Week 1 Foundations of Convolutional NNs

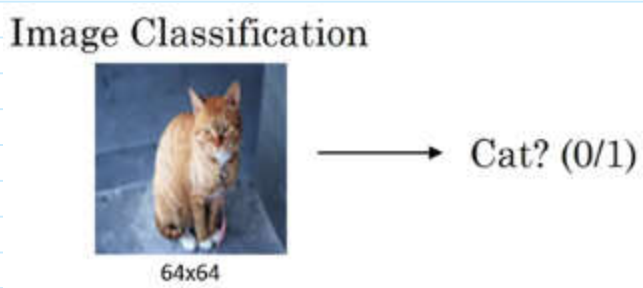
Sunday, 12 August, 2018 11:35

Computer Vision

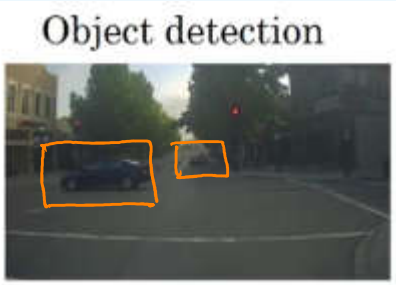
Sunday, 12 August, 2018 11:40

Computer vision functional uses:

- car driving
- face recognition
- object recognition / image classification
  cats, fruits, locations, etc
- art creation (neural style transfer)

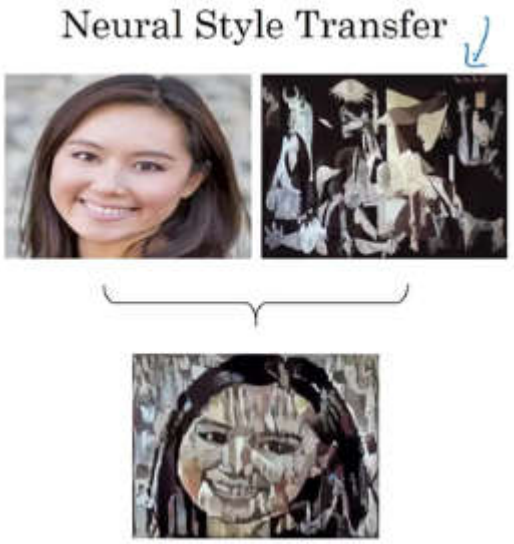


Screen clipping taken: 2018-08-12 11:53



Screen clipping taken: 2018-08-12 11:54

Not only determine if the image shows the item of interest but also how many items there are and their location.



Screen clipping taken: 2018-08-12 11:59

- CV is a very prolific domain with many innovations that can be translated

to other ML domains (i.e. speech recognition).

DEEP LEARNING & COMPUTER VISION CHALLANGES

Input feature set can get to be very large!

→ Cat? (0/1)

64x64 x3 ← color channels (RGB)

64 × 64 × 3 = 12,288 features

(though a 64 × 64 image is very small)

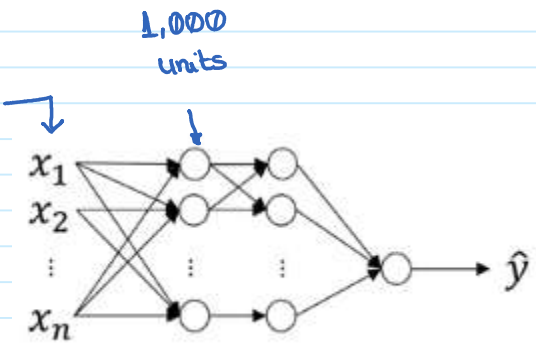
Screen clipping taken: 2018-08-12 12:02

1000 × 1000 × 3 = 3 million features

Screen clipping taken: 2018-08-12 12:04

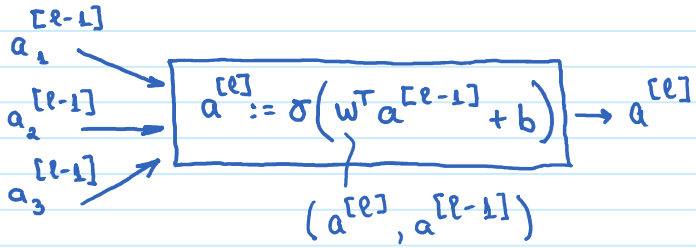
For a 3M input features DL NN with 1,000 hidden units on the 1st layer we have:

3M features each feature is a pixel R,G or B in the image



Screen clipping taken: 2018-08-12 12:05

REMEMBER:



For a standard Fully Connected layer 1 network the weights tensor $w^{[1]}$ is a $(1000, 3M)$ matrix containing 3B parameters.

→ it is very difficult to get enough data to prevent OVERFITTING.
(i.e. way too many parameters for each data example)

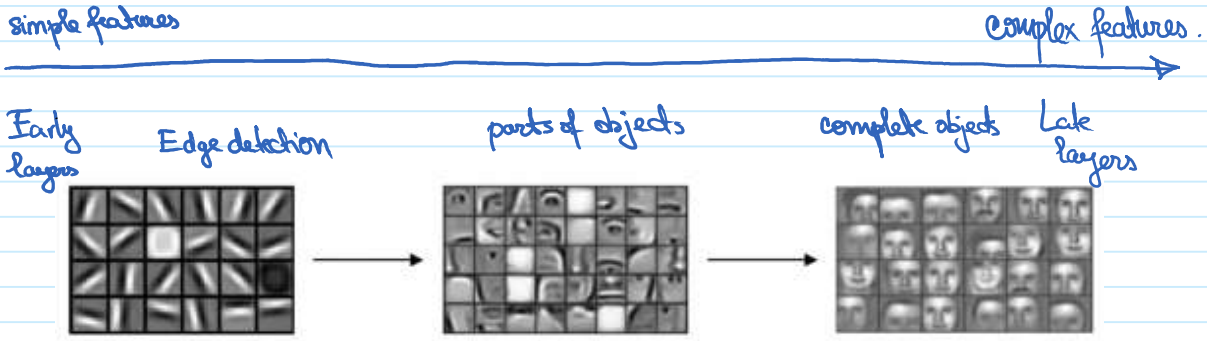
→ MEMORY REQUIREMENTS are not feasible

To get around the huge input features problem we use CONVOLUTION.

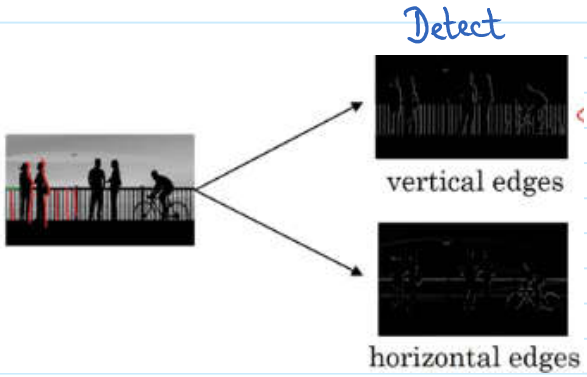
Edge Detection example

Sunday, 12 August, 2018 13:29

EDGE DETECTION - CONVOLUTION MOTIVATING EXAMPLE



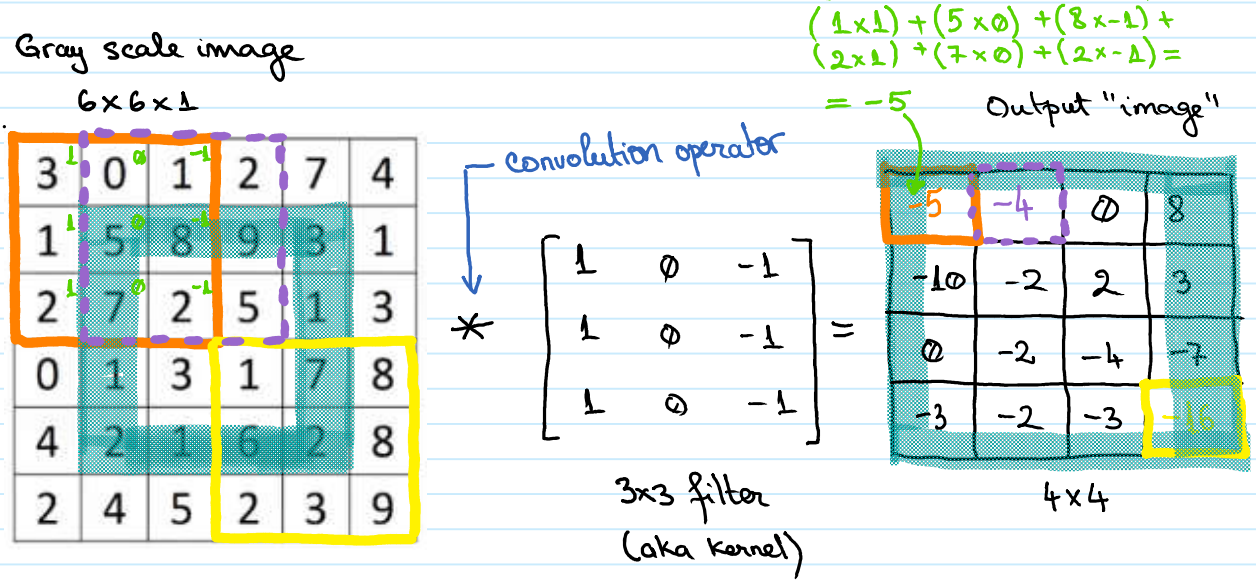
Screen clipping taken: 2018-08-12 13:32



Screen clipping taken: 2018-08-12 13:35

VERTICAL EDGE DETECTION:

- Convolution operation:



Screen clipping taken: 2018-08-12 13:37

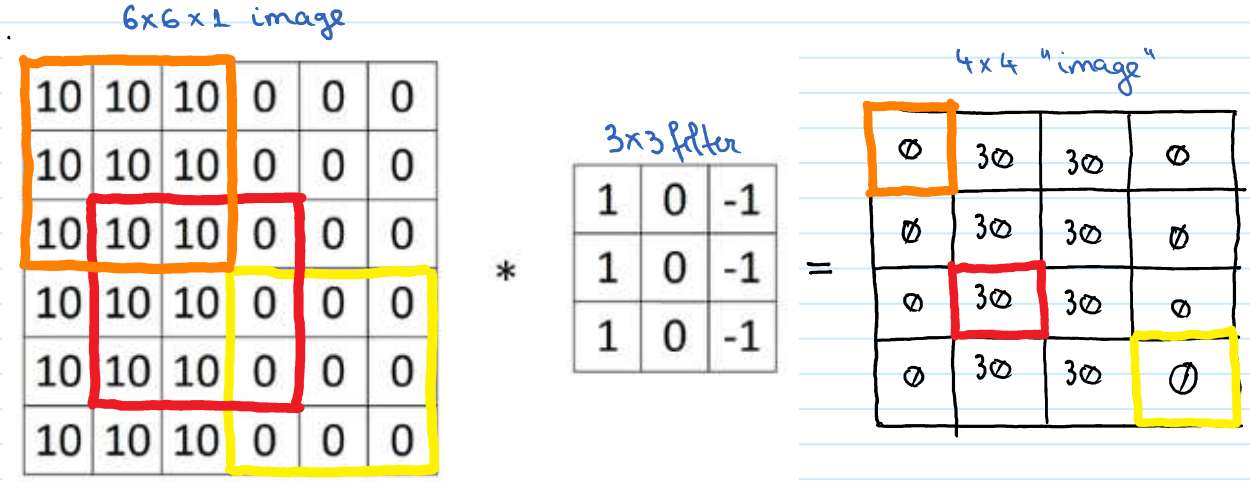
In practice programming languages don't use "*" they use a function call:

Python: conv_forward

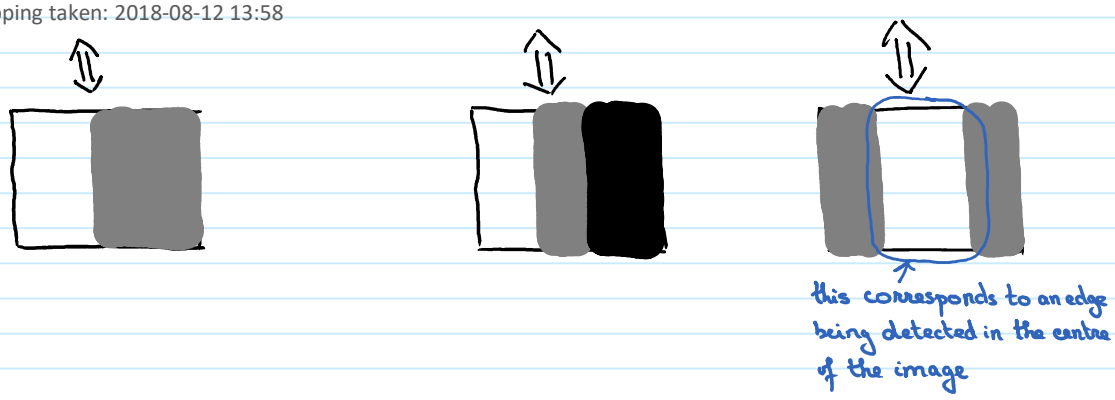
Tensorflow: tf.nn.conv2d

Keras: conv2D

- How does convolution detect edges?



Screen clipping taken: 2018-08-12 13:58



The "edge" appears to be very thick (wide) b/c the input image is very small. If we use a 1000x1000 input image the detected edges in the output "image" will look more appropriately sized.

- INTUITION (gained from interpreting the filter)

A vertical image is an image where there are bright pixels on the left and dark pixels on the right

→ What about bright pixels on the right and dark on the left? See next slides.

More edge detection

Sunday, 12 August, 2018 14:14

Goals: - Learn difference b/w positive & negative images
→ i.e. light to dark vs dark to light edge detectors
- Other types of edge detectors
- How to train an algorithm to learn edge detection.

• POSITIVE & NEGATIVE IMAGES

· From previous example we had:

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

*

1	0	-1
1	0	-1
1	0	-1

=

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

Screen clipping taken: 2018-08-12 14:29

What happens if the input image is flipped horizontally and we still use the same filter?

0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10

*

1	0	-1
1	0	-1
1	0	-1

=

0	-30	-30	0
0	-30	-30	0
0	-30	-30	0
0	-30	-30	0

Screen clipping taken: 2018-08-12 14:25

The output "image" gets the central band inversed.

- OTHER TYPES OF EDGE DETECTORS:

1	0	-1
1	0	-1
1	0	-1

Vertical

Screen clipping taken: 2018-08-12 14:30

1	1	1
0	0	0
-1	-1	-1

Horizontal

Screen clipping taken: 2018-08-12 14:30

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10

Screen clipping taken: 2018-08-12 14:34

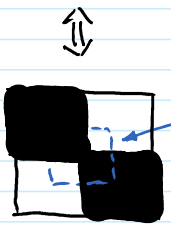
horizontal edge detector

1	1	1
0	0	0
-1	-1	-1

$$= \begin{matrix} 0 & 0 & 0 & 0 \\ 30 & 10 & -10 & -30 \\ 30 & 10 & -10 & -30 \\ 0 & 0 & 0 & 0 \end{matrix}$$

positive edge

negative edge



these intermediate values are an artifact of working with small input images. They reflect the fact that the filter captures part of the positive edge on the left and part of the negative edge on the right

→ again for larger images (1000×1000) this intermediate values effect will be much less pronounced.

$$\begin{matrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{matrix}$$

Sobel vertical filter

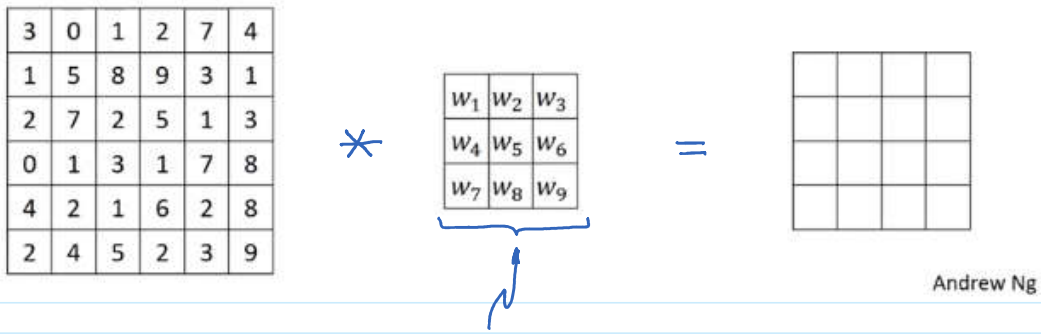
emphasize the middle row more.

$$\begin{matrix} 3 & 0 & -3 \\ 10 & 0 & -10 \\ 3 & 0 & -3 \end{matrix}$$

Scharr vertical filter.

• LEARNING TO DETECT EDGES

As Deep Learning developed we learned that maybe we don't need to create the filters ourselves we'll get the NN to determine what is the best filter to use.



Screen clipping taken: 2018-08-12 14:54

we treat the 9 numbers of the filter as parameters we can learn using back propagation.

We'll see that the NN will learn the right filter that captures the data pattern we want. It will detect multiple types of low level features using filters that are more effective than hand designed filters.

→ vertical edges, horizontal edges, diagonal edges etc.
(45°, 75°, 63°...)

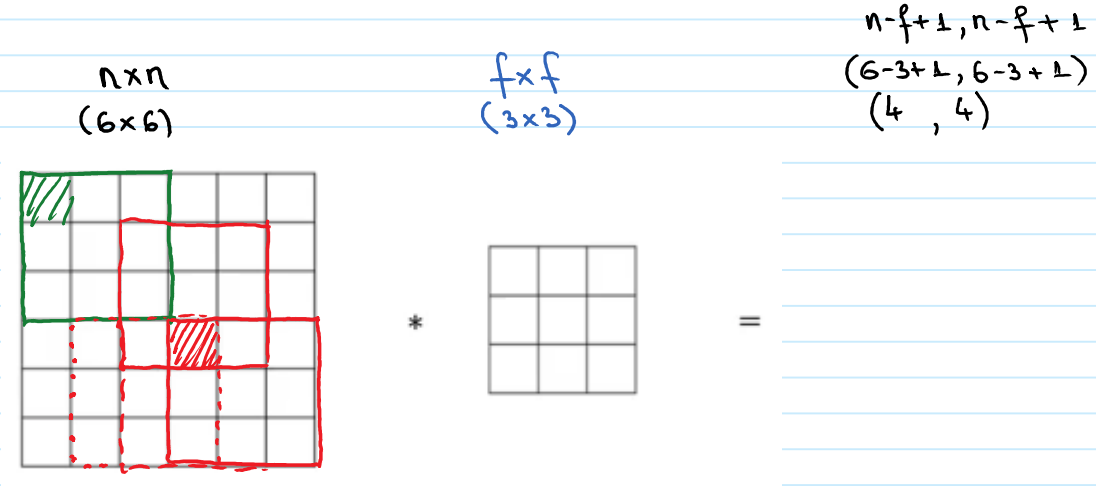
Underlying the learning process is the application of the convolution operation which determines what back propagation optimizes on (i.e. what features it needs to detect).

→ we'll see how to use back propagation to do this later in the week.

Padding

Sunday, 12 August, 2018 15:03

Given two matrixes of dimensions $n \times n$ and $f \times f$ the convolution resultin matrix is of size $(n-f+1, n-f+1)$



Screen clipping taken: 2018-08-12 16:40

There are two downsides to this:

1) everytime we convolve image size shrinks.
   - Maybe we don't want the image to shrink everytime we detect edges.
   - If we have a 100 layer deep NN and after each layer we are shrinking the image a little then after 100 layers we have a very small image left.

2) the information in the pixels on the perimeter is used less than the pixels in the center. We are throwing away of the information at the edges
   i.e. the information in the corner pixels is used only once while central pixel information is used multiple times
   → it affects the values of multiple cells in the output

SOLUTION:

- Pad the image with an additional p pixels "thick" perimeter

  New padded image size = $n + 2p$

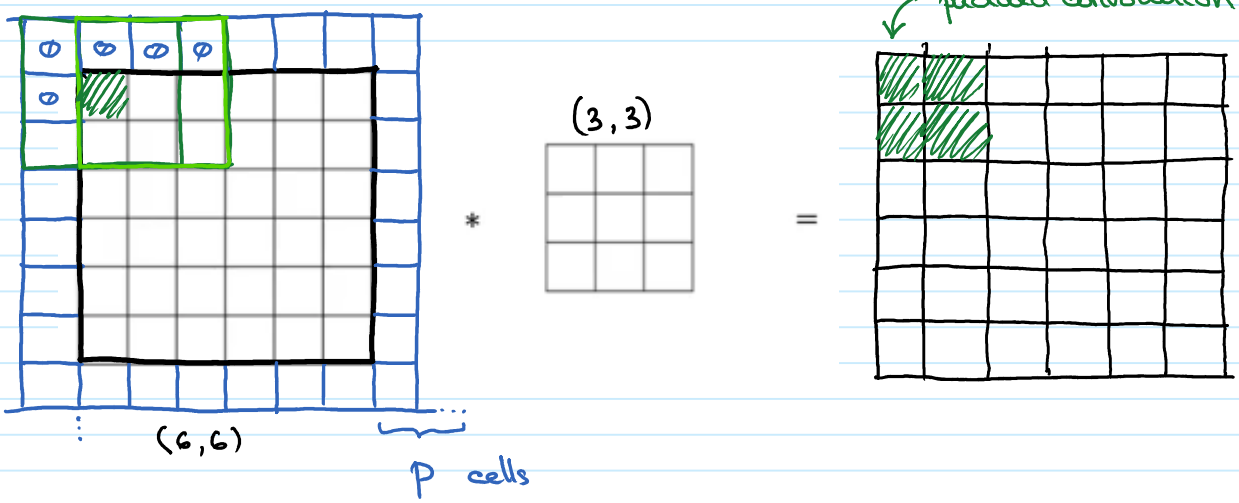
  ↑ pad on both sides of the matrix.

To maintain output image size to same size as input:

$$\underbrace{(n+2p) - f + 1}_{\text{output size of padded convolution}} = \underbrace{n}_{\text{unpadded input size.}} \Rightarrow 2p - f + 1 = 0$$

$$\Rightarrow \boxed{p = \frac{f-1}{2}}$$

- By convention we pad with 0.



Padded size: (8,8) 6×6

We could pad with more than the minimum # of pixel that prevent shrinking

CONVOLUTION TYPES

- VALID (no padding):

  $p = 0$

$n \times n \quad * \quad f \times f \quad \rightarrow \quad n-f+1 \times n-f+1$

$6 \times 6 \quad * \quad 3 \times 3 \quad \rightarrow \quad 4 \times 4$

- SAME: Pad so that input & output have the same size.

$$p = \frac{f-1}{2}$$

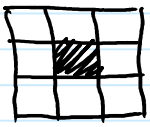
When f is odd p is an integer. Almost everyone uses odd size filters in CV

$3 \times 3 \rightarrow p = 1$

$5 \times 5 \rightarrow p = 2$

There are 2 reasons why we have odd size filters in Computer Vision:

- if p is even we need assymetric padding (i.e pad more on left than right.
- it is nice to have a single pixel be the center of the image. so we can talk about the position of the filter



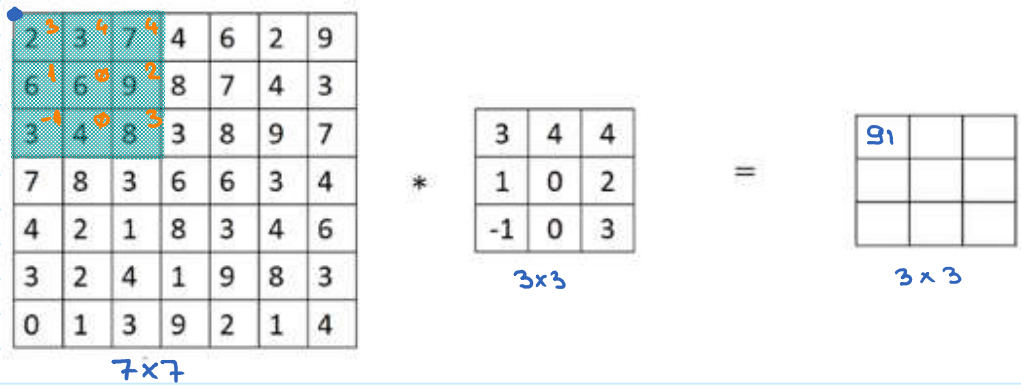
- STRIDED:

→ see next notes.

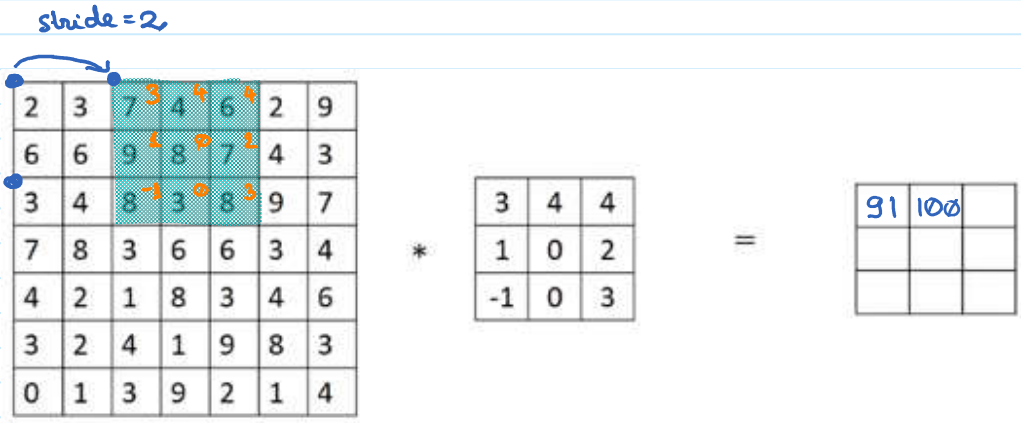
Strided Convolution

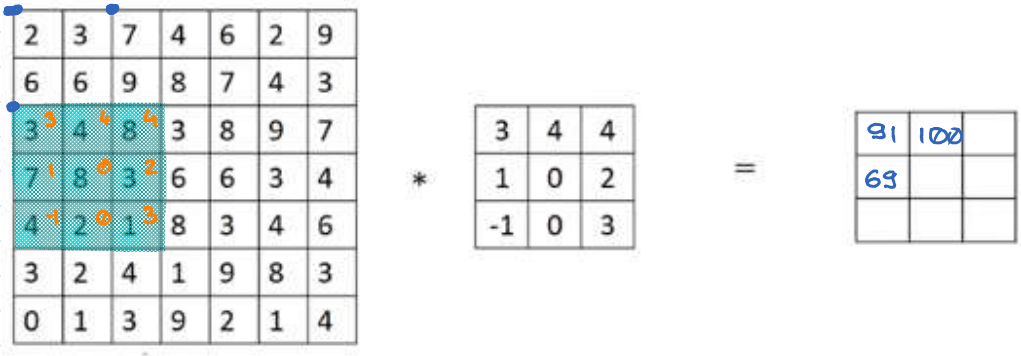
Sunday, 12 August, 2018 17:56

Another method for performing convolution where the filter doesn't move one pixel at a time, instead it "strides" 2 or more pixels at a time.



Screen clipping taken: 2018-08-12 18:05





2	3	7	4	6	2	9
6	6	9	8	7	4	3
3	4	8	3	8	9	7
7	8	3	6	6	3	4
4	2	1	8	3	4	6
3	2	4	1	9	8	3
0	1	3	9	2	1	4

$*$

3	4	4
1	0	2
-1	0	3

$=$

91	100	83
69	91	127
44	72	74

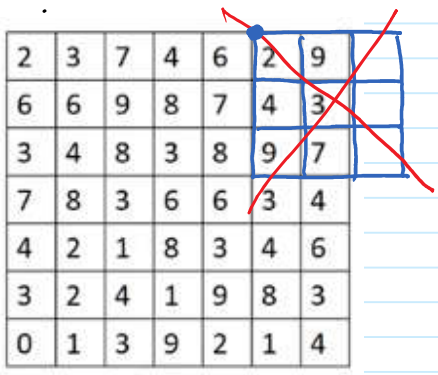
For a $n \times n$ image convolved with an $f \times f$ filter using padding p and a stride of s.

The output image is of size:

$$\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$$ ← floor of

$$\frac{7+0-3}{2} + 1 = 3$$

We take the floor of the fraction if its result is not an integer. This is because by convention the convolution is implemented as not running the computation if the filter falls outside the image with padding after "jumping" s strides.



Screen clipping taken: 2018-08-12 18:22

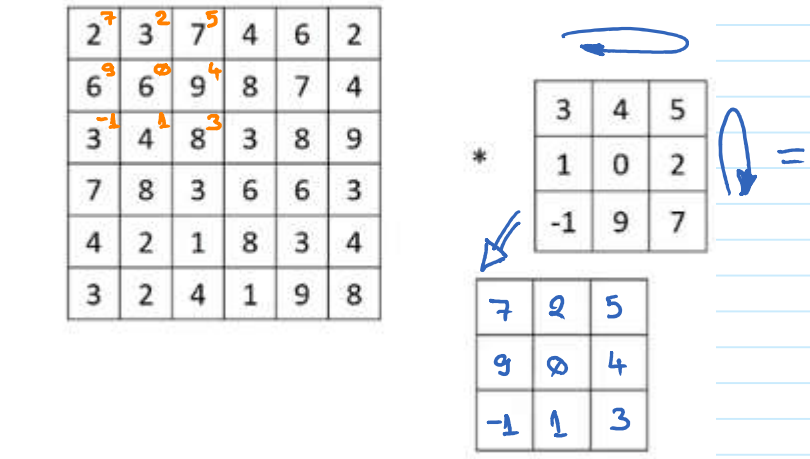
Cross-correlation vs convolution

Sunday, 12 August, 2018 18:24

In Deep-Learning literature by convention we call convolution what other fields (general math, signal-processing) calls cross-correlation.

The original definition of convolution includes an additional step before resuming the multiplication & addition operations where the filter is mirrored both horizontally & vertically.

Convolution in math textbook:



Screen clipping taken: 2018-08-12 18:30

By including the flipping the original convolution enjoys the property of

ASSOCIATIVITY: $(A*B)*C = A*(B*C)$

which is useful for some signal processing problems.

SUMMARY:

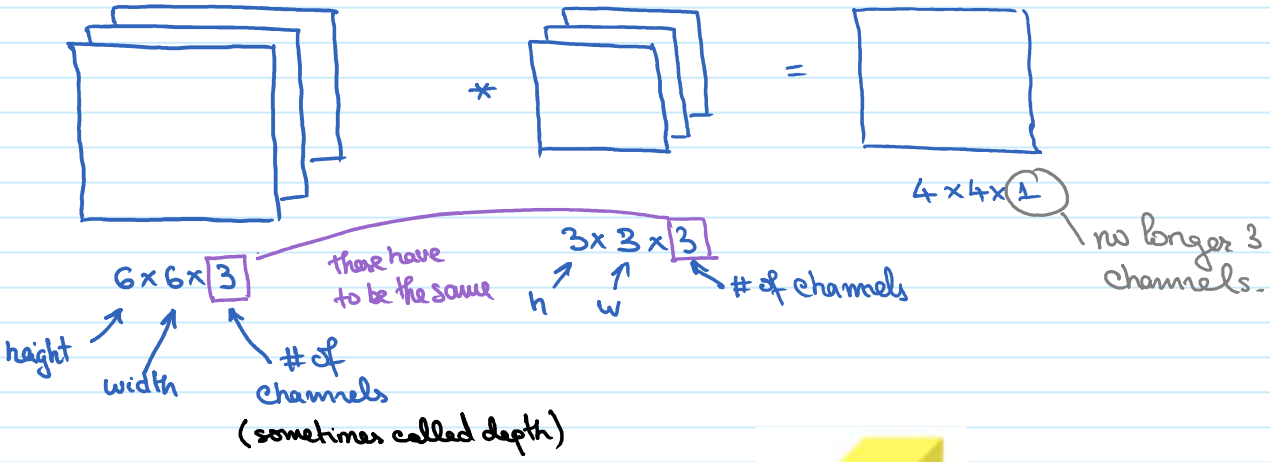
We don't bother with the flipping operation in Deep Learning.

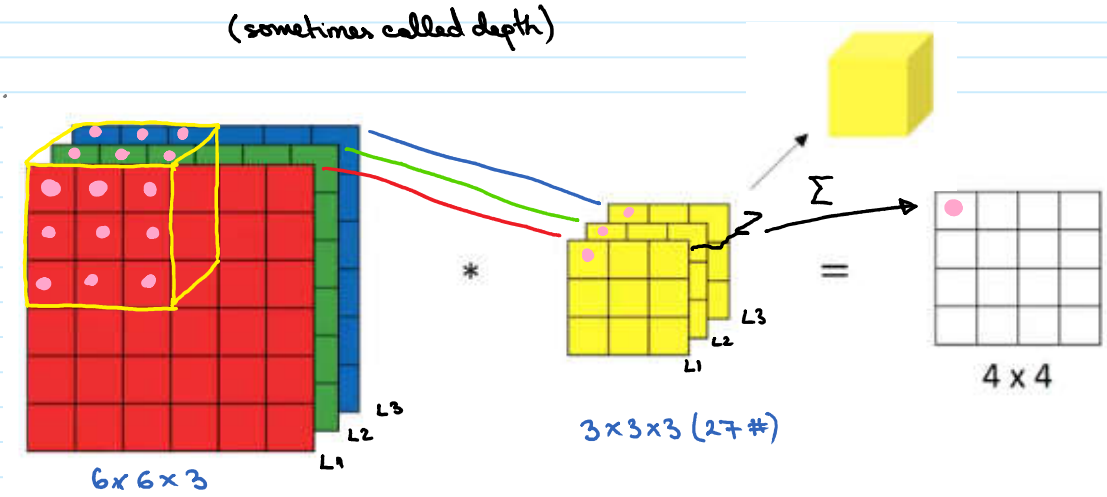
Convolutions over Volumes

Sunday, 12 August, 2018 18:38

Convolutions on RGB images

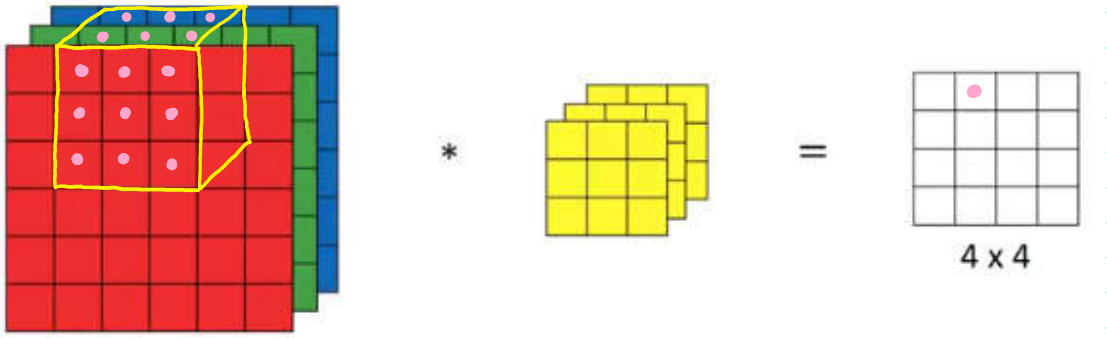
Let's say we want to detect features in an RGB image.

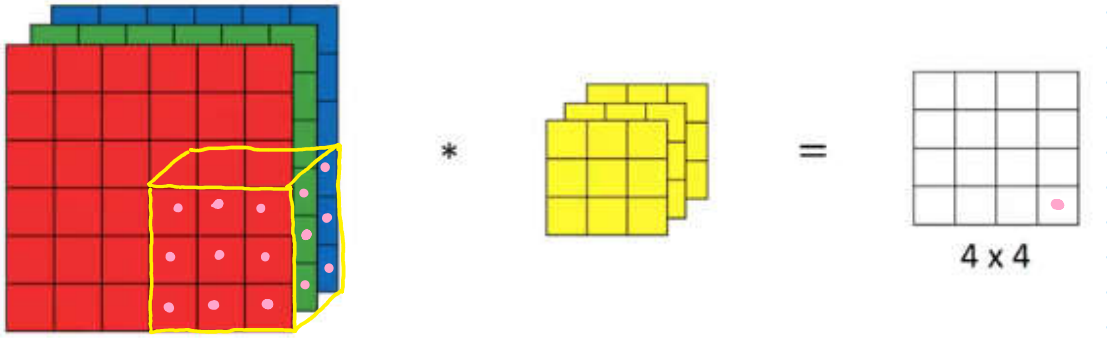




Screen clipping taken: 2018-08-12 18:43

Each layer in the image is multiplied with its equivalent layer in the filter and then all the layer values are added (we are adding 27 products)





What does volume convolution allow us to do?

- Let's say we want to detect edges in the Red channel. We set the filter to be

R:

1	0	-1
1	0	-1
1	0	-1

G:

0	0	0
0	0	0
0	0	0

B:

0	0	0
0	0	0
0	0	0

- if we want edges on all colors:

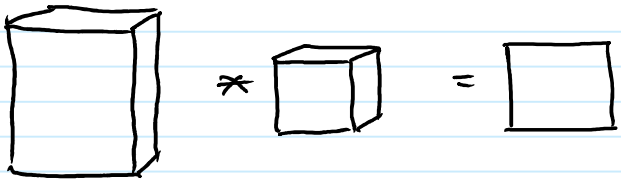
R = G = B =

1	0	-1
1	0	-1
1	0	-1

By convention we include all three channels in the filter (even if some channels are 0) so that it matches the # of channels in the input image.

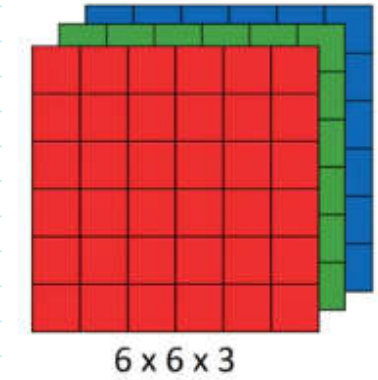
Remember:

3D * 3D = 2D

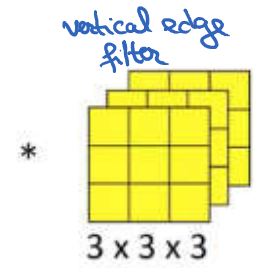


MULTIPLE FILTERS:

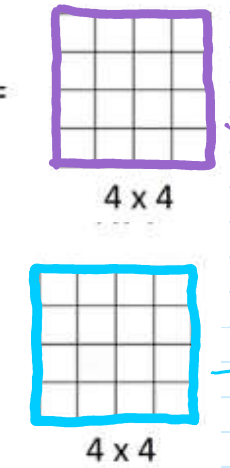
Say we want to apply both a horizontal & vertical edge detection filter.



Screen clipping taken: 2018-08-12 19:06



horizontal edge filter



4 x 4 x 2 output volume resulted from applying 2 different filters (each filter detects a specific feature / pattern)

SUMMARY:

input image		filter		output
$n \times n \times n_c$	*	$f \times f \times n_c$	$\rightarrow$	$n-f+1 \times n-f+1 \times n_f$

n_c: # of channels.

n_f: # of filters used

`# of channels.

'# of filters used

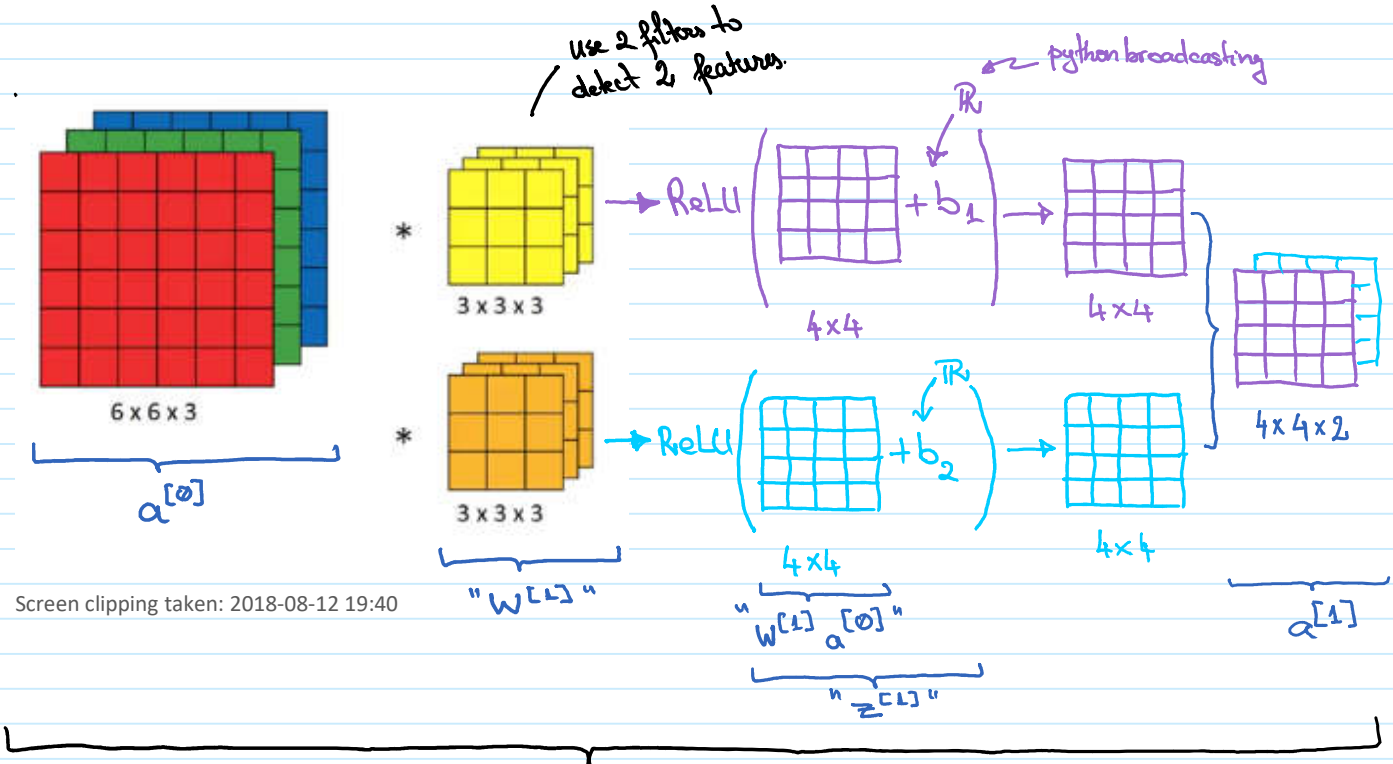
this applies for convolution w/ no padding & stride of 1.

This allows us to:

- operate directly on RGB images.
- ★ we can now apply many filters to detect many features (hundreds of features sometimes). aka patterns

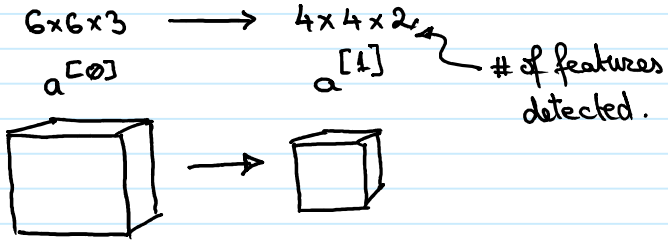
One Convolutional Network Layer

Sunday, 12 August, 2018 19:38



Screen clipping taken: 2018-08-12 19:40

One layer of a convolutional Neural Network.



Forward prop in a standard NN:

$$z^{[1]} = W^{[1]} \overbrace{a^{[0]}}^{x} + b^{[1]}$$

$$a^{[1]} = g^{[1]}(z^{[1]})$$

In a CNN the convolution stands for "W a" (the linear operation)

We then add biases

Apply non-linearity (ReLU)

Combine output features.

OF PARAMETERS IN ONE LAYER EXERCISE

If you have 10 filters that are 3 x 3 x 3 in one layer of a neural network, how many parameters does that layer have?

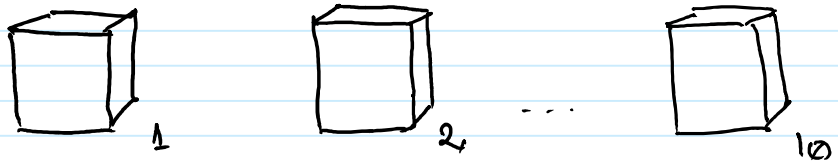
Screen clipping taken: 2018-08-12 20:05

Each filter is equivalent to W matrix so we have $3 \times 3 \times 3 = 27$ w parameters.

For each filter we apply a single bias value (broadcast 27x)

→ one bias per detected feature.

So we have 27 w + 1b = 28 parameters per filter.



$3 \times 3 \times 3 = 27$ w

1b

28 params

Total $28 \times 10 = 280$ parameters.

COOL OBSERVATION:

No matter how large the input feature set is (1000 x 1000 or 5k x 5k) the # of parameters stays constant @ 280 (determined by # of filters we want to use).

So even if we have very large images we won't overfit the data.

SUMMARY OF NOTATION:

For a convolutional layer l

$f^{[l]}$ = filter size $(f^{[l]} \times f^{[l]})$

$p^{[l]}$ = padding (valid or same)

$s^{[l]}$ = stride.

$n_c^{[l]}$ = number of filters

Each filter is: $f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$

Activations: $a^{[l]} \rightarrow n_H^{[l]} \times n_w^{[l]} \times n_c^{[l]}$

$A \rightarrow m \times n_H^{[l]} \times n_w^{[l]} \times n_c^{[l]}$

m: # of examples when applying vectorization

Weights: $f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]}$

$n_c^{[l]}$: # of filters in layer l

Bias: $n_c^{[l]}$ - we'll see that it is more convenient to represent this as a $(1, 1, 1, n_c^{[l]})$

Input: $n_H^{[l-1]} \times n_w^{[l-1]} \times n_c^{[l-1]}$

n_H: height, n_w: width

Output: $n_H^{[l]} \times n_w^{[l]} \times n_c^{[l]}$

$$n_{H/w}^{[l]} = \left\lfloor \frac{n_{H/w}^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor$$

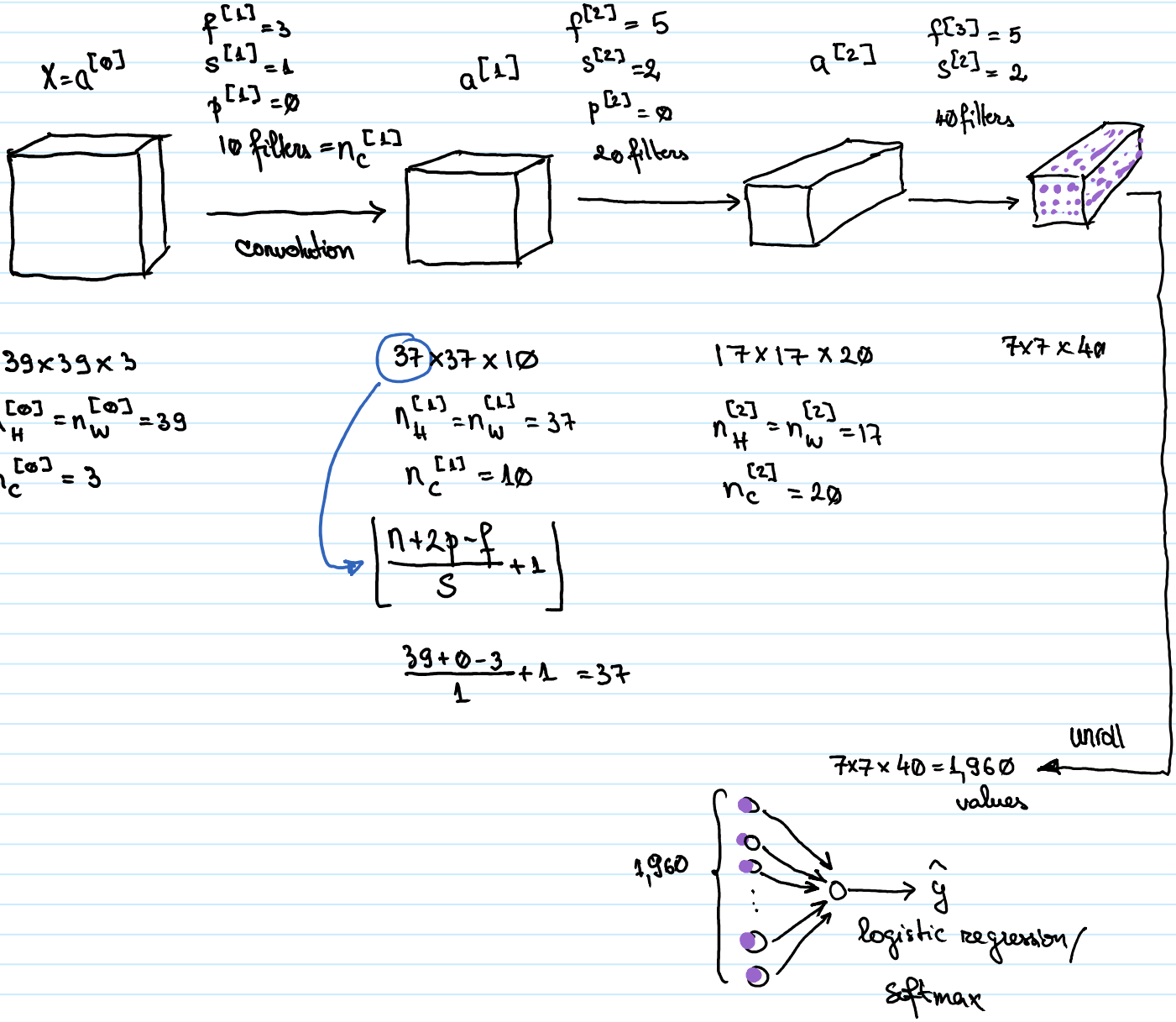
Simple Convolutional Network

Sunday, 12 August, 2018 20:35

We want to do an image classification project (i.e. is it a cat → 0 or 1)

The input images are going to be small $39 \times 39 \times 3$

And we use 10 filters to detect 10 features



$$\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$$

$$\frac{39+0-3}{1} + 1 = 37$$

A lot of the turning of Conv Nets consists of selecting hyper-parameters like filter size, stride, padding, how many filters to use.

As we go deeper in the Conv Net we typically decrease image size while increasing channel size.

$$39 \times 39 \times 3 \rightarrow 37 \times 37 \times 10 \rightarrow 17 \times 17 \times 20 \rightarrow 7 \times 7 \times 40$$

TYPES OF LAYERS IN A CONVOLUTIONAL NET:

- Convolutional (Conv)
- Pooling (Pool)
- Fully connected (FC)

Most of the layers are Conv with a few Pool / FC.

Pooling Layers

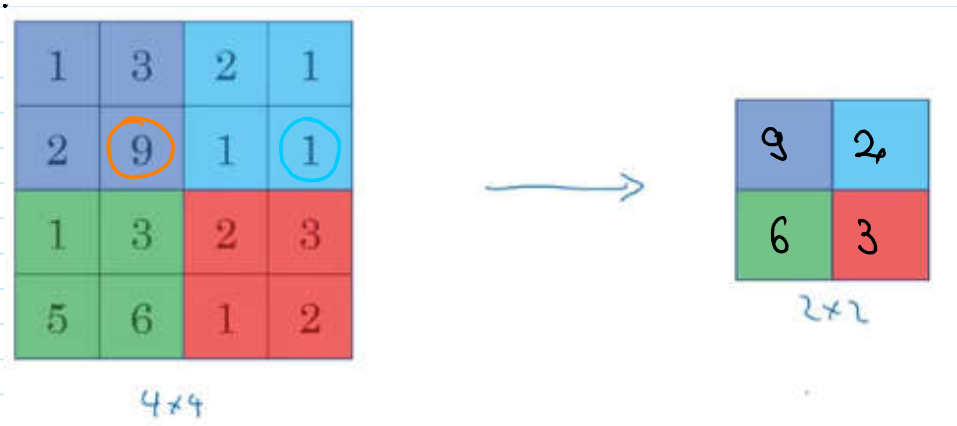
Sunday, 12 August, 2018 21:03

Raison d'être:

- Reduce the size of the representation
- Speed up computation
- Improve feature robustness.

EXAMPLE OF POOLING

We apply a Max Pooling algorithm:



Screen clipping taken: 2018-08-12 21:07

Take a max over each 2×2 region stepping every 2 steps:

$f = 2$
$s = 2$
} hyperparameters for Pooling

Interestingly there are no parameters to optimize in a POOLING layer.
→ we only have hyperparameters.

INTUITION:

We can think of each 4×4 values as indicative of the presence of some feature

(i.e. the higher the value the more "prominent the feature is present) - maybe a vertical edge or maybe an eye.

A high value indicates the feature is in that region

While a low value indicates the feature is not in that region.

So the max pooling appears to "preserve" the location of the prominent features while compressing the feature set

This understanding is not very well supported by additional theories

Though it is <u>STRONGLY</u> supported by empirical studies.

ANOTHER MAX POOLING EXAMPLE:

5x5

1	3	2	1	3
2	9	1	1	5
1	3	2	3	2
8	3	5	1	0
5	6	1	2	9

→ $f=3$, $s=1$

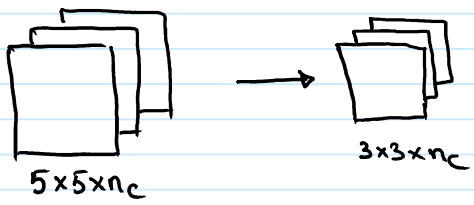
3x3

9	9	5
9	9	5
8	6	9

Screen clipping taken: 2018-08-12 21:27

Same $\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$ output size dimension as previously used apply here

For 3D Pooling the output is resolved for each "channel" independently.

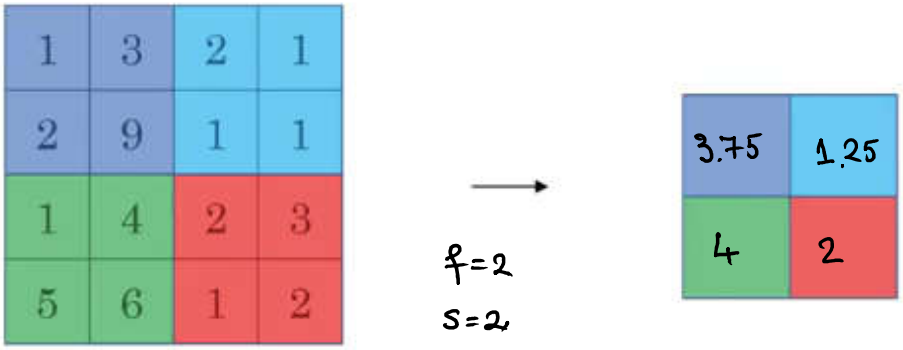


AVERAGE POOLING

Not used very much

→ exception sometimes very deep in a NN we collapse the representation from 7x7x1000 to 1x1x1000 using Average Pooling.

Instead of taking the max it takes the average of a region.



Screen clipping taken: 2018-08-12 21:34

SUMMARY OF POOLING

- Common choices of hyperparameters:

  $f=2$, $s=2$ } Shrinks representation by x 2x

  We don't normally use padding

- We can use either Max or Avg padding.

- Size of i/o

  Input: $n_H \times n_W \times n_C$

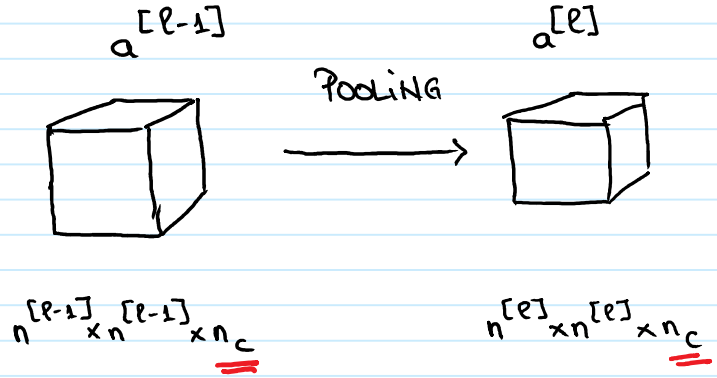
  ↓

  Output: $\left\lfloor \frac{n_H - f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n_W - f}{s} + 1 \right\rfloor \times n_C$

- No parameters to train.

POOLING BACKPROP NOTES (Mi):

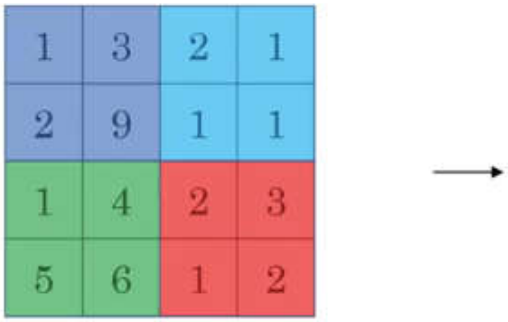
- Forward prop:

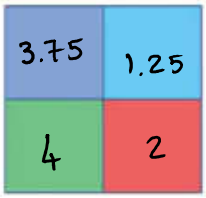


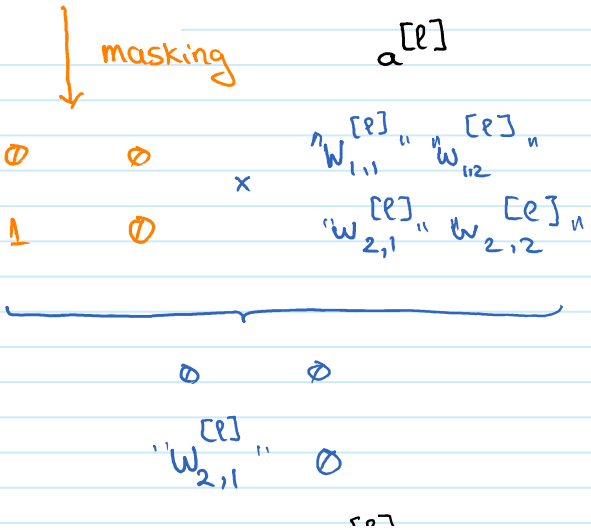
- Backprop:

Looking @ a single n_c layer.

$a^{[l-1]}$







for each entry in $a^{[x]}$ there are (f,f) entries in $a^{[l-1]}$.

CNN Examples

Sunday, 12 August, 2018 22:10

Say we are trying to do hand-written digit recognition:

→ inspired by Yan Lecun's LeNet 5.

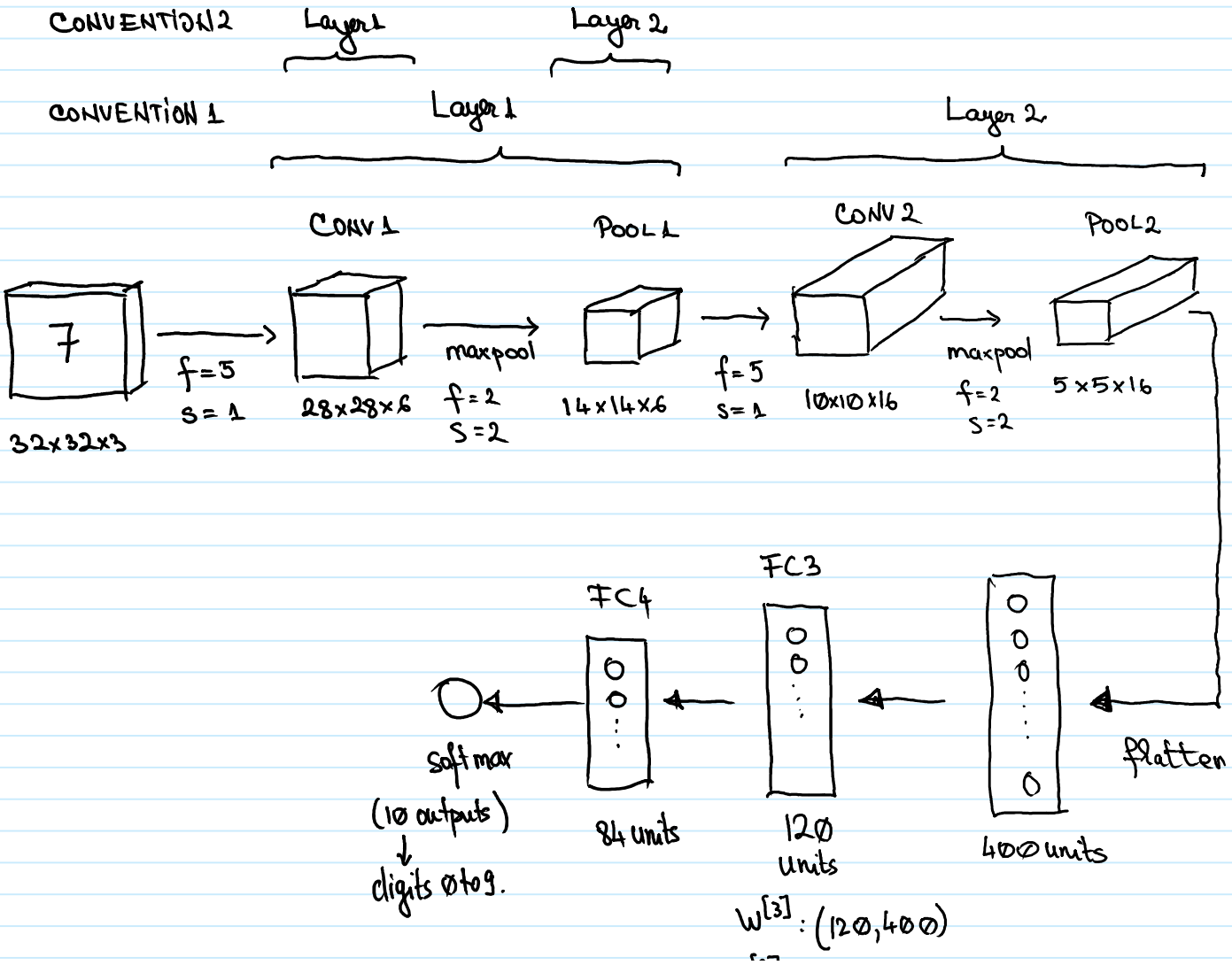
There are 2 conventions as to what to call a layer in a ConvNet.

→ people usually report only the layers that have parameters.

As the pooling layer has no parameters (only hyperparameters) it is not normally counted as a layer.

So we will use convention 1 from here on.

We combine the Conv layer & the Pooling layer.



$b^{[3]}: (120, 1)$

GENERAL TRENDS

- One common guideline for choosing a hyperparameters is to use what others have used.
- Another common pattern is to start with Conv + Pool layers followed by fully connected layers, followed by a softmax layer.

CONV – POOL – CONV – POOL – FC – FC – FC – SOFTMAX

- n_H & n_W ↓ and n_C ↑ as we go deeper in the network.

	Activation shape	Activation Size	# parameters
Input:	(32,32,3)	3,072 $a^{[0]}$	0
CONV 1 (f=5, S=1)	(28,28,8)	6,272	208
POOL 1	(14,14,8)	1,568	0 ←
CONV 2 (f=5,S=1)	(10,10,16)	1,600	416
POOL 2	(5,5,16)	400	0 ←
FC3	(120,1)	120	48,001
FC4	(84,1)	84	10,081
Softmax	(10,1)	10	841

Screen clipping taken: 2018-08-12 22:33

Max Pooling layers have no parameters

Conv layers have few parameters.

Most of the parameters are in the fully connected layers

Activation size drops slowly as we progress deeper in the Conv Net.

Reviewing examples of architectures others have used is the best way to learn

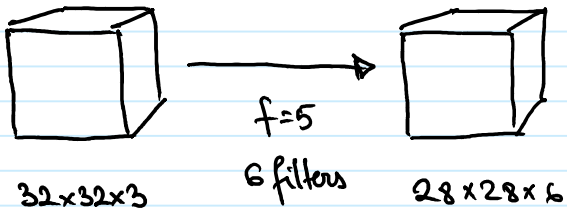
how to design a ConvNet

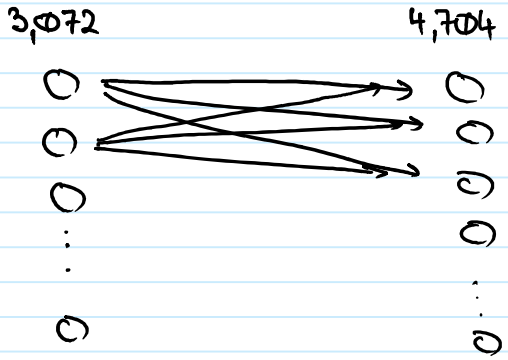
Why use convolutions?

Sunday, 12 August, 2018 22:42

ADVANTAGES OVER FULLY CONNECTED NN:

- parameter sharing
- sparcity of connections.





If we were to fully connect it we would have 3,072 x 4,704 = 14M parameters to train. This is a lot of parameters to train for such a small network.

If we use the CONV Net we need

5x5 = 25 + 1 = 26 parameters per filter

6x26 = 156 parameters in total for the conv net.

This small # of parameters is due to (in part) parameter sharing.

PARAMETER SHARING:

A feature detector (i.e. a vertical edge detector) that is used in one part of

the image is probably used in other parts of the image also.

→ so we don't need to duplicate "feature detectors" locally.

This is true for both low level & high level features.

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

*

1	0	-1
1	0	-1
1	0	-1

3x3

=

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

Screen clipping taken: 2018-08-12 22:52

SPARCITY OF CONNECTIONS:

In each layer, each output value depends only on a small number of neighbouring inputs

→ this assumes features are locally correlated.

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

*

1	0	-1
1	0	-1
1	0	-1

=

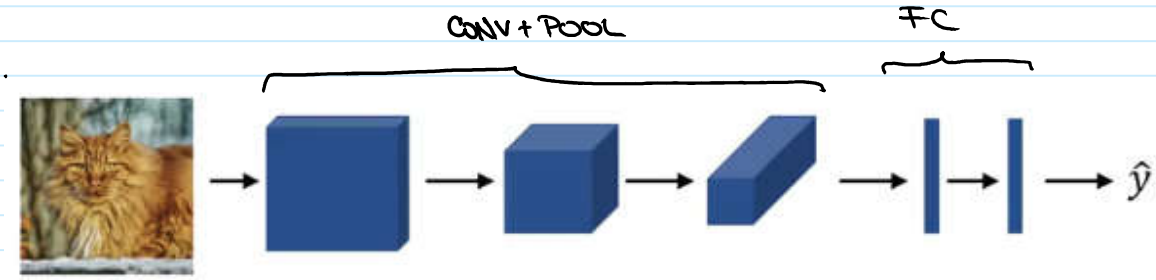
0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

ConvNets are good at translational invariance

→ if a picture of a cat is shifted a few pixels left it is still a cat.

The learned convolutional structure helps encode that the same label should be applied to a shifted image.

PUTTING IT ALL TOGETHER



Screen clipping taken: 2018-08-12 22:59

Cost $J = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$ can be minimized by starting from a randomly initialized parameters W & b.

Use gradient descent (or RMS prop or ADAM) to minimize the loss function.

Week 2 Case Studies

Tuesday, 14 August, 2018 14:20

Why case studies?

Tuesday, 14 August, 2018 14:23

CV research has spent a lot of time on how to put together the building blocks discussed previously (ConvNets, Pooling, Fully Connected layers).

Features of Successful Architectures can be used in new architectures of CV

Classic Networks

Tuesday, 14 August, 2018 19:12

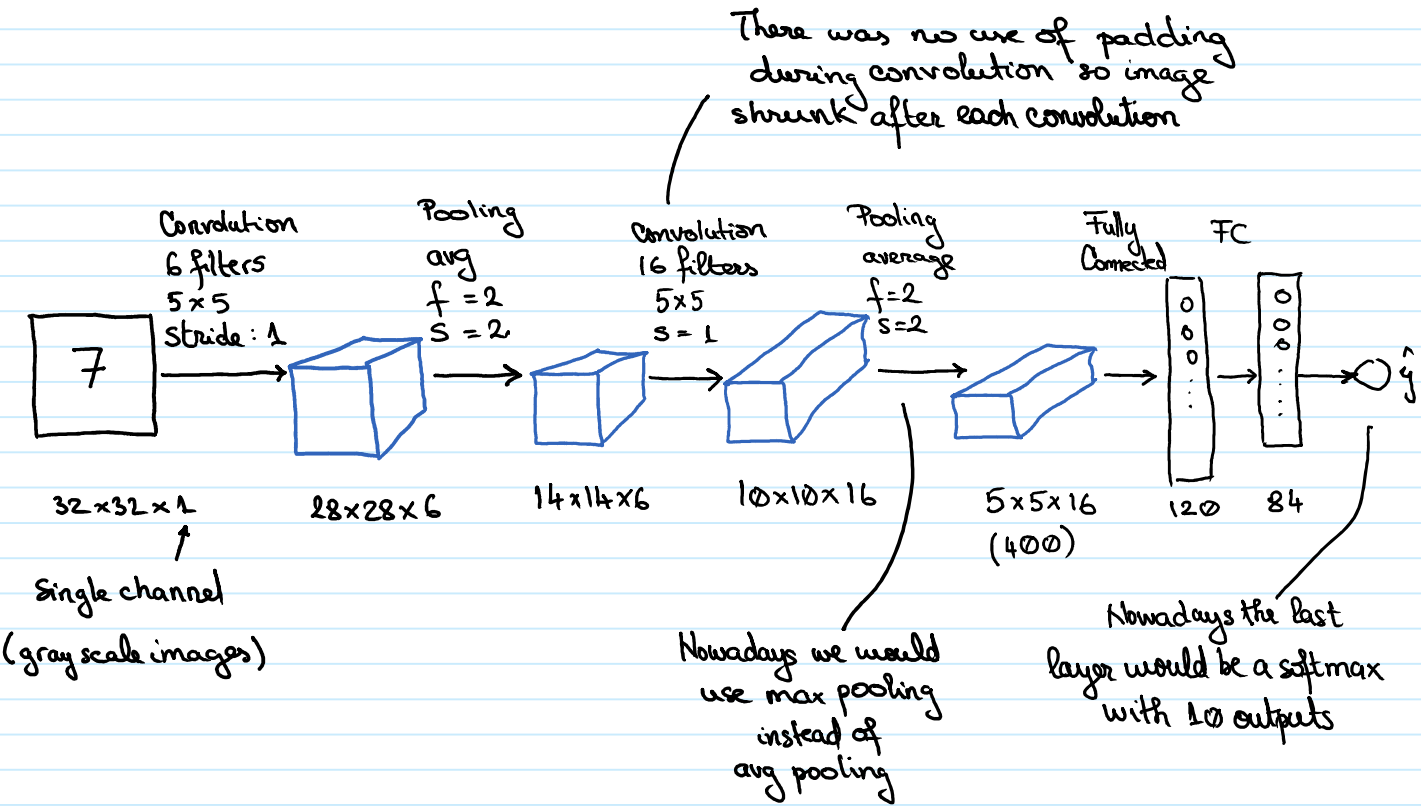
Goal: Learn about some classic Deep Learning Computer Vision classifiers like: LeNet 5, AlexNet & VGG-16

LENET 5

[LeCun et al., 1998. Gradient-based learning applied to document recognition]

Screen clipping taken: 2018-08-14 19:17

GOAL: Recognize hand-written digits



- Original network implementation used 60K parameters (small)
  → today we often see 10M to 100M parameters networks.

- $n_H, n_W \downarrow$, $n_C \uparrow$ } as we go deeper into the network

- We notice a pattern of layers that is still common today:

  CONV → POOL → CONV → POOL → FC → FC → output

ADVANCED NOTES

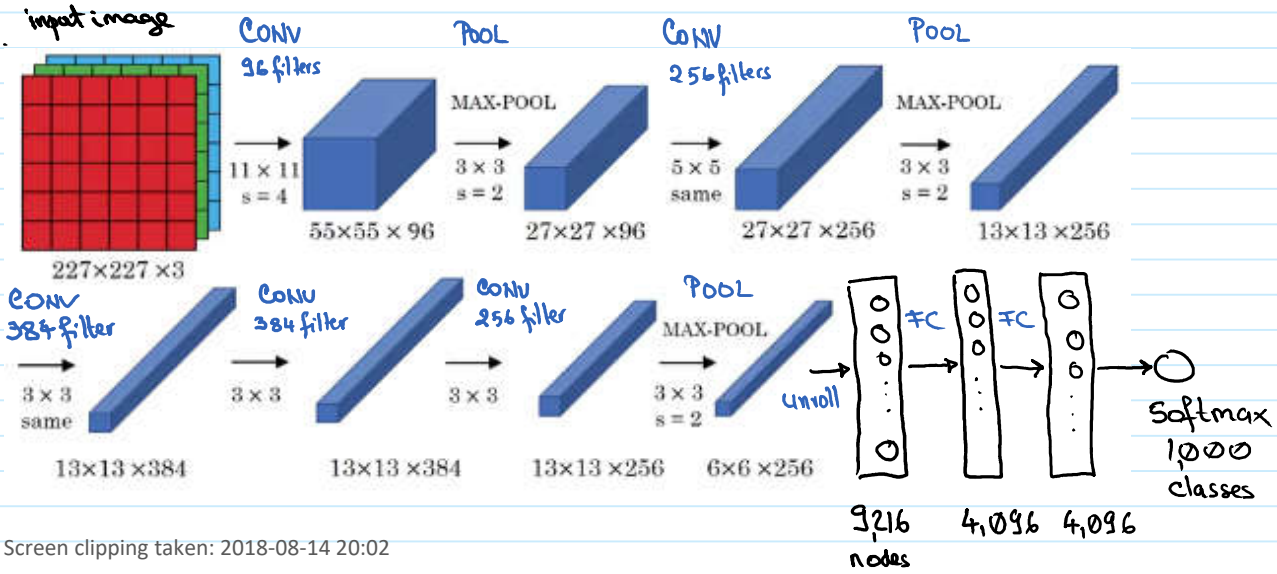
- Sigmoid & tanh were used instead of RELU.
- To save computation time the filters used by LeNet 5 would only look at a limited # of channels from the input.
  So the n_C in the normal: $n_H \times n_W \times n_C \rightarrow f \times f \times n_C$
  were not the same value b/w input and filters
- A non-linearity was added after pooling.

Focus on reading section 2 & 3.

ALEX NET

[Krizhevsky et al., 2012. ImageNet classification with deep convolutional neural networks]

Screen clipping taken: 2018-08-14 19:58



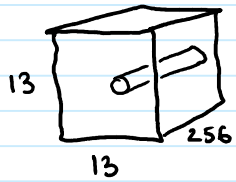
Screen clipping taken: 2018-08-14 20:02

- AlexNet was similar to LeNet 5 but much larger (60M parameters instead of 60K)

- It was also using ReLU activation which improved performance.

ADVANCED NOTES:

- Paper states they used 224x224x3 input images but calculations use 227x227x3
- B/C GPUs were not very fast back then they used a complicated way to train across two GPUs (split the training of the layers to be run on 2 GPUs)
- They used an ~~obsolete~~ type of layer called Local Response Normalizations.



Look across all 256 values of a volume and normalize them
→ for each position of (13x13) we might not want many neurons with high activation
→ this doesn't help much

- This paper convinced most in the CV to take a serious look at Deep Learning.

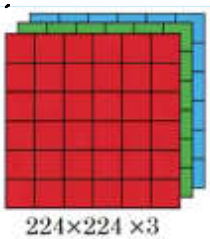
VGG-16

[Simonyan & Zisserman 2015. Very deep convolutional networks for large-scale image recognition]

Screen clipping taken: 2018-08-14 20:16

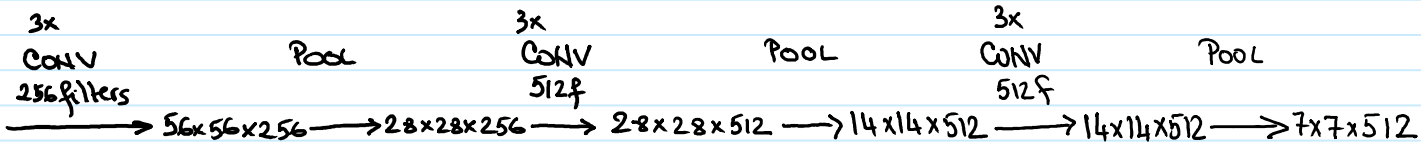
Approach was meant to simplify the large NN while maintaining performance.
→ decrease the # of hyperparameters.

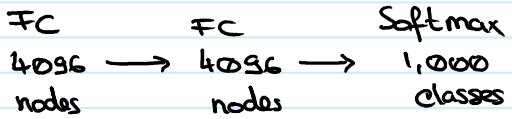
- All CONV layers used a 3x3 filter, stride = 1, same
  MAX-POOL layers used were 2x2 w/ stride = 2



2x CONV 64 filters → 224x224x64 → POOL → 112x112x64 → 2x CONV 128 filters → 112x112x128 → POOL → 56x56x128

Screen clipping taken: 2018-08-14 20:22





- The 16 in VGG-16 indicates there are 16 layers in the network
- Network has 138M parameters (large by today's standard)
- Network has a very regular structure
  - → double number of features after each CONV layers while halfing image size
  - $n_H, n_W \downarrow$ & $n_c \uparrow$
- There's also a VGG-19 (performs about the same).

Residual Networks

Tuesday, 14 August, 2018 20:34

[He et al., 2015. Deep residual networks for image recognition]

Screen clipping taken: 2018-08-14 20:41

GOAL: Learn about SKIP CONNECTIONS

→ taking the activation from one layer & feeding it to another layer much deeper in the network

→ this results in Res Nets (Residual Networks) being built.

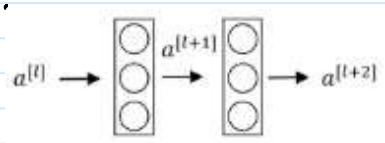
MOTIVATION:

Very, very deep NN are difficult to train because vanishing/exploding gradient problems. Skip Connections help address it.

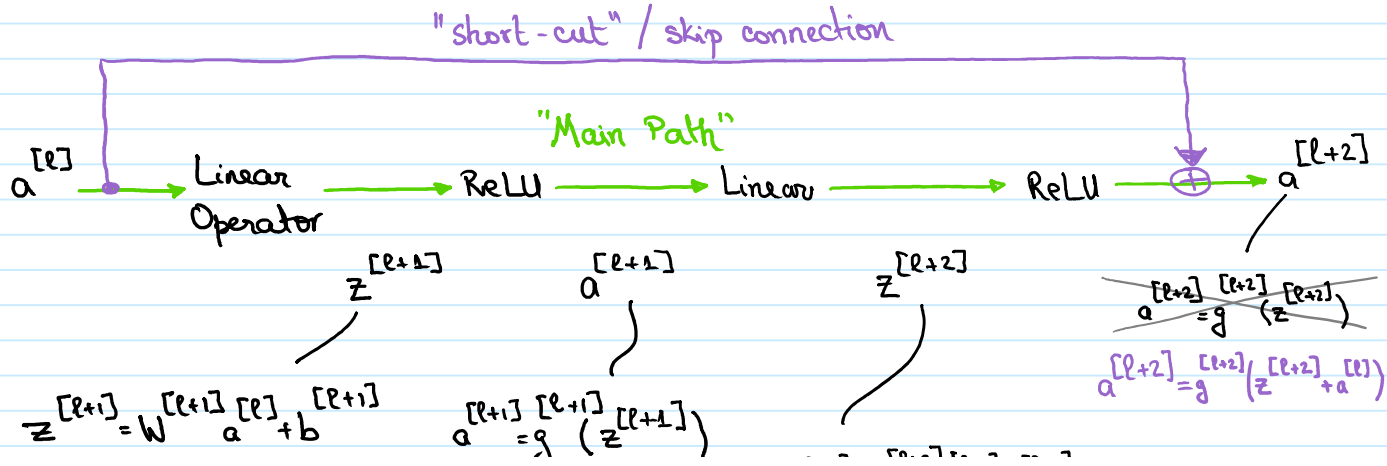
→ we can build NN with up to 1000 layers using this technique.

RESIDUAL BLOCK:

Res Nets are built using Residual Blocks.

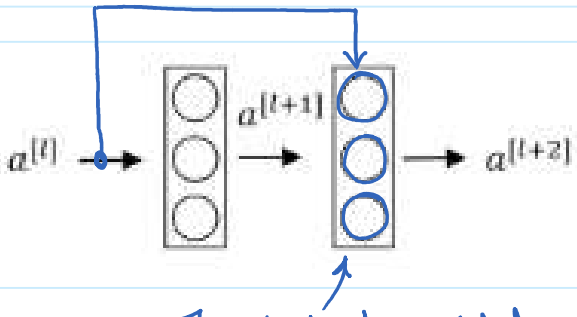


Screen clipping taken: 2018-08-14 20:42



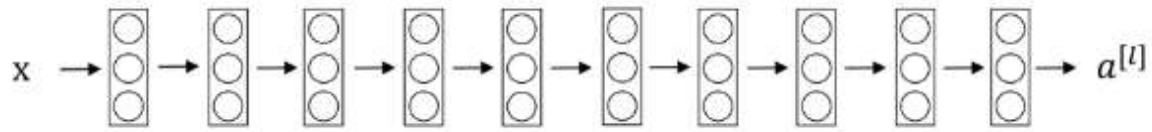
$z^{[l+2]} = W^{[l+2]} a^{[l+1]} + b^{[l+2]}$

By using the "short-cut" we are injecting information from $a^{[l]}$ into $l+2$
→ the addition of $a^{[l]}$ into $l+2$ makes layer l to $l+2$ a residual block.



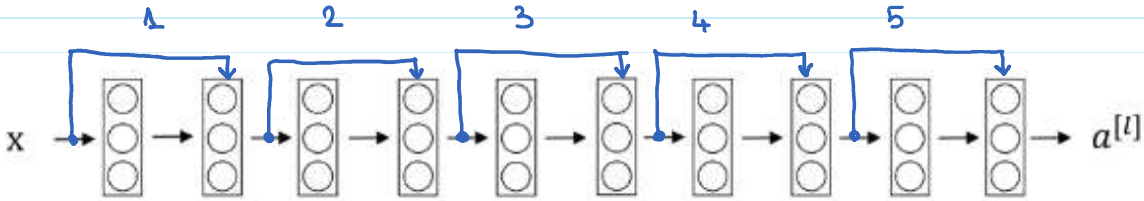
The shortcut is added before applying the ReLU (i.e. getting $a^{[l+2]}$)

PLAIN NETWORK

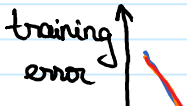


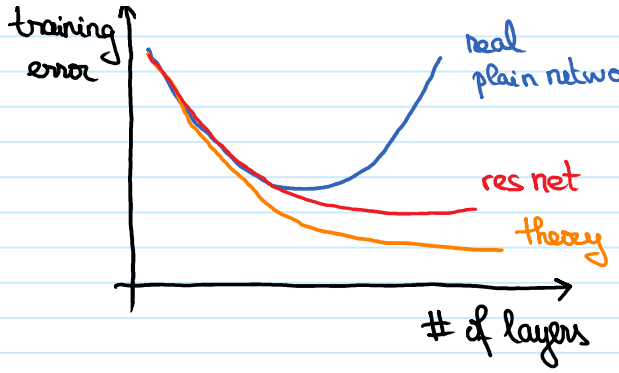
Screen clipping taken: 2018-08-14 21:10

RESIDUAL NETWORK (w/ 5 Residual Blocks)



When using a standard training algorithm the training error starts rising b/c of vanishing / exploding gradient problems.





problems.

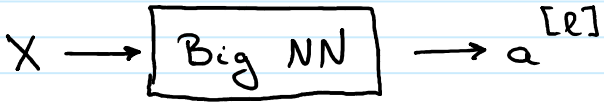
How ResNets work?

Wednesday, 15 August, 2018 13:00

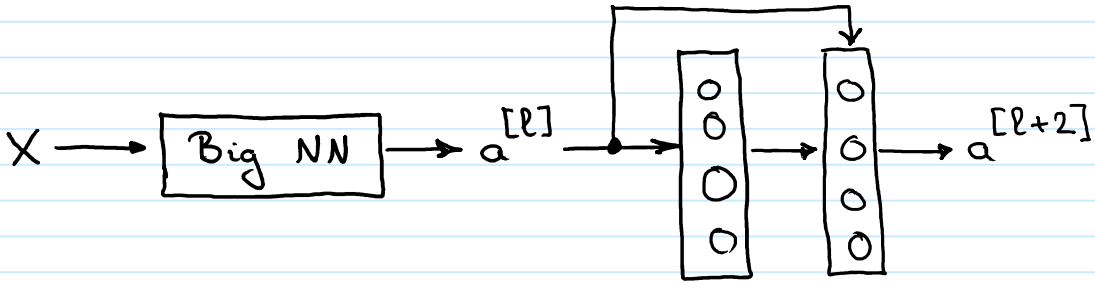
Goal: show how the network performance does not degrade as its depth increases when using Res Nets

EXAMPLE

Given a large NN



We develop it by adding a residual block



- assume all activation functions are ReLU activations

  $\Rightarrow$ all activations $a^{[\,]} \geq 0$ because ReLU outputs ≥ 0

$$a^{[l+2]} = g\left(z^{[l+2]} + a^{[l]}\right)$$

$$= g\left(w^{[l+2]} a^{[l+1]} + b^{[l+2]} + a^{[l]}\right)$$

if we were to use L2 Regularization (weight decay)
that would tend to shrink the values of $w^{[l+2]}$
(also shrink $b^{[l+2]}$) $\Rightarrow$ $w^{[l+2]} \to 0$ & $b^{[l+2]} \to 0$

So when applying L2 Regularization $a^{[l+2]}$ becomes.

$$a^{[l+2]} = g(W^{[l+2]} a^{[l+1]} + b^{[l+2]} + a^{[l]})$$

$$= g(a^{[l]}) = \text{ReLU}(a^{[l]}) \quad a^{[l]} \geq 0 \Rightarrow g(a^{[l]}) = a^{[l]}$$

$$a^{[l+2]} = a^{[l]}$$

This shows that the identity function is easy for the Residual Block to learn.

→ this means that adding the Residual Block does not degrade the performance of the NN ($a^{[l]}$ can be easily "copied" to $a^{[l+2]}$ when L2 regularization is used which makes the Regularization Block "transparent" to deeper layers)

But our goal is to improve performance (not merely keeping it from degrading).

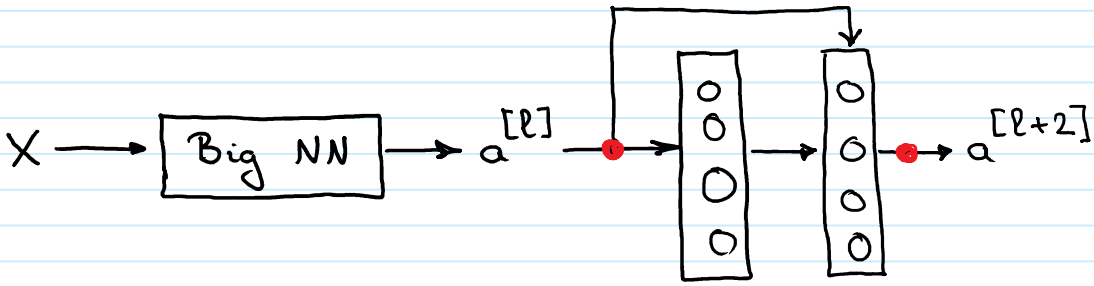
- These Res Blocks not only can easily learn the identity function (not degrading performance) but if we are lucky they can also learn other more complex functions deep in the NN
  → this improves performance (it's easier for Gradient Descent to use these functions to improve performance)

- Very deep plain NN (without Res Blocks) are having a very difficult time learning parameters that output even something as simple as the identity function
  → this makes the output worse

OBSERVATION:

When we calculate $a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$ we are assuming $z^{[l+2]}$ & $a^{[l]}$ have the same dimensions.

→ this imposes the use of SAME CONVOLUTION in Res Nets



use "same" convolution.

If $a^{[l+2]}$ does not have the same dimensions as $a^{[l]}$ we use a helping operation (multiply $a^{[l]}$ by W_s) to make the dimensions the same

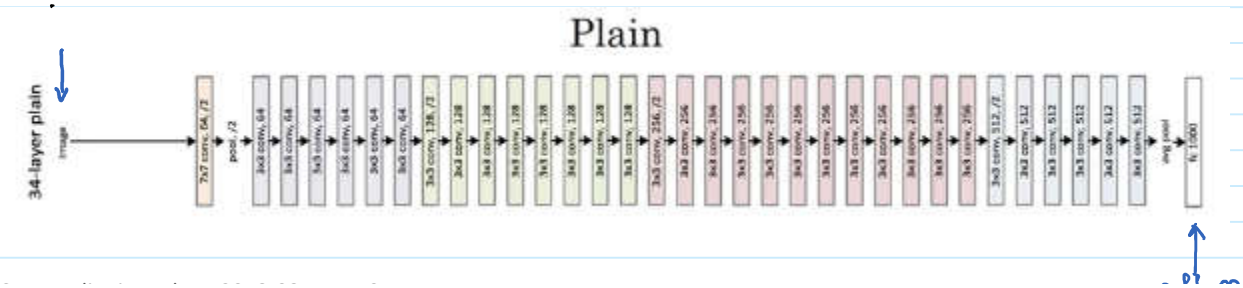
$$a^{[l+2]} = g(z^{[l+2]} + W_s a^{[l]})$$

256 ; $W_s \in \mathbb{R}^{256 \times 128}$; 128

W_s could be:

- a matrix where the parameters are learned
- fix matrix that just $\emptyset$ pads $a^{[l]}$ to make it 256 dimensional.

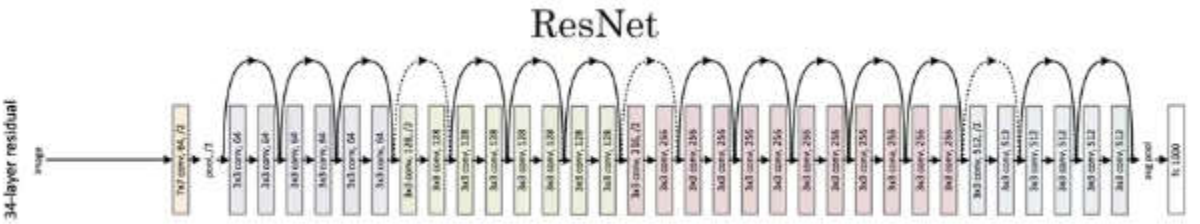
RES NET PAPER:



Screen clipping taken: 2018-08-15 14:07

soft max output

Add skip connections to get:



Screen clipping taken: 2018-08-15 14:07

3×3 same Convolution (allows $z^{[l+2]} + a^{[l]}$ to work)

pooling layer after which we need to adjust the dimension using W_s.

1x1 convolution

Wednesday, 15 August, 2018 15:42

Goal : Understand what a 1×1 convolution does.

→ it is not just multiplying by one number.

[Lin et al., 2013. Network in network]

Screen clipping taken: 2018-08-15 23:27

Processing an input through a 1×1 convolution layer when we have a single channel input doesn't change the shape of the input. It

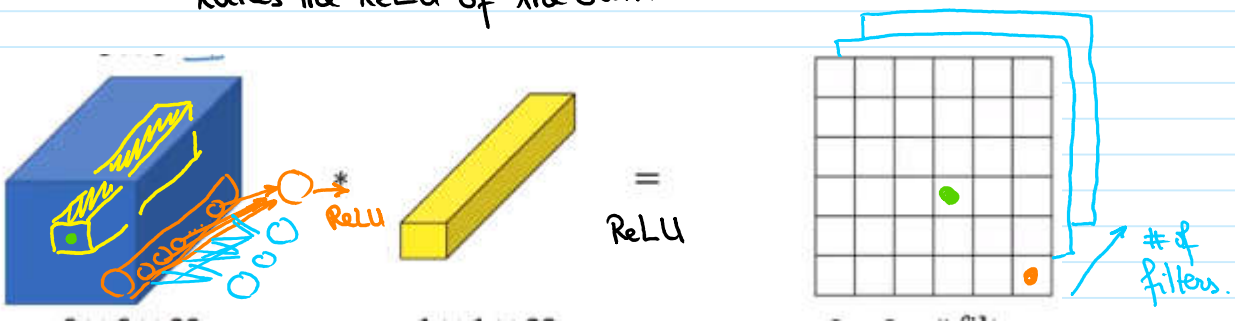
- multiply the values by the 1×1 value
- add non-linearity by passing the product through ReLU

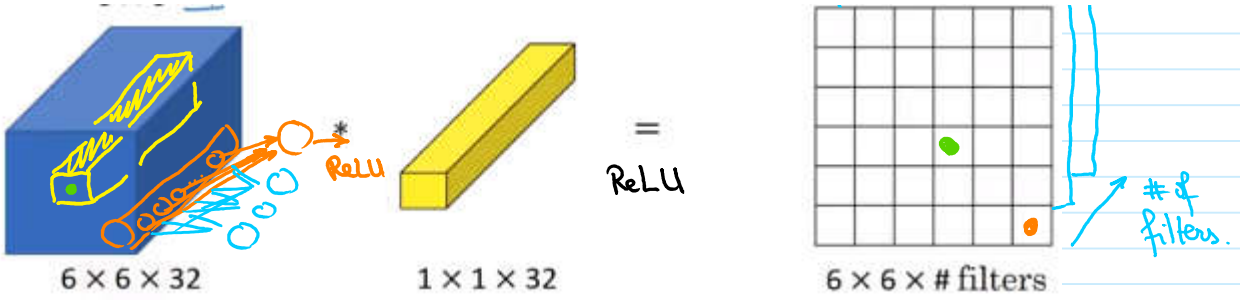
The addition of the ReLU allows the network to learn a more complex function by adding non-linearity

Screen clipping taken: 2018-08-15 23:27

However when the input is multi-channel the 1×1 convolution

- multiplies all the values in the channels
- sums up the channels (flattens the result accross channels)
- takes the ReLU of the sum





Screen clipping taken: 2018-08-15 23:29

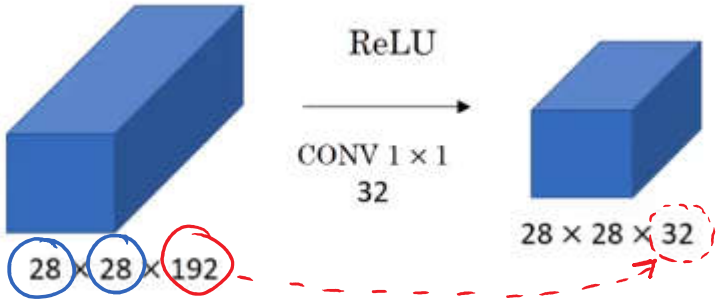
We can think of one of the 1×1×32 convolution filters as if it was one neuron taking as inputs a "slice" of channels (32 values), multiplying them by the same weight then suming the products and taking the ReLU of the sum.

→ multiple 1×1 filters correspond to multiple "neurons"

1×1 convolution is also known as Network in Network.

APPLICATION OF 1×1 CONVOLUTIONS

Using Pooling we can decrease the height (n_H) & width (n_W) of the activation volume



Screen clipping taken: 2018-08-16 00:29

- We use 1×1 convolution to shrink the number of channels (n_c)

  → we use 32 filters of 1×1×192 filters

  this allows us to save on compute resources.

- We can also increase the number of channels in the volumes by adding more 1×1 filters.

Inception networks motivation

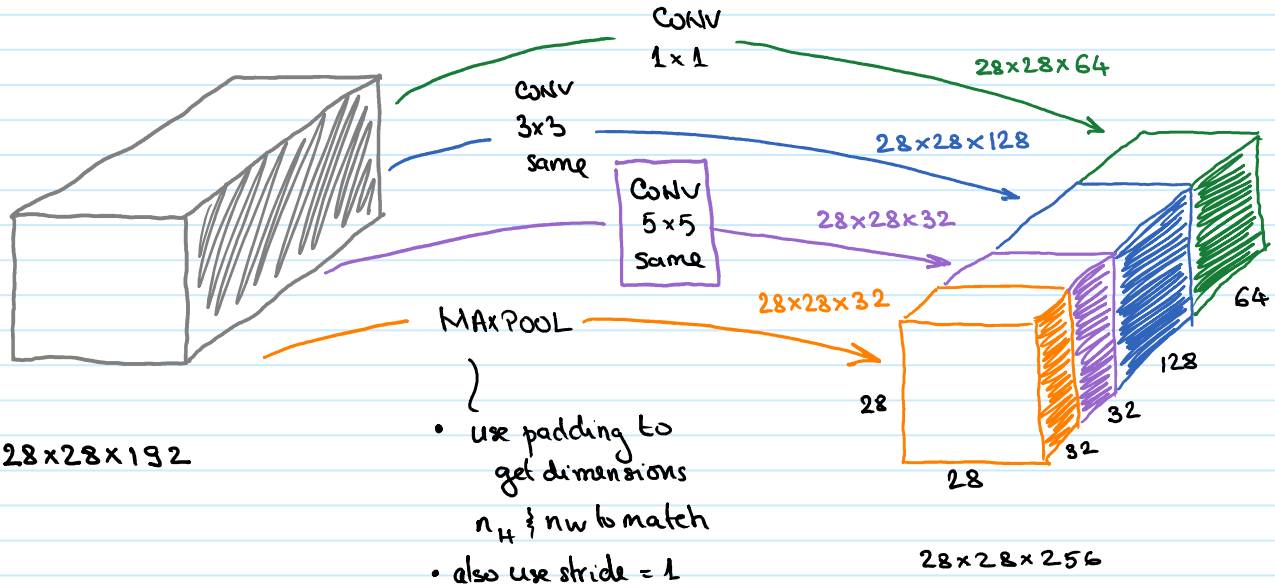
Thursday, 16 August, 2018 11:32

Goal: Understand why we would want to use inception networks

[Szegedy et al. 2014. Going deeper with convolutions]

Screen clipping taken: 2018-08-16 12:16

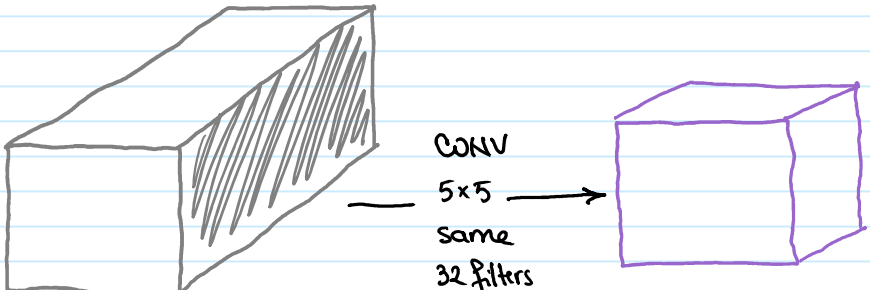
An inception layers combines convolution & pooling layers into one output that concatenates ("stack-up") the individual operation results.

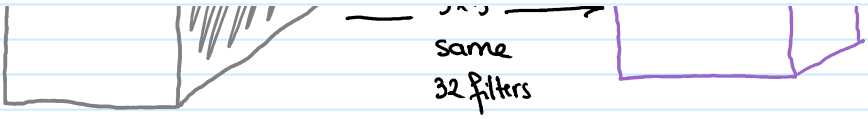


So instead of us picking what are the filter sizes (CONV layers) or what is the pooling size we can do all of them at once and let the network learn which one works best.

MAJOR ISSUE

Computational cost: we'll only focus on the CONV 5×5 same block above.





28x28x192 28x28x32

Each of the 32 filters making up the output channels are 5x5x192

→ we need to compute:

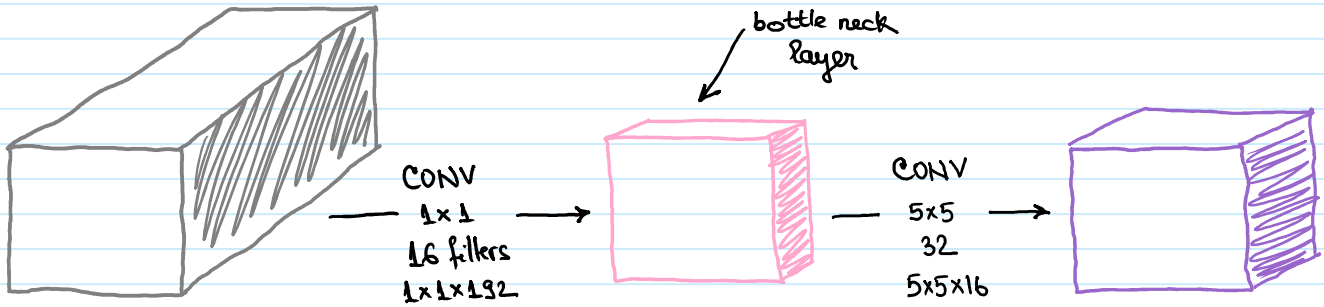
28x28x32 × 5x5x192 = 120M calculations

(28x28x32: # of output values; 5x5x192: computations per output value)

we can reduce this 10x by using 1x1 matrix convolutions

→ so 12M calculations

Reducing the # of computations by using 1x1 convolutions:



28x28x192 28x28x16 28x28x32

# of computations: 28x28x16 x 1x1x192 = 2.4M 28x28x32 x 5x5x16 = 10.0M

Total: 2.4M + 10.0M = 12.4M

Note: # of additions is similar to # of multiplications so we are only calculating # of multiplication.

QUESTION: How much does shrinking the representation using a bottle-neck layer hurt performance?

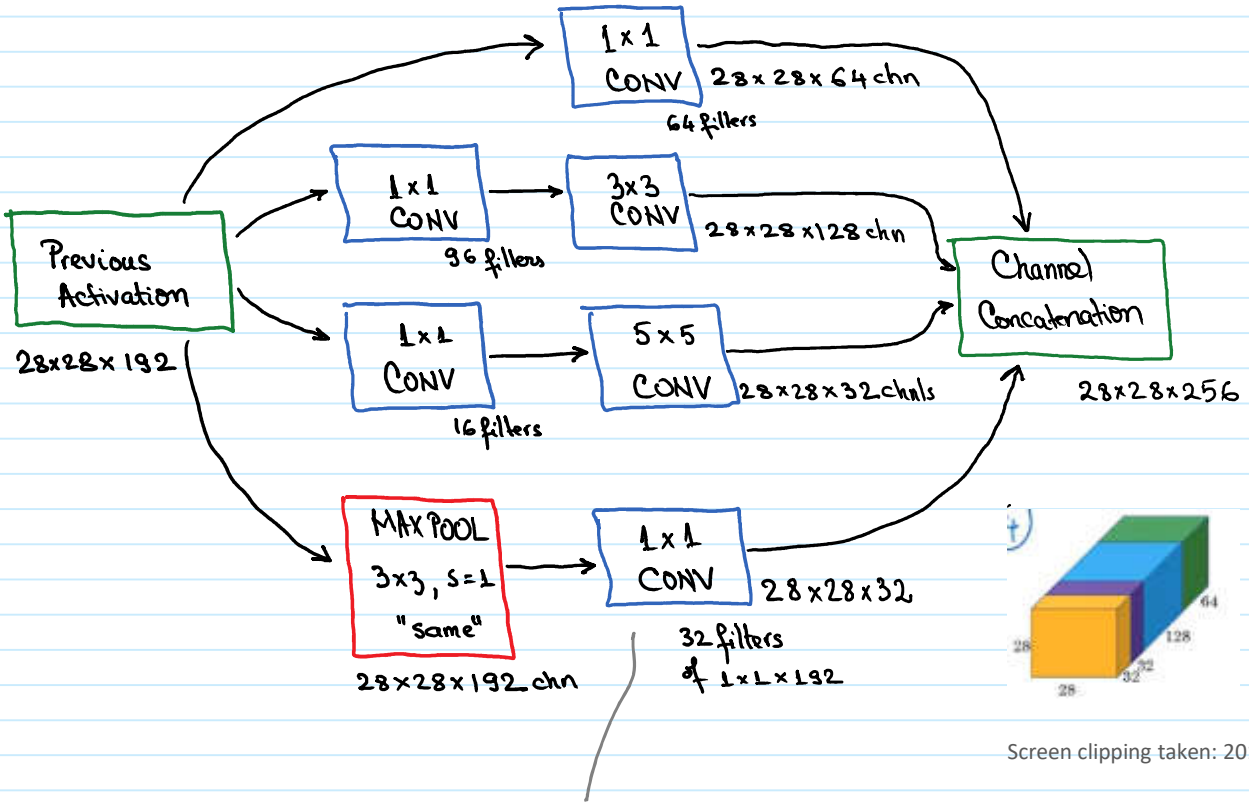
As long as the bottle-neck layer is "implemented within reason" we

can shrink the representation by quite a lot without affecting performance much.

Inception Networks

Thursday, 16 August, 2018 12:51

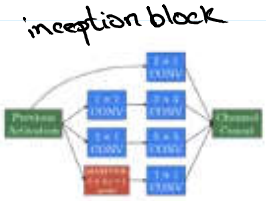
Goal: Understand how inception networks are implemented

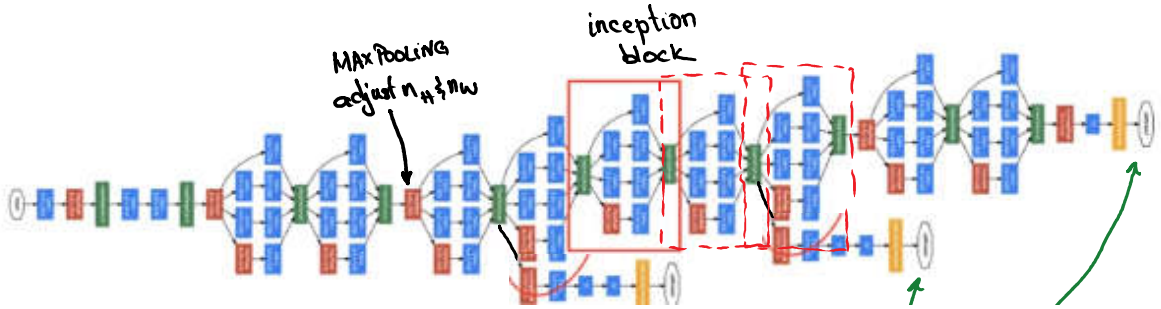


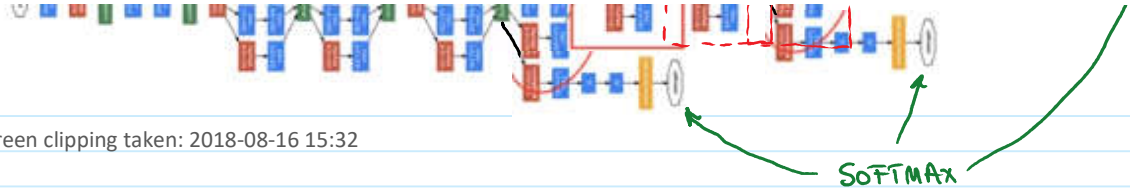
Screen clipping taken: 2018-08-16 15:31

add the 1×1 CONV to decrease 192 chn to 32 chn so we don't end up with MAX POOL taking over all the channels in the final layer.

Inception network







Screen clipping taken: 2018-08-16 15:32

There's additional side branches that (similar to the last layer) use SOFTMAX to try to make a prediction.

→ this is used to ensure the features computed in the intermediate layers are not off when predicting the output label. Using them has a regularizing effect (prevents overfitting)

Inception network was developed by Google where it was called GoogLeNet The name of the network is cited as bein inspired by the meme below:

http://knowyourmeme.com/memes/we-need-to-go-deeper A

Screen clipping taken: 2018-08-16 15:45

Practical Advice - Use open-source modules

Thursday, 16 August, 2018 15:46

- Many NN are difficult to replicate just from reading research papers.
- Use open source implementations of pre-trained NN that can be downloaded mostly from GitHub

Practical Advice - Transfer Learning

Thursday, 16 August, 2018 15:46

Labeled Data sets that can be used to train networks

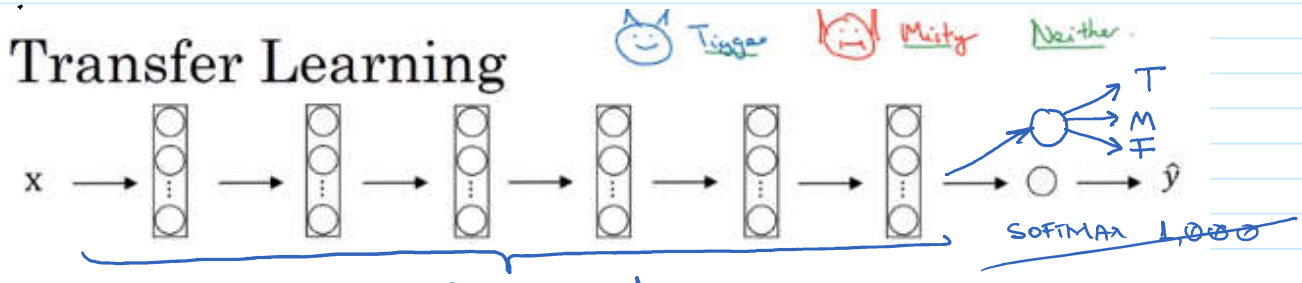
ImageNet, MS coco, Pascal

Use transfer learning to train your network from pre-trained NN available online

EXAMPLE - SMALL TRAINING SET

Training NN to recognize two cats (Tiger & Misty)

- there is very little training data available with just these cats (Tiger and Misty)
- but there are many NN that have been trained by others on large data sets that can be downloaded. ImageNet is one of them and it has a SOFTMAX output layer with 1,000 classes
- download one of these NN (both code & weights)



Screen clipping taken: 2018-08-16 17:23

- Train just the last layer parameter (freeze the upstream layers weights)
  - many of the Open Source NN are friendly & have hooks that allow freezing of early layers.

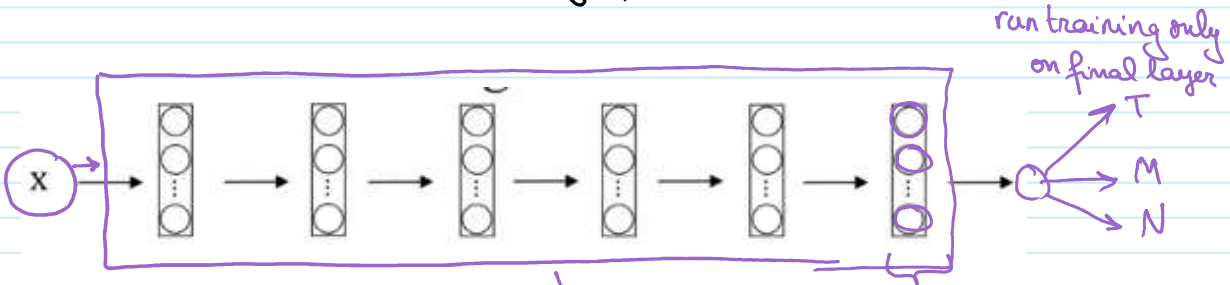
i.e. trainable parameters = ∅

freeze = 1

☆ NEAT SPEED-UP TRICK

Because the layers before the last layer are "frozen" we can "precompute" the activations of the Tigger & Misty training data and rerun the learning optimization only on the last layer.

→ this prevents us from recomputing the activations every time we run a training epoch.



Screen clipping taken: 2018-08-16 17:35

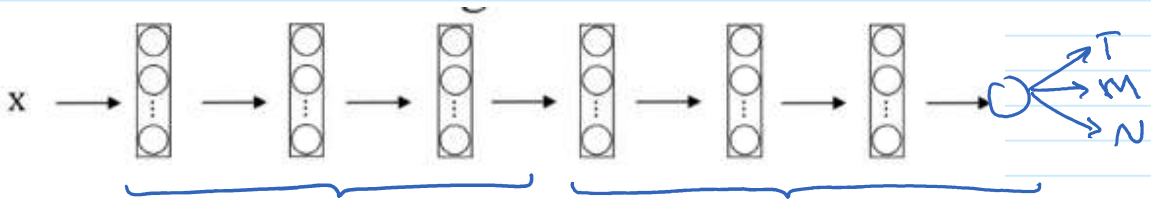
pass all the training data through these frozen layers

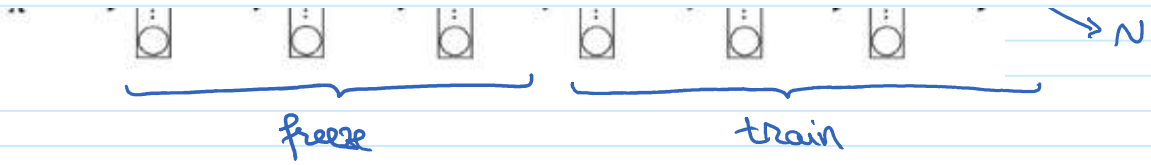
these activations are "fixed" as the training data is constant and the upstream layers are frozen so save it to disk as input for last layer

EXAMPLE - LARGER DATA SET

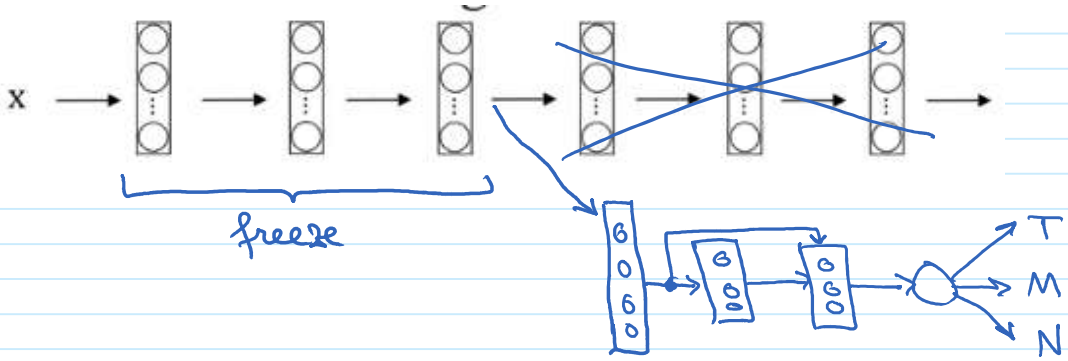
Freeze fewer downstream layers.

- Option A: keep the "unfrozen" layers





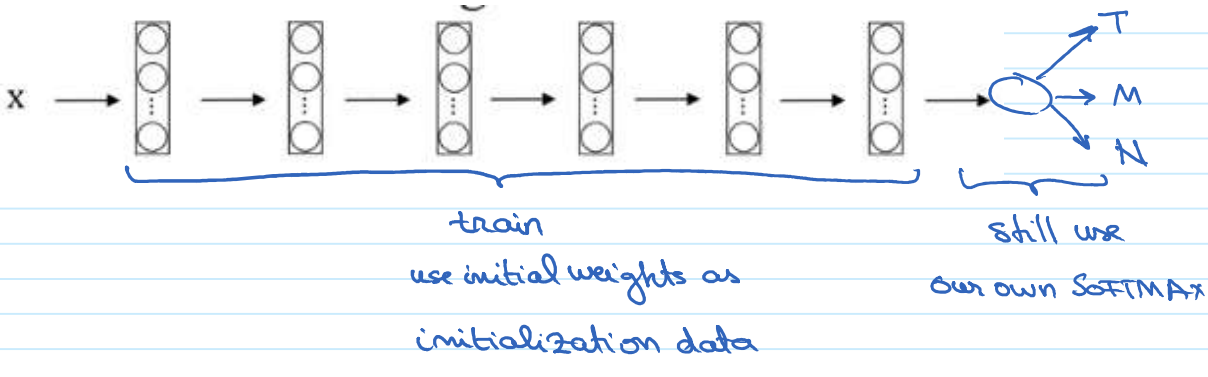
- Option B: Discard "unfrozen" layers and replace them with our own layers.



more data allows us to train more layers not just the end SOFTMAX Layer

- Option C: Use all layers in the network to continue training the network

→ in this case we would use the previous weights as initialization data.



TAKEAWAY:

Unless we have a very large amount of data start by using someone's

else NN and use transfer learning to bootstrap our own network

Practical Advice - Data Augmentation

Thursday, 16 August, 2018 21:36

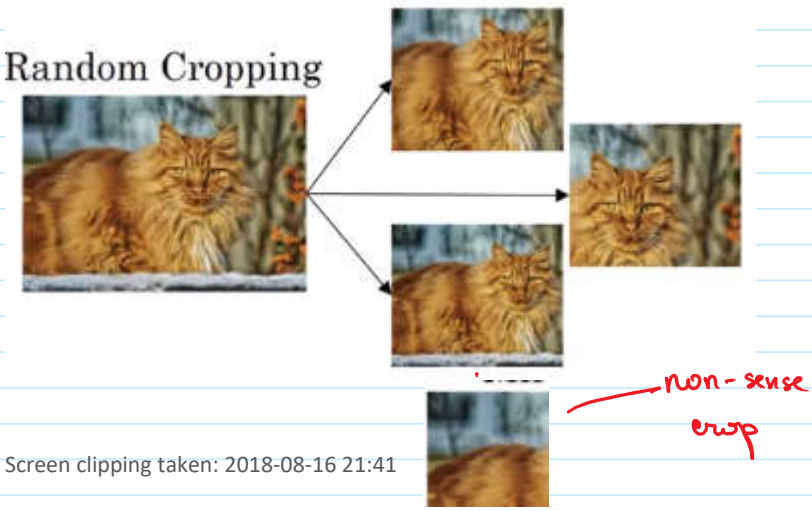
Computer Vision - like many DeepLearning tasks - requires as much data as possible. Data augmentation comes in handy.

Data augmentation techniques:

- Mirroring:

Screen clipping taken: 2018-08-16 21:39

- Random Cropping:



Screen clipping taken: 2018-08-16 21:41

Random cropping works as long as we don't take a non-sense crop.

- Rotation } used less

- Rotation
- Shearing

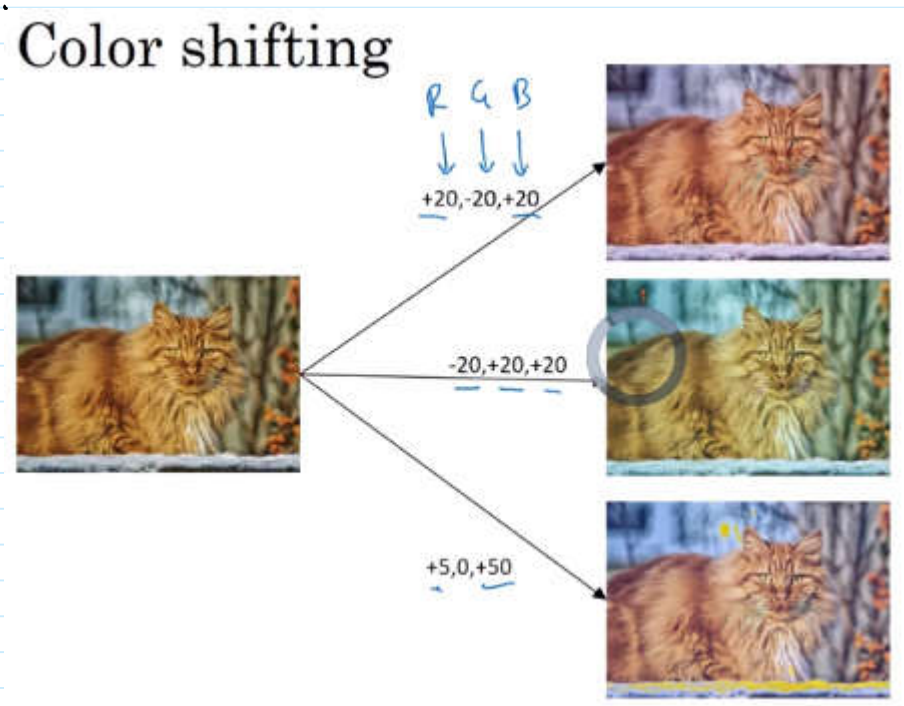
} used less

- Color Shifting
  - draw the color channel distortion factor from a tight distribution
  - this accounts for different types of illumination in real life.
  - there are many ways to sample RGB for distortion purposes.

    PCA - principal color analysis (used in Alex Net paper)

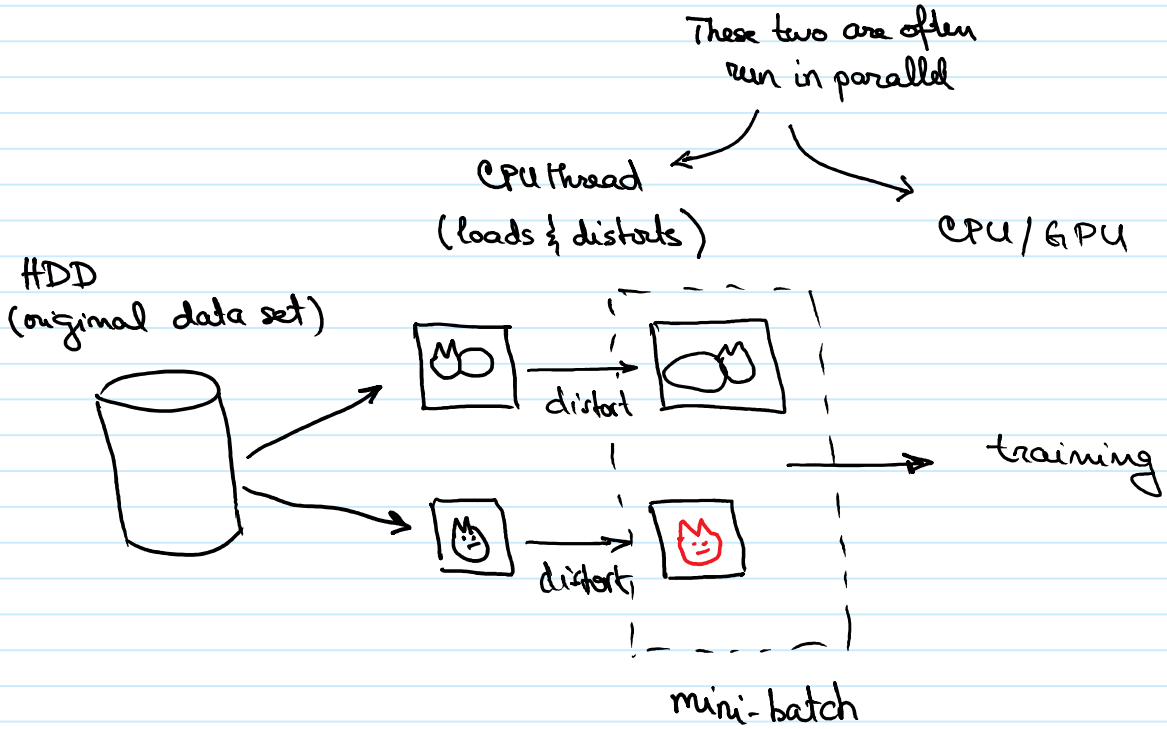
    - aka PCA color augmentation
    - Keeps the color tint the same.

      i.e. if image has a lot of R & B and little G (purple tint) PCA adds/subtract a lot of R & B and little G.



Screen clipping taken: 2018-08-16 21:51

IMPLEMENTATION DETAILS



END NOTES:

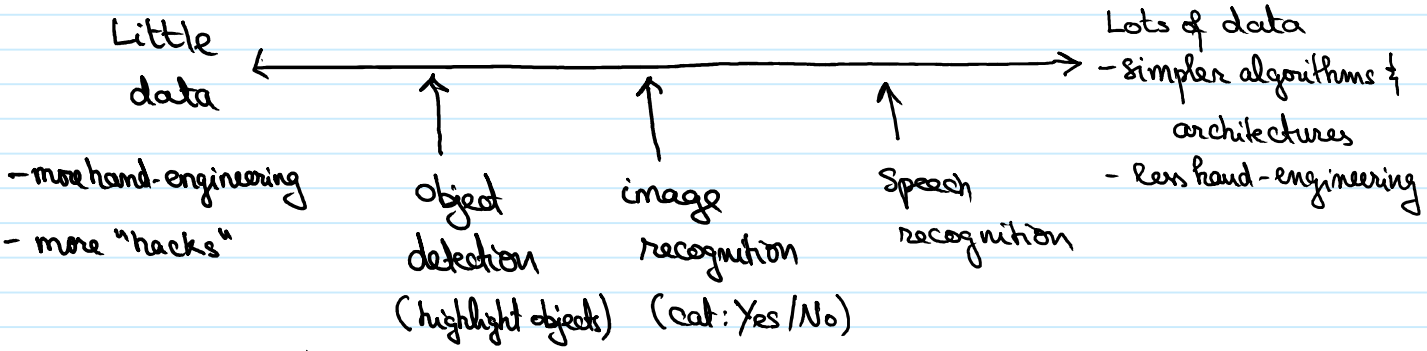
Similar to CV NN development Data Augmentation has many associated hyperparameters. Using someone else's work as a starting point allows for faster progress.

The State of Computer Vision

Thursday, 16 August, 2018 22:00

Goal: learn about key observations from Deep learning specific to CV

DATA VS HAND-ENGINEERING



Screen clipping taken: 2018-08-16 22:05

- There are two source of knowledge for creating a ML classifier:
  - Labeled data (x, y)
  - Hand-engineering
    - features
    - network architecture
    - other
    - hyperparameter choices

These two balance each other. As CV doesn't have much labeled data the field of CV is much more hand-engineering heavy (complex algorithms)

→ more recently as more data is available this artisanal

work is becoming simpler.
→ transfer learning helps dramatically.

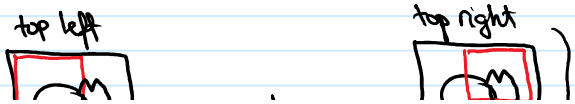
- Field is focused on improving performance on standard data sets
  - benchmarks / winning competitions.

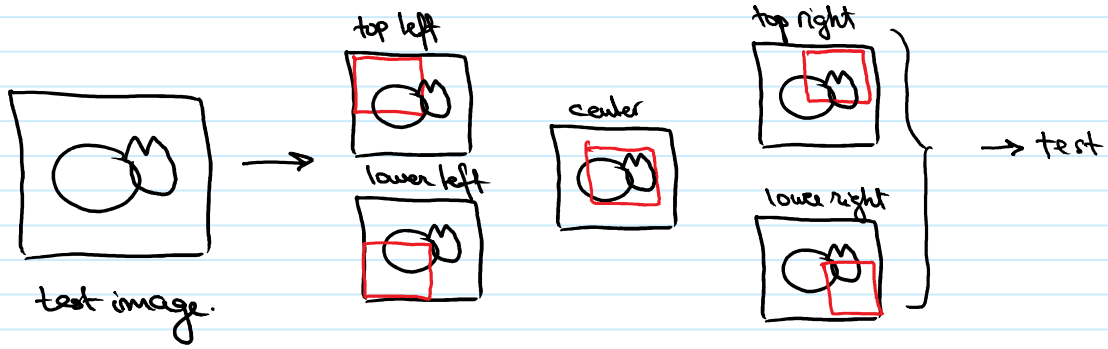
This will get you published but does not necessarily translate into systems that can be deployed in the industry

What helps win competitions ? NOT USED IN PRODUCTION

- ensembling: train several NN independently and average their outputs.
  → initialize 3 to 15 NN randomly and average their outputs (j do not average their weights)
  → this improves performance by 1% - 2%
  → this however slows down training 3X - 15X and as we need to keep these NN around this technique also uses a lot of storage space which is not feasible (unless you have huge budgets)

- multi-crop at test time: run classifier on multiple version of test images and average the result
  → this is a form of data augmentation at test time
  → an example is the "10-crop" or "5-crop"





→ this also slows down data processing

SUMMARY

- Use architectures of networks published in the literature
- Use open source implementations if possible
  → these guys figured out the finicky details (learning rate decay, hyperparameters)
- Use pretrained models and fine-tune on your dataset
  → spent compute resources to train a NN

Screen clipping taken: 2018-08-16 22:31

- if we invent a new architecture we probably have to do all this on our own.

Week 3 Object Detection

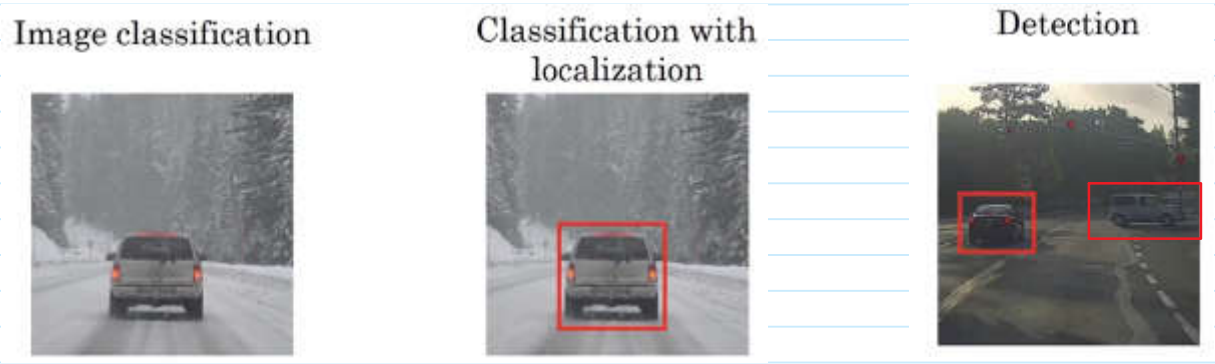
Sunday, 19 August, 2018 19:41

Object Localization

Sunday, 19 August, 2018 19:42

To build up the Object Detection classifier we are first going to understand Object Localization.

IMAGE CLASSIFICATION VS LOCALIZATION VS DETECTION



Screen clipping taken: 2018-08-19 22:02

Label it as car or not (single obj)

Label it as car or not

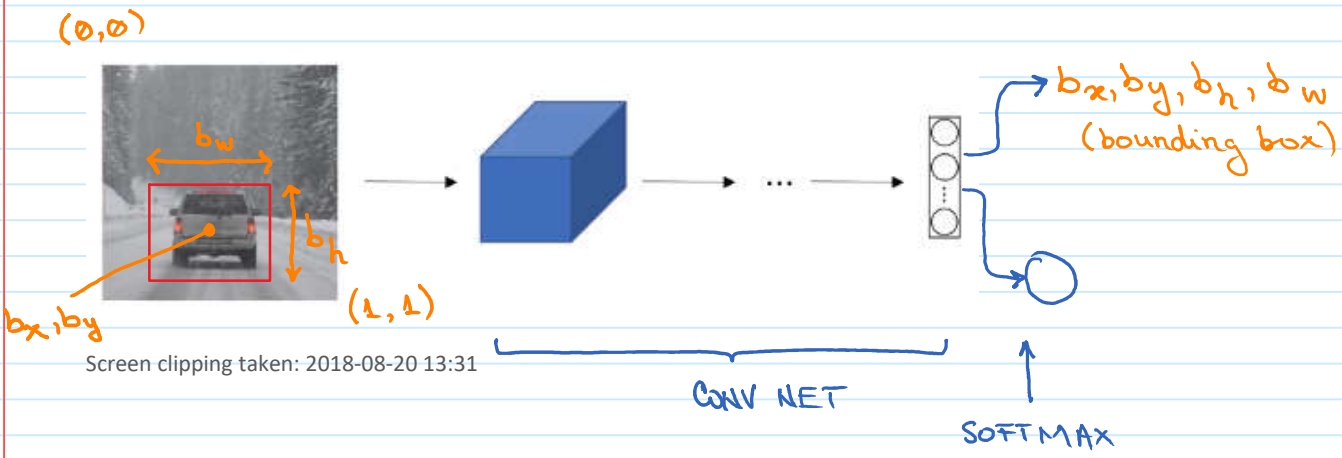
Localize where car is (single obj)

Label it

Localize one or more

Localize other objects too

CLASSIFICATION WITH LOCALIZATION



Screen clipping taken: 2018-08-20 13:31

Problem: Input picture output what class is in the picture and localize the object in the image.

SOFTMAX class labels:

1) Pedestrian

2) car

3) motorcycle

4) background
(none-of-the-above)

The training set not only includes the label of the image but also b_x, b_y, b_w, b_h.

$b_x = 0.5$

$b_y = 0.75$

$b_h = 0.3$

$b_w = 0.4$

DEFINE TARGET LABEL y

The NN now outputs b_x, b_y, b_h, b_w and class label (1-4) [or maybe probability of class label].

Define target label y as follows:

$x =$

$$\begin{bmatrix} p_c \\ b_x \\ b_y \end{bmatrix} \quad \begin{bmatrix} 1 \\ b_x \\ b_y \end{bmatrix} \quad \begin{bmatrix} 0 \\ ? \\ ? \end{bmatrix}$$

"don't care"

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} \quad \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad \begin{bmatrix} p_c \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix}$$

← don't care

Where :

- p_c : is there an object? (probability an obj of the class other than background is present.

  For label class 1-3 $p_c = 1$

  For label class 4 $p_c = 0$

- b_x, b_y, b_h, b_w : if there is an obj of class 1-3 output its location (its bounding box)

- c_1, c_2, c_3 : if there is an obj which class (pedestrian, car, motorcycle) is it?

  → assume only one of the classes is in the picture

Loss FUNCTION USED

Remember : y : ground truth

$\hat{y}$: NN output

Use squared error (easier to explain) :

$$\mathcal{L}(\hat{y}, y) = \begin{cases} (\hat{y}_1 - y_1)^2 + (\hat{y}_2 - y_2)^2 + \dots + (\hat{y}_8 - y_8)^2, & \text{if } y_1 = 1 \ (p_c = 1) \\ (\hat{y}_1 - y_1)^2, & \text{if } y_1 = 0 \end{cases}$$

$$\mathcal{L}(\hat{y}, y) = \begin{cases} (\hat{y}_1 - y_1)^2, & \text{if } y_1 = 0 \end{cases}$$

where 1 - 8 denote each of the features in y $(\underbrace{p_c, b_x, b_y \dots c_3}_{8 \text{ features}})$

In practice the loss function can be more complex:

p_c } Logistic regression loss — Logistic Regression Loss

b_x
b_y } Squared error.
b_h
b_w

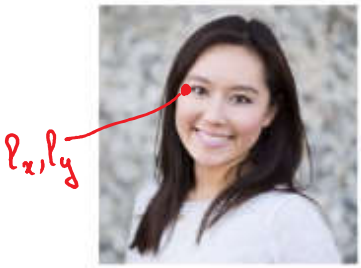
c_1
c_2 } Log likelyhood loss — Softmax Classifier
c_3

Landmark Detection

Monday, 20 August, 2018 18:15

Goal : Understand how to get a NN to output coordinates x, y for important points in the data (i.e. landmarks)

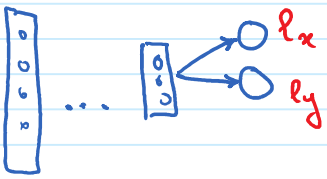
b_x, b_y, b_h, b_w



Screen clipping taken: 2018-08-20 18:17

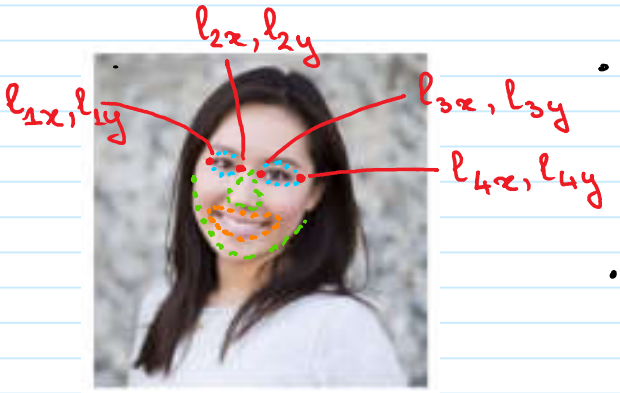
EXAMPLE : FACE EXPRESSION

Train a NN so it outputs where the corner of someone's eye is (l_x, l_y)



NN outputs two more numbers.

What if we want to output coordinates for all corners of the eyes like below



Screen clipping taken: 2018-08-20 18:22

- Or what if we want to output a much finer shape around the eye
- Or what if we want to output the points around the mouth to determine the shape of the mouth and whether the person is frowning

the mouth and whether the person is frowning or smiling.

- What if we want to output the edges of the face or shape of the nose ?

→ Let us define a set of points (i.e. 64) we'll call landmarks

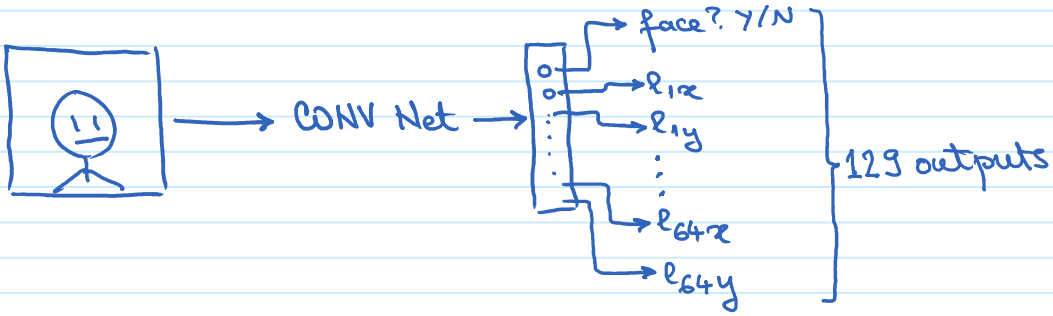
l_{1x}, l_{1y}

l_{2x}, l_{2y}

⋮

l_{64x}, l_{64y}

- These landmarks are part of the training data set.
- We then train NN to identify these key regions on a face.



- This is used as a basic building block for:
  - emotional recognition
  - Snapchat Augmented Reality filters (add dog ears to your face)

DIFFICULT TASK:

- Laboriously label the training set.

EXAMPLE: BODY POSE

EXAMPLE: BODY POSE

Screen clipping taken: 2018-08-20 18:43

Similarly to Face Expression detection come up with key landmarks as part of the training data that capture the position of the body.

l_{1x}, l_{1y}

$\vdots$

l_{32x}, l_{32y}

CAVEAT: Labels have to be consistent among all training images (i.e (l_{1x}, l_{1y}) always identifies the center of the head)

Object Detection

Monday, 20 August, 2018 19:15

Goal: Learn to use a CONV Net to perform object Detection using the Sliding Windows Detection Algorithm.

CAR DETECTION EXAMPLE

- Final objective: detect & localize cars in images.

Screen clipping taken: 2018-08-20 19:19

- First step: create a labeled training set with closely cropped images of cars.
  (i.e. the image is only the car - everything else is cropped out)

Training set:

x	y
	1
	1
	1
	0
	0

Screen clipping taken: 2018-08-20 19:20

Use this training set to create a CONV Net that outputs car Yes/No.

 → CONV Net → Car Yes/No (0 or 1)

Screen clipping taken: 2018-08-20 19:23

- We then use this CONV Net into a Sliding Windows Detection Algorithm

  1) Pick a sliding window size

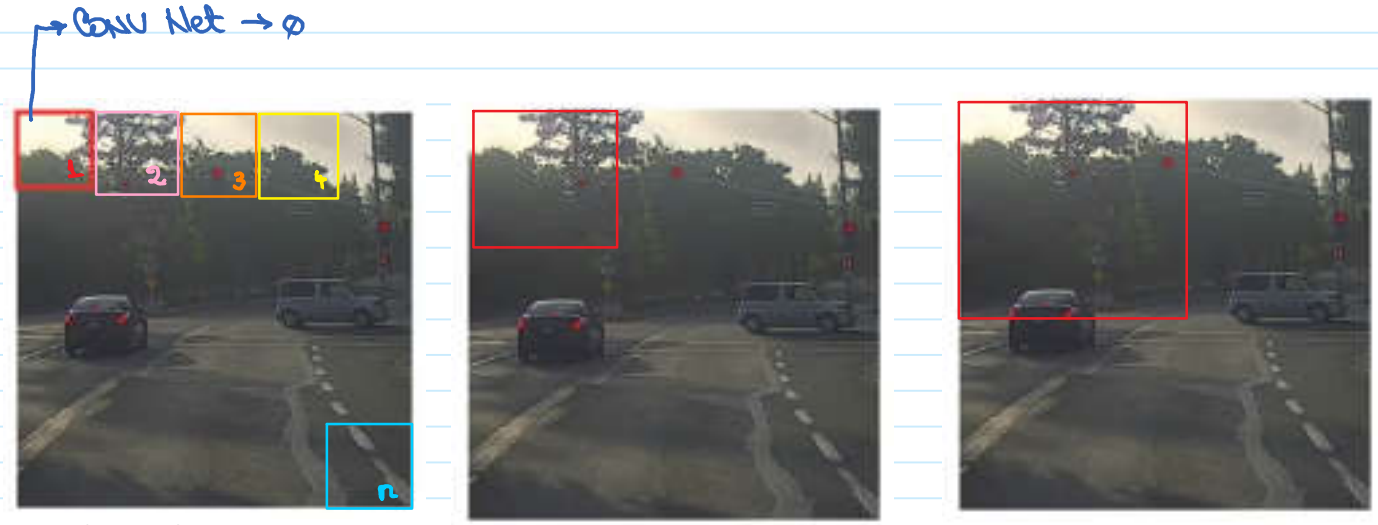
  2) Input into the Conv Net a section of the entire image and check if a car is detected or not.

  3) Repeat step 2 until the entire image has been tiled

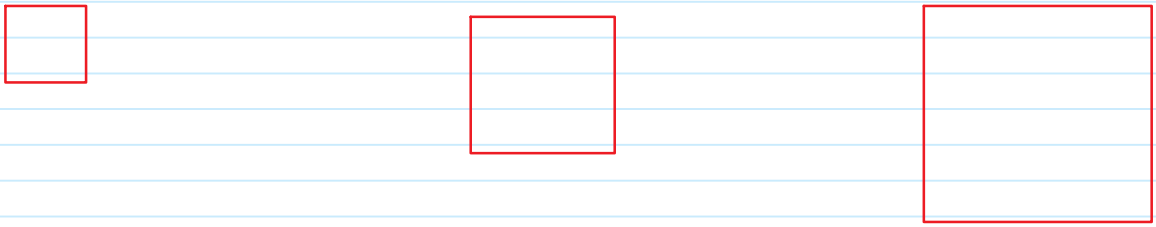
  4) Repeat 1-3 with a slightly larger window.

  5) Repeat 1-3 with even larger window size.

Note: Stride size is a parameter.



Screen clipping taken: 2018-08-20 19:26



Using this algorithm it is highly likely that every car in the image will be detected

SLIDING WINDOW DISADVANTAGE: COMPUTE COST

→ there are many croppings that need to be checked and CONV Nets are expensive to run.

→ increasing the stride (fewer crop regions) negatively impacts sensitivity so we would rather keep stride small

Previously (when CONV Nets were not used) simple linear classifiers were used. These are running much faster (though they are not as performant) so Sliding Window Detection could be used feasibly.

Solution to this problem in next slide.

Sliding Window (CONV Implementation)

Monday, 20 August, 2018 22:00

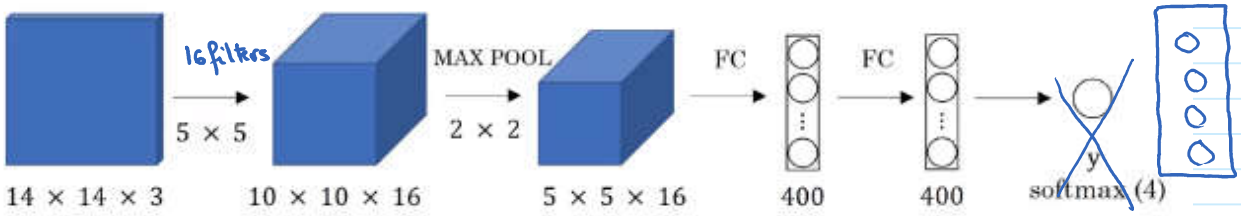
Goal: Learn how to implement the Sliding Window Detection using Convolution so that we need less compute and speed it up to a level that makes it feasible to run in production

A tool we will use to implement SWD using convolution is to turn Fully Connected layers into CONV layers.

TURNING FC LAYERS INTO CONV LAYERS

Working Example: We have an image detection algorithm like below

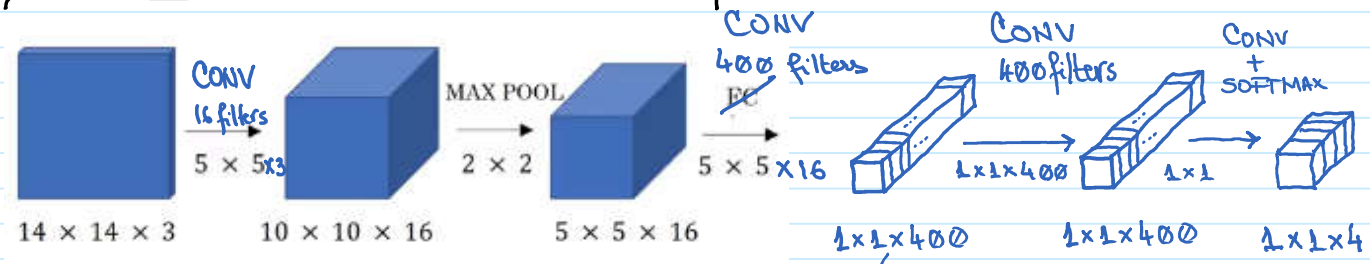
To make the change to CONV layers we'll make the output 4 numbers each corresponding to the probability the output is one of the four classes SOFTMAX classified (car, pedestrian motorcycle, background)



Screen clipping taken: 2018-08-20 22:07

For each node in the FC layer there are $5 \times 5 \times 16 = 400$ associated weights (one from each cell in the $5 \times 5 \times 16$ volume)

The first layers are the same as before



Screen clipping taken: 2018-08-20 22:15

Each cell in the 1x1x400 has associated with it 5x5x16 = 400 weights. Each of these weights corresponds to one of the cells in the 5x5x16 volume

This is also equivalent to a FC layer.

CONVOLUTION IMPLEMENTATION OF SLIDING WINDOWS DETECTION

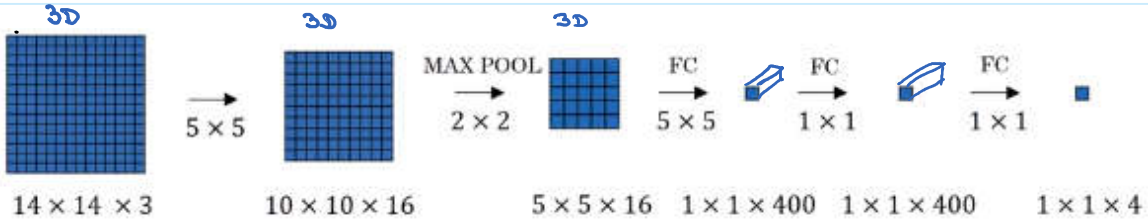
As presented in:

[Sermanet et al., 2014, OverFeat: Integrated recognition, localization and detection using convolutional networks]

Screen clipping taken: 2018-08-21 17:25

Assume the input image into the CONV Net (i.e. run on one window) is 14x14x3

- small numbers make it easy to explain
- for ease of drawing represent the volumes as 2D instead of 3D

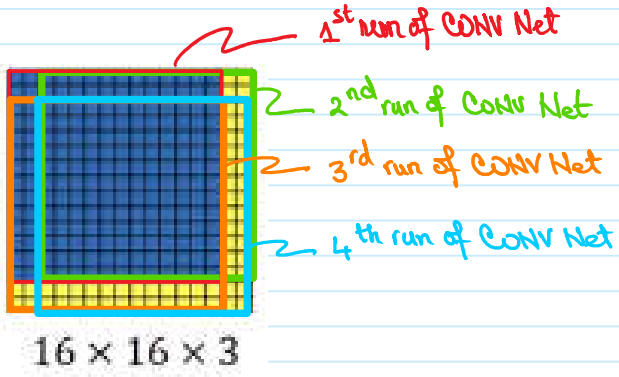


Screen clipping taken: 2018-08-21 17:27

SOFTMAX output

Assume the test set image is 16x16x3 (yellow stripe). We need to run the CONV Net

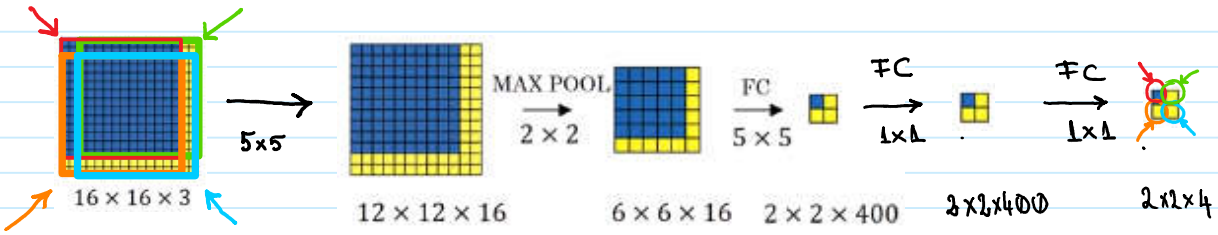
4 times to cover the test set image (using stride = 2)



Screen clipping taken: 2018-08-21 17:37

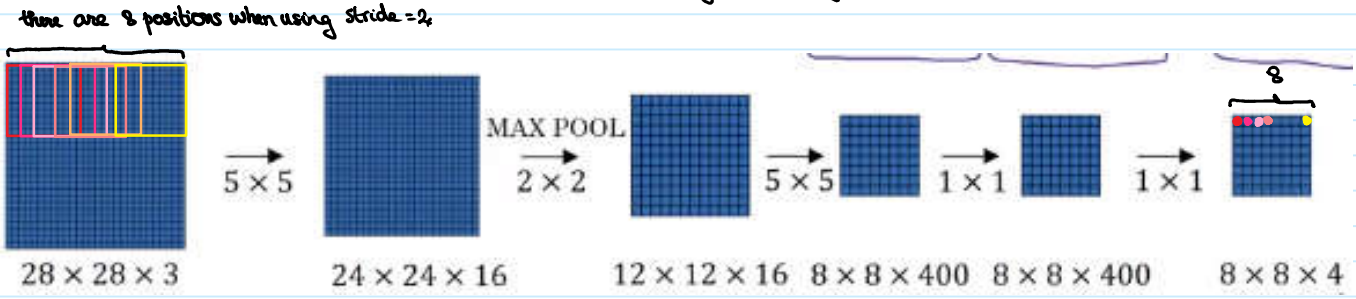
But a lot of the computations done in these 4 passes are duplicates (i.e. they are performed repeatedly for each pass)

We can share the results of these computations among the CONV Net passes by using a convolution implementation of the entire classifier



Screen clipping taken: 2018-08-21 18:59

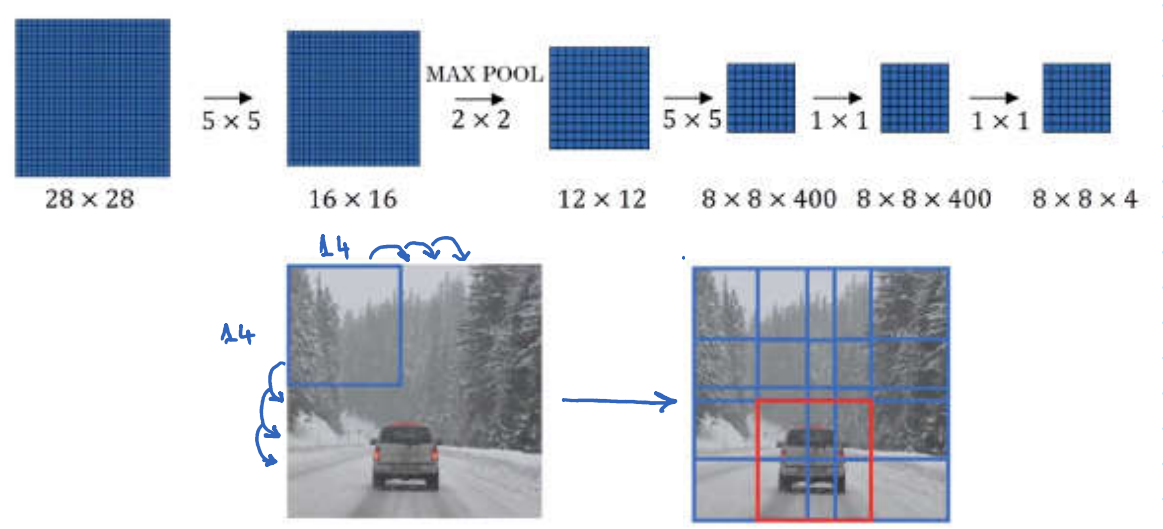
Let's say we want to run SWD on larger images (28×28×3)



Screen clipping taken: 2018-08-21 19:12

SUMMARY :

- Normally we would feed each window into the CONV Net to label which ones have Cars present in them.
- Using the convolution implementation we sweep accross the entire image and detect which windows are likely to be cars in a single pass through the NN

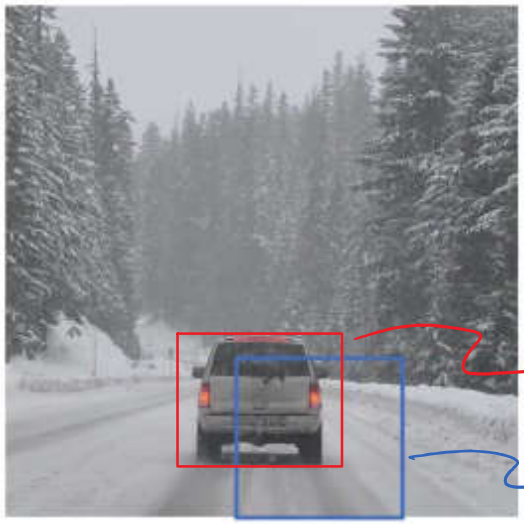


Screen clipping taken: 2018-08-21 19:26

Bounding Box Predictions

Wednesday, 22 August, 2018 00:02

Goal: Learn how to improve the accuracy of the bounding box used in object detection



PROBLEMS:

- None of the boxes of the SWD match well with the position of the car
- The shape of the box is not square it is more of a horizontal rectangle.

Screen clipping taken: 2018-08-22 00:05

SOLUTION: YOLO ALGORITHM

[Redmon et al., 2015, You Only Look Once: Unified real-time object detection]

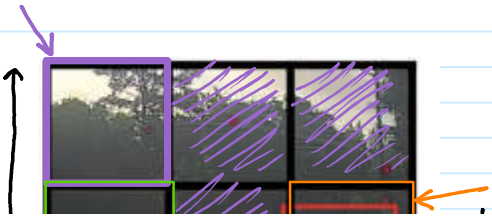
Screen clipping taken: 2018-08-22 10:57

Note: This is one of the harder papers to figure out the details.

- Divide the image into a grid (i.e. 3x3)

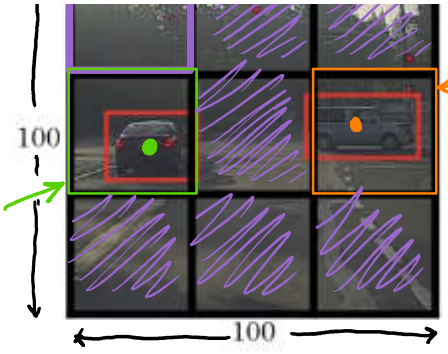
  Grid in this case is 3x3 – in practice it would be much finer - maybe 19x19

- Apply the image classification & localization from Classification with Localization algorithm to each of the 3x3 cells (9 cells)



- For each of the 9 cells the algorithm will use as training data an 8 dimensional vector label y

  p_c: 0, 1 (object present/absent)



p_c: 0, 1 (object present/absent)

b_x, b_y: center of bounding box coordinates (if obj present)

b_h, b_w: bounding box height/width (if obj present)

c_1, c_2, c_3: object classes: pedestrian, car, motorcycle (if obj present)

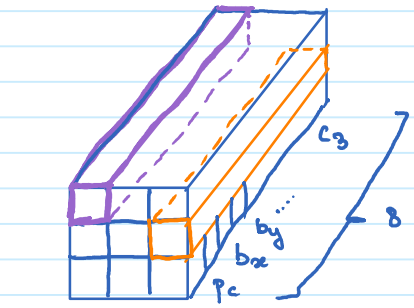
Screen clipping taken: 2018-08-22 10:59

- YOLO algorithm assigns the object to the grid cell cell containing the b_x, b_y point (midpoint of bounding box) → an object is assigned only to one grid cell.

  i.e. in the case above even though the central cell contains parts of both cars we'll consider it devoid of cars ($p_c = 0$)

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} \quad \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix} \quad \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

don't care



target output volume is $3 \times 3 \times 8$

TRAINING:

- Training data for NN uses an input image of $100 \times 100 \times 3$:
- We train the NN to output the $3 \times 3 \times 8$ volume.

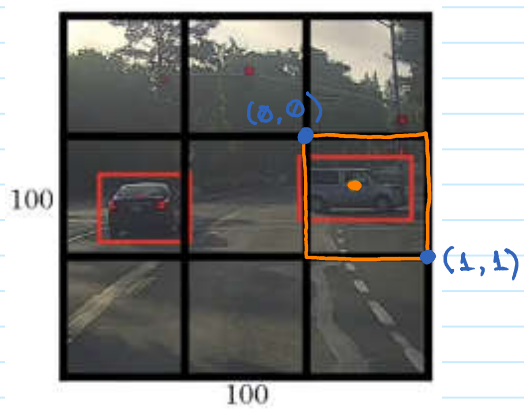
Input → CONV → MAX POOL → ... → Output $\hat{y}$

TESTING:

- Run forward prop and for each of the 3x3 (9) outputs check p_c to determine if there is an object present in the cell (where it is & what it is)
- As long as there is only a single object in a grid cell the algorithm will perform Ok (when using a fine grid multiple obj per cell are rare).
- For multiple objects in a grid cell see later notes.

HOW DO WE ENCODE B_x, B_y, B_H, B_W

One reasonable convention is:



$$y = \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix} \begin{matrix} \\ \rightarrow 0.4 \\ \rightarrow 0.3 \\ \rightarrow 0.9 \\ \rightarrow 0.5 \\ \\ \\ \end{matrix}$$

must be inside grid cell coordinates (0,1)

for bounding boxes larger than grid cell b_h, b_w can be larger than 1.

Screen clipping taken: 2018-08-22 13:36

The YOLO paper uses a more complex convention:

- σ function ensures $b_x, b_y \in [0,1]$

- exponential parametrization $b_h, b_w > 0$
- etc.

SUMMARY:

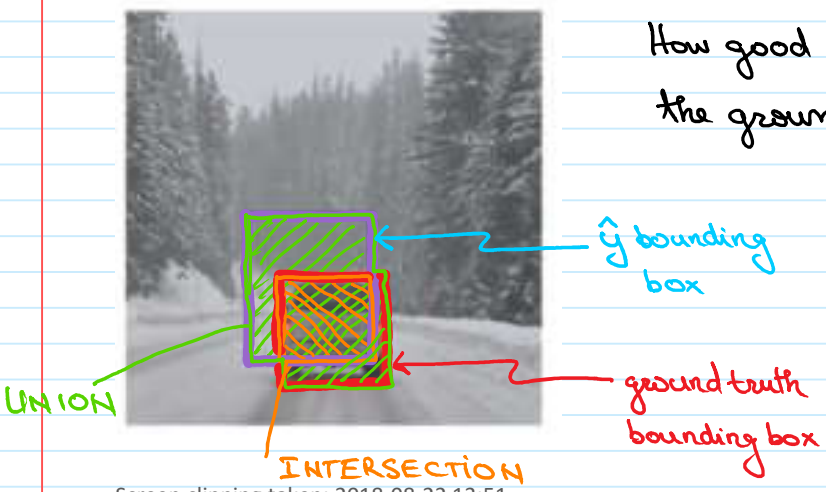
- By using b_x, b_y, b_h, b_w the YOLO algorithm localizes objects much more precisely (no longer limited by window location & window size)
- This is a convolutional implementation
  - → algorithm is <u>not</u> implemented 9 times (once for each cell in the 3 x 3 grid)
  - → computation is shared amongst all cells
  - → algorithm runs fast enough to be used for real-time image detection

Intersection over Union

Wednesday, 22 August, 2018 13:48

Goal: Learn how to tell if the object detection classifier is working well?

EVALUATE OBJECT LOCALIZATION



Screen clipping taken: 2018-08-22 13:51

How good is the $\hat{y}$ bounding box prediction wrt the ground truth?

Use INTERSECTION OVER UNION (IoU)

$$IoU = \frac{\text{area of } \blacksquare}{\text{area of } \blacksquare}$$

One convention is to consider the localization to be good enough to be called correct if $IoU \geq 0.5$

→ perfect overlap ⇒ $IoU = 1$

→ sometimes more stringent IoU (> 0.6) are used.

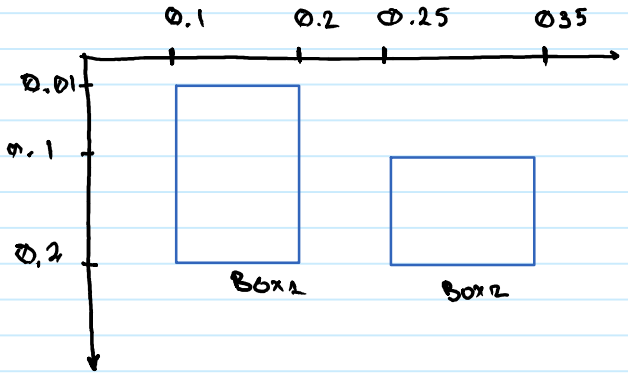
In general IoU measures the degree of overlap between two bounding boxes.

→ how similar are two boxes.

11) Don't confuse it with "I owe you" promisory note!

EXAMPLES:

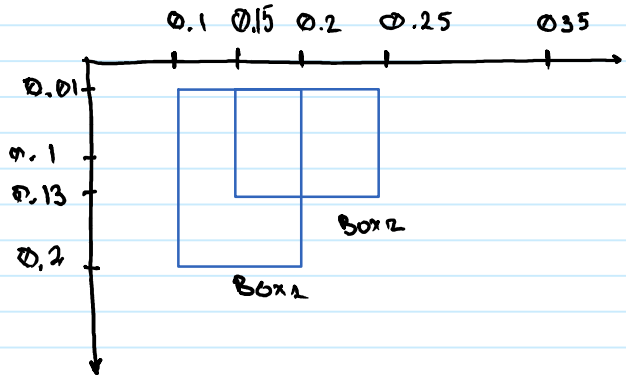
- No intersection



Box1: (0.1, 0.01, 0.2, 0.2)

Box2: (0.25, 0.1, 0.35, 0.2)

- Intersection



Box1: (0.1, 0.01, 0.2, 0.2)

Box2: (0.15, 0.01, 0.25, 0.13)

Non-max Suppression

Wednesday, 22 August, 2018 14:06

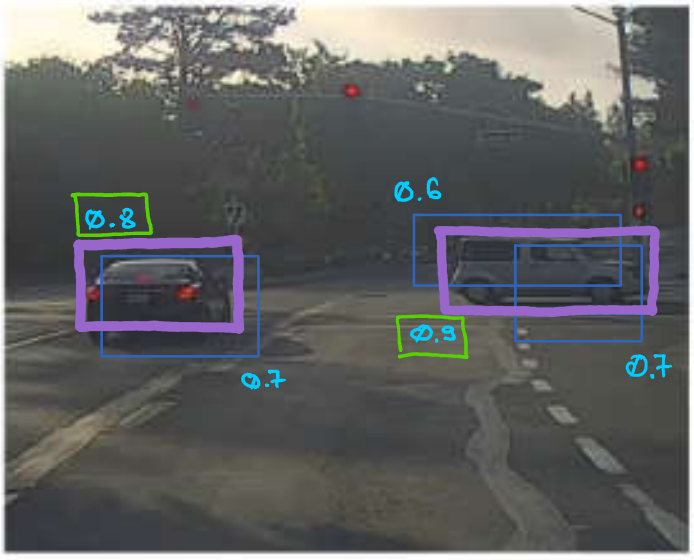
Goal: Learn how to ensure the object detection algorithm detects each object only once.

EXAMPLE OF PROBLEM

Screen clipping taken: 2018-08-22 18:50

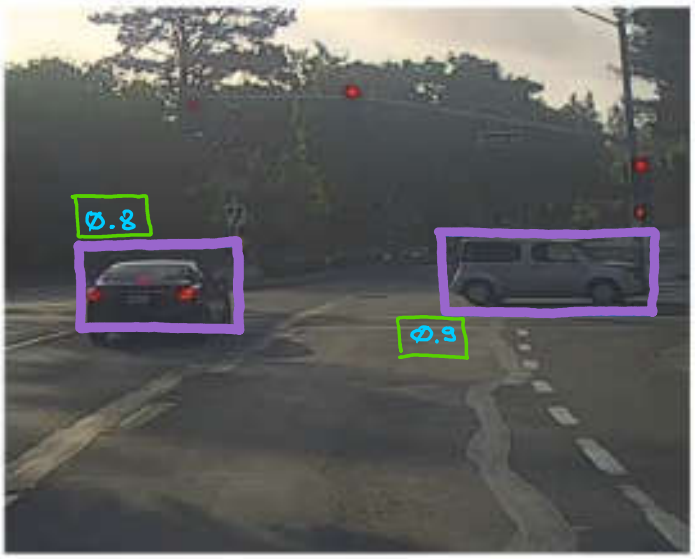
NON-MAX SUPPRESSION SOLUTION

- Non-max suppression cleans up the cells indicating they contain a car so that only one car is detected per group of cells.
- Let's say for each car we have a few cells that identify the car.
- Each one of these cells has a probability associated with it that indicates how likely it is expected for a car to be there (similar to P_c - the assignment actually uses $p_c \times c_1$ or c_2 or c_3)



Screen clipping taken: 2018-08-22 18:55

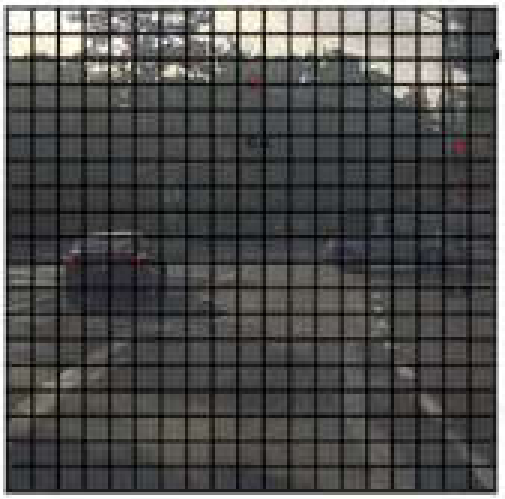
- Iterate through the probabilities of detection and find the bounding box associated with the maximum probability.
- The non-max suppression section then suppresses all the remaining bounding boxes that have high IoU with the maximum probability bounding box.
- The remaining boxes are the output.



NON-MAX SUPPRESSION ALGORITHM

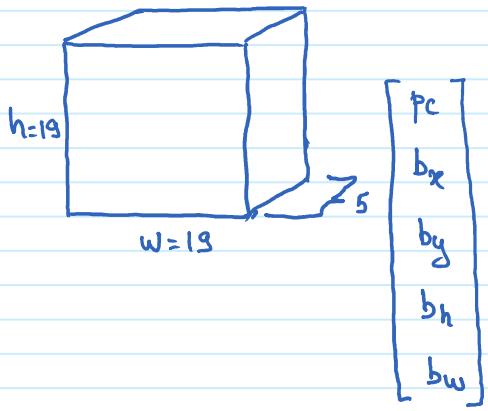
Let's pretend that for this implementation we are only detecting cars

(i.e. no pedestrians or motorcycles) so we only have c_2 out of c_1, c_2, c_3, c_4



19×19

So our output volume is 19×19×5



Screen clipping taken: 2018-08-22 19:12

- For each cell of the grid (19×19 = 361) we discard any bounding boxes associated with a very low probability of containing a car

  i.e. $p_c \leq 0.6$

- While there are any remaining boxes that were not processed do:

  [a processed box is either one that was removed or identified as an output]

  - pick the box with the largest probability (p_c) and use this as the output prediction.
  - discard any remaining box with a high IoU with the output prediction (i.e. IoU ≥ 0.5)

NOTE: If we were to also try to detect the other classes the output volume would have 3 additional components for the 3rd dimension

c_1, c_2, c_3

→ in this case we would independently carry out non-max suppression on each of the classes

(TO BE IMPLEMENTED IN THIS WEEK ASSIGNMENT)

Anchor Boxes

Wednesday, 22 August, 2018 19:27

Goal: Learn to use Anchor Boxes to allow each grid cell to detect multiple objects.

OVERLAPPING OBJECTS EXAMPLE:

[Redmon et al., 2015, You Only Look Once: Unified real-time object detection]

Screen clipping taken: 2018-08-22 19:31

Screen clipping taken: 2018-08-22 19:32

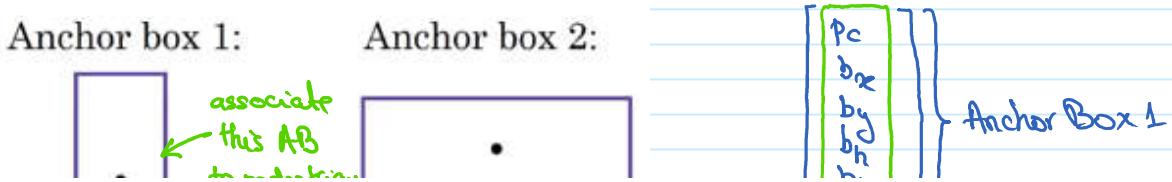
In the image on the left both the pedestrian and the car have their midpoint in aproximately the same location within the same grid cell.

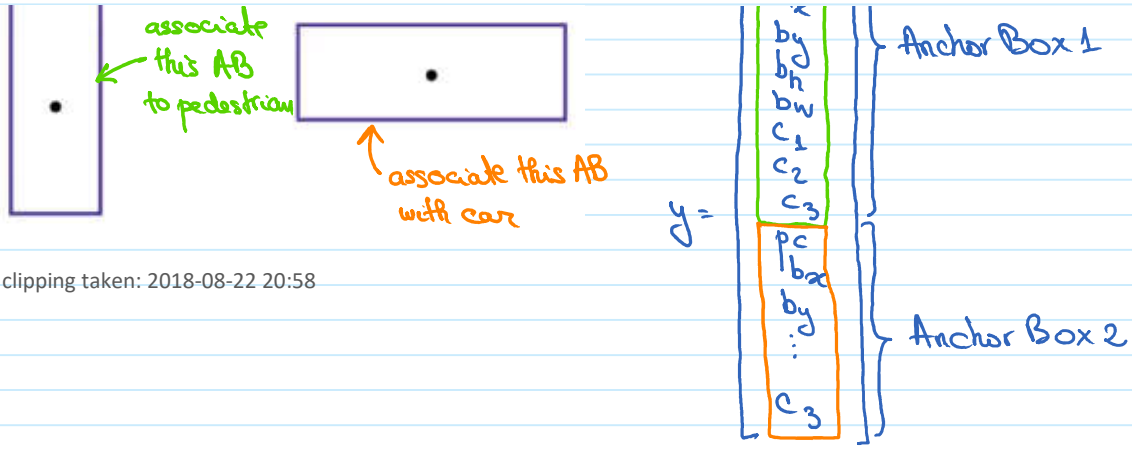
So for this grid cell the output would have to pick only one of the classes for the object detected.

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

SOLUTION USING ANCHOR BOXES:

- Predefine two shapes called "Anchor Boxes". For each Anchor Box associate multiple predictions (we use 5 or more Anchor Boxes in general)





Screen clipping taken: 2018-08-22 20:58

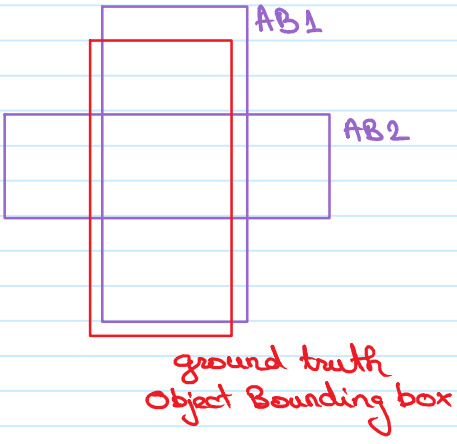
- The Anchor Boxes are associated with either the car or pedestrian based on how similar their shape is.

ANCHOR BOX ALGORITHM

Previously: Assign each object in the training image to the grid cell where the object's midpoint is located.

Output y: $3 \times 3 \times 8$

Anchor Box: Assign each object in the training image to the grid cell where the object's midpoint is located and to the anchor box for the grid cell with highest IoU to the object bounding box



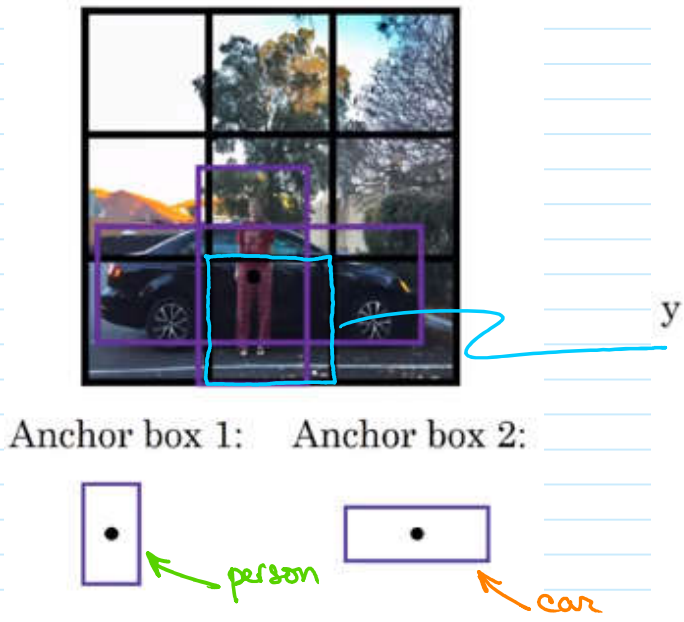
→ the object gets assigned not only to a grid cell but to a pair (grid cell, anchor box)

Output y: $3 \times 3 \times 16$

2×8

2 anchor boxes — y was 8 dimensional.

ANCHOR BOX EXAMPLE:



$y =$

Both pedestrian & car | What if car only?

	Both pedestrian & car	What if car only?	
p_c	1	∅	Anchor Box 1
b_x	b_x	?	
b_y	b_y	?	
b_h	b_h	?	
b_w	b_w	?	
c_1	1	?	
c_2	∅	?	
c_3	x	?	
p_c	1	1	Anchor Box 2
b_x	b_x	b_x	
b_y	b_y	b_y	
b_h	b_h	b_h	
b_w	b_w	b_w	
c_1	∅	∅	
c_2	1	1	
c_3	∅	∅	

Screen clipping taken: 2018-08-22 21:18

PROBLEM 1: this 2 Anchor Box implementation does not handle well the case when there are 3 object classes in the same grid cell (pedestrian, car & motorcycle)

→ would need to implement some sort of tie breaker

PROBLEM 2: also this implementation with two different shaped Anchor Boxes doesn't handle well two objects with similar shaped bounding boxes in the same grid cell (i.e. two cars)

→ again come up with a tie breaker

→ in practice if the grid is set to a fine tesselation (19 x 19) the chances of PROBLEM1 or PROBLEM2 occurring are small.

ADDITIONAL ADVANTAGE OF ANCHOR BOXES

Allow the network to specialize better. As the skinny tall Anchor Boxes get associated with pedestrians and the fat low Anchor Boxes are associated with cars some output units specialize in detecting skinny obj like pedestrians while other specialise in detecting wide objects cars.

How TO CHOOSE ANCHOR BOXES?

→ choose them by hand. 5 or 10 Anchor Box shapes that cover the type of objects we want to detect.

→ a more advanced way, as suggested in a later YOLO paper, would be to use a K-MEANS algorithm to group together the types of object shapes we want to detect and then use this to choose the Anchor Box shapes we want.

YOLO Algorithm

Wednesday, 22 August, 2018 21:46

Goal: Put together all the components of the YOLO object detection algorithm

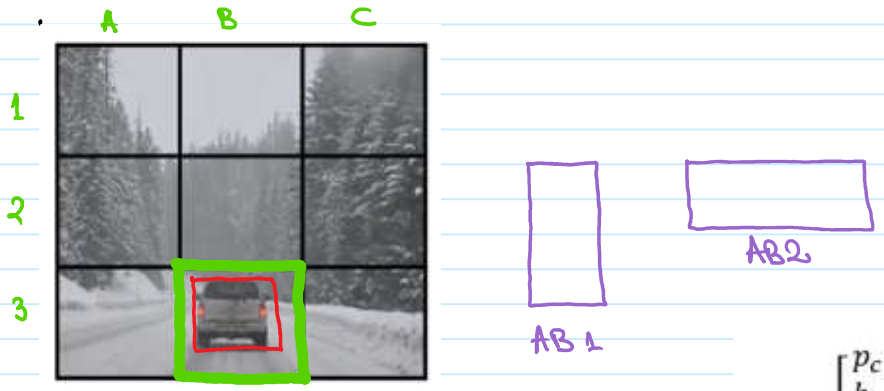
[Redmon et al., 2015, You Only Look Once: Unified real-time object detection]

Screen clipping taken: 2018-08-22 21:48

Assume we want to detect 3 types of objects
1 pedestrian
2 car
3 motorcycle

- In this case we don't need to explicitly set the background class (class 4).

1 CONSTRUCTING TRAINING SET



Screen clipping taken: 2018-08-22 22:20

If we use 2 anchor boxes the output y has dimensions

$3 \times 3 \times 2 \times 8$ or $3 \times 3 \times 16$

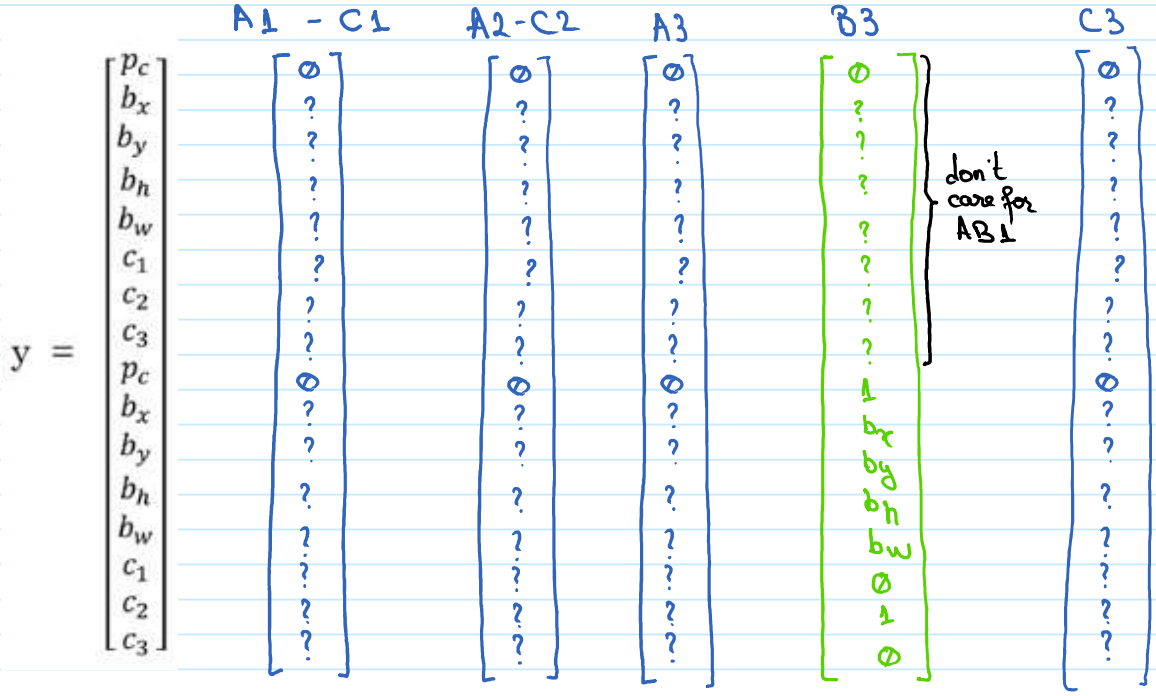
grid h, grid w, # of anchor boxes, parameters per anchor box: 5 + # of classes

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

p_c + bounding box; classes; anchor Box 1

To construct a training set we go through each of the 3×3 (9) grid cells of a

training image and form the apropriate output y (ground truth).



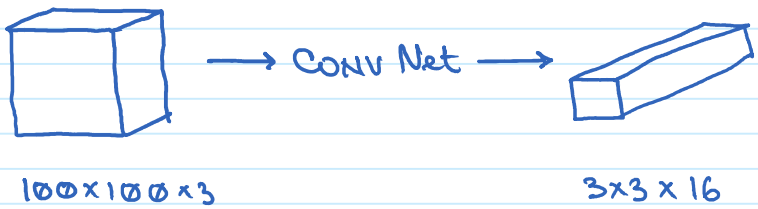
Assuming the bounding box for the car is slightly wider than tall it get assigned to Anchor Box 2.

- So all the y outputs for A1 to C3 get combined into the 3x3x16 output volume

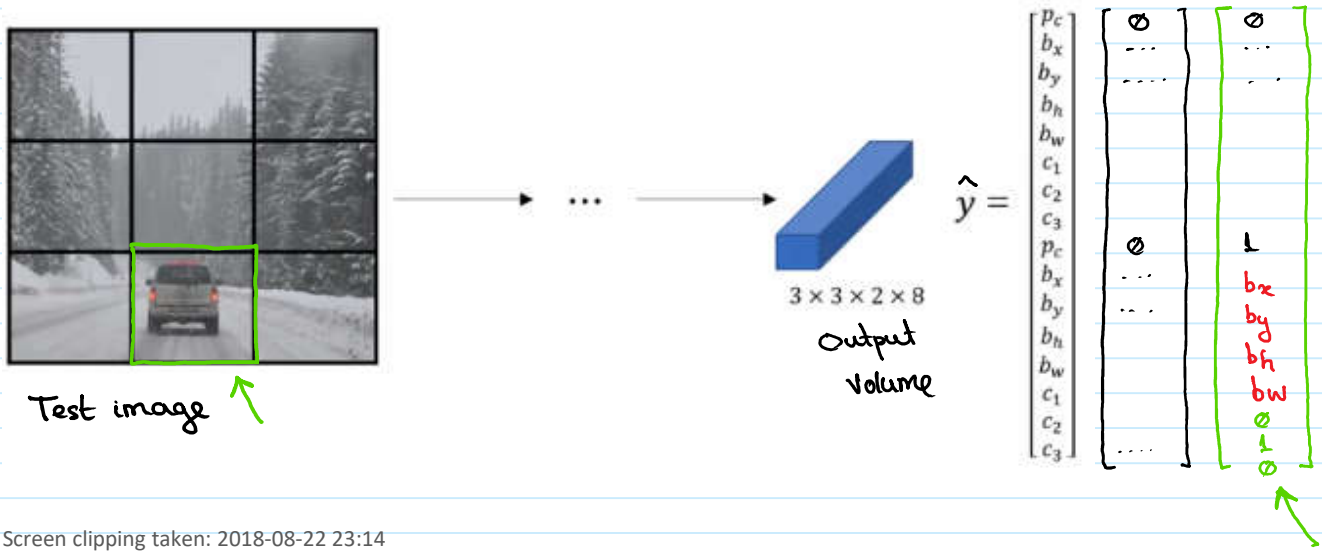
  → in practice it would be a 19x19x (5x8)

  use 5 Anchor Boxes.

- We use this Y output to train a CONV Net that takes as input X images of 100x100x3



2. MAKING A PREDICTION



Screen clipping taken: 2018-08-22 23:14

We would then take the 3x3x2x8 volume and run it through max suppression.

As we are using 2 Anchor Boxes for each grid cell we are getting 2 bounding boxes

1. For each grid call, get 2 predicted bounding boxes.
2. Get rid of low probability predictions.
3. For each class (pedestrian, car, motorcycle) use non-max suppression to generate final predictions.

Screen clipping taken: 2018-08-22 23:21

INPUT | Step 1 | Step 2 | Step 3 (OUTPUT)

Screen clipping taken: 2018-08-22 23:22

SUMMARY:

YOLO is one of the best computer vision algorithm incorporating the best ideas of object detection literature.

Region Proposals

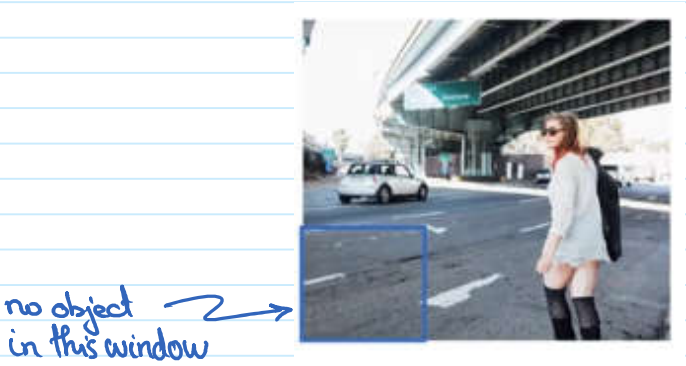
Wednesday, 22 August, 2018 23:27

Goal: Learn about a different way to do object detection.

[Girshik et. al, 2013, Rich feature hierarchies for accurate object detection and semantic segmentation]

Screen clipping taken: 2018-08-22 23:28

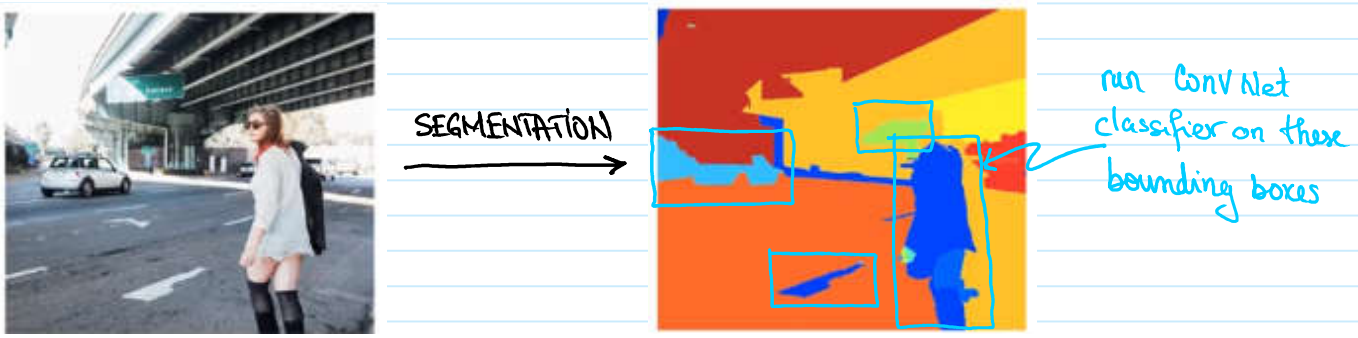
One issue associated with the standard object detection (using convolution) is that it runs convolution on regions of the image where there is clearly no object.



Screen clipping taken: 2018-08-22 23:30

R-CNN (REGIONS WITH CONVOLUTIONAL NN):

- Run the CNN only on a few regions of the image instead of the entire image.
- The region selection is done by running a segmentation algorithm to determine what could be objects



Screen clipping taken: 2018-08-22 23:36

- The segmentation algorithm will output ~2,000 bounding boxes on which we would run our classifier.
  - → this decreases the number of position drastically.
  - → especially if we are running the Conv Net on different bounding box shapes (skinny or wide) or different scales.

- The basic R-CNN algorithm is still quite slow!

FASTER ALGORITHMS

ORIGINAL

- [Girshik et. al, 2013. Rich feature hierarchies for accurate object detection and semantic segmentation]

R-CNN:	Propose regions. Classify proposed regions one at a time. Output label + bounding box.

Screen clipping taken: 2018-08-22 23:44

this is a new bounding box determined by running the classifier on the proposed region bounding box & finding an object

FASTER V1

- [Girshik, 2015. Fast R-CNN]

Fast R-CNN:	Propose regions. Use convolution implementation of sliding windows to classify all the proposed regions.

Screen clipping taken: 2018-08-22 23:46

FASTER V2

- [Ren et. al, 2016. Faster R-CNN: Towards real-time object detection with region proposal networks]

Faster R-CNN: Use convolutional network to propose regions.

Screen clipping taken: 2018-08-22 23:50

→ this is still slower than YOLO.

Week 4 Face Recognition

Thursday, 23 August, 2018 17:50

* What is face recognition

Friday, 24 August, 2018 15:36

Demo of Face Recognition & Liveness detection (detect if the face is of a live human or just a picture)

FACE VERIFICATION VS FACE RECOGNITION

Verification (aka one-to-one problem) 1:1

- Input image, name/ID
- Output whether the input image is that of the claimed person

Screen clipping taken: 2018-08-24 15:49

Recognition 1:K

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

Screen clipping taken: 2018-08-24 15:51

Recognition problem is much harder than Verification problem.

→ Assume we have a verification system that is 99% accurate

→ Assume $K = 100$ (i.e. we have 100 people to choose from in our database)

→ If we use the verification system to check if a face is recognized

- For each face query we have 100 faces in the database to compare against.
- The probability of making no mistakes in 100 verifications is:

$$(99\%)^{100} = 36.6\%$$

→ We need higher accuracy for the verification algorithm if we are to use it with the Recognition algorithm.

Accuracy exploration IMI

Saturday, 25 August, 2018 13:19

Someone posted this table in the forum:

	A	B	C	D	E	F	G
1	Failure Rate		Attempt Size (K)				
2			1	10	100	1,000	10,000
3	Accuracy	90.0000%	10.00%	65.13%	100.00%	100.00%	100.00%
4		99.0000%	1.00%	9.56%	63.40%	100.00%	100.00%
5		99.9000%	0.10%	1.00%	9.52%	63.23%	100.00%
6		99.9900%	0.01%	0.10%	1.00%	9.52%	63.21%
7		99.9990%	0.00%	0.01%	0.10%	1.00%	9.52%
8		99.9999%	0.00%	0.00%	0.01%	0.10%	1.00%

Screen clipping taken: 2018-08-24 16:06

If a method is 90% accurate it means if we call it 10 times

9 times it is correct

1 time it is wrong

In other words if we call the method once we have a 0.1 or $\frac{1}{10}$ chance of it being wrong.

- What is the chance of it being wrong at least once in 10 calls?

  In this case we use the shortcut of calculating the chance α of it being correct in all 10 calls and then taking the complement of it $(1-\alpha)$ to find the answer. This gets around counting all the ways to get at least one wrong answer.

  Probability all 10 calls are correct $= \alpha =$

  $= (P\ 1^{st} \text{ call correct})$ AND $(P\ 2^{nd} \text{ call correct})$ AND ... $(P\ 10^{th} \text{ call correct}) =$

  $= \frac{9}{10} \times \frac{9}{10} \times \dots \frac{9}{10} =$

  $= \left(\frac{9}{10}\right)^{10} = 0.35 = \alpha$

Probability at least one call in 10 is wrong = $1 - \alpha = 0.65$

NOTE: This assumes each call (guess) is independent of the other calls.

- What is the chance of at least one call in 100 being wrong?

$$1 - \left(\frac{9}{10}\right)^{100} = 1.00 \quad (100\%)$$

If we generalize for a given accuracy x the chance of at least one wrong call in Y calls being wrong is: $\boxed{1 - x^Y}$

When we use Recognition with verification if a person needs to be identified successfully their face needs to be compared against all the faces in the database without any errors. Say there are Y faces in the database.
If each verification call has an accuracy of x the chance of no wrong calls being made when a person needs to be identified is x^Y.

→ because of the exponent Y (number of faces to compare against) we need a very high accuracy x to make sure we don't make wrong calls.

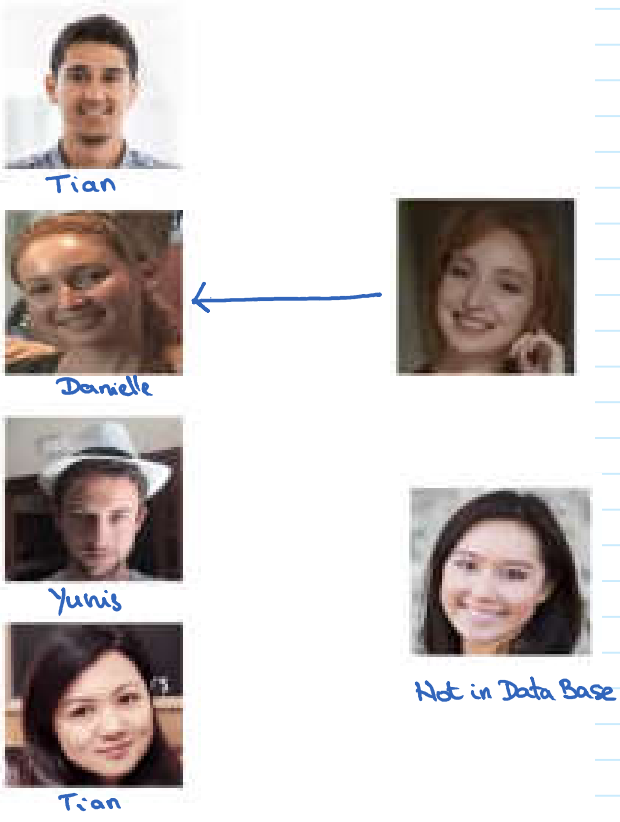
One shot learning

Saturday, 25 August, 2018 13:29

Goal: Understand the effect of One-shot-learning problem on the implementation of Face Recognition

One-Shot-Learning: train a network to identify a person from a single training example.

→ Deep learning classifiers don't work well when we have a single training example.

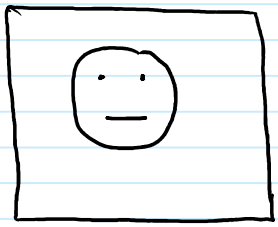


One-Shot-Learning is needed in most cases where we have only one image of an employee available to be used for training.

Screen clipping taken: 2018-08-25 13:43

SOLVING ONE-SHOT-LEARNING PROBLEMS

- APPROACH A



→ ConvNet → 0

(Softmax 5 outputs)

4 employees + unknown

Problem: - Training set has too few elements to train the NN.

- When a new person joins the entire NN needs to get retrained.

- **APPROACH B**

Learn a SIMILARITY FUNCTION:

→ Given two images as input img1 & img2 output the degree of similarity between them.

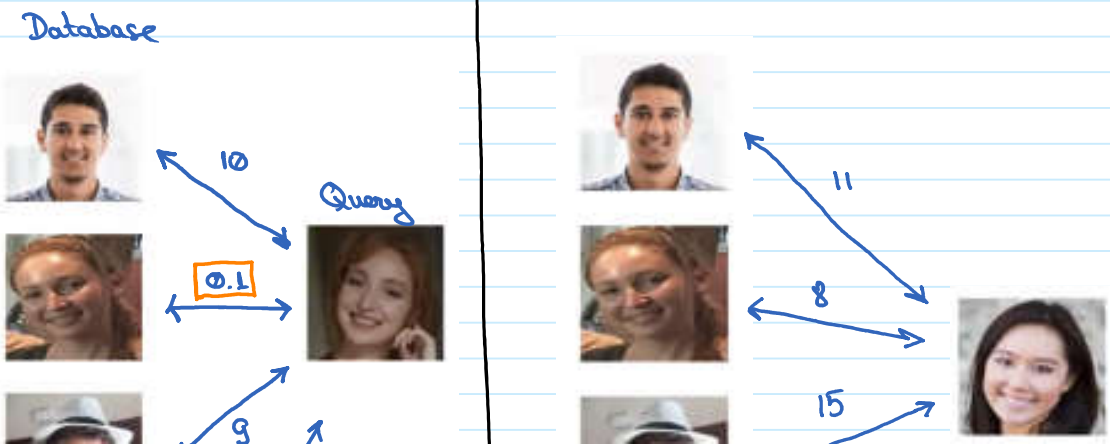
$d(img_1, img_2)$ = degree of difference

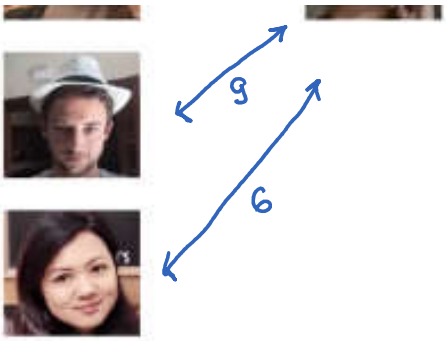
→ same person $d(img_1, img_2) \leq \tau$ (hyperparameter)

→ different persons $d(img_1, img_2) > \tau$

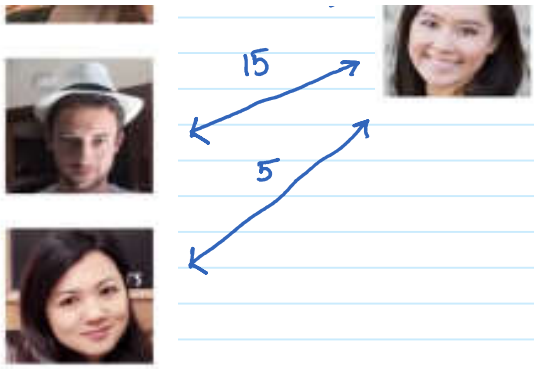
To perform Face Recognition using the Similarity Function we compare the query face to all images in our database and get a similarity score for each pair.

Query img is in Database	Query image is not in Database





There exists a pair with a small difference score



There are no pairs with small difference scores

So as long as we can train a NN to learn a Similarity Function we can then use it to solve the Face Recognition Problem (and the One-Shot-learning problem)

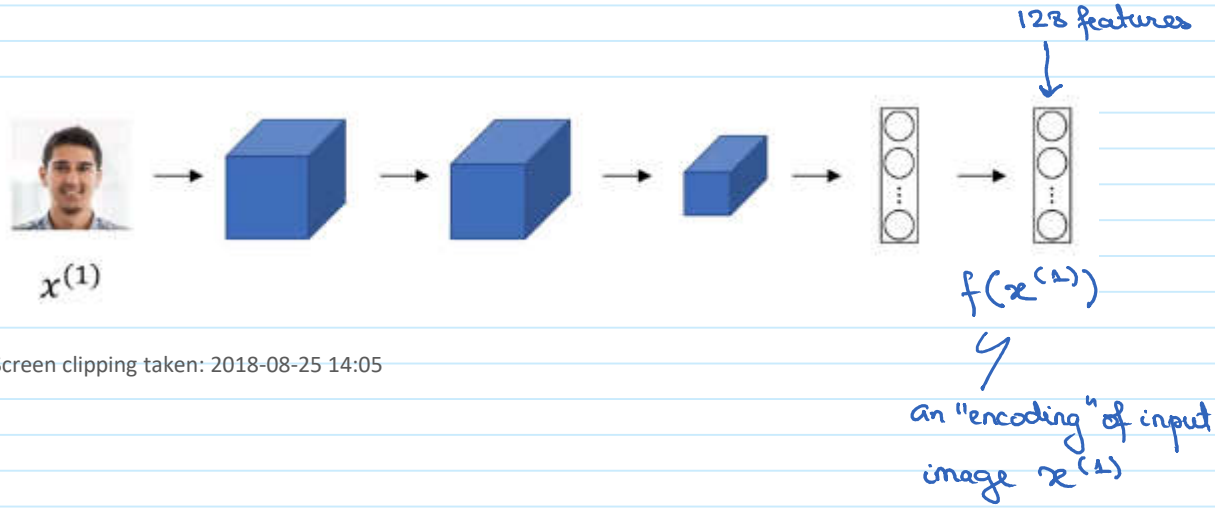
Siamese Network

Saturday, 25 August, 2018 14:03

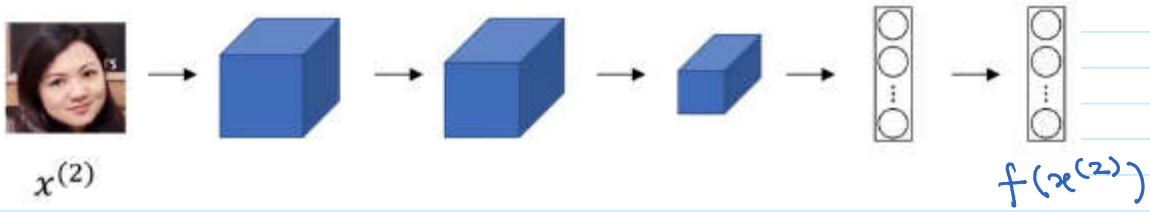
Goal: Learn how to implement a NN that solves the Similarity Function using the Siamese Network architecture.

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

Screen clipping taken: 2018-08-25 14:05



Screen clipping taken: 2018-08-25 14:05



Screen clipping taken: 2018-08-25 14:08

If we want to compare picture $x^{(1)}$ with picture $x^{(2)}$ we use the same NN we used for $x^{(1)}$ to get an "encoding" of picture $x^{(2)}$.

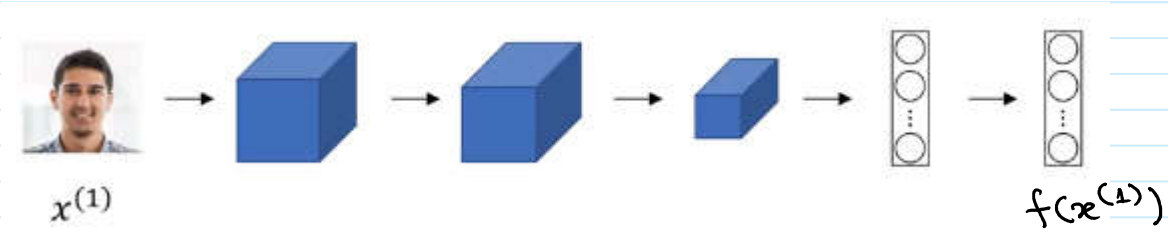
We then calculate the distance between the two images as the L2 Norm between the two encodings of the images.

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

Running two different inputs through two identical Conv NN and comparing the outputs is called SIAMESE NEURAL NETWORK ARCHITECTURE.

GOAL OF TRAINING

We want to train a NN so that its encoded output can be used in a function $d()$ that indicates when the two input pictures are of the same person.



- Parameters of NN determine an encoding $f(x^{(i)})$ (128 features)
- Learn parameters so that if $x^{(i)}$, $x^{(j)}$ are pictures of
  - the same person $\|f(x^{(i)}) - f(x^{(j)})\|_2^2$ is small
  - different persons $\|f(x^{(i)}) - f(x^{(j)})\|_2^2$ is large

So how do we define this objective function?

Triplet Loss

Saturday, 25 August, 2018 14:26

Goal: Learn how to train a NN by defining & applying gradient descent on the triplet loss function so that it gives us an encoding of pictures of faces that we can then use to compare faces with

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

Screen clipping taken: 2018-08-25 14:41

To apply the Triplet Loss function we need to compare pairs of images.

Same person

Similar Encodings

Anchor A

Positive P

Different persons

Different Encodings

Anchor A

Negative N

Screen clipping taken: 2018-08-25 14:42

Triplet Loss terminology:

Distance between Anchor & Positive is small (for same person)

——"—— Anchor & Negative is large (for different persons)

We are always looking at 3 images at a time:

1) Anchor image (A)

2) Positive image (P)

3) Negative image (N)

So we want:

$$\underbrace{\|f(A) - f(P)\|_2^2}_{d(A,P)} \leq \underbrace{\|f(A) - f(N)\|_2^2}_{d(A,N)}$$

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

- A trivial way to solve this is to learn
  - $f() = \vec{0}$ for any input $\Rightarrow \|0 - 0\|^2 - \|0 - 0\|^2 \leq 0$
  - $f() = \vec{k}$ for any input $\Rightarrow \|k - k\|^2 - \|k - k\|^2 \leq 0$

To prevent the NN from learning these cases we'll redefine our learning criterion to:

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

hyperparameter $\alpha > 0$

OR:

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

also known as "margin"
(terminology used in support vector machines)

OR:

$$\|f(A) - f(P)\|_2^2 + \alpha \leq \|f(A) - f(N)\|_2^2$$

We use L2 Norm to determine the distance between two vectors.

In this case we use α to push the $d(A,P)$ and $d(A,N)$ pairs away from each other during training.

Example:

Let $d(A,P) = 0.5$ and $\alpha = 0.2$

then we wouldn't be satisfied with a NN that outputs

$d(A, N) = 0.51$.

we would want at least a $d(A,N) = 0.7$ to achieve the 0.2α margin.

TRIPLET LOSS FUNCTION DEFINITION

Given 3 images A, P, N (anchor, positive - same as anchor, negative - different than anchor) we define the loss $\mathcal{L}$ as follows:

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

So as long as we were able to train a network where the norms differences plus α are less than 0 the $\mathcal{L}$ is 0 (we have no loss)

But if the norms differences plus α are greater than 0 than the loss $\mathcal{L}$ is also greater than 0 and the training will try to minimize it.

The overall cost function for the entire training set is:

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

So if we have a set of 10k images with 1k different people in them we'll generate triplets of A, P & N from the 10k images to train the network.

OBSERVATION:

- We need multiple pictures of the same person to train the network (A & P).
- But after training we don't need to have multiple images of the same person to test the Face Recognition classifier.

How to choose A,P,N triplets?

During training, if A,P,N are chosen randomly, $d(A,P) + \alpha \leq d(A,N)$ is easily satisfied.

Screen clipping taken: 2018-08-26 20:32

This is because for random faces

$$\|f(A) - f(P)\| + \alpha \leq \|f(A) - f(N)\|$$

is most often satisfied by default.

And this means the classifier is not "learning" much.

To make sure we are training the classifier effectively we want to choose A,P,N pairs that are "HARD" to train on.

→ in particular choose A,P,N triplets that result in

$$d(A,P) \approx d(A,N)$$

→ in this case the classifier will try to decrease $d(A,P)$ or increase $d(A,N)$.

→ choosing "hard" triplets results in the training computational efficiency to increase (gradient descent will converge on the minimum faster)

More details are presented in the paper:

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

Naming algorithms trivia:

Domain of inquiry + "Net" or

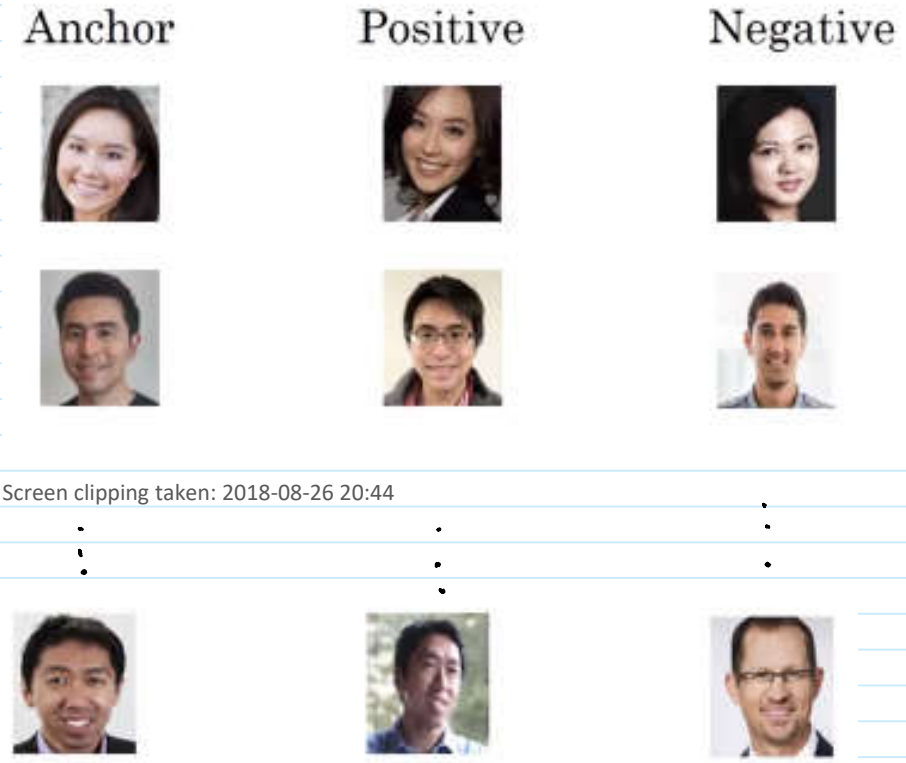
"Deep" + Domain of inquiry

i.e.
Face Net , Deep Face

TRAINING SETS USING TRIPLET LOSS:

Using a training set of triplets we try to minimize cost function J which causes the Gradient Descent to tune the network parameters so that the output encoding for two images will cause $d(x^{(i)}, x^{(j)})$ to be:

- small when images are of the same person
- large when images are of different persons.



Screen clipping taken: 2018-08-26 20:44

Commercial Face Recognition classifiers are trained on 1M+ images or

even 10M or 100M images.

→ these are large data sets that are uncommon for today's standards.

→ some of the companies have released pre-trained classifier that can be used so we don't need to train our own.

→ it's probably better to use these pre-trained models.

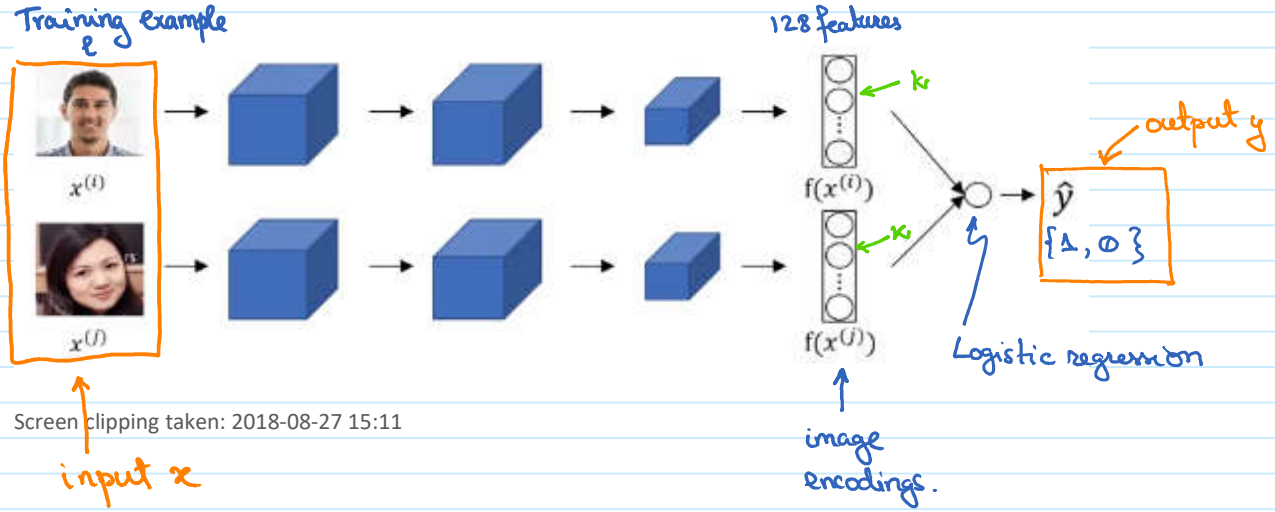
Face verification & binary classification

Monday, 27 August, 2018 15:09

Goal : Learn other methods of training NN to do Face Recognition.

BINARY CLASSIFICATION (ALTERNATIVE TO TRIPLET LOSS)

Train a siamese networks as below:



Screen clipping taken: 2018-08-27 15:11

Feed the encoding output to a logistic regression node which outputs

0 : different person faces

1 : same person faces

Instead of feeding the outputs of the CONV Nets directly into the Logistic Regression node we feed the absolute value of the difference :

$$\hat{y} = \sigma\left(\sum_{k_i=1}^{128} w_i \left| f(x^{(i)})_{k_i} - f(x^{(j)})_{k_i} \right| + b \right)$$

in this case k_i indicates one of the features (out of 128 features) that is being fed into the Logistic Regression node.

Alternative $\hat{y}$:

$$\sum_{k=1}^{128} \left(f(x^{(i)})_k - f(x^{(j)})_k \right)^2$$

Alternative $\hat{y}$:

$$\hat{y} = \sigma\left(\sum_{k=1}^{128} w_k \frac{\left(f(x^{(i)})_k - f(x^{(j)})_k\right)^2}{f(x^{(i)})_k + f(x^{(j)})_k} + b\right)$$

aka χ^2 (chi squared) formula.

These alternatives are explored in:

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

Screen clipping taken: 2018-08-27 15:26

SUMMARY:

We only use two images (instead of three - for Triple Loss) to train & test the network.

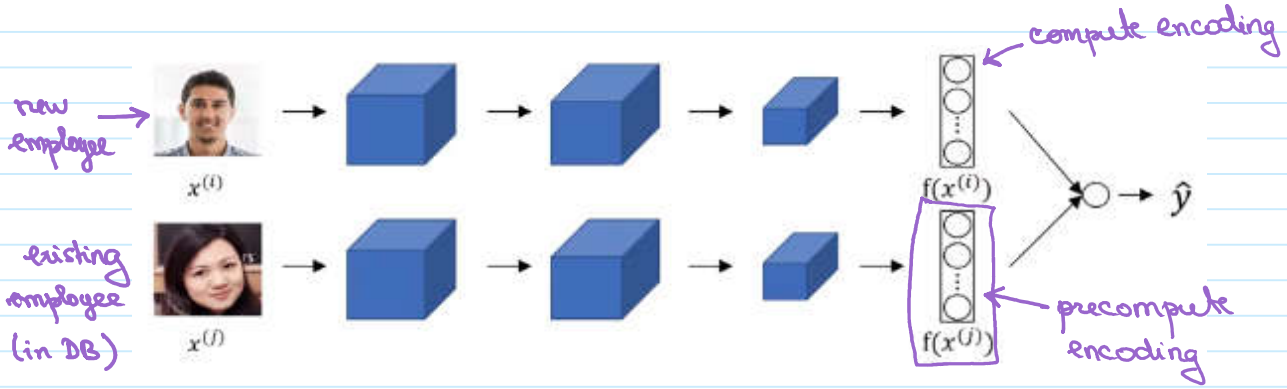
x	y	
	1	"same"
	0	"different"
	0	"different"
	1	"same"

Screen clipping taken: 2018-08-27 15:36

This method also works well.

COMPUTATIONAL TRICK:

When the NN is deployed precompute the encodings for all the current entries in the database so only new employees need their encoding computed at test time.



This way we don't need to store images and we are saving compute at test time.

Precomputing can also be used for the Triple Loss Classifiers.

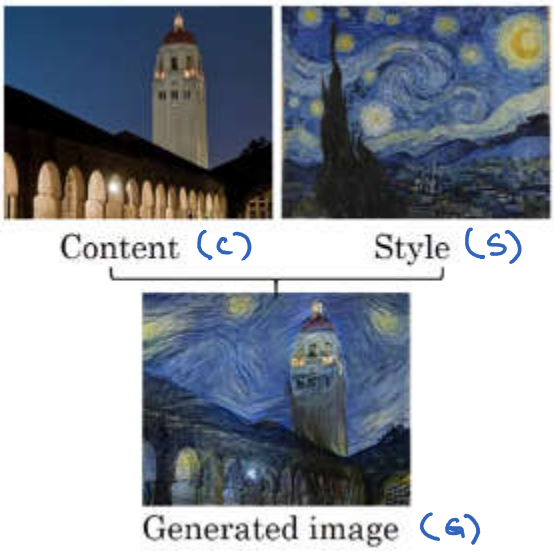
Week 4 Neural Transfer

Thursday, 23 August, 2018 17:52

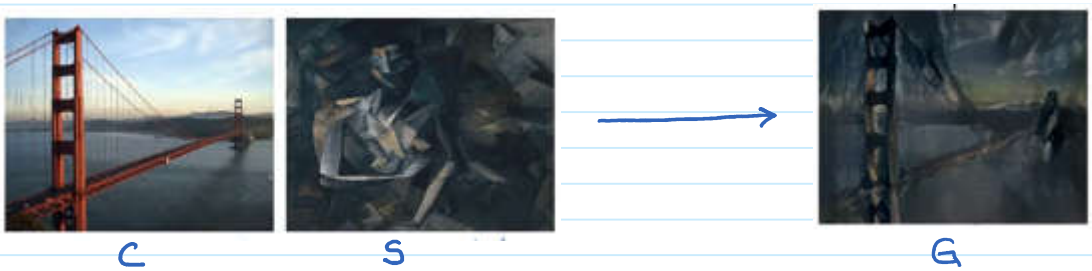
What is Neural Style Transfer?

Monday, 27 August, 2018 15:43

Images generated by Justin Johnson



Screen clipping taken: 2018-08-27 15:39



Screen clipping taken: 2018-08-27 15:40

To understand how Neural Style Transfer works we need to understand the features generated at different levels (both shallow & deep) of a ConvNet.

→ what is each layer outputting?

What are Deep ConvNets learning?

Monday, 27 August, 2018 15:44

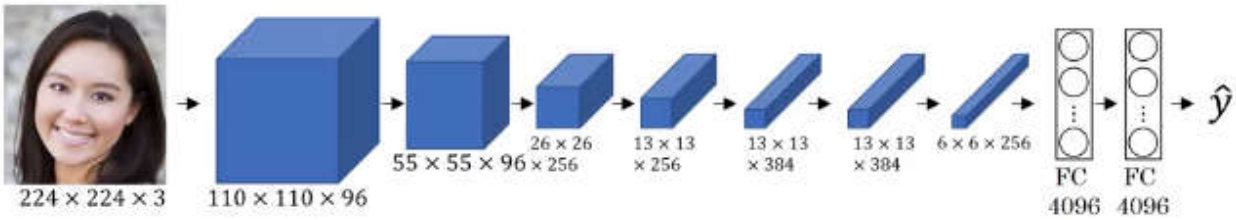
Goal: Learn what each node in a Deep Conv Net is learning.

As described in:

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

Screen clipping taken: 2018-08-27 18:48

Given an Alex Net like Conv Net



Screen clipping taken: 2018-08-27 18:32

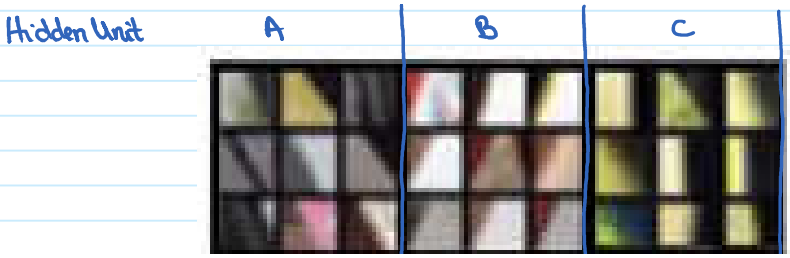
VISUALIZE SHALLOW LAYERS

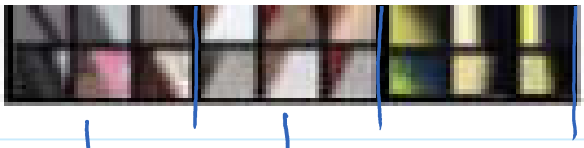
Pick an unit from layer 1 of the trained NN and then pass the training set through the NN and find which are the images that maximize the unit's activation

→ it might be only a portion of each image → an image patch.

A hidden unit in layer 1 will take as input only a small portion of the entire image - so we'll only visualize these small image patches that affect the activation of a single hidden unit.

→ for example these are 9 image patches from 9 training example images that maximize the activation of the hidden unit.



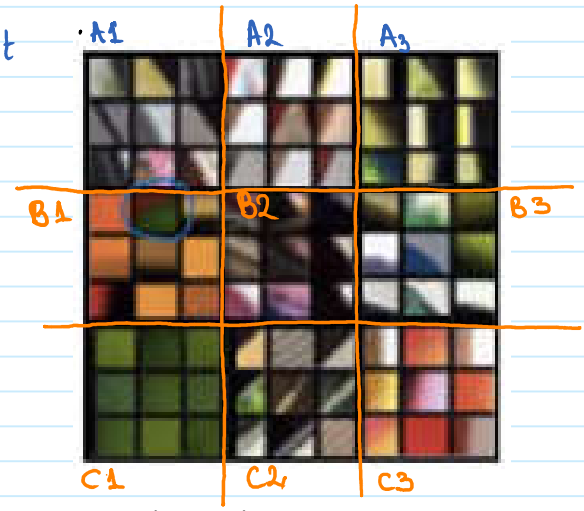


Screen clipping taken: 2018-08-27 18:56

Max Activated by images that look like

More generally:

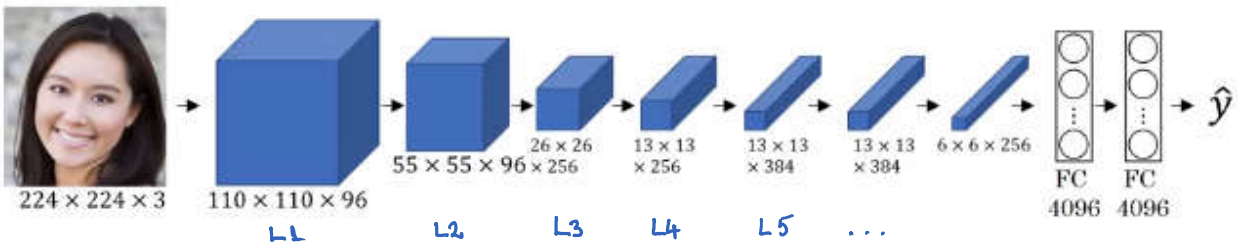
Hidden Unit



Screen clipping taken: 2018-08-27 19:00

So hidden units in Layer 1 appear to have maximum activation when processing simple patterns (vertical, horizontal lines, etc)

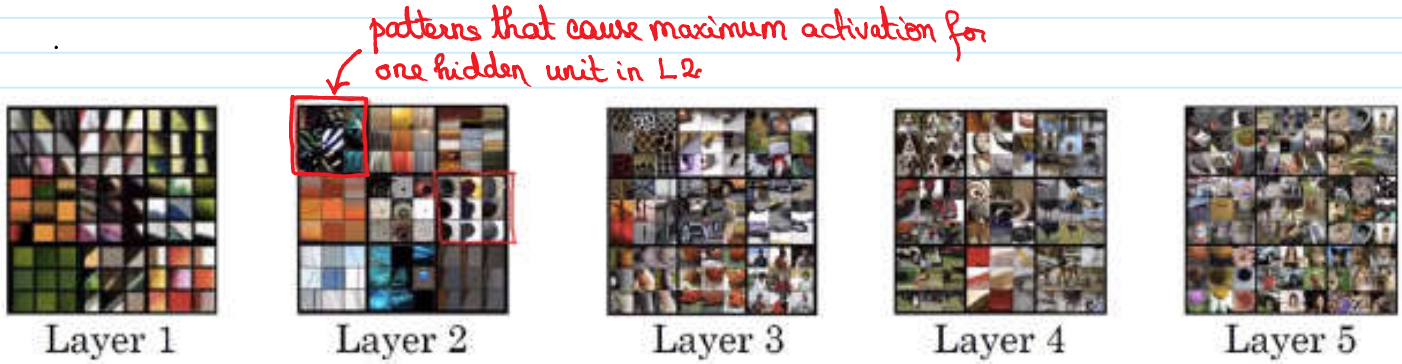
VISUALIZE DEEP LAYERS



In deeper layers a hidden unit is affected by larger image patches from the input image

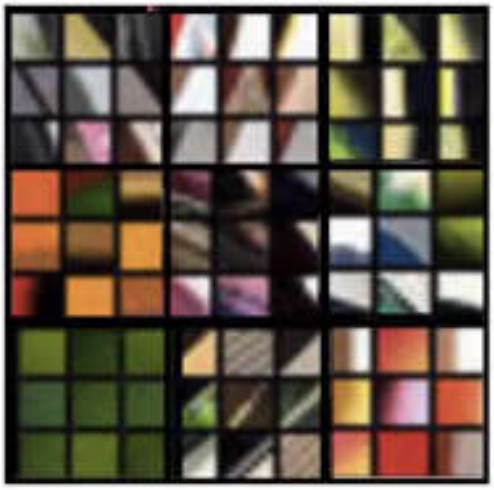
→ for the last Fully Connected layers (the deepest) each unit is affected by the entire image.

NOTE: the images below are not scaled to properly show the larger patch of the input image for the deeper layers.



Screen clipping taken: 2018-08-27 23:44

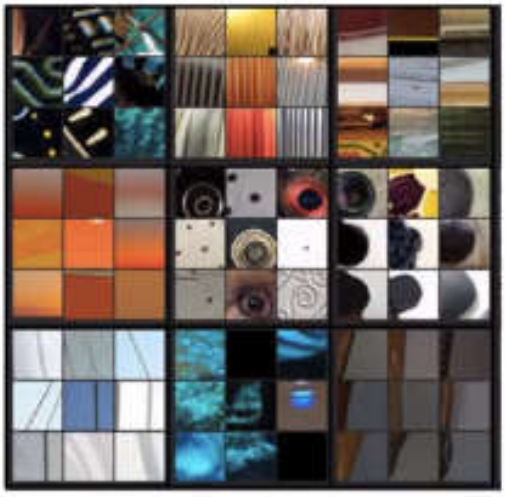
Layer 1



Screen clipping taken: 2018-08-27 23:48

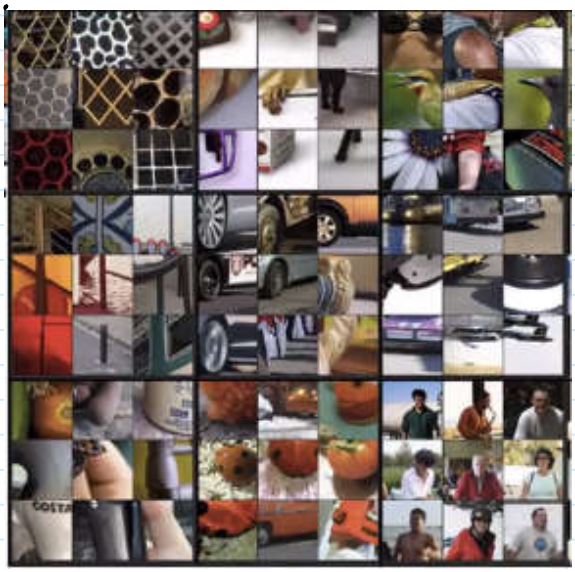
Layer 2

Layer 2



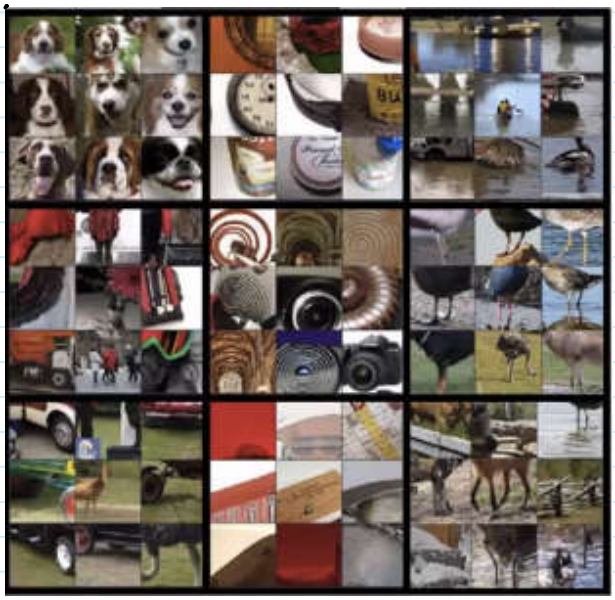
Screen clipping taken: 2018-08-27 23:49

Layer 3



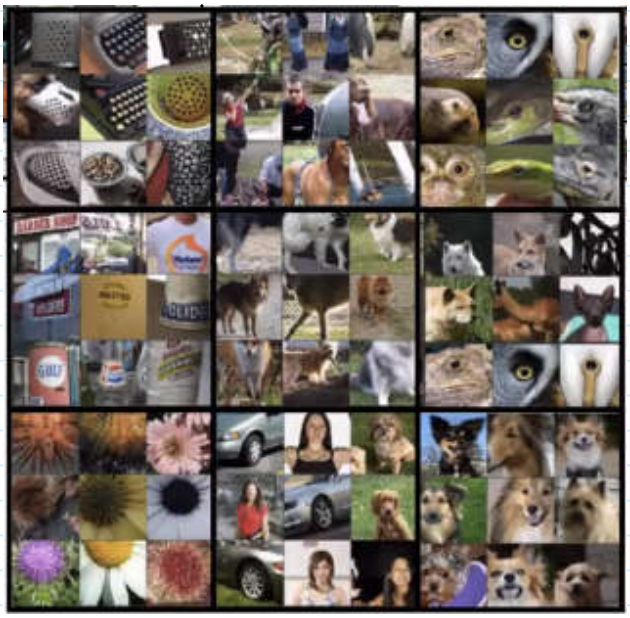
Screen clipping taken: 2018-08-27 23:50

Layer 4



Screen clipping taken: 2018-08-27 23:51

Layer 5



Screen clipping taken: 2018-08-27 23:52

As we go deeper in the network the pattern that maximally activate the hidden nodes are becoming more complex and more difficult to describe.

Cost function

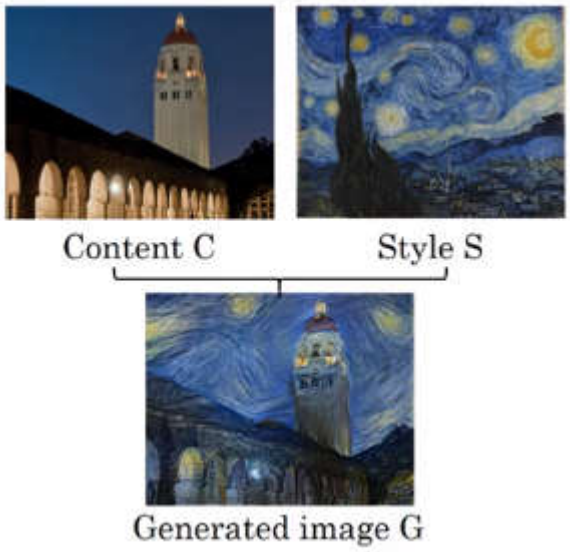
Tuesday, 28 August, 2018 00:06

Goal: Define the cost function that will be used to generate the output of Neural Style Learning

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

Screen clipping taken: 2018-08-28 00:09

Problem formulation: Generate an image G given a content image C and a style image S.



Screen clipping taken: 2018-08-28 00:10

We define a cost function $J(G)$ that evaluates how good the generated image is and it is used by Gradient Descent to train the networks.

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

how similar is image G to the content image C

how similar is image G to the style of image S.

Note: Only one weight (α or β) is needed but original paper used both.

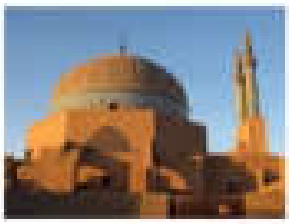
BUILD GENERATED IMAGE G:

1) Initiate G randomly

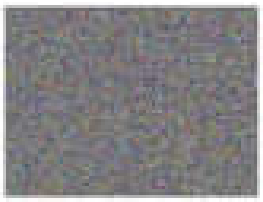
$G: 100 \times 100 \times 3$ ← RGB

2) Use gradient descent to minimize $J(G)$

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



Screen clipping taken: 2018-08-28 00:18



1

2

Screen clipping taken: 2018-08-28 00:19

Content cost function

Tuesday, 28 August, 2018 00:20

Goal: Learn how to define the Content cost component of the Neural Style Transfer Cost Function

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

Screen clipping taken: 2018-08-28 00:23

Overall cost function:

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

Screen clipping taken: 2018-08-28 00:23

CONTENT COST COMPONENT:

- If we use a shallow layer to compute the content cost we impose simple patterns similar to the content image to be used
  i.e. the generated image will resemble the content image even for simple patterns

- If we use a deeper layer to compute the content cost we are asking for more complex patterns from the content image to be present in the generated image, but not necessarily the simple patterns.
  i.e. if there's a "dog" shape in the content image there should be a "dog" shape in the generated image.

So we'll use some middle layer (neither too shallow, nor too deep) to compute the content image.

1) Choose a hidden layer l to compute content cost.

2) Use a pre-trained Conv Net (i.e. VGG16) to obtain a generated image

3) Determine how similar the content image C is to the generated image G to incentivize the Gradient Descent to make image G similar to C.

→ Let $a^{[l](C)}$ & $a^{[l](G)}$ be the activations of layer l when passed the Content & Generated images as input in the NN.

→ if $a^{[l](C)}$ & $a^{[l](G)}$ are similar, we say both images have similar content.

$$J_{Content}(C,G) = \frac{1}{2} \| a^{[l](C)} - a^{[l](G)} \|_2^2$$

constant doesn't matter much cause it's supercaded by α.

these are both vectors that we need to unroll to compute L2 Norm

(element wise sum of squared differences of the two activation vectors)

Style cost function

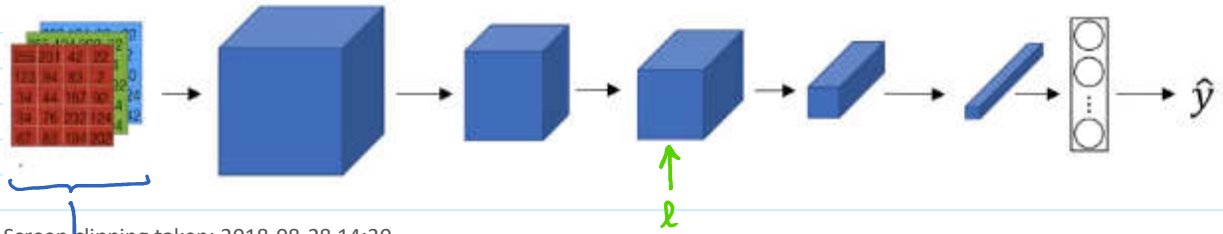
Tuesday, 28 August, 2018 14:25

Goal : Learn how to define the Style Cost component of the Neural Style Transfer Cost function

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

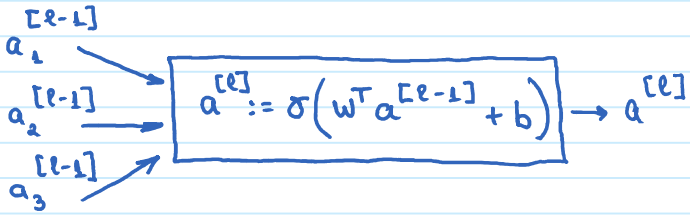
Screen clipping taken: 2018-08-28 14:29

WHAT DOES THE "STYLE" OF AN IMAGE MEAN?



Screen clipping taken: 2018-08-28 14:30

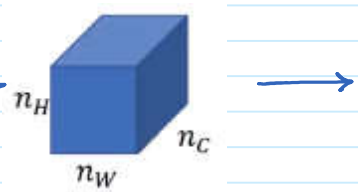
INPUT IMAGE

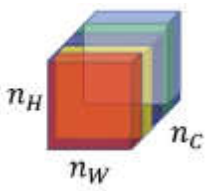


- Let's say we choose layer l's activations to measure "style" on.
- Define style as correlation between activations accross the different channels of layer l's

  How correlated are the activations accross different channels?

layer l's activations

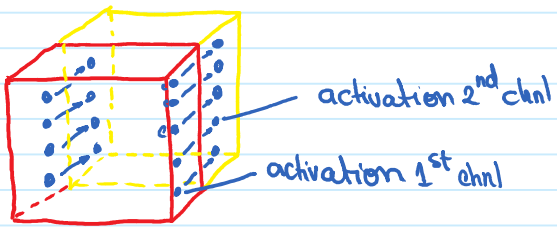




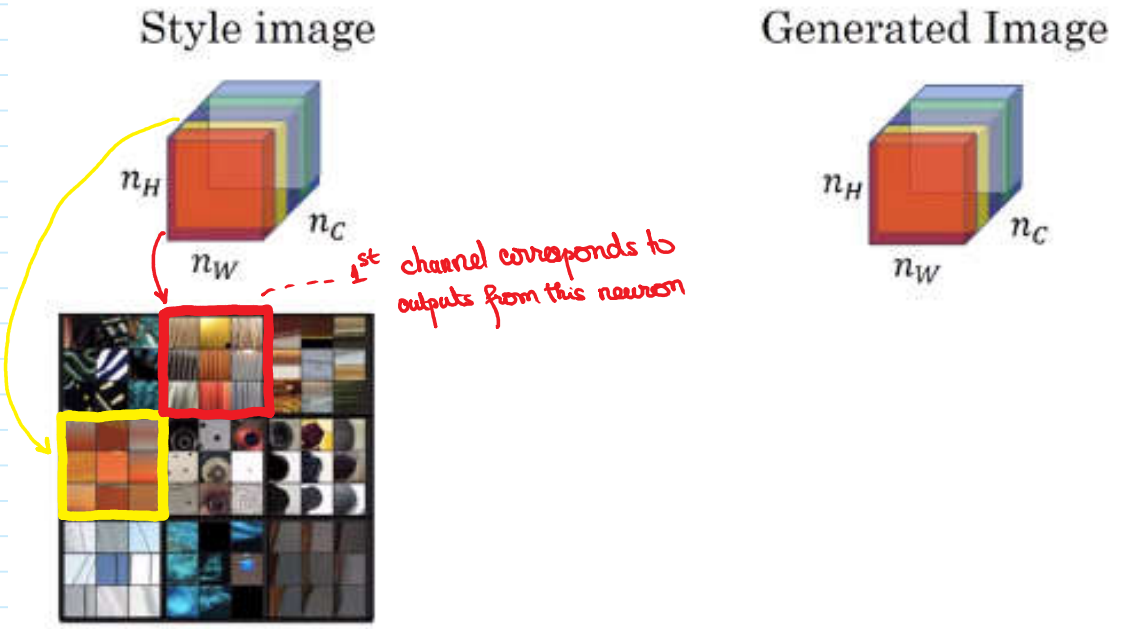
(in practice there are many more channels.

Screen clipping taken: 2018-08-28 14:39

Looking at the first two channels measure the correlation between the equivalent activations ($n_H \times n_W$)



Intuition about why correlation of activation captures style



Screen clipping taken: 2018-08-28 15:16

- If chnl 1 activations are correlated with chnl 2 activations whenever chnl 1 has a vertical texture chnl 2 has an orange tint.
- If they are not correlated this paired pattern (vertical ↔ orange tint) is not present.

The correlation tells us which of these high level patterns go together.

→ the degree of correlation tells us how often these patterns go together

★ The "style" can be imagined as the degree of correlation in different parts of the image.

→ how often do "vertical textures go with orange hues."

- We then compare the style in the input Style Image (S) with the style in the Generated Image (G) and try to make the styles similar.

FORMAL DEFINITION OF STYLE

Given an image we will compute its **Style Matrix** that contains all the correlations discussed above.

Let $a^{[l]}_{i,j,k}$ denote the activation at position i, j, k in hidden layer l.

$i \rightarrow$ height, $j \rightarrow$ width, $k \rightarrow$ channel indexes

Let $G^{[l]}$ be a square matrix of size $n_c^{[l]} \times n_c^{[l]}$

$n_c^{[l]} \rightarrow$ number of channels in layer l

where $G^{[l]}_{k,k'}$ measures how correlated the activations in channel k are to the activations in channel k'

$\rightarrow k, k' \in [1; n_c^{[l]}]$

Example:

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

where k & k' index over all $n_c^{[l]}$ channels of layer l.

Observations:

- In Linear Algebra G matrices are known as "Gram matrices".
- If $a_{i,j,k}$ & $a_{i,j,k'}$ are large the product is large while if one is large

ant the other one is small the product is smaller.

- Even though we used the term correlation G measures the unnormalized cross covariance because we are not subtracting out the mean and we are only multiplying the numbers.

So we compute the style matrix G for both the Style Image (S) and for the Generated Image (G)

$$G^{[l](S)}_{k,k'} = \sum_{i}^{n_H^{[l]}} \sum_{j}^{n_W^{[l]}} a^{[l](S)}_{i,j,k} \times a^{[l](S)}_{i,j,k'} \quad \text{(style of Style Image)}$$

$$G^{[l](G)}_{k,k'} = \sum_{i}^{n_H^{[l]}} \sum_{j}^{n_W^{[l]}} a^{[l](G)}_{i,j,k} \times a^{[l](G)}_{i,j,k'} \quad \text{(style of Generated Image)}$$

The **Style Cost Function** of layer l is:

$$J^{[l]}_{style}(S,G) = \underbrace{\frac{1}{\left(2 n_H^{[l]} n_W^{[l]} n_C^{[l]}\right)^2}}_{\text{this normalization doesn't matter much because } J_{style} \text{ is weighted by hyperparameter } \beta. \text{ (authors used it though)}} \underbrace{\left\| G^{[l](S)} - G^{[l](G)} \right\|_F^2}_{\text{Because Gs are matrices we use Frobenius Norm}}$$

$$\sum_{k} \sum_{k'} \left(G^{[l](S)}_{k,k'} - G^{[l](G)}_{k,k'} \right)^2$$

STYLE COST FUNCTION:

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

Screen clipping taken: 2018-08-28 16:47

It turns out we get more visually pleasing results if we use the Style Cost Function from multiple layers

→ Shallow layers measure correlations of simple visual features (i.e. edges)

→ deeper layers measure correlations of more complex features (i.e. vertical striations or dog shapes)

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

↑ hyperparameter allows us to weight different layers' styles differently

OVERALL COST FUNCTION

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

Use Gradient Descent to minimize it.

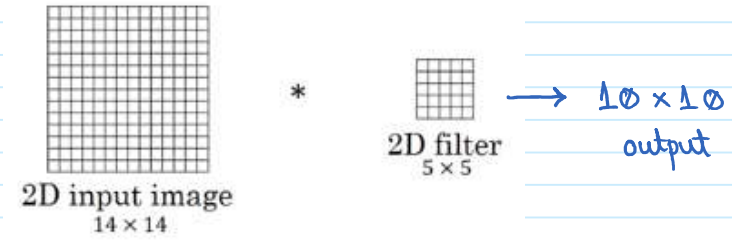
1D and 3D generalization

Tuesday, 28 August, 2018 16:57

Goal: Learn how to apply convolution on 1D & 3D data.

CONVOLUTION IN 2D & 1D

- 2D convolution:



Screen clipping taken: 2018-08-28 17:01

OR

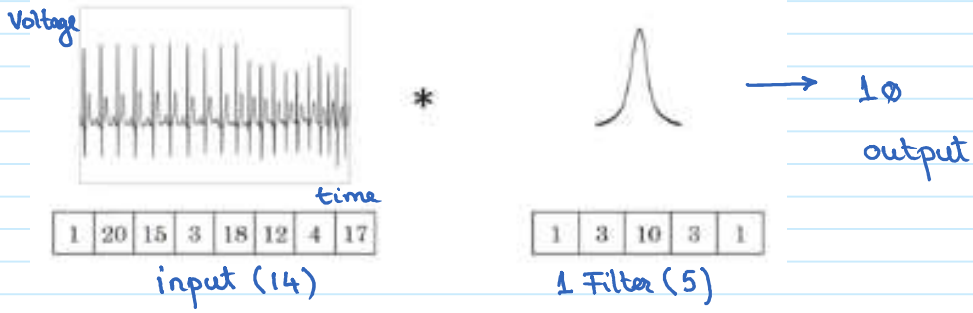
$14 \times 14 \times 3 * 5 \times 5 \times 3 \longrightarrow 10 \times 10 \times 16$

(16 filters)

REMEMBER: each filter is used to detect a specific feature (pattern)

- 1D convolution:

- EKG signals (heart beat)



Screen clipping taken: 2018-08-28 17:03

1st layer: $(14) * (5) \longrightarrow (10 \times 16)$

(16 filters)

2nd layer $(10 \times 16) * (5 \times 16) \longrightarrow (6 \times 32)$

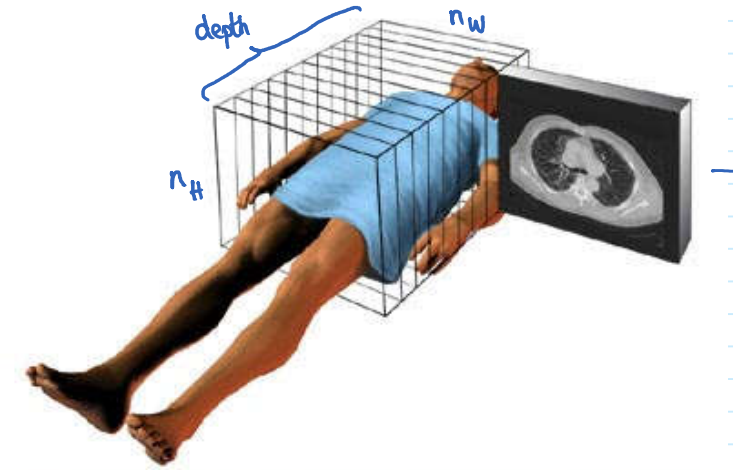
32 filters

NOTE: For many problems containing 1D data (or sets of sequences) we actually use Recurrent Neural Networks (see next course)

CONVOLUTION IN 3D

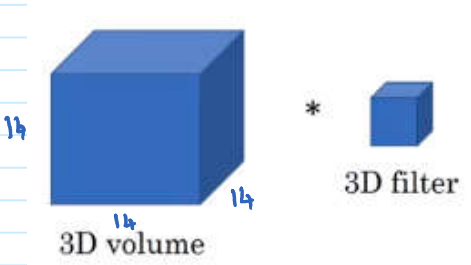
Detecting features (patterns) in 3D data.

- Example 1: CT scan data contains 3D visual volumes



Screen clipping taken: 2018-08-28 17:15

To simplify



Screen clipping taken: 2018-08-28 17:19

1st layer: $(14 \times 14 \times 14 \times 1) * (5 \times 5 \times 5 \times 1) \rightarrow 10 \times 10 \times 10 \times 16$ (16 filters)

H W D # of channels (i.e. different scanning frequency)

2nd layer: $(10 \times 10 \times 10 \times 16) * (5 \times 5 \times 5 \times 16) \longrightarrow 6 \times 6 \times 6 \times 32$

32 filters

- Example 2: Movie data
  - different slices are the frames in the movie → 3rd dimension is time.
  - detect
    - motion
    - people's actions