Convolutional Neural Networks

1. Foundations of convolutional neural networks (CNN)

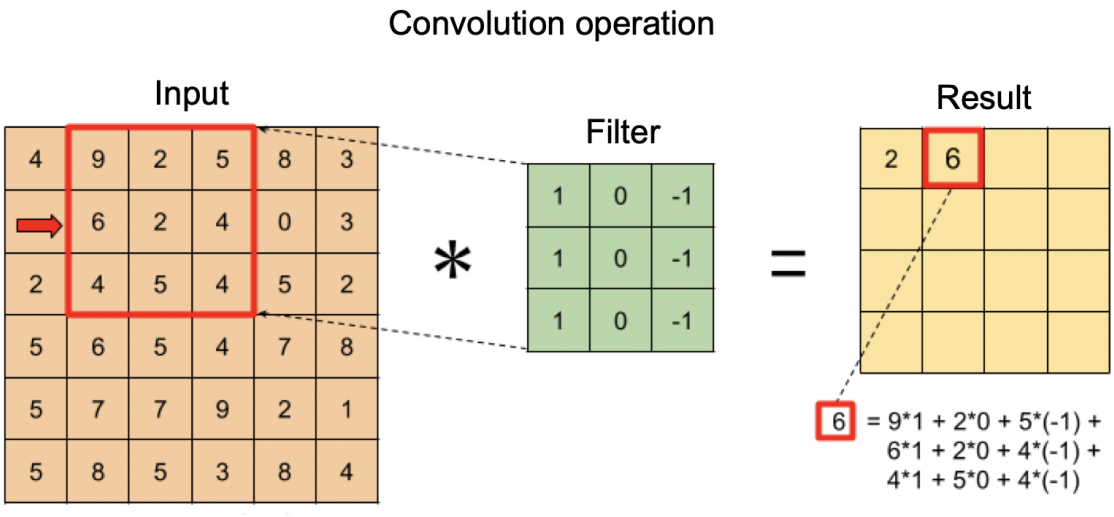
1.1. Computer vision

- Why computer vision
  - Rapid advances in computer vision are enabling brand new applications
  - Cross-fertilization into other areas, because the computer vision research community has been so creative and so inventive in coming up with new neural network architectures and algorithms
- Computer vision applications
  - Image classification
  - Object detection
  - Neural style transfer

- One challenge of computer vision: the inputs can get really big
    - Easily overfitting
    - Infeasible computational and memory requirements

1.2. Convolution operation

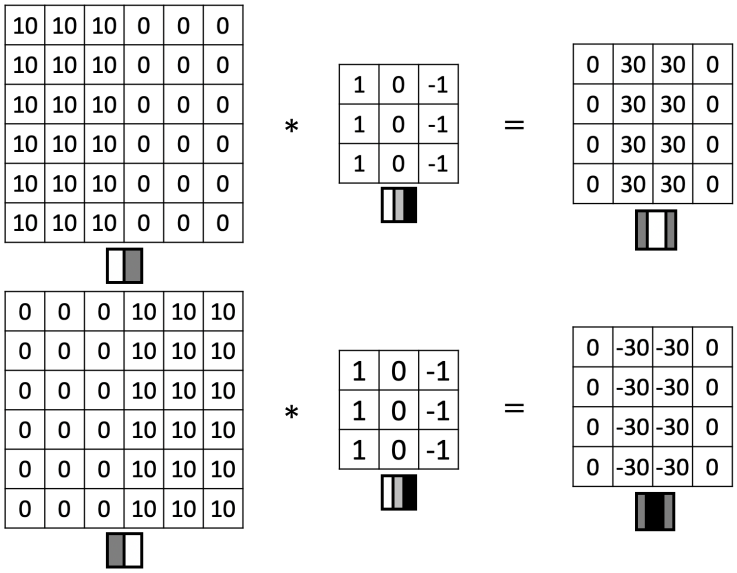
- How convolution operation works



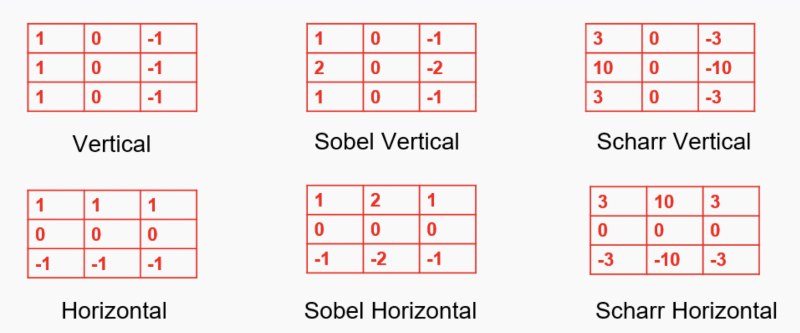
Thechnically, this operation is called cross correlation by mathematicians

- Code
    - Python: `conv_forward`
    - TensorFlow: `tf.nn.conv2d`
    - Keras: `Conv2D`
- Edge detection, an example application of convolution
    - Edge example

    - Vertical edge detection



- Hand-coded edge detection filters



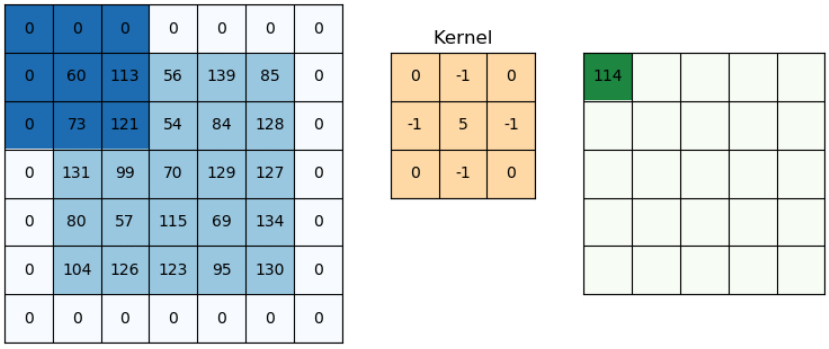
Dimension: $f \times f$, where f is usually odd

- Learn the filter parameters from neural network

1.3. Padding

- Downside of convolution operation
  - (-) Shrinking output
  - (-) throwing away information from edges
- Padding on the input matrix

  Common to zero-pad the border. In the example below, the padding p = 1.



- Valid and same convolutions
  - **Valid convolution:** no padding

$$n \times n \ image \quad * \quad f \times f \ filter \quad \rightarrow \quad (n-f+1) \times (n-f+1) \ output$$

  - **Same convolution:** pad so that output size is the same as the input sizes

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

- Padding conventions

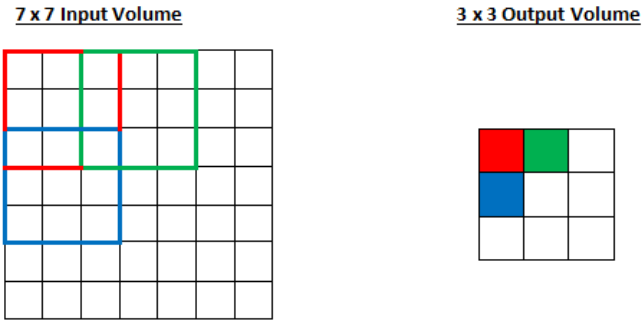
  The most commonly used padding mode in CNN is "same" padding.
  - Same padding preserves the height and width of the input images or tensors, which make designing a network architecture more convenient.
  - For valid padding, the volume of the tensors would decrease substantially in neural networks with many layers, which can be dangerous to the network performance.
  - In practice,it is recommended that you preserve the spatial size using same padding for the convolutional layers and decrease the spatial size via pooling layers instead.

- Advantages of padding
  - (+) Allows you to use a CONV layer without necessarily shrinking the height and width of the volumes. This is important for building deeper networks, since otherwise the height/width would shrink as you go to deeper layers.
  - (+) Helps us keep more of the information at the border of an image. Without padding, very few values at the next layer would be affected by pixels as the edges of an image.

1.4. Strided convolution

- Stride example

  In the example below, stride s = 2.

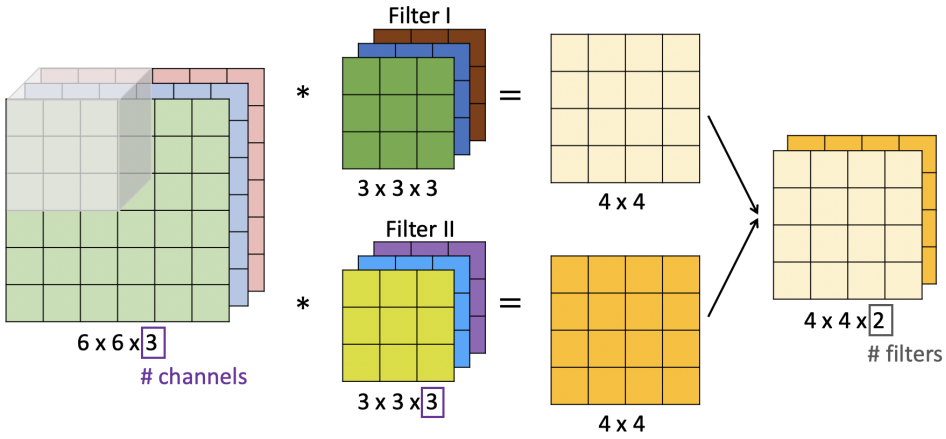


Output dimension: $\left\lfloor \frac{n+2p-f}{s}+1 \right\rfloor \times \left\lfloor \frac{n+2p-f}{s}+1 \right\rfloor$

By convention, the filter must be fully contained in the input image to do convolution.

1.5. Convolution layer

- Convolutions over volume



$$n \times n \times n_c \quad * \quad f \times f \times n_c \quad \rightarrow \quad (n-f+1) \times (n-f+1) \times n_c'$$

- Notation
  - $f^{(l)}$: filter size
  - $p^{(l)}$: padding size
  - $s^{(l)}$: stride size
- Dimensions
  - Input: $n_H^{(l-1)} \times n_W^{(l-1)} \times n_C^{(l-1)}$
  - Output: $n_H^{(l)} \times n_W^{(l)} \times n_C^{(l)}$, where

$$n_H^{(l)} = \left\lfloor \frac{n_H^{(l-1)} + 2p^{(l)} - f^{(l)}}{s^{(l)}} + 1 \right\rfloor$$

$$n_W^{(l)} = \left\lfloor \frac{n_W^{(l-1)} + 2p^{(l)} - f^{(l)}}{s^{(l)}} + 1 \right\rfloor$$

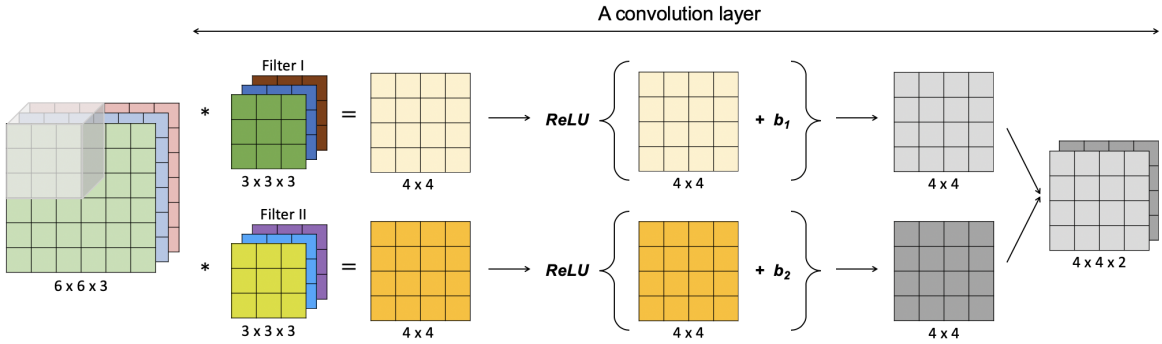
- Each filter: $f^{(l)} \times f^{(l)} \times n_C^{(l-1)}$

  Weights: $f^{(l)} \times f^{(l)} \times n_C^{(l-1)} \times n_C^{(l)}$

  Bias: $1 \times 1 \times 1 \times n_C^{(l)}$

- Activation: $a^{(l)} \rightarrow n_H^{(l)} \times n_W^{(l)} \times n_C^{(l)}$

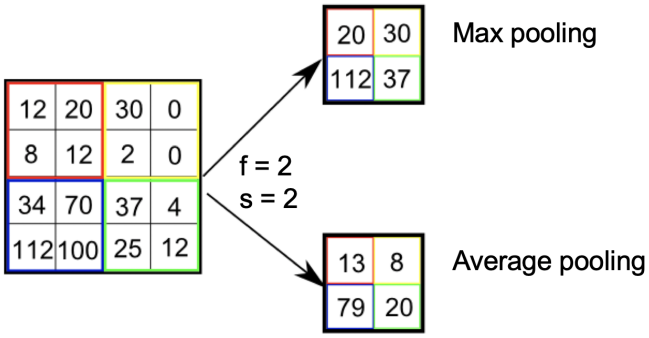
  Activation vectorized: $A^{(l)} \rightarrow m \times n_H^{(l)} \times n_W^{(l)} \times n_C^{(l)}$



1.6. Pooling layer

- Hyperparameters
  - f: filter size
  - s: stride size
  - Max or average pooling
  - Pooling usually does not use any padding

- Pooling
  - **Max-pooling layer:** slides an (f, f) window over the input and stores the max value of the window in the output.
  - **Average-pooling layer:** slides an (f, f) window over the input and stores the average value of the window in the output.



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

  - Applies to each channels independently
  - No parameters for backpropagation to learn

- Advantages of pooling in ConvNet

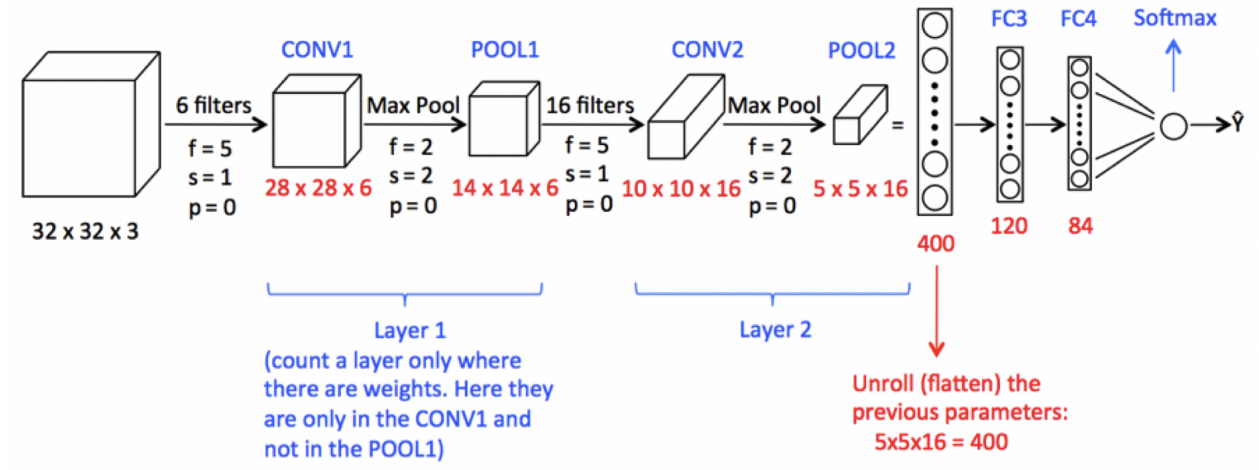
  - (+) Reduces the size of the input
  - (+) Speeds up the computation
  - (+) Makes feature detectors more invariant to its position in the input

1.7. CNN schematics

- Types of layer in a convolutional network

  - Convolution (CONV)
  - Pooling (POOL)
  - Fully-connected (FC)

- Workflow



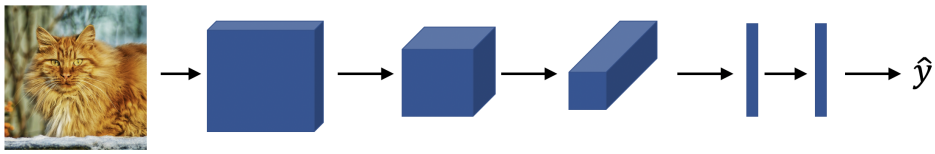
	Activation shape	Activation Size	# parameters
Input:	(32,32,3)	3,072	0
CONV1 (f=5, s=1)	(28,28,8)	6,272	208
POOL1	(14,14,8)	1,568	0
CONV2 (f=5, s=1)	(10,10,16)	1,600	416
POOL2	(5,5,16)	400	0
FC3	(120,1)	120	48,001
FC4	(84,1)	84	10,081
Softmax	(10,1)	10	841

- From left to right, the height and width often decrease, and the number of channels often increase.
- In order to be able to build very deep networks, we usually only use pooling layers to downsize the height/width of the activation volumes while convolutions are used with "same" padding. Otherwise, we would downsize the input of the model too quickly.

1.8. Why convolutions

- Advantages of convolutional layers over fully connected layers
  - (+) **Parameter sharing:** A feature detector (such as a vertical edge detector) that's useful in one part of the image is probably useful in another part of the image.
    - Reduces the total number of parameters, thus reducing overfitting.
    - Allows a feature detector to be used in multiple locations throughout the whole input image/input volume. Good at capturing **translation invariance** (e.g. over different places in a picture) because you are applying the same filter.
  - (+) **Sparsity of connections:** In each layer, each output value depends only on a small number of inputs.
- Putting all together

Training set $(x^{(1)}, y^{(1)}) \dots (x^{(m)}, y^{(m)})$.



$$\text{Cost } J = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

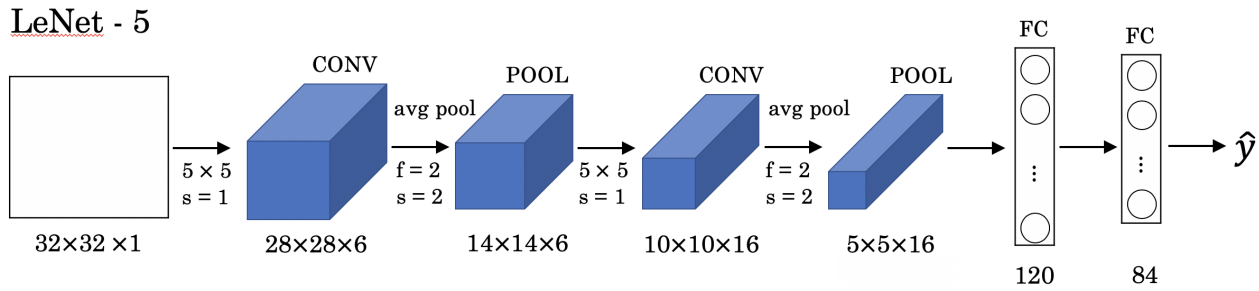
Use gradient descent to optimize parameters to reduce J

- Learn more about convolutional neural networks
  - An intro to ConvNet and Image Recognition on YouTube
  - An intuitive guide to ConvNet on Medium

2. CNN case studies

2.1. Classic networks

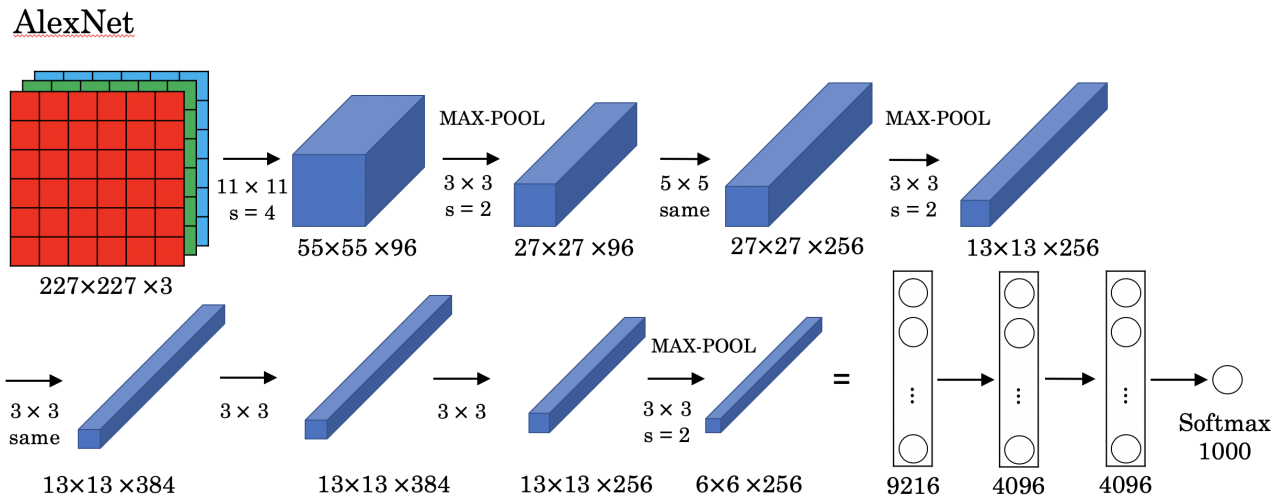
- **LeNet-5**



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

  - The goal of LeNet-5 was to recognize handwritten digits on grayscale images of 32 by 32 by 1.
  - From left to right, the height and width decrease, and the number of channels increases.
  - ~60K parameters

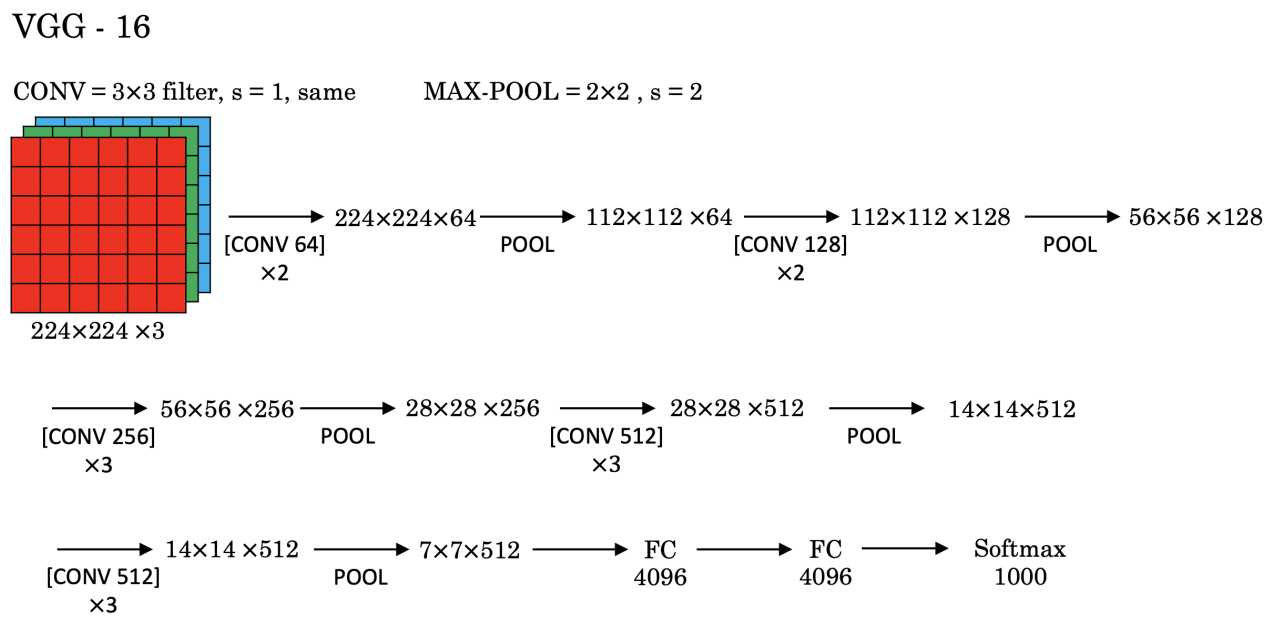
- **AlexNet**



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

- Similar to LeNet-5 but much bigger
- Uses ReLu activation rather than sigmoid/tanh
- ~60M parameters

- **VGG**



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

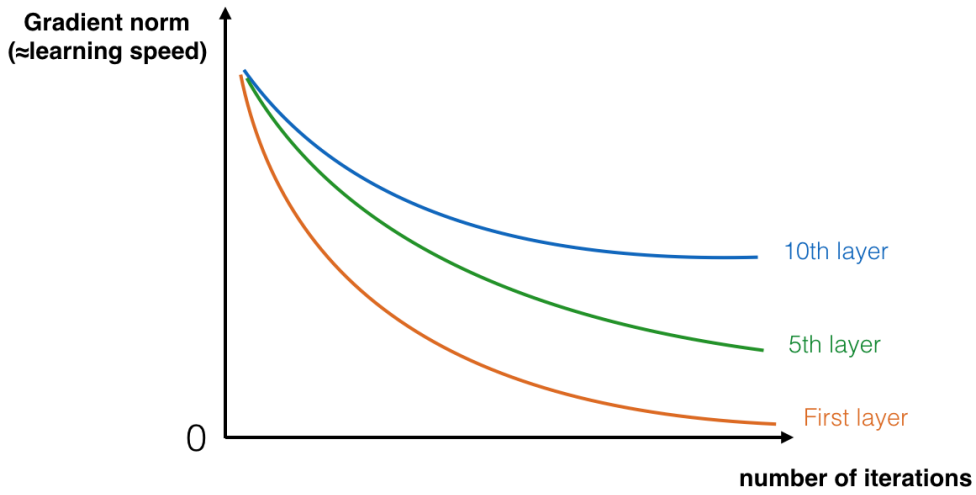
- Relatively deep CNN, 16 layers that have weights
- Simplified architecture, with same hyperparameters among CONV layers, and same hyperparameters among POOL layers, except for the number of channels that roughly doubles for every step.
- ~138M parameters

2.2. ResNet

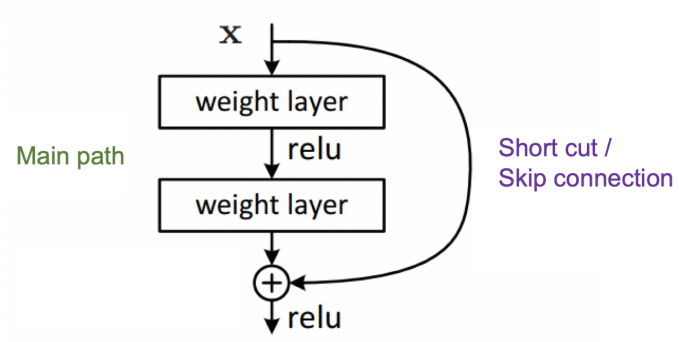
- **The problem of very deep neural networks**

The main benefit of a very deep network is that it can represent very complex functions. It can also learn features at many different levels of abstraction, from edges (at the lower layers) to very complex features (at the deeper layers). However, **very deep "plain" networks don't work in practice because they are hard to train due to vanishing gradients.**

- Vanishing gradient is a huge barrier to training: very deep networks often have a gradient signal that goes to zero quickly, thus making gradient descent unbearably slow. More specifically, during gradient descent, as you backprop from the final layer back to the first layer, you are multiplying by the weight matrix on each step, and thus the gradient can decrease exponentially quickly to zero (or, in rare cases, grow exponentially quickly and "explode" to take very large values).
- During training, you might therefore see the magnitude (or norm) of the gradient for the earlier layers descrease to zero very rapidly as training proceeds:



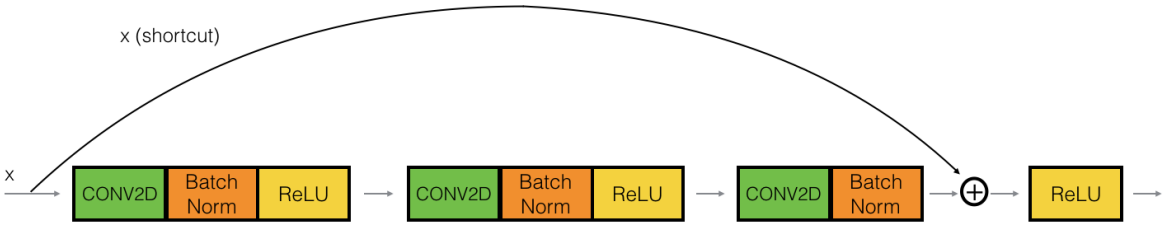
- **Residual block**



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

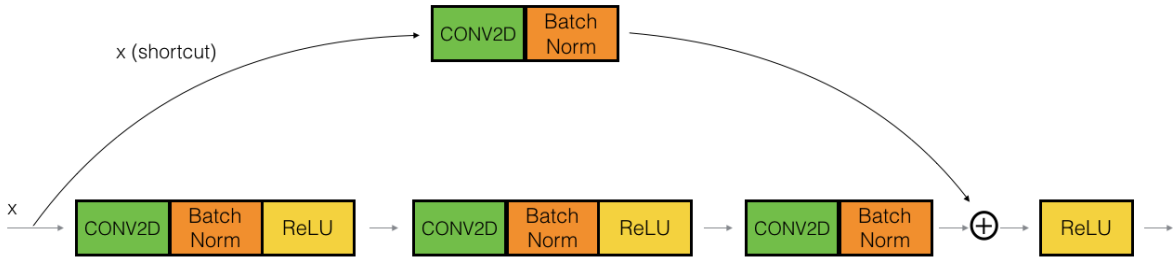
- **The identity block**

  The input activation $a^{(l)}$ usually has the same dimension as the output activation $z^{(l+2)}$ by using "same" padding.
  $a^{(l+2)} = g(z^{(l+2)} + a^{(l)})$



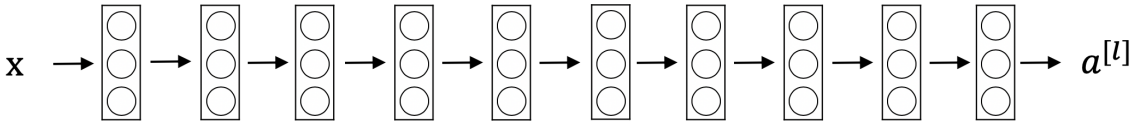
- **The convolution block**

  When $z^{(l+2)}$ and $a^{(l)}$ don't have the same dimension, implement an additional matrix to convert $a^{(l)}$'s dimension to $z^{(l+2)}$'s.
  $a^{(l+2)} = g(z^{(l+2)} + w_s a^{(l)})$

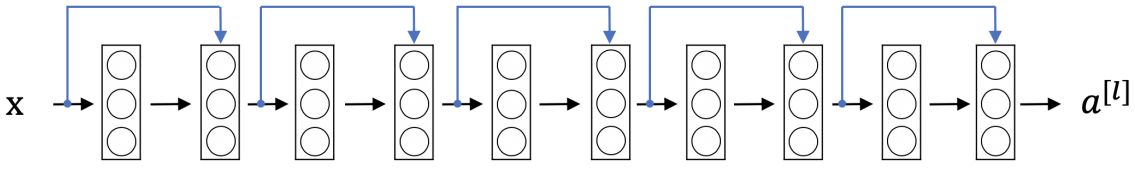


- **Residual network**

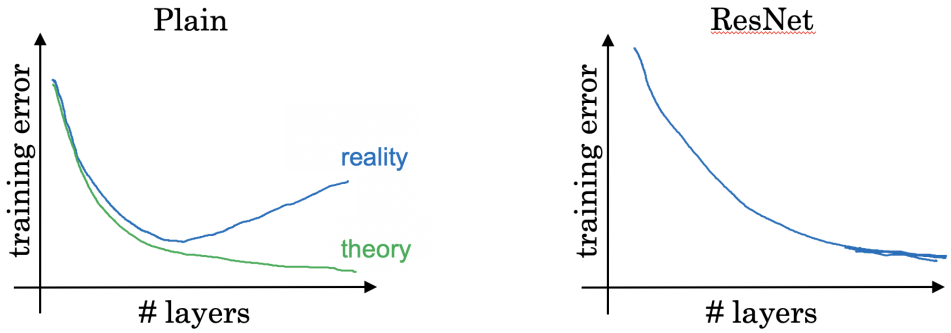
Plain Network



Residual Network



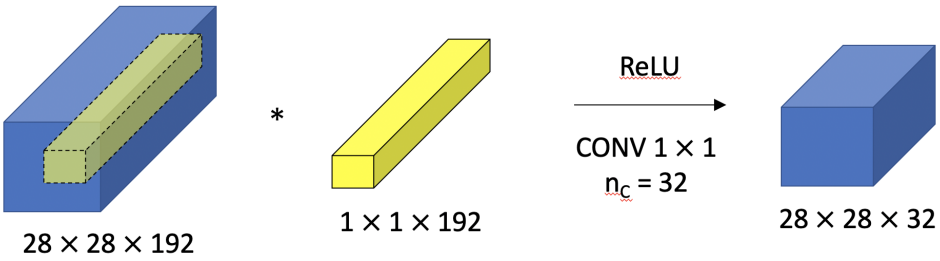
- (+) The skip connection helps to address vanishing gradient problem.
- (+) The skip-connection makes it easy for a ResNet block to learn an identity mapping between the input and the output of the ResNet block.



- **Why ResNets work**
  - (+) Allows to train much deeper neural networks without hurting the performance. Adding two layers in the neural network doesn't hurt the network's ability to do as well as this simpler network without these two extra layers, because identity function is easy for residual block to learn (not so easy in plain network).
  - (+) Helps with the vanishing and exploding gradient problems. Using a skip-connection helps the gradient to be directly backpropagated to earlier layers, and thus helps you to train deeper networks. May lead to a simpler architecture.

2.3. Inception neural networks

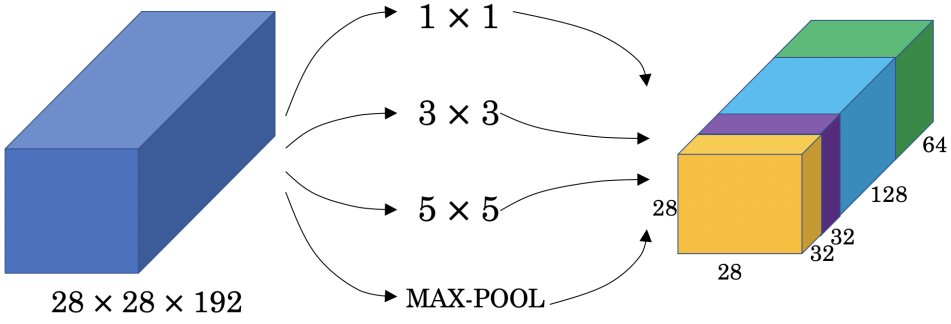
- **Networks in networks and 1 x 1 convolutions**



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

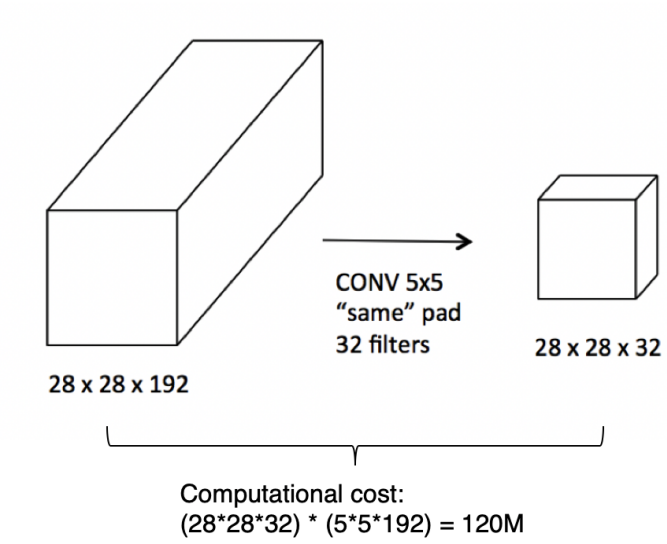
  - (+) Allows to shrink the number of channels and therefore, saves computation in some networks
  - (+) Adds non-linearity by allowings to learn the more complex function

- **Inception network motivation**

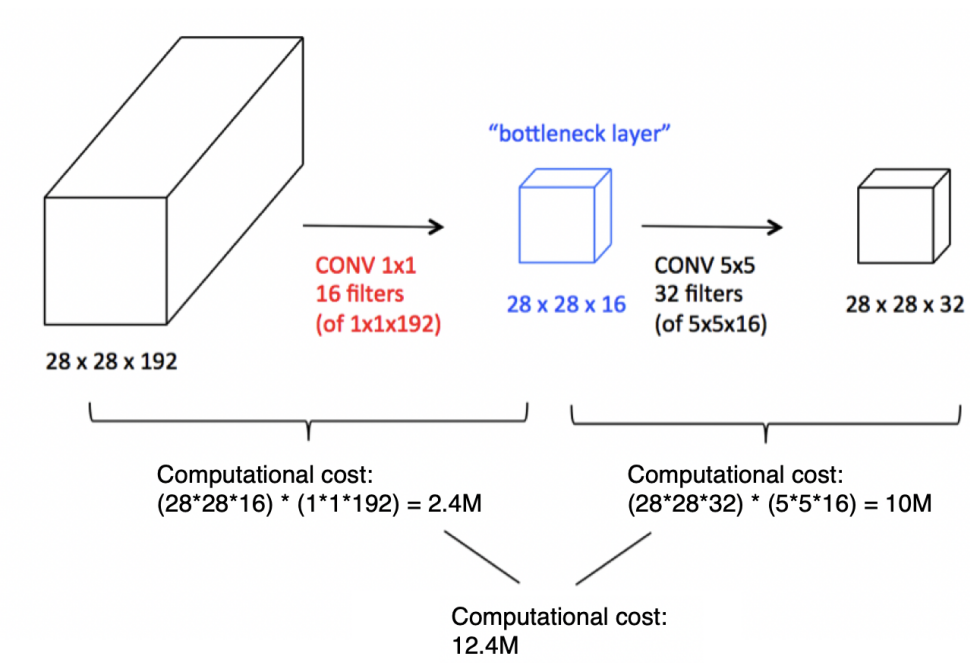


[Szegedy et al. 2014. Going deeper with convolutions]

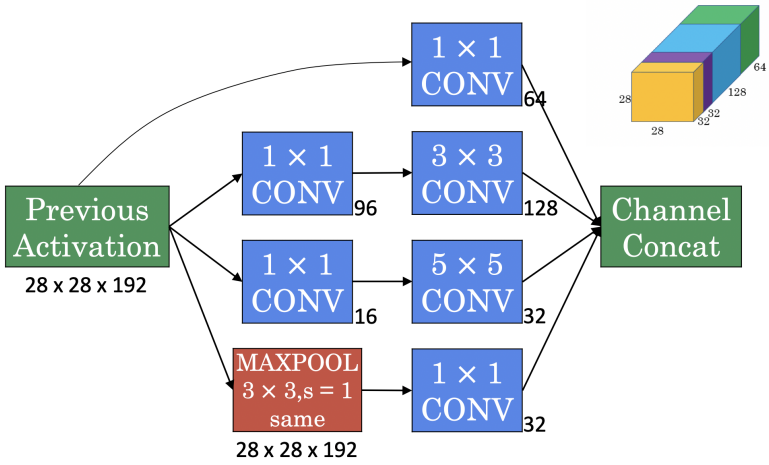
- **Use 1 x 1 convolutions to solve the computational cost problem**
  - Convolution can be very costly



  - Use 1 x 1 convolution to reduce cost by a factor of ~10

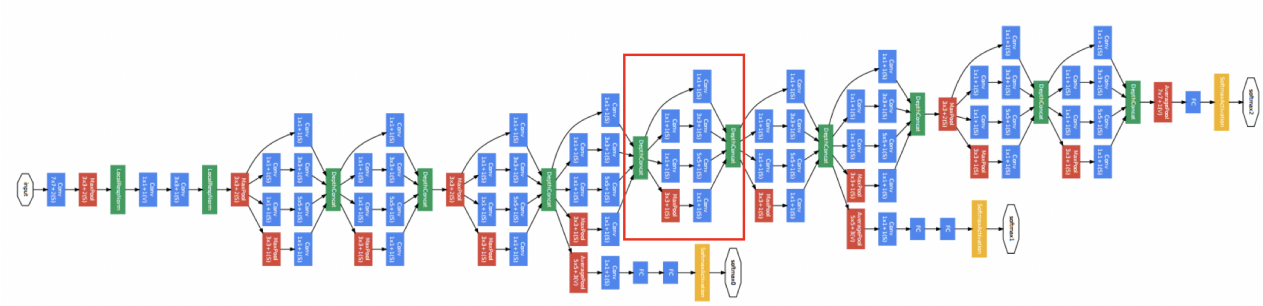


- **Inception module**



Building block of the inception network
[Szegedy et al., 2014, Going Deeper with Convolutions]

- **Inception network**



This particular network inception network is also known as "GoogleNet".
[Szegedy et al., 2014, Going Deeper with Convolutions]

  - (+) The additional outputs from the hidden layers help to ensure that the features computed even in the hidden layers not too bad for predicting the output class of a image, which appears to have a regularizing effect on the inception network and helps prevent this network from overfitting.

3. Practical advices for using CNN

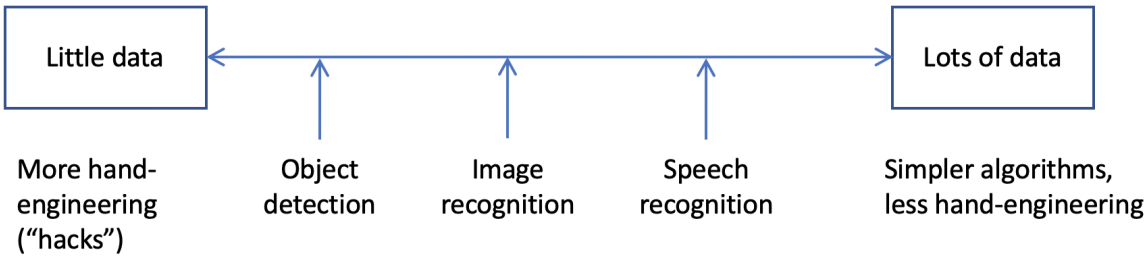
3.1. General tips

- Use open source implementation instead of re-inventing the wheels
  - Use architectures of networks published in the literature
  - Use open source implementations if possible
- Transfer learning
  - Use pretrained models and fine tune on your dataset
- Data augmentation to improve the performance of computer vision systems

3.2. Data vs. hand-engineering

- 2 sources of knowledge

- Labeled data
- Hand-engineered features/network architecture/other components



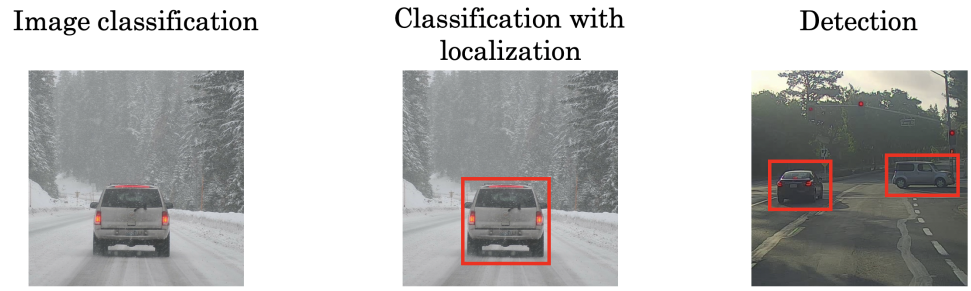
3.3. Tips for doing well on benchmarks/winning competitions

- Warning: not necessarily practical and rarely used when building production systems
- Ensembling: train several networks independently and average their outputs
- Multi-crop at test time: run classifier on multiple versions of test images and average results

4. Object detection

4.1. Object localization

- Classification, localization, and detection



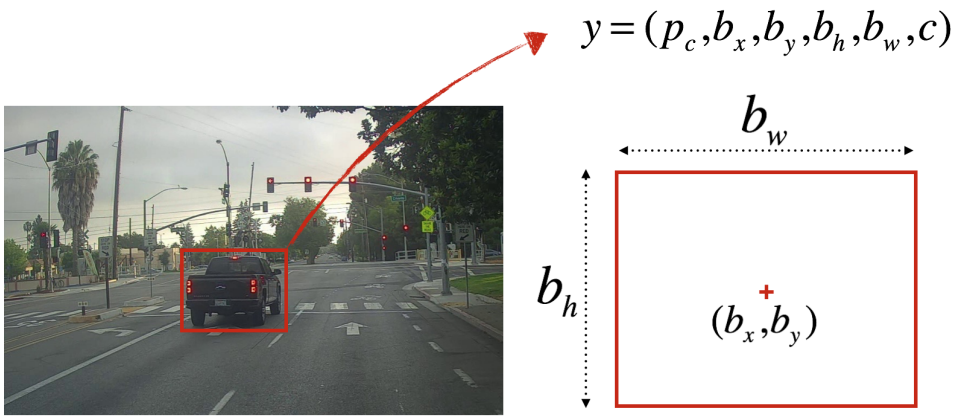
  - **Classification:** Label as a car
  - **Localization:** Label as a car AND specify a bounding box around the position of the car in the image. Records midpoint (bx, by), height bh, and width bw.
  - **Detection:** There can be multiple objects and/or of different categories to localize.

- Defining the target lable y

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

  - p_c represents whether there is any object in the image (0 or 1).
  - b_x, b_y, b_h, and b_w are the object's midpoint coordinates, the object's height, and the object's width that define the bounding box.

- c_1, c_2, and c_3 are the dummy estimates of class 1, 2, and 3.



$p_c = 1$: confidence of an object being present in the bounding box

$c = 3$: class of the object being detected (here 3 for "car")

- Defining the loss function

$$E(\hat{y}, y) = \begin{cases} \sum_{elements\ in\ y} (\hat{y} - y)^2 & if\ y_1 = 1 \\ (\hat{y}_1 - y_1)^2 & if\ y_1 = 0 \end{cases}$$

Alternatively, use log loss or logistic regression loss

4.2. Landmark detection

- Landmark detection

  Outputs the X,Y coordinates of different landmarks you want to recognize in an image. The identity of each landmark must be consistent across different images.

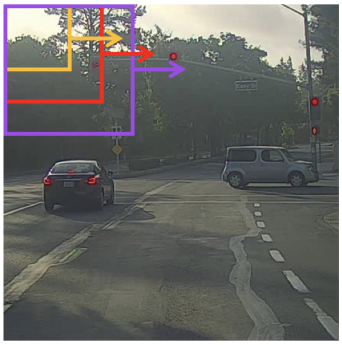
  E.g. face recognition

- Target label y

  E.g. (xi, yi) of all landmarks i

4.3. Object detection with sliding windows

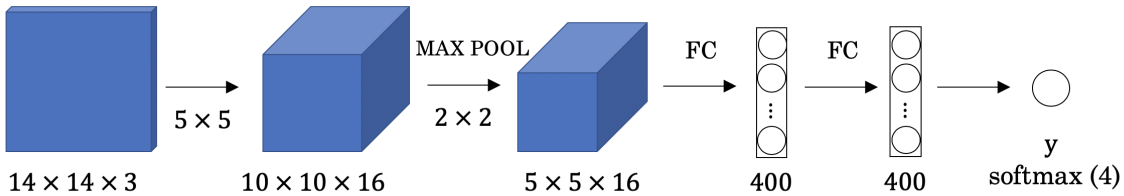
- Sliding windows detection
  - Step 1: Slide a rectangular region across the entire image, and pass each of the cropped image into the object localization ConvNet to classify zero or one for each position as some stride.
  - Step 2: Resize the region and repeat Step 1.

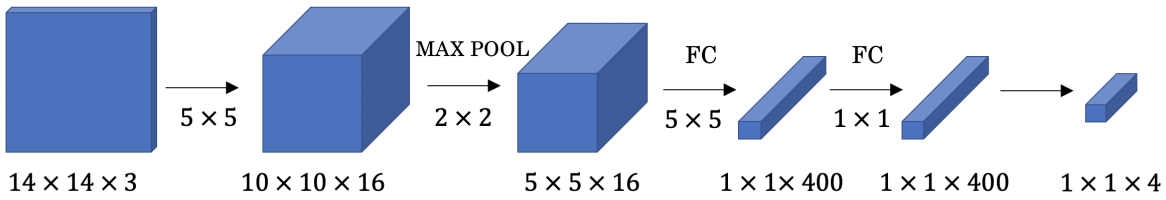


- Disadvantage of sliding windows detection with ConvNets
    - (-) Huge computation cost due to running so many different sliding windows with ConvNets independently/sequentially.

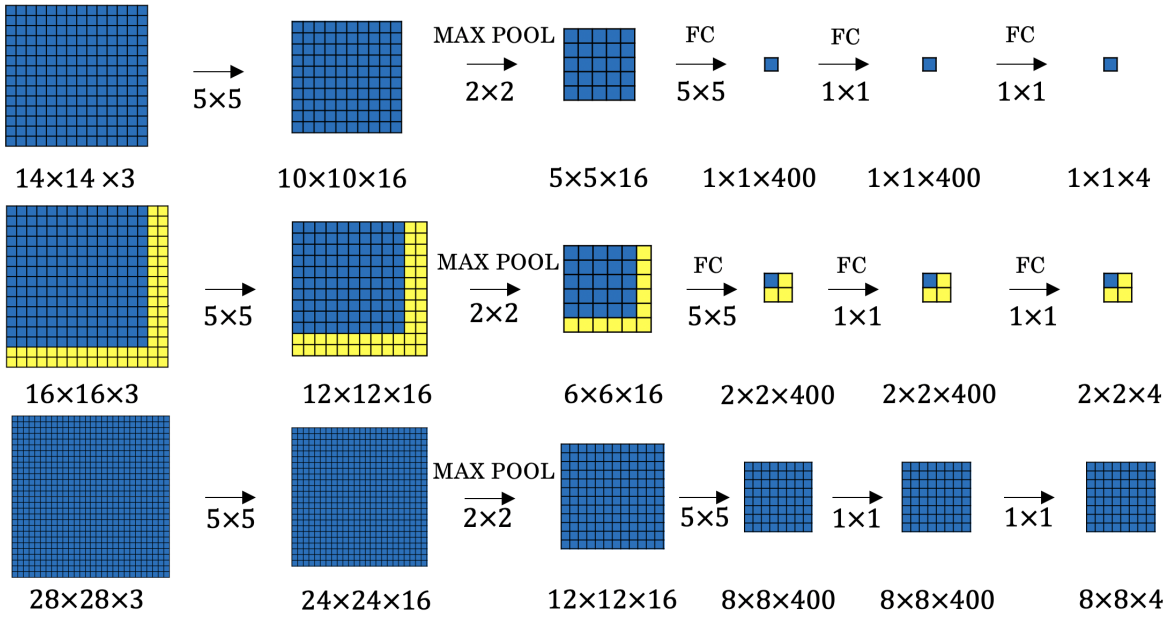
4.4. Convolutional implementation of sliding windows

- Turning FC layer into convolutional layers





- Implementing sliding windows convolutionally



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

- (+) Allows different ConvNets that have highly duplicated tasks to share a lot of computation.

  Instead of running ConvNets on different subsets of the input image independently, the convolutional implementation combines all into one computation and shares a lot of the computation in the common regions.

4.5. Bounding box predictions with YOLO

YOLO ("you only look once") is a popular algoritm because it achieves high accuracy while also being able to run in real-time. This algorithm "only looks once" at the image in the sense that it requires only one forward propagation pass through the network to make predictions. After non-max suppression, it then outputs recognized objects together with the bounding boxes.

- **Output accurate bounding box with YOLO algorithm**

  - Algorithm

    1. Place a grid (e.g. a 3x3 grid, or a 19x19 grid) on the image.
    2. Assign each object to the grid cell that contains the **midpoint** of that object.
    3. Each grid cell has an output that specifies whether there is any object, the bounding box of the object (fractions relative to the height and width of the grid cell), and the class of the object.

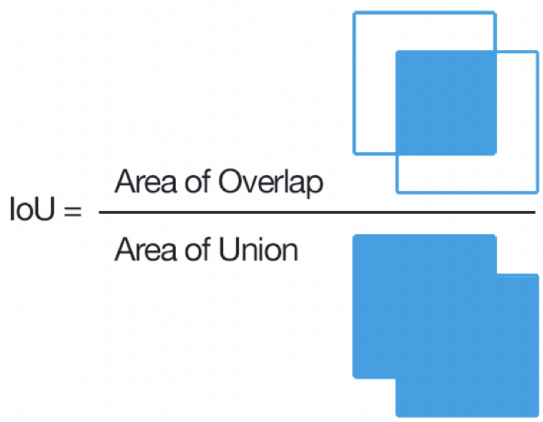
    Note that if the midpoint of an object falls into a grid cell, that grid cell is responsible for detecting that object. [Redmon et al., 2015, You Only Look Once: Unified real-time object detection]

  - Pros and cons

    - (+/-) YOLO works well so long as there aren't more than one object in each grid cell. For each object, even if the object spans multiple grid cells, that object is assigned only to one of the grid cells.

    - (+) Allows the network to output bounding boxes of any aspect ratio, and output much more precise coordinates that aren't just dictated by the stripe size of the sliding windows classifier.
    - (+) Efficient algorithm because it is a convolutional implantation, where you use one ConvNet with a lot of shared computation between all of the grid cells.

- **Intersection over union (IoU) to evaluate object detection algorithm**



  IoU is a measure of the overlap between 2 bounding boxes. By convention, "correct" if IoU ≥ 0.5

- **Non-max suppression to clean up bounding boxes**

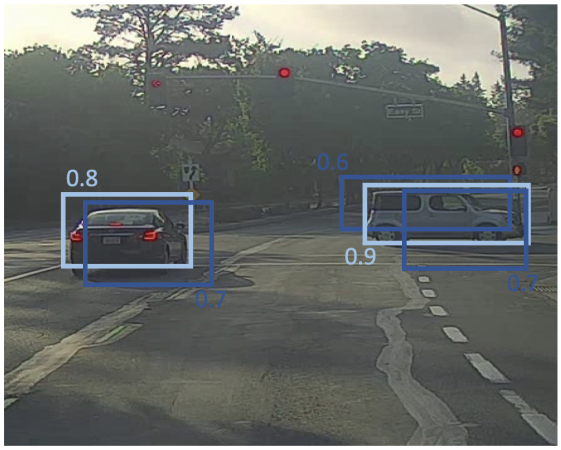
    - Problem to address

      Because you're running the image classification and localization algorithm on every grid cell, it's possible that more than one of the grid cells think they have detected an object. So the object detection algorithm might end up with multiple detections of each object.

    - Algorithm

      Non-max suppression keeps the bounding box that has the maximum probability of detection (pc) of the object, and suppress the overlapping ones that are non-maximal.

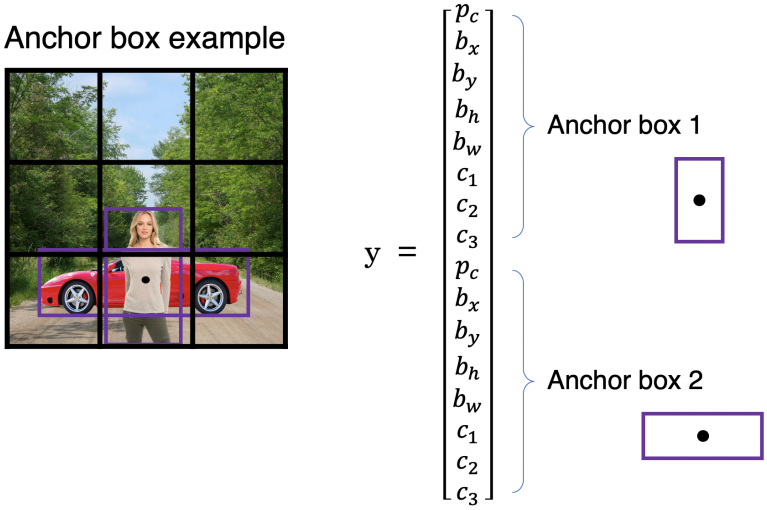
        1. Discard all boxes with pc ≤ 0.5
        2. While there are any remaining boxes:
            - Pick the box with the largest pc, output that as a prediction.
            - Discard any remaining box with IoU ≥ 0.5 with the box output in the previous step.



- **Anchor boxes**
  - Problem to address

    One of the problems with object detection is that each of the grid cells can detect only one object. What if a grid cell wants to detect multiple objects?

  - Algorithm



    Each object in training image is assigned to grid cell that contains object's midpoint and anchor box for the grid cell with highest IoU.

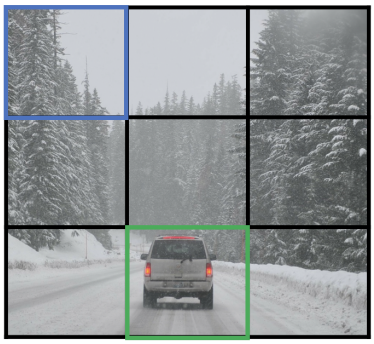
  - How to choose anchor boxes
    - By hand: Typically choose 5-10 anchor boxes that spans a variety of shapes to cover the types of objects.
    - Automatically: Use a K-means algorithm to group together the types of objects shapes you tend to get. And then select a set of anchor boxes that are most stereotypically representative of the objects.
  - Pros and cons
    - (-) Does not handle well when there are two anchor boxes but three objects in the same grid cell; or two objects associated with the same grid cell, but both of them have the same

anchor box shape.

  - (+) Allows learning algorithm to specialize so that some of the outputs can specialize in detecting white, fat objects like cars, and some of the output units can specialize in detecting tall, skinny objects like pedestrians.

- **Putting everything together: YOLO algorithm**

  - Training and predicting



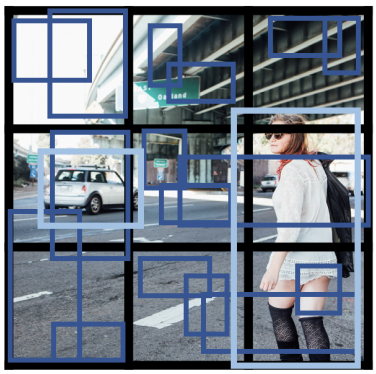
1 - pedestrian
2 - car
3 - motorcycle

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

y is 3×3×2×8

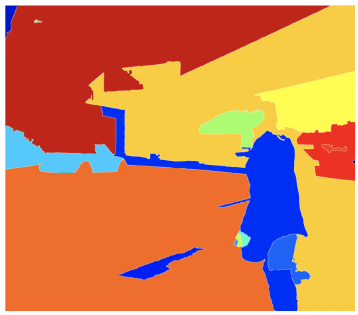
  - Outputting the non-max suppressed outputs

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



4.6. Region proposals

- **Region proposal: R-CNN (Regions with CNN)**

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

Rather than running sliding windows on every single window, select just a few windows and run ConvNet classifier on these windows. Use segmentation algorithm to find blobs, and run classifier only on those blobs.

  - (-) R-CNN very slow

- **Faster algorithms**

  - Original **R-CNN:** Propose regions. Classify proposed regions one at a time. Output label + bounding box. [Girshik et. al, 2013. Rich feature hierarchies for accurate object detection and semantic segmentation]
  - **Fast R-CNN:** Propose regions. Use convolution implementation of sliding windows to classify all the proposed regions. [Girshik, 2015. Fast R-CNN]
    - (-) The clustering step to propose the regions is still quite slow.
  - **Faster R-CNN:** Use convolutional network to propose regions. [Ren et. al, 2016. Faster R-CNN: Towards real-time object detection with region proposal networks]

R-CNN algorithms are usually slower than YOLO algorithm.

5. Special applications: face recognition & neural style transfer

5.1. Face recognition

- **Face verification vs. face recognition**

  - Verification (1:1 matching problem)
    - Input image, name/ID
    - Output whether the input image is that of the claimed person
  - Recognition (1:K matching problem)
    - Has a database of K persons
    - Get an input image
    - Output ID if the image is any of the K persons (or “not recognized”)

    Face recognition requires much higher accuracy compared to face verification.

- **One shot learning**

  Learning from one example to recognize the person again. Historically, deep learning algorithms don't work well if there is only one training example.

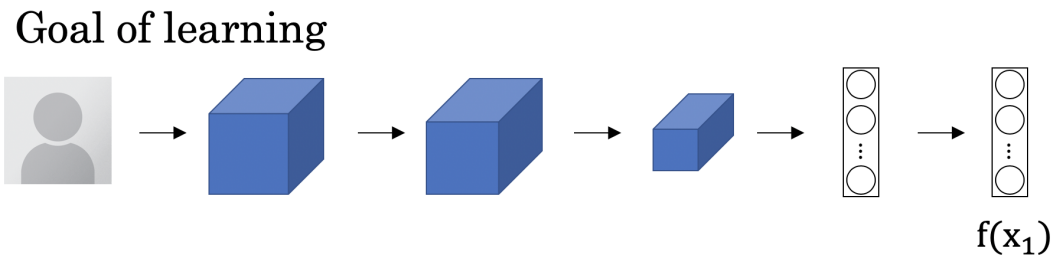
  - Learning a "similarity" function

    d(img1,img2) = degree of difference between 2 images

    - If d(img1,img2) ≤ τ, "same"
    - If d(img1,img2) > τ, "different"

- **Siamese network**

  Run two identical CNNs on two different inputs and then compare the ouputs.



  Parameters of NN define an encoding $f(x_i)$

  Learn parameters so that:

  If x_i, x_j are the same person, $\|f(x_i) - f(x_j)\|^2$ is small.

  If x_i, x_j are different persons, $\|f(x_i) - f(x_j)\|^2$ is large.

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

- **Triplet loss function for face verification**

  Define an applied gradient descent to learn the parameters of the neural network so that it outputs a good encoding for the pictures of faces.
  [Schroff et al.,2015, FaceNet: A unified embedding for face recognition and clustering]

  - Goal

    Want $d(A, P) - d(A, N) + \alpha \leq 0$, where
    $d(A, P) = \|f(A) - f(P)\|^2$, $d(A, N) = \|f(A) - f(N)\|^2$, and α is margin

- Loss function
  - Given 3 images A, P, N

$$E(A, P, N) = max(\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 + \alpha,\ 0)$$

  - Overall loss

$$J = \sum_{i=1}^{m} E(A_i, P_i, N_i)$$

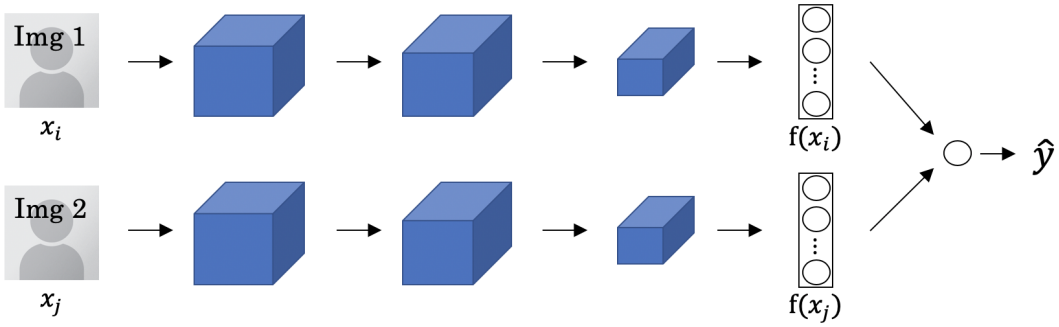
  Needs multiple pictures of the same person to train the model. After training, can apply the model to one shot learning problem.

- Choosing the triplets A, P, N
  - During training, if A,P,N are chosen randomly, $d(A, P) + \alpha \leq d(A, N)$ is easily satisfied.
  - Choose triplets that are "hard" to train on. Choosing triplets carefully increases the computational efficiency of your learning algorithm.

- **Binary classification for face verification**

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

  - Learning the similarity function

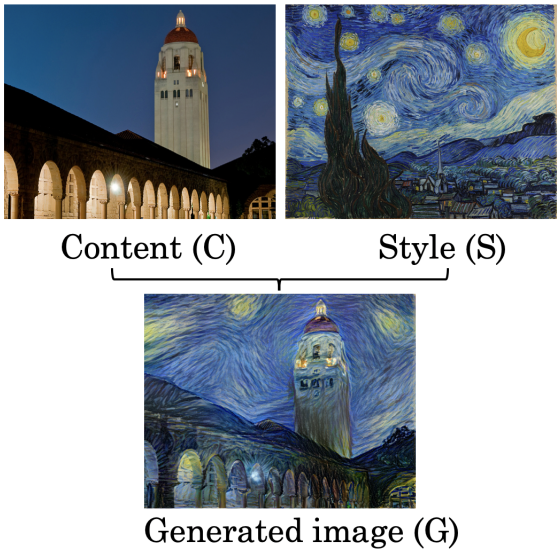


$\hat{y} = \sigma\left(\sum_k w_k \left|f(x_i)_k - f(x_j)_k\right| + b\right)$, where k represents a unit of the encoding layer.

Can also use Chi-square similarity $\hat{y} = \sigma\left(\sum_k w_k \frac{(f(x_i)_k - f(x_j)_k)^2}{f(x_i)_k + f(x_j)_k} + b\right)$

5.2. Neural style transfer

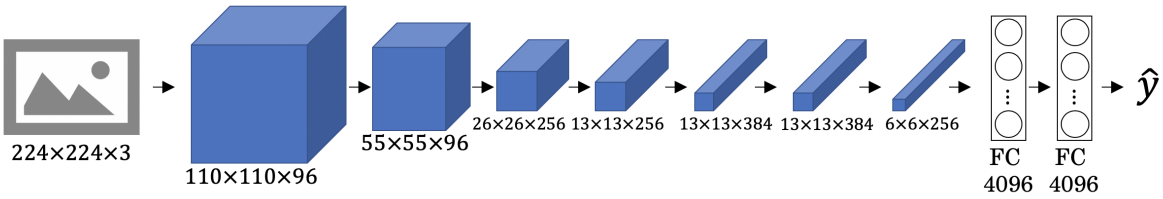
- **Example of neural style transfer**

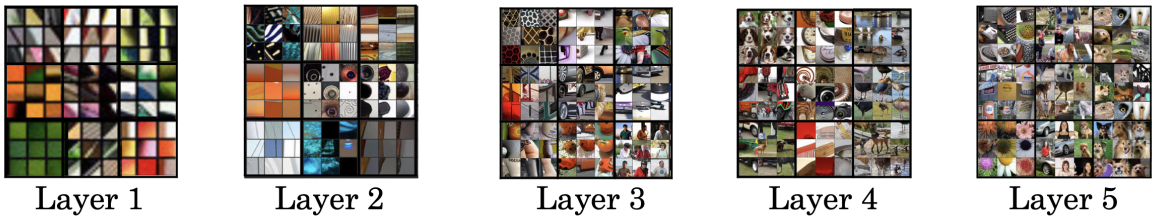


[Image credit: Justin Johnson]

- **Visualizing what a deep network is learning**

  1. Pick a unit in layer 1. Find the nine image patches that maximize the unit's activation.
  2. Repeat for other units.





Deeper layers detect more complex objects.
[Zeiler and Fergus., 2013, Visualizing and understanding convolutional networks]

- **Cost function of the generated image**

  - Neural style transfer cost function

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

    - $J_{content}(C, G)$ measures how similar is the content of the generated image (G) to the content of the content image (C).
    - $J_{style}(S, G)$ measures how similar is the style of the generated image (G) to the style of the style image (S).

- α and β are the hyperparameters that specify the relative weighting between the content cost and the style cost.

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

- Find the generated image G
  1. Initiate G randomly

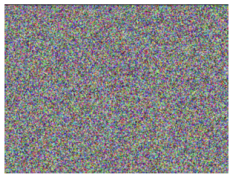
     G: 100×100×3

  2. Use gradient descent to minimize J(G)

     Updating the pixle values of G

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





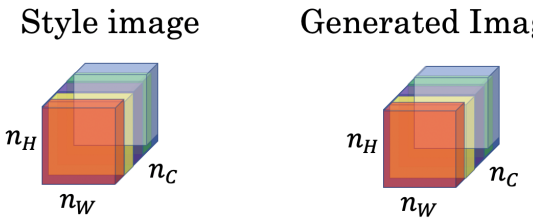

[Gatys et al., 2015. A neural algorithm of artistic style.]

- **Content cost function**
  - The content cost function is computed using one hidden layer's activations. The hidden layer l should be not too shallow, not too deep. Use pre-trained ConvNet (E.g., VGG network). Let $a_C^{(l)}$ and $a_G^{(l)}$ be the activation of layer l on the images.
  - If $a_C^{(l)}$ and $a_G^{(l)}$ are similar, both images have similar content.

    $J_{content}(C, G) = \frac{1}{2} \left\| a_C^{(l)} - a_G^{(l)} \right\|^2$, which is element-wise sum of squares of activation differences between C and G.

- **Style cost function**

- Meaning of the “style” of an image
  - The style cost function for one layer is computed using the "Gram matrix" of that layer's activations. The overall style cost function is obtained using several hidden layers.
  - In each layer, define style as correlation (unnormalized cross-covariance) between activations across channels. The correlation measures how often the textures occur together and don't occur together in different parts of an image.



- Style matrix

  Let $a^{(l)}_{i,j,k}$ = activation at (i, j, k). The style matrix $G^{(l)}$ is $n^{(l)}_C \times n^{(l)}_C$.

  $$G^{(l)}_{kk'} = \sum_{i=1}^{n^{(l)}_H} \sum_{j=1}^{n^{(l)}_W} a^{(l)}_{ijk}\ a^{(l)}_{ijk'}$$

  , which is also known as the "Gram matrix" in linear algebra.

- Cost function
  - Style cost function of each layer

    $$J^{(l)}_{style}(S,G) = \frac{1}{C}\left\|G^{(l)}_S - G^{(l)}_G\right\|^2_F = \frac{1}{(2n^{(l)}_H n^{(l)}_W n^{(l)}_C)^2} \sum_k \sum_{k'} \left((G^{(l)}_{kk'})_S - (G^{(l)}_{kk'})_G\right)^2$$

    , where

    $$(G^{(l)}_{kk'})_S = \sum_{i=1}^{n^{(l)}_H} \sum_{j=1}^{n^{(l)}_W} (a^{(l)}_{ijk})_S\ (a^{(l)}_{ijk'})_S$$

    for style image

    $$(G^{(l)}_{kk'})_G = \sum_{i=1}^{n^{(l)}_H} \sum_{j=1}^{n^{(l)}_W} (a^{(l)}_{ijk})_G\ (a^{(l)}_{ijk'})_G$$

    for generated image

  - Overall style cost function

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