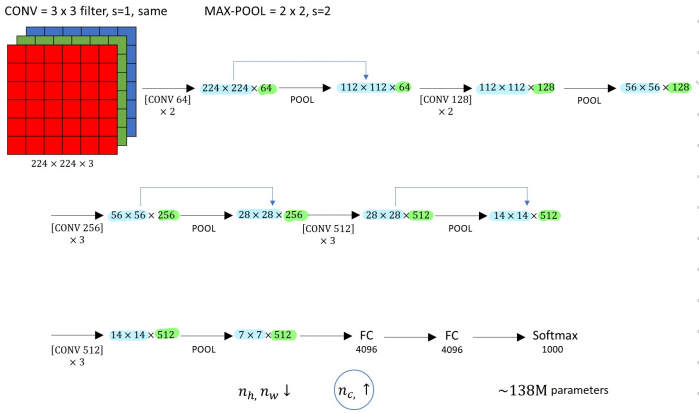
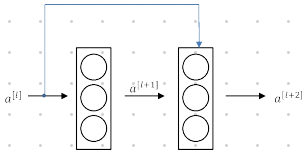
* VGG16: → 16 LAYERS

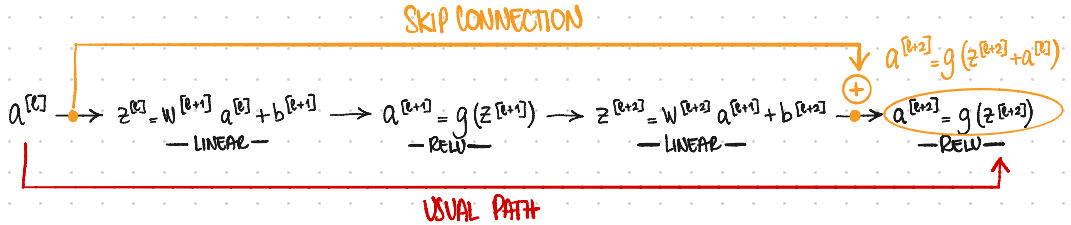



* as it gets deeper, $n_h$ and $n_w$ go ↓ by a factor of 2.
$n_c$ goes ↑ usually by a factor of 2.

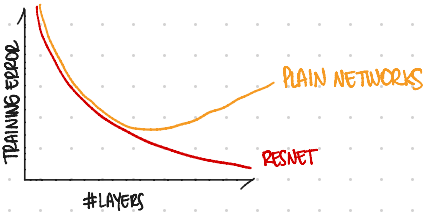
* ResNets:

* ResNets are built over residual blocks

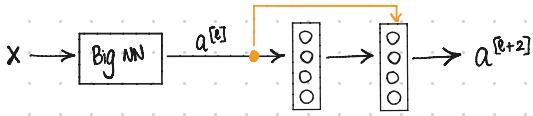



* residual blocks allow you to train bigger networks



* identity functions are easy to learn for residual blocks




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

L2 REGULARIZATION

IF $w^{[l+2]} = 0 \vee b^{[l+2]} = 0 \Rightarrow a^{[l+2]} = g(a^{[l]}) = a^{[l]}$

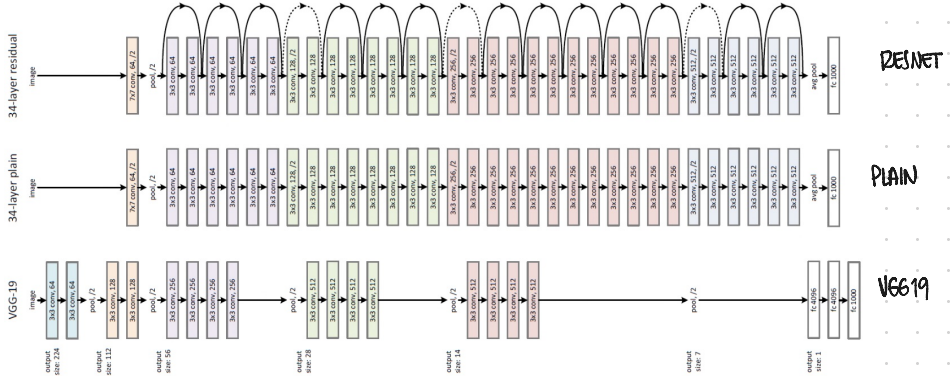
RELU: $a \geq 0$

* it usually uses same convolution, so that it preserves dimensions

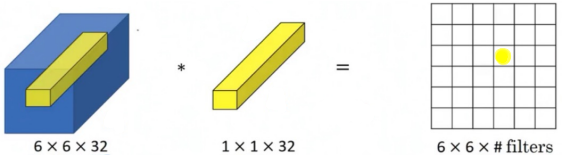
* if $a^{[l+2]}$ happens to be of dimension 256 and $a^{[l]}$ of dimension 128, you add an extra matrix and preserve dimensions:

$$g(w^{[l+2]} a^{[l+1]} + b^{[l+2]} + W_s a^{[l]})$$

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



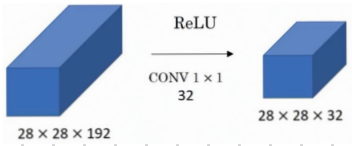

## NETWORKS IN NETWORKS AND 1×1 CONVOLUTIONS




NETWORK IN NETWORK

* one way of thinking about the 32 numbers in the filter with the 32 numbers in the input is as if you had a neuron that is taking an input 32 numbers and multiplying each of these 32 numbers in one slice of the same position highlighted by the 32 ⊛ channels.

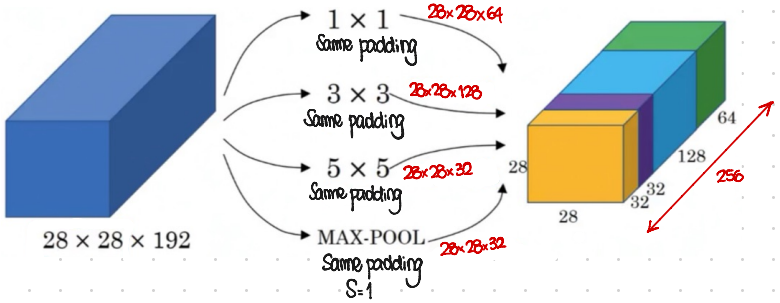
* when is it useful? when you want to reduce the # channels in an input → save computational power.




* it is also useful for building inception (NN)

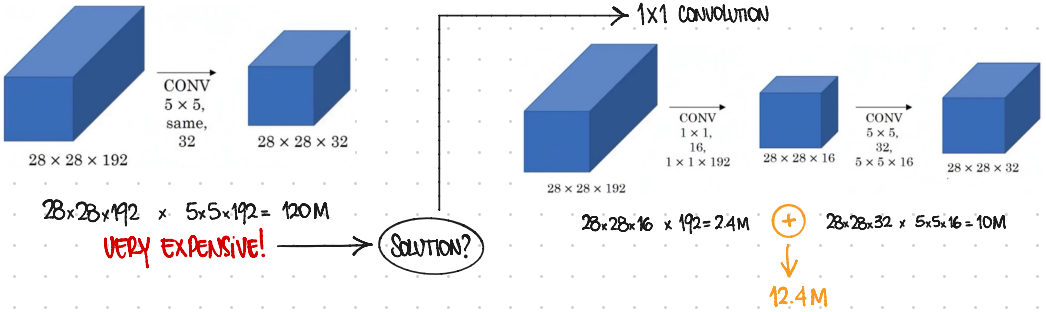
## INCEPTION NETWORK

* instead of choosing what filter size you want in a conv layer, or if you want a conv + pooling layer, YOU DO THEM ALL!

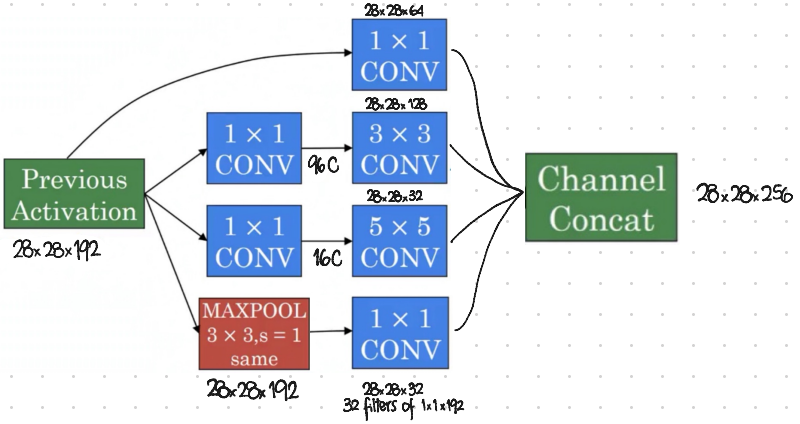



* instead of needing to pick one of these filter sizes or pooling, you can do them all and just concatenate all the outputs and let the (NN) learn whatever parameters it wants to use, whatever combination of filter sizes it wants.

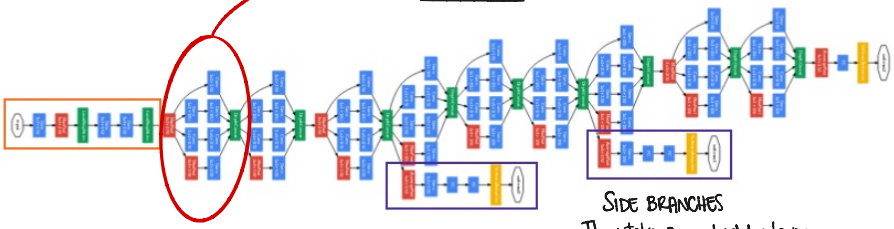
* PROBLEM → computational cost! let's see the 5x5 filter example:



*one inception block:




## GOOLENET



SIDE BRANCHES

They take some hidden layer and tries to make a prediction

It appears to have a regularization effect that prevents overfitting.

## TRANSFER LEARNING

*rather than training the weights from scratch, you often make much faster progress if you download weights someone else has already trained.

* you freeze the parameters in the already-trained layers and you just train the parameters associated with your last layer (e.g. softmax layer).

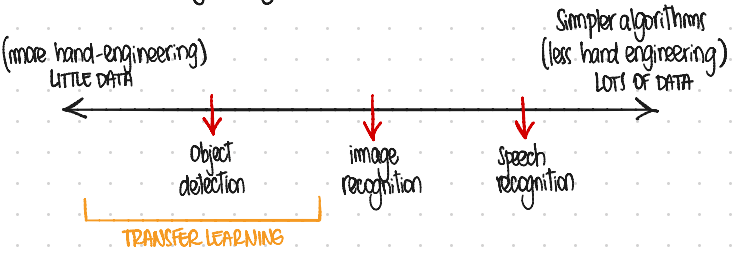
* if you have a large dataset, you can freeze some of the first layers and train the rest.

## DATA AUGMENTATION

* often used to improve the performance on computer vision systems.
* common methods → mirroring
  - random cropping
  - rotation
  - color shifting: you take ± values and add them to the original RGB image

## STATE OF COMPUTER VISION

* data vs. hand-engineering:




* sources of knowledge → labeled data
  - hand-engineered features

* tips for doing well on benchmarks:

(#1) ensembling: train several (NN) independently and average outputs. (it takes more computer memory).

(#2) multi-crop at test time: run classifier on multiple versions of the test images and average results.

## OBJECT LOCALIZATION

* figuring out where in the picture is the object we want to locate.
* usually the image has ONE object that you need to localize. If it has >2, it's object detection.
* you should change your (NN) to have a few more output units that output a bounding box.

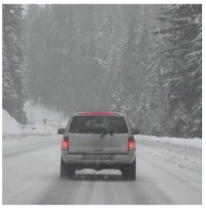
* target label $y$, subject to 4 classes (1: person, 2: car, 3: motorcycle, 4: none)

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

$p_c$: IS THERE AN OBJECT? (PROBABILITY THAT ONE OF THE CLASSES IS THERE)

$b_x, b_y, b_h, b_w$: BOUNDING BOX LOCATION

$c_1, c_2, c_3$: CLASSES



$$y = \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

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

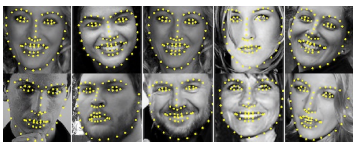
$y_8$: # OF ELEMENTS IN VECTOR $y$

* you care about how good your (NN) is at identifying $y_1$ → IS THERE AN OBJECT?
If there is none, if $y_1 = 0$, then you don't care about the other measures: $(\hat{y}_2 - y_2)^2 + (\hat{y}_3 - y_3)^2 \dots$

* In practice, one usually uses log likelihood loss, squared error, and logistic regression loss.
  - log likelihood loss → CLASSES
  - squared error → BOUNDING BOX
  - logistic regression loss → $y_1$, $p_c$: IS THERE AN OBJECT?

## LANDMARK DETECTION

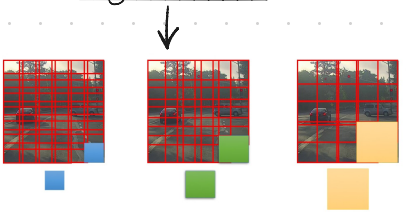
* you can have a (NN) output x and y coordinates of important points in the image → LANDMARKS
* Snapchat, IG filters



* you get labeled data with the coordinates you are interested in.
* you can do the same with poses pictures.

# OBJECT DETECTION

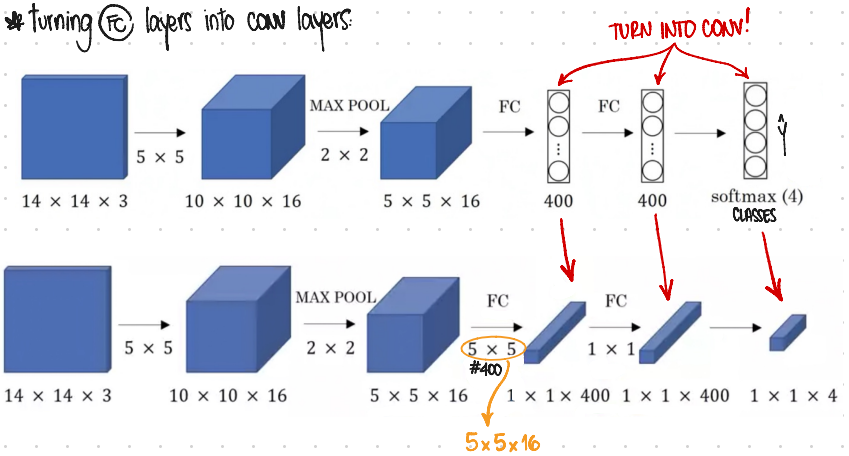
* You train a (NN) and then use a Sliding window detection.



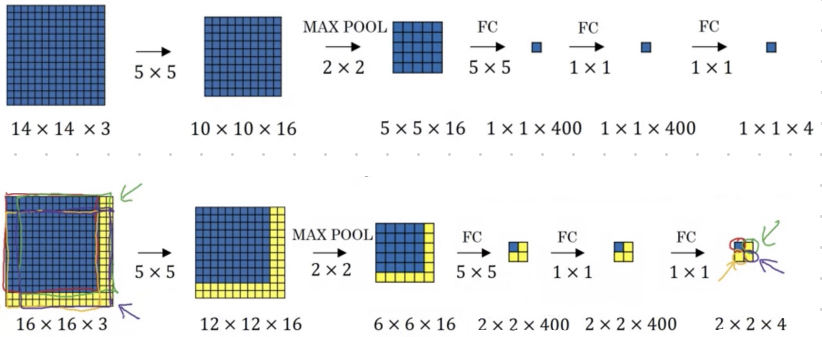
* You feed the trained (NN) just a portion of the image. Then you use a larger window and feed it to the (NN).

* Sliding windows can be computationally expensive! How to solve this? CONVOLUTIONAL IMPLEMENTATION!

* Turning (FC) layers into conv layers:




* convolution implementation for sliding windows:







* instead of forcing you to run 4 propagations on 4 subsets of the input image independently, it combines all 4 into one form of computation and shares a lot of computation in the region of the image that are common.