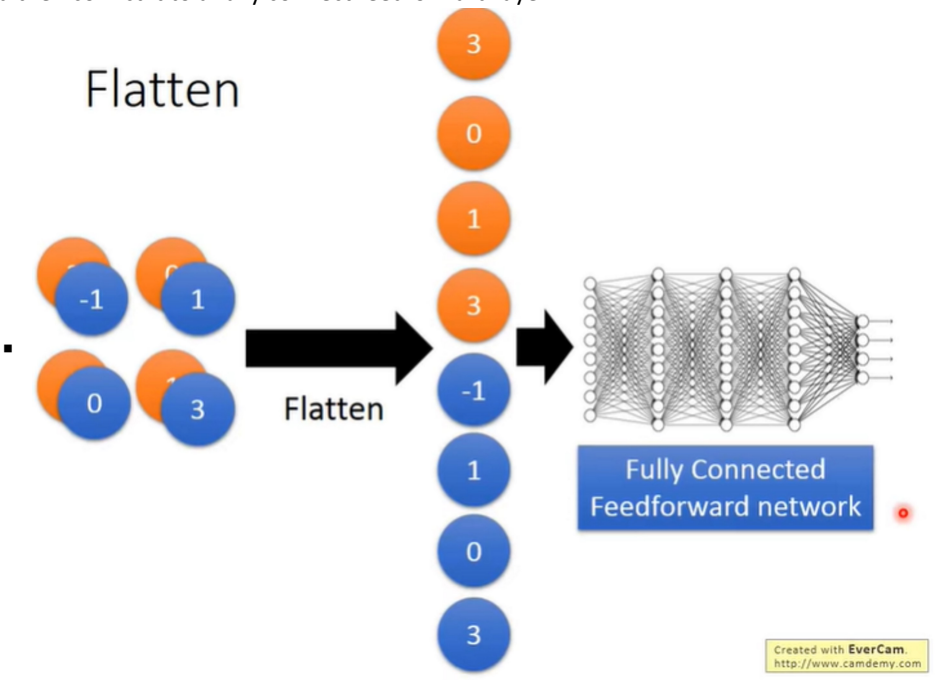
- Good thing: We have toolkit, so We don't really need to know the implementation detail

- Max pooling(or subsampling)
  - Extract the max in a subset

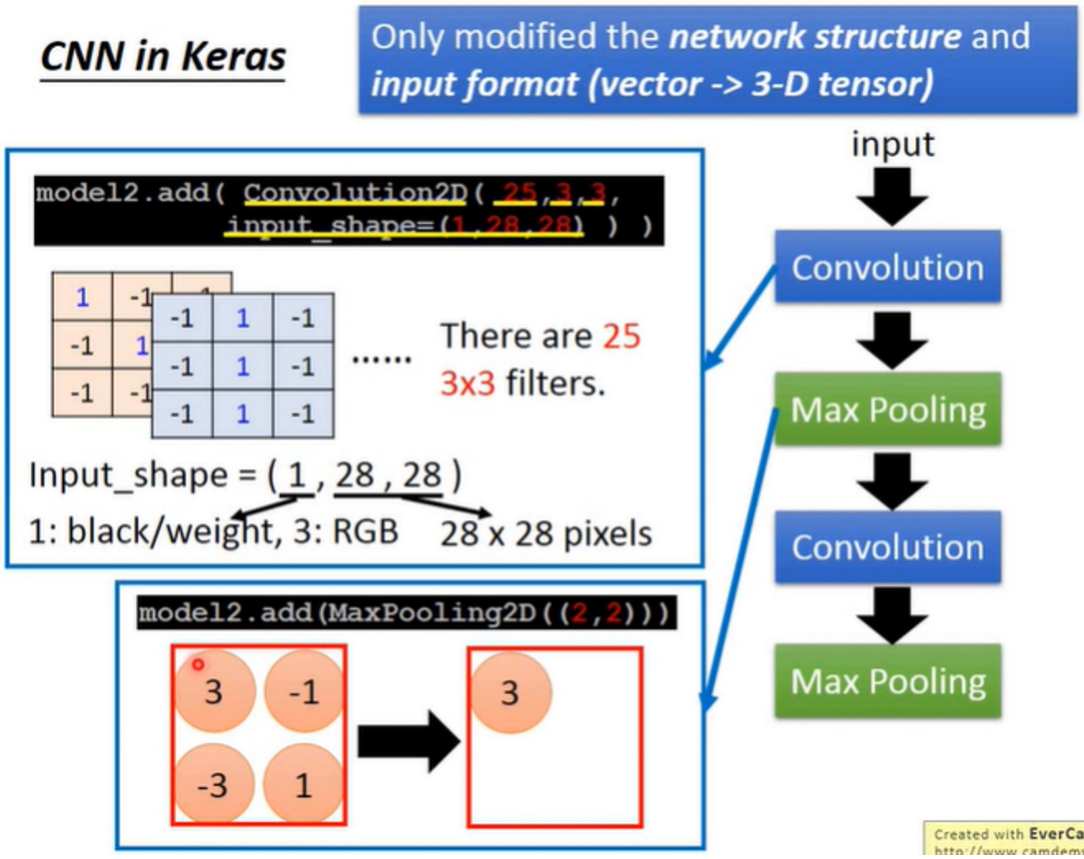


  - Problem: could you still do the derivative? ==> yes, discuss later
  - After this, we have a new image, with less parameter. The number of image is equal to the number of filter. Each filter is a channel.

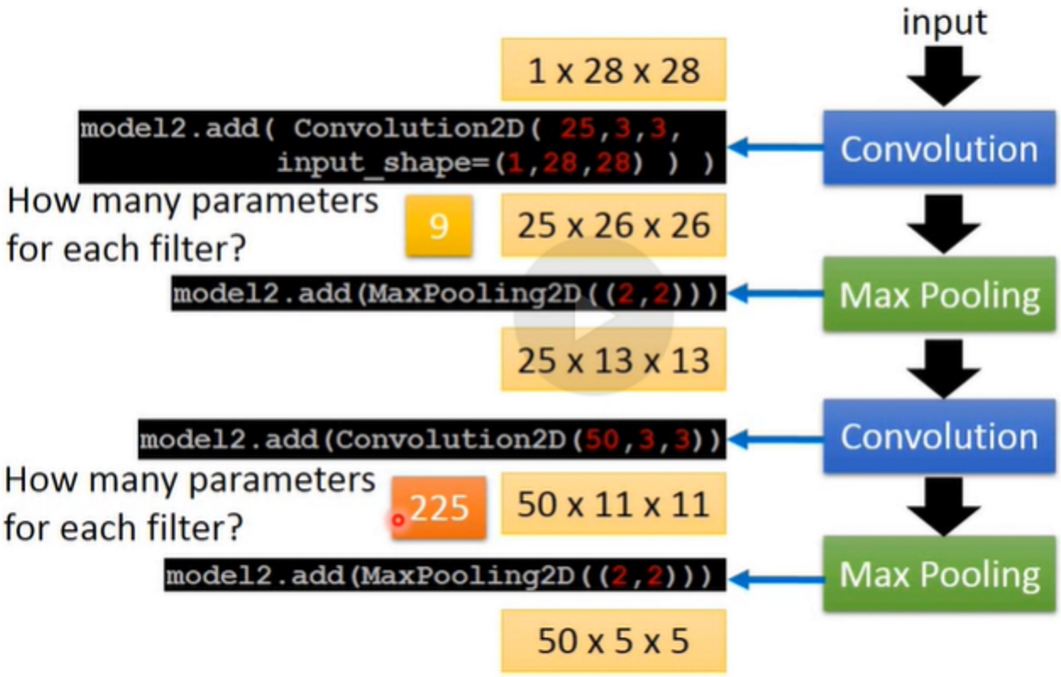
- 

    - You can repeat this process many times
- Flatten:
    - pull your features map into a long and thin line(vector)
    - And then connct it to a fully connect feedforward layer
- 

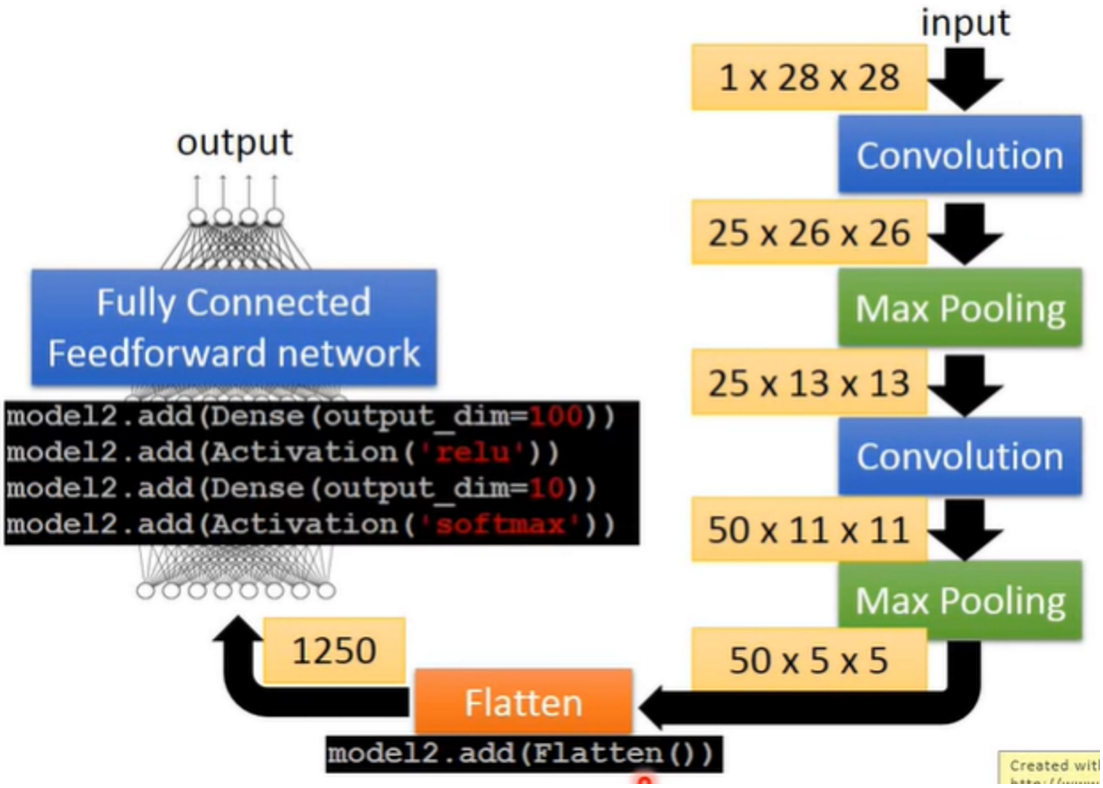
- CNN in Keras
    - Keras don't like vector, he want tensor. So, you need to modify your network structure and import (Vector --> 3D tensor)
        - Tensor is a high dimension vector
        - Why 3D? --> (High, Width, and color
    - mode2.add(Convolution2D(25, 3, 3, input_shape(1,28,28)))
        - 25 means 25 filters, and 3*3 is the filter size.
        - input_share: it means the input image has size 28*28, and they all B-W, so 1D image. If it's color, it would be 3D (RGB)
    - model2.add(MaxPooling2D((2,2)))
        - the pooling layer is in size (2*2)



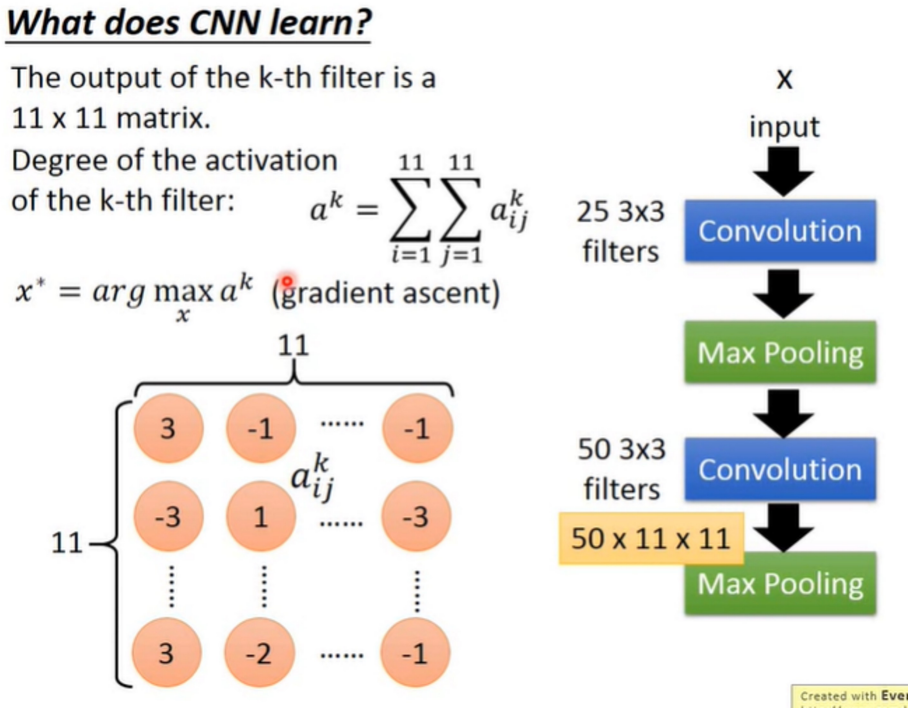
- The shape of output would be: 25 * 26 * 26, so 25 feature image, each with shape 26 * 26
- After the max pooling, the shape would be: 25 * 13 * 13
- You can repeat this process many times
  - Q: how many parameters in each Conv layer? --> First layer: 3*3 = 9, second layer: 25* 3*3 = 225



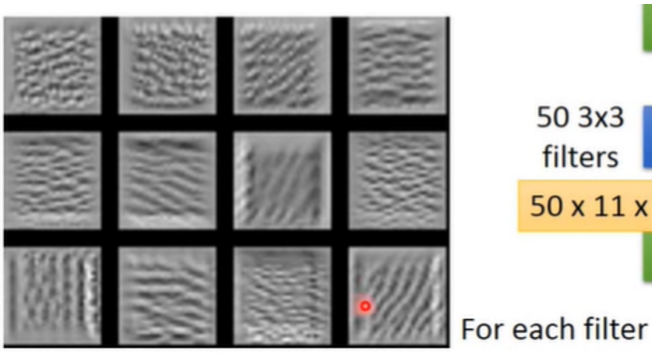
- Flatten + fully connected: model2.add(Flatten())



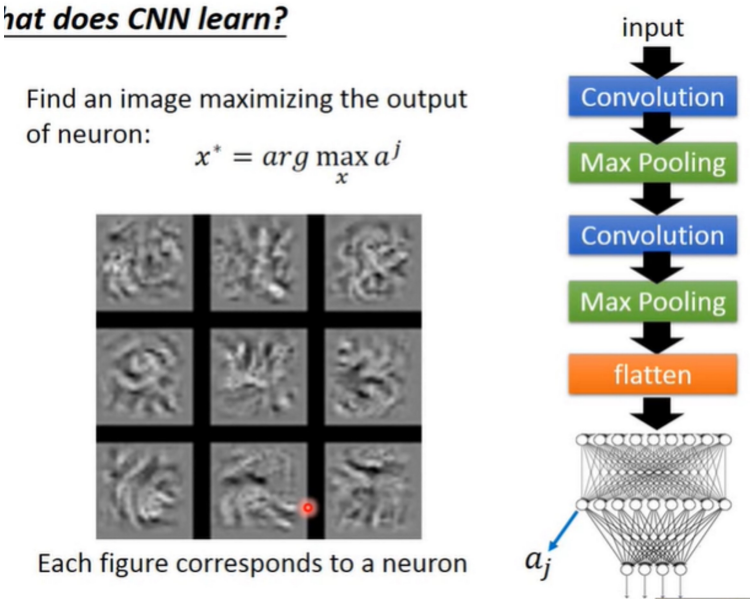
- Live Demo
  - Some people say CNN is a black box, no idea how to interpret this ourput
  - In my understanding, you try to give the intellegience to computer, it's has to be this way, something so complicate for human to understand. --> As least, it make you feel it's called intelligence. ^_^!
- What does CNN learned? How to analysis the job of certain filter?
  - Now, we have k filter, with 11*11 shape, and we want to find a image that the degree had been activated by filter in maximum, let's call this image as X*. So, we have 50 filter, X* denotes those 50 images that has the most degree being activated.
  - How: use gradient ascent



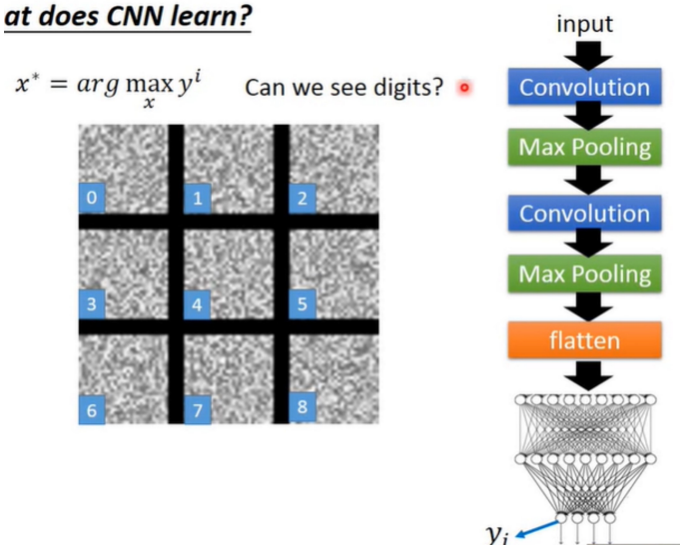
  - Noticed each filter has different job in detecting different pattern/feature, e.g. some stright line, some diagonal line…. the texture is different

- 

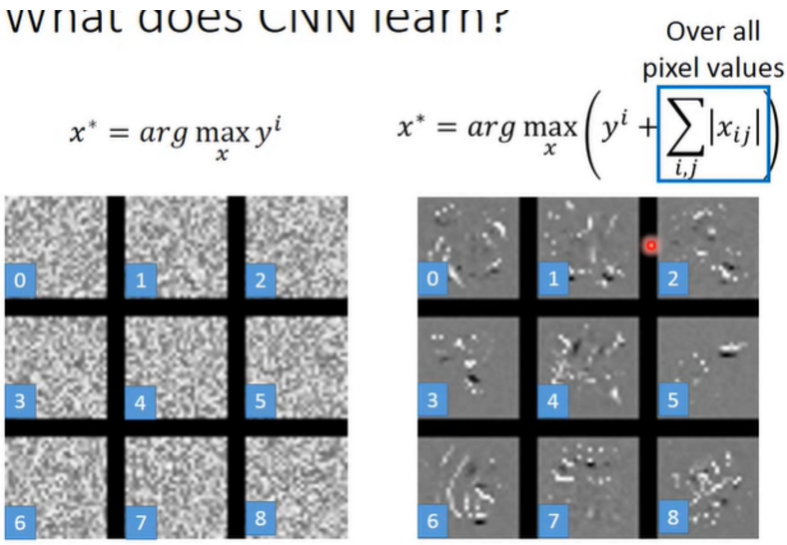
- What are those 50 images after output from flatten layer? --> There are 50, but take 9 as example



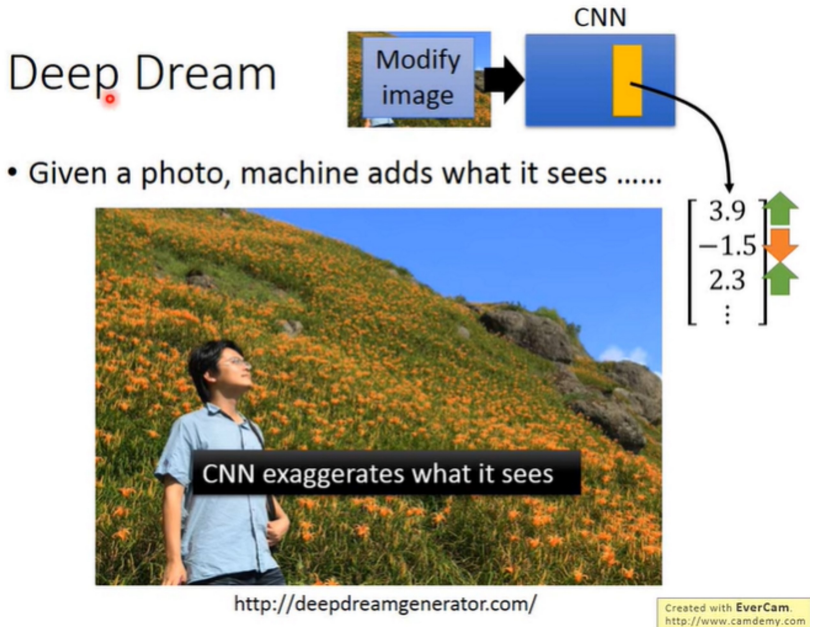
- What are the 10 image after output layer? Can we see the digits? --> We might not see the digit, but, in fact, the CNN model can see. The point is the image in CNN land is different in real land(Human land), what they learn is not interpretable by human!



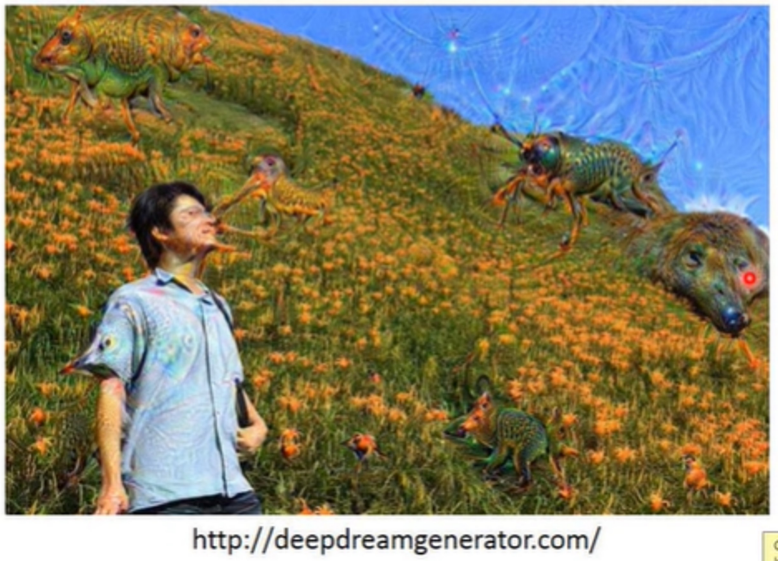
  - The fact had been observed by many researchers: Deep Neural Networks are Easily Fooled, https://www.youtube.com/watch?v=M2IebCN9Ht4
- Is there a way to make it looking better? --> Yes, add some constrain, or **regularization** on x*.
  - Some mistake on formulate: y^i - ….



- Deep Dream:
  - Given a phone, machine will adds what he see
  - Exaggerate the value of hidden layer

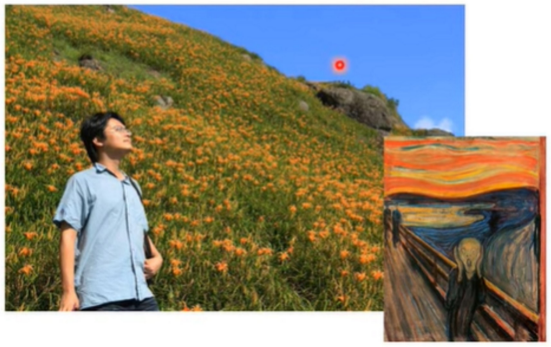


  - And throw this image to Deep Dream, you will see something interesting:



- Deep Style

Deep Style

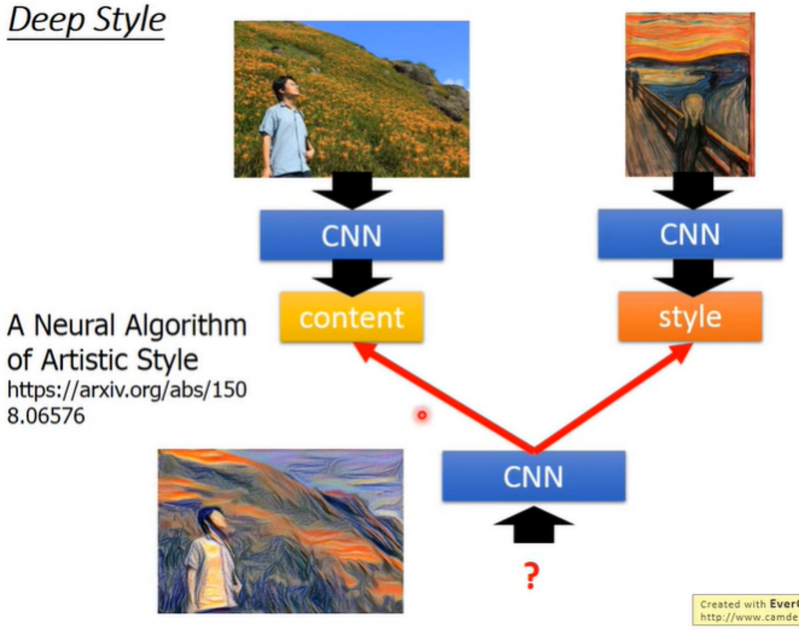
- Given a photo, make its style like famous paintings
  - 

https://dreamscopeapp.com/ Created with http://www.

- Given a photo, make its style like famous paintings
  - 

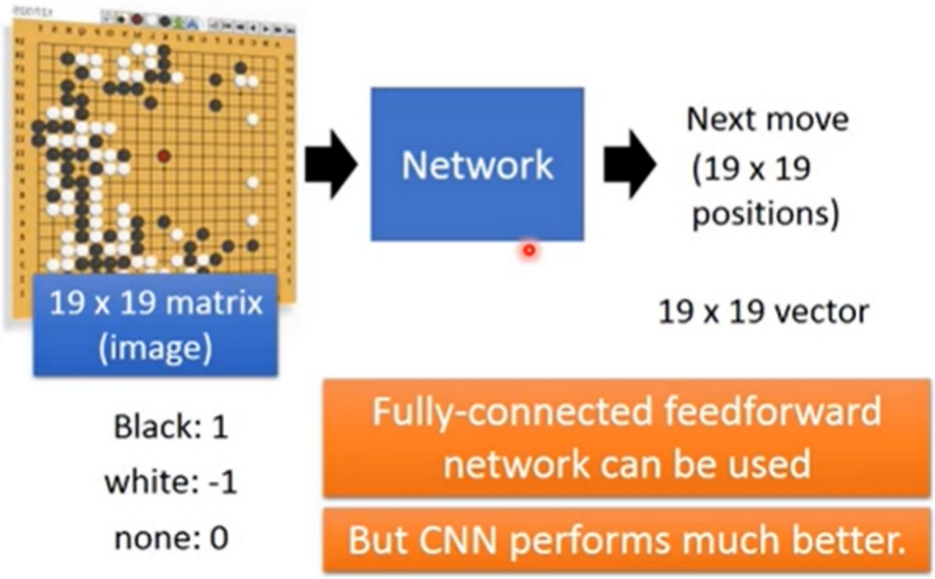
https://dreamscopeapp.com/ Created with E

- How does these was implemented?
  - Reference: A Neural Algorithm of Artistic Style, https://arxiv.org/abs/1508.06576
  - Throw your image into CNN --> Get a filter output.
  - Throw the artist image into CNN --> Get another image
  - we don't really care the actually value of output, but the correlation between two output
  - Now, you try to ask your machine/CNN to generate a image that could maximize the content in your image and the style from artist image (e.g. Gradient decent)

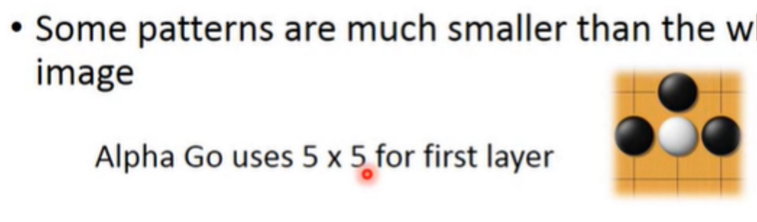


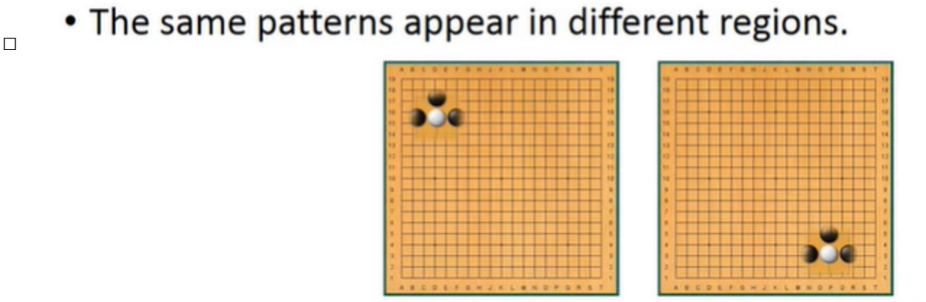
- More application: Playing GO
  - 收集一堆棋谱，let Black =1, white = -1, otherwise = 0， 然后丢给CNN，就像train

image 一样。



- Why CNN for playing GO?
  - Why it's necessary to use CNN?
    - Only when we know the image/application has previous three property!
  - Why it working in playing GO?
    - Because Go has similar structure as image: 1)Some patterns are much smaller than the whole image. 2)The same pattern appear in different regiosn. 3)No Maxpooling? max pooling这个架构



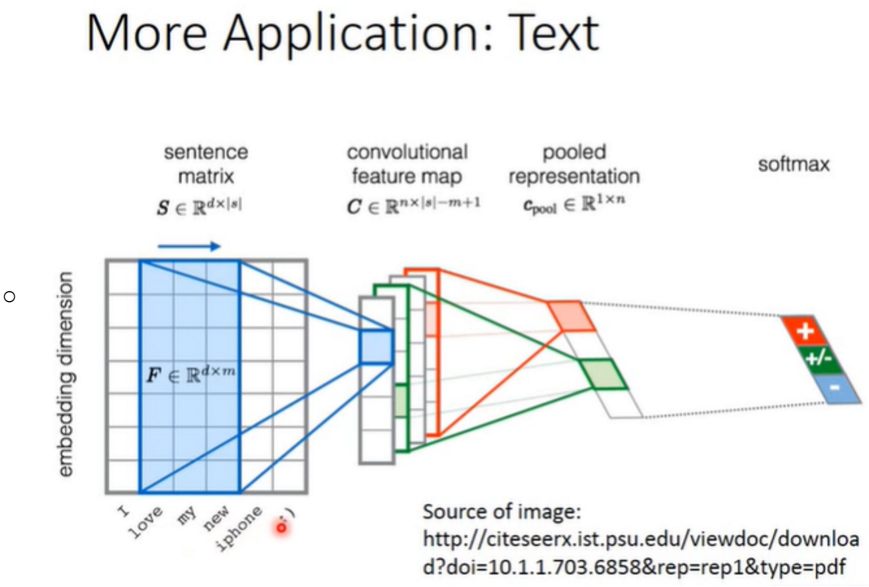


- Subsampling the pixels will not change the object

→ Max Pooling How to explain this???

- **Neural network architecture.** The input to the policy network is a $19 \times 19 \times 48$ image stack consisting of 48 feature planes. The first hidden layer zero pads the input into a 23×23 image, then convolves k filters of kernel size 5×5 with stride 1 with the input image and applies a rectifier nonlinearity. Each of the subsequent hidden layers 2 to 12 zero pads the respective previous hidden layer into a 21×21 image, then convolves k filters of kernel size 3×3 with stride 1, again followed by a rectifier nonlinearity. The final layer convolves 1 filter of kernel size 1×1 with stride 1, with a different bias for each position, and applies a softmax function. The match version of AlphaGo used $k = 192$ filters; Fig. 2b and Extended Data Table 3 additionally show the results of training with $k = 128$, 256 and 384 filters.

- More Application: Text
  - Give a word sequenc --> tell me it's positive/negative/neutral emotion?
  - Known as Word embedded, each word can be represented as a vector
  - You can apply CNN in test analysis, but there are some difference: 1)The blue Box is your filter, and you move along the word sequence, and your feature map is a vector. The number of vectors equals the number of filter.



- To learn more:
  - The methods of visualization in these slides
    - https://blog.keras.io/how-convolutional-neural-networks-see-the-world.html
  - More about visualization
    - http://cs231n.github.io/understanding-cnn/
  - Very cool CNN visualization toolkit
    - http://yosinski.com/deepvis
    - http://scs.ryerson.ca/~aharley/vis/conv/

- How to let machine draw an image
  - PixelRNN
    - https://arxiv.org/abs/1601.06759
  - Variation Autoencoder (VAE)
    - https://arxiv.org/abs/1312.6114
  - Generative Adversarial Network (GAN)
    - http://arxiv.org/abs/1406.2661