Part4. Convolutional Neural Network

1. Foundations of CNN
2. Deep Convolutional Models
3. Computer Vision Problems
4. Special Applications

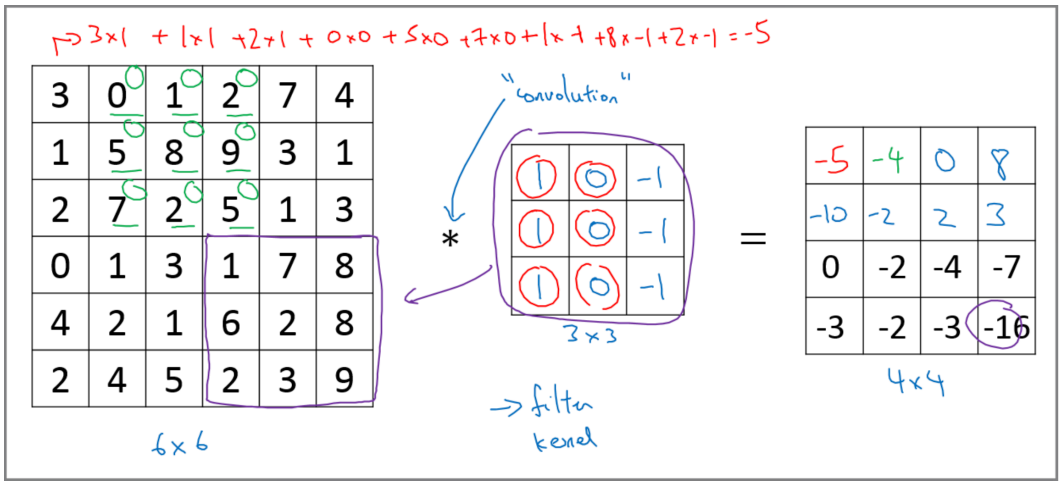
1. Foundations of CNN

Computer vision

Computer vision is one of the applications that are actively growing thanks to ML. (Self driving cars, Face recognition etc.) Rapid changes to computer vision are making new applications that weren't possible a few years ago. Computer vision deep learning techniques always evolve making a new architectures which can help us in other areas other than computer vision. Examples of a computer vision problems includes image classification, object detection and neural style transfer.

Edge detection example

- The convolution operation is one of the fundamental blocks of a CNN. One of the examples about convolution is the image edge detection operation.
- Early layers of CNN might detect edges, the middle layers will detect parts of objects and the later layers will put these parts together to produce an output.



Vertical Edge Detection

- If we applied this filter to a white region followed by a dark region, it should find the edges in between the two colors as a positive value. But if we applied the same filter to a dark region followed by a white region it will give us a negative values. To solve this, we can use the abs function to make it positive.
- Horizontal edge detection

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

- There are a lot of ways we can put number inside the horizontal or verticel edge detections filter. For example, here are the vertical **Sobel** filter and **Scharr** filter, taking care of the middle row.

1	0	-1
2	0	-2
1	0	-1

3	0	-3
10	0	10
3	0	-3

- What we learned in the deep learning is that we don't need to hand-craft these numbers, we can treat them as weights and then learn them. It can learn horizontal, angled, or an edge type automatically rather than getting them by hand.
- If you make the convolution operation in TensorFlow, you will find the function **tf.nn.conv2d**. In Keras, you will find **Conv2d** function.

Padding

- We want to apply convolution operation multiple times, but if the image shirnks, we will lose a lot of data on this process. Also the edges pixels are used less than other pixels in the center of an image.
- To solve these problems, we can pad the input image before convolution by adding some rows and columsn to it. We will call the padding amount p, the nymber of rows/columns that will be inserted in the image. In almost all the cases, $p = 0$.
- If a $n \times n$ matrix is convolved with $f \times f$ filter(kernel) and padding p, it results in $(n + 2p - f + 1, n + 2p - f + 1)$ matrix.
- Same convolution is a convolution with a padding so that the output size is the same as the input size. $(p = \frac{f-1}{2}, f \text{ is usually odd.})$

Stride

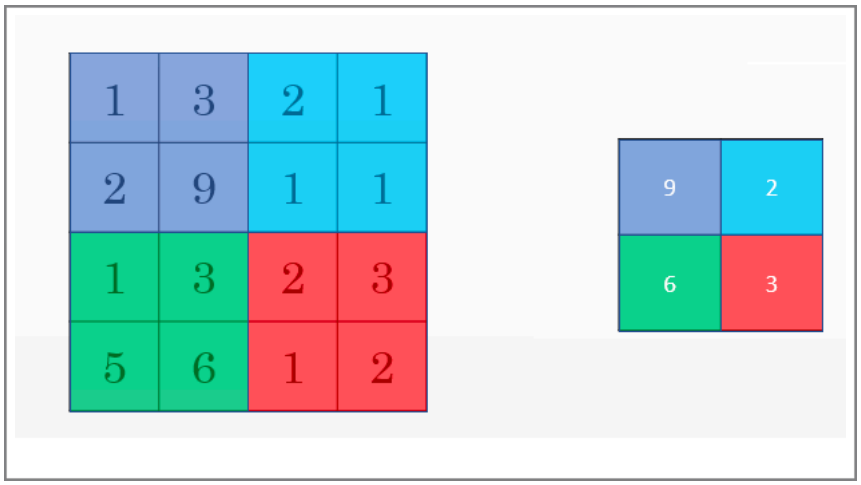
- When we are making the convolution operation, we use s to tell the number of pixels we will jump when we are convolving filter/kernel.

- If a $n \times n$ matrix is convolved with $f \times f$ filter(kernel) and padding p and stride s, it results in $(\frac{n + 2p - f}{s} + 1, \frac{n + 2p - f}{s} + 1)$ matrix. In case of fraction, we can take floor of the value.

Pooling layers

- Other than the conv layers, CNN often uses pooling layers to reduce the size of the inputs, speeding up computation and making some of the features more robust.

- Max pooling and Average pooling

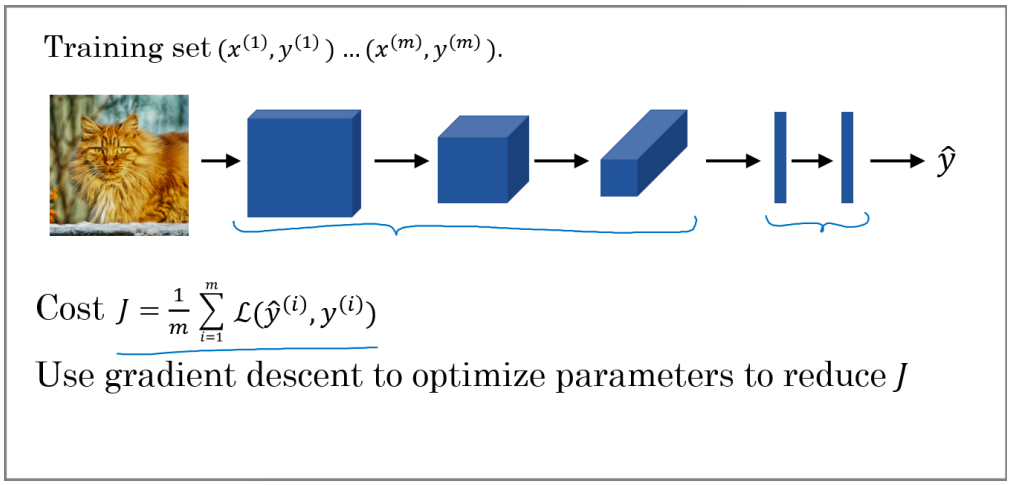


Average Pooling

- Max pooling is used more often than Average pooling in practice.

Why convolutions?

- Two main advantages of Convs are:
  - Parameters 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.
  - Sparsity of connections: In each layer, each output value depends only on a small number of inputs which makes its translation invariant.



Putting it all together

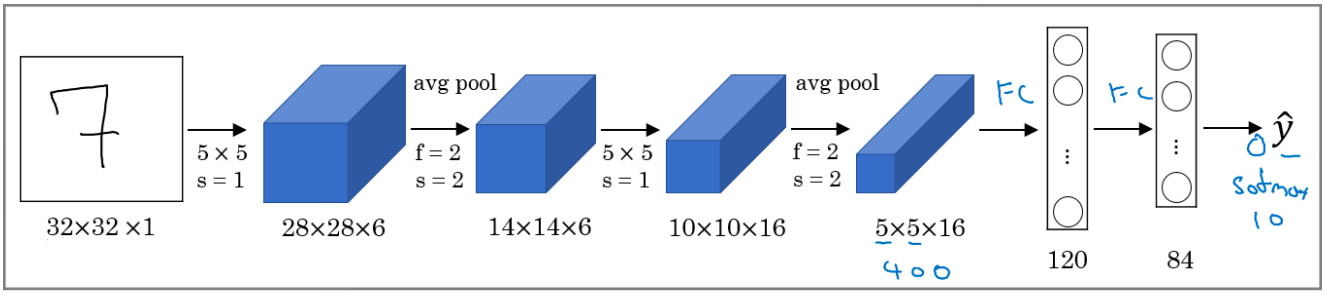
2. Deep Convolutional Models

Why look at case studies?

We learned about Conv layer, pooling layer, and fully connected layers. It turns out that computer vision researchers spent the past few years on how to put these layers together. To get some intuiotions, you have to see the examples that has been made since some neural network architecture that works well in some tasks can also work well in other tasks.

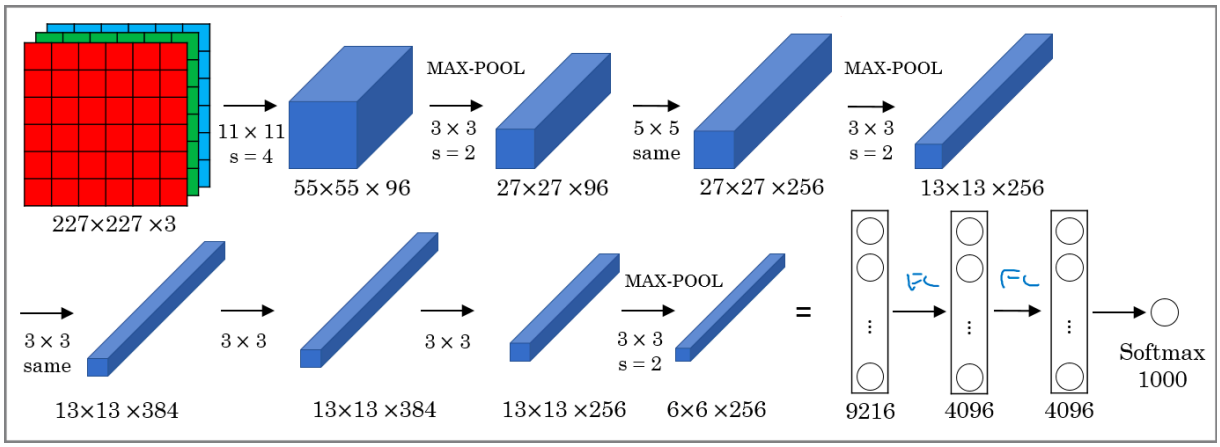
Classic networks

1. **LeNet-5**



LeNet-5

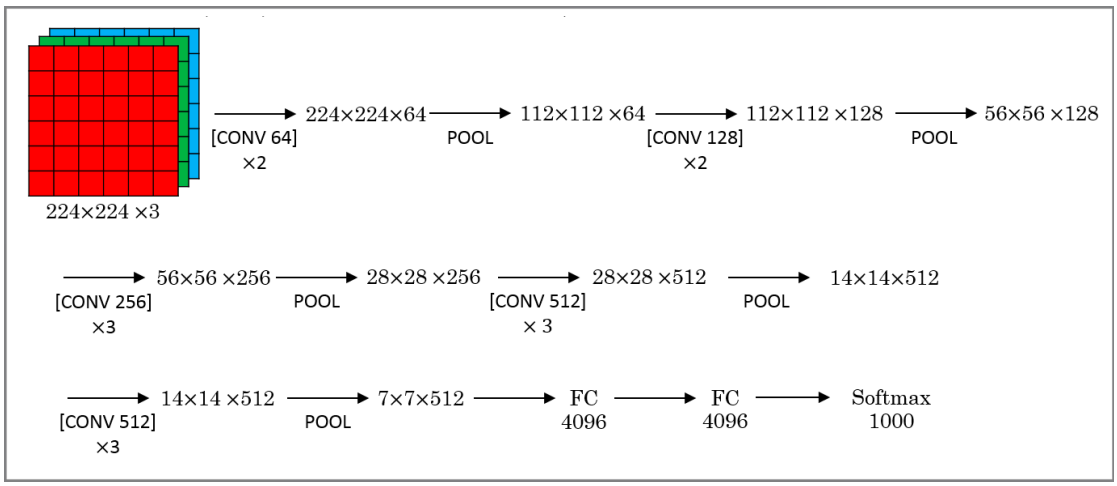
2. **AlexNet**



AlexNet

- Similar to LeNet-5, but bigger with 60 million parameters.
- The original paper contains multiple GPUs and Local Response Normalization (RN).
  - Multiple GPUs were used because the GPUs were not so fast back then.
  - Researchers proved that Local Response normalization doesn't help much, so for now don't bother yourself for understanding or implementing it.

3. VGG-16



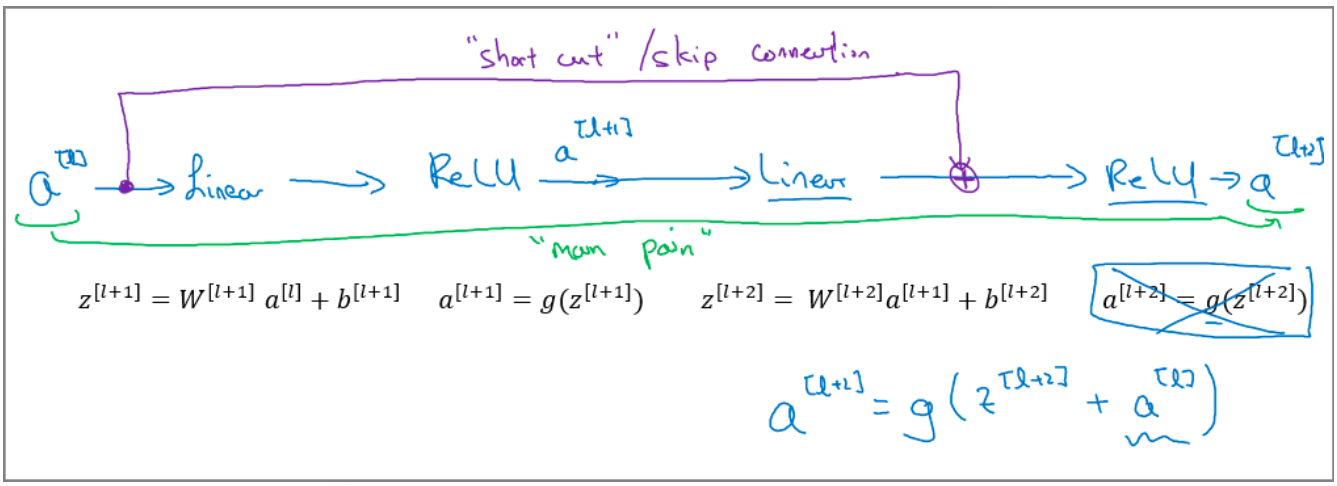
VGG-16

Instead of having a lot of hyperparameters, have some simpler network focusing on having only these two blocks: CONV and MAX_POOL.

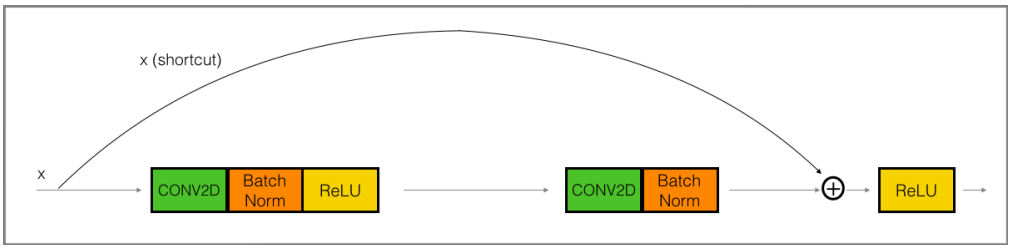
- This network is large even by modern standards. It has around 138 miilion parameters.
- There are another version called **VGG-19,** which is a bigger version. But most people uses the VGG-16 instead of the VGG-19 since it does almost the same work.

4. Residual Networks (ResNets)

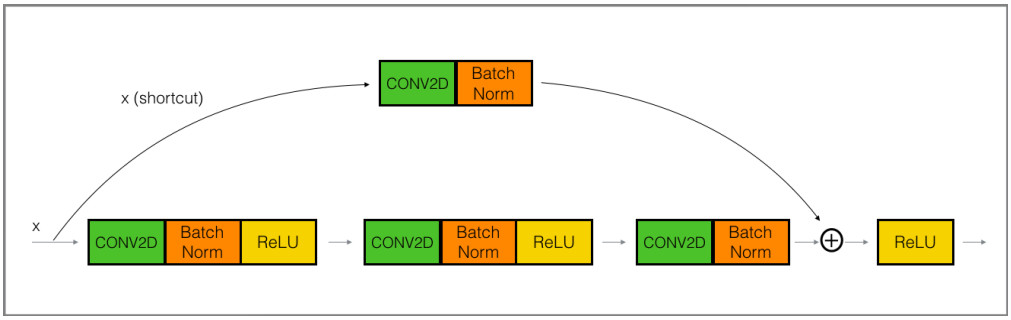
- Theoretically, plain networks will get a better solution as they go deeper, but in reality, due to the vanishing and exploding gradients problems, the performance of the network suffers.
- Skip connection: take the activation from one layer and suddenly feed it to another layer even much deeper in NN, which allows you to train large NNs even with depths greater than 100.



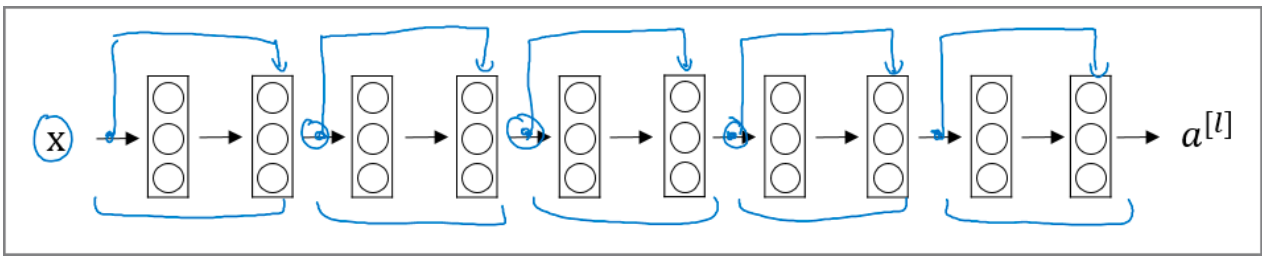
Residual Block



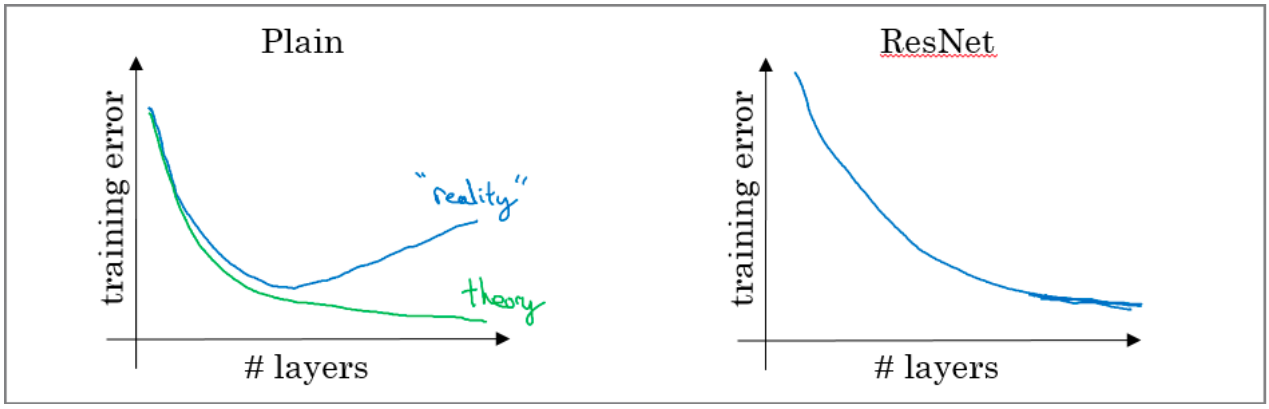
Identity Block



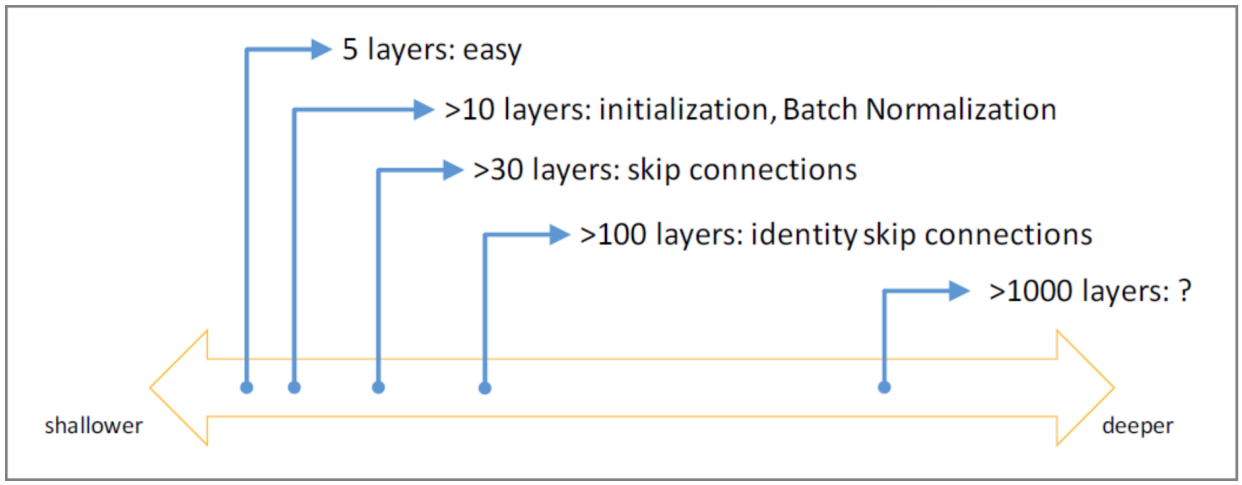
Convolutional Block



Residual Network



Superior Performance of ResNets

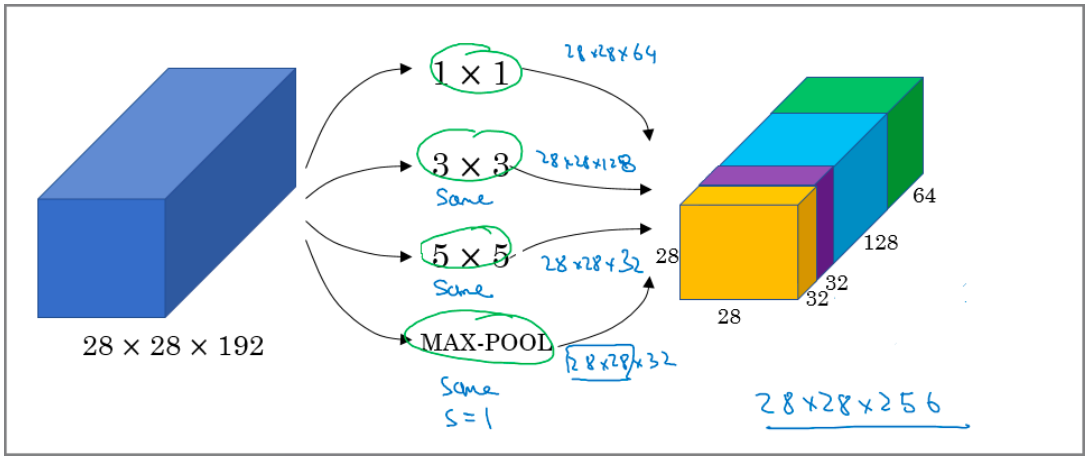


Spectrum of Depth

5. Network in Network and 1×1 convolutions

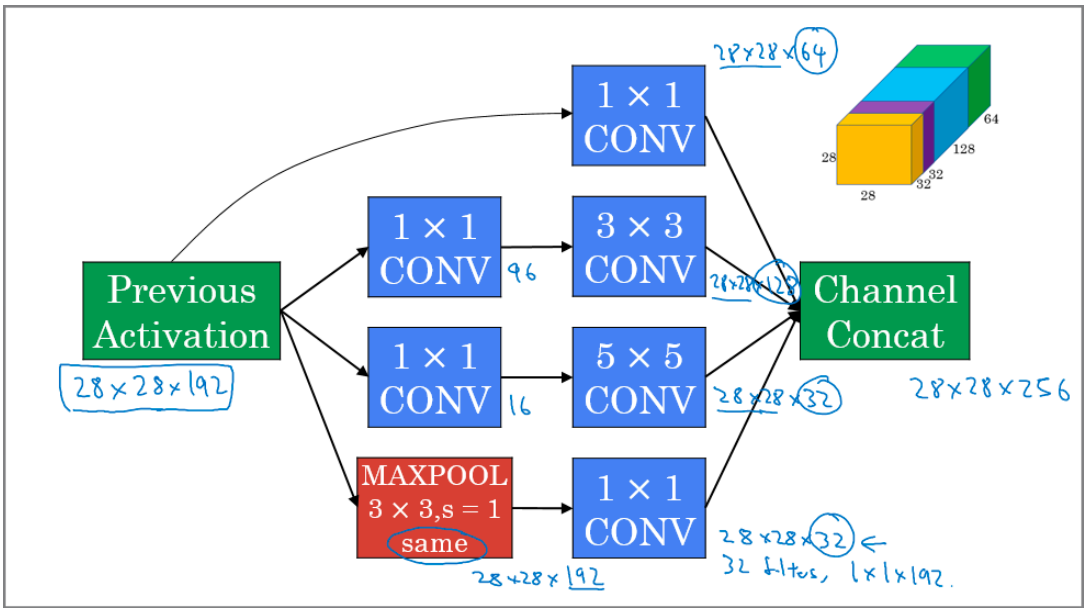
- A 1×1 convolution (also called Network in Network) is useful when:
  - Want to shrink the number of channels to save a lot of further computations. (Feature Transformation)
  - With specified number of 1×1 filters with the same number of input channels, the 1×1 Conv. can learn non-linear operator.
- Replace fully connected layers with 1×1 convolutions!

6. Inception Network

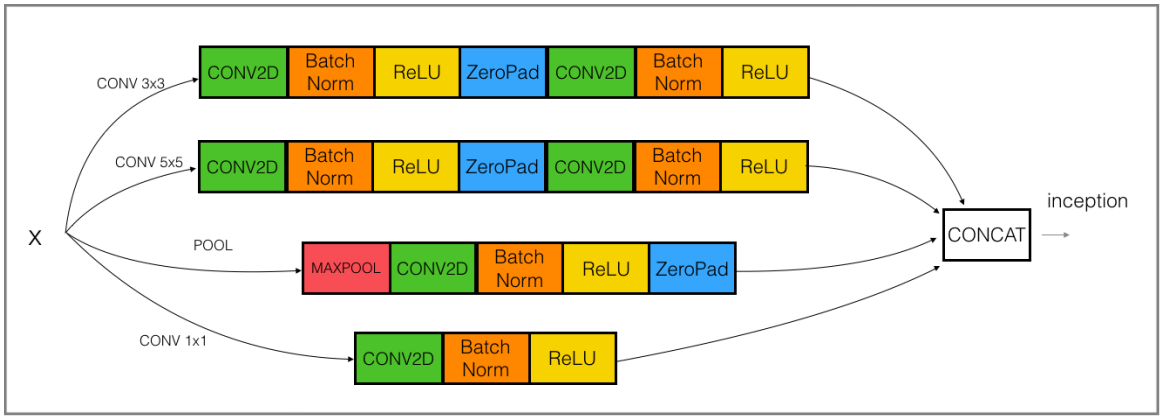


Naive Version of Inception Module

- Use 1×1 convolution to reduce computational cost



Dimension-reducted Version of Inception Module



Example of inception model in Keras

- The Inception network consists of concatenated blocks of the Inception module.
- There are 3 Softmax branches at different positions to push the network toward its goal, and helps to ensure that the intermediate features are good enough for the network to learn.
- Since the invention of the Inception module, the authors and others have built other vesions of this network, such as Inception v2, v3 and v4. Also, there is a network that has used the inception module and the ResNet together.

Using Open-Source Implementation

It turns out that a lot of NNs are difficult to be replicated due to some trivial details that may not be presented on its papers, such as learning decay or parameter tuning. Thus, a lot of deep learning researchers are disclosing their code on Internet through Github.

Transfer Learning

If you are using a specific NN architecture that has been trained before, you can use this pretrained parameters instead of random initialization to solve your problem, which helps boost the performance of the NN. The pretrained model might have trained on a large datasets like imageNet, MsCOCO, or Pascal and took a lot of time to learn those parameters. Using those pretrained models can save you a lot of time.

Go online and download a good NN with its weights, remove the softmax activation layer and put your own one and make the network learn only the new layer while other layers are fixed/frozen.

One of the tricks that can speed up your training is to run the pretrained NN without final softmax layer and get an intermediate representation saved on disk. Use these representations to a shallow NN. This can save you the time needed to run an image through all the layers.

Data Augmentation

- If the amount of data increases, your deep NN will perform better. The majority of computer vision applications need more data right now. Data augmentation is one of the techniques that deep learning uses to increase the performance of deep NN.
- Data Augmentation also has some hyperparameters. A good place to start is to find an open source data augmentation implementation and then use it or fine tune these hyperparameters.
  - Mirroring, Random Cropping, Rotation, Shearing, Local Warping etc.
  - Color Shifting
    - Add to R, G, and B some distortions that will make the image identified as the same for the human, but is slightly different for the computer.
    - PCA color augmentation

State of Computer Vision

- Since there is not enough data in a lot of computer vision problems, it relies a lot on hand engineering. Specifically, object detection problems have scarce data, requiring a more complex NN architectures.
- Tips for doing well on computer vision problems:
  - Ensembling
    - Train several networks independently and average their outputs. Mering down some classifiers.

    - After you decide the best architecture for your problem, initialize some of that randomly and train them independently.
    - Might slow down your production by the number of ensembles. Also, it takes more memory as it saves all the models in the memory. (Used in competiton, not in a real production)
- Multi-crop at test time
    - Run classifier on multiple versions of test versinos and average results
- Use open source code

3. Computer Vision Problems

Image Classification

- Classify an image to a specific class. The whole image represents one class. No need to know exactly where the object is. Usually one objecct is presented.
- Use a Conv Net with a softmax attached to the end of it.

Image Classification with Localization

- Given an image, want to learn the class of the image and where the class locates in the image. Need to detect a class and a rectangle of where that object is. Usually one object is presented.
- Use a Conv Net with a softmax attached to the end of it and 4 numbers, bx, by, bh, bw to tell you the location of the class in the image. The dataset should contain this four numbers with the class, too.

Image Detection

- Given an image, want to detect all the objects in the image that belong to specific classes, with their location. An image can contain more than one object with different classes.

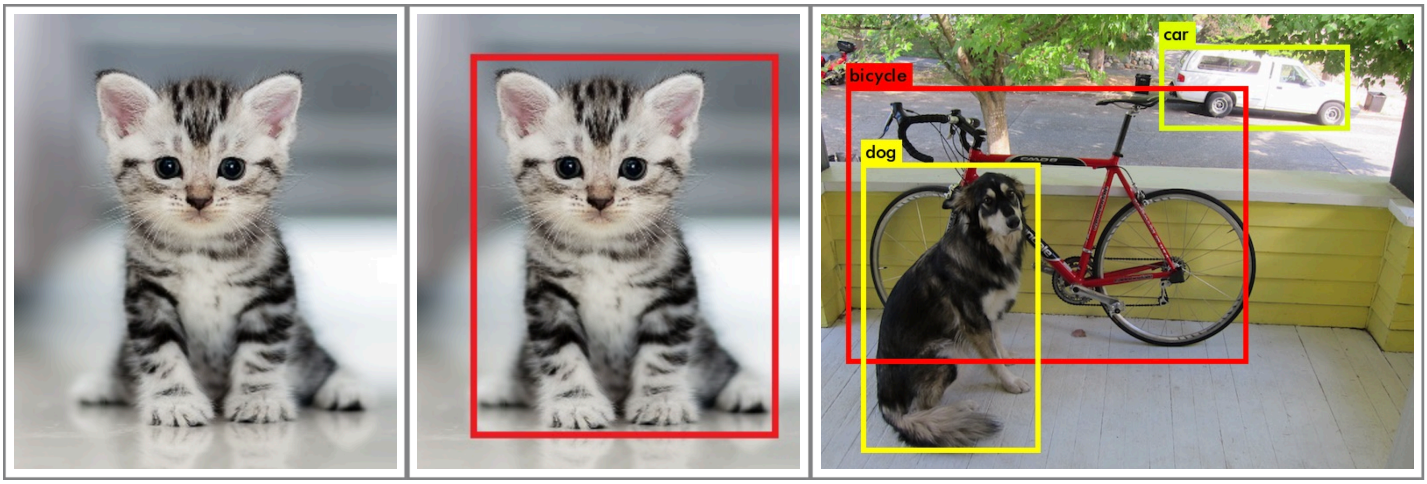


Image Classification Image Localization Image Detection

Landmark Detection

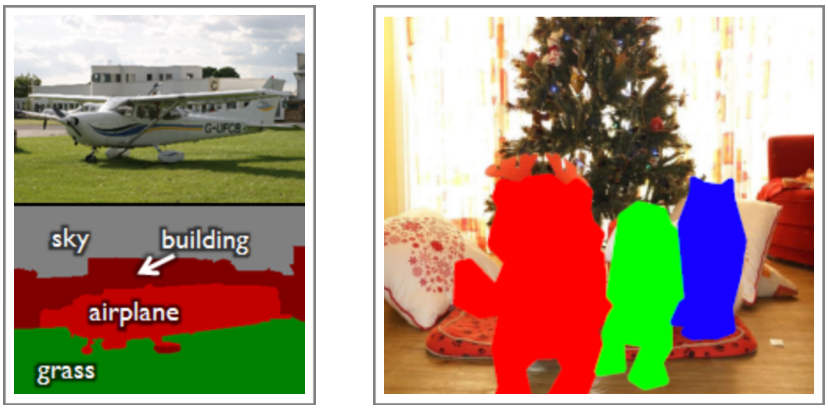
- want to output some points
- corners of the eyes, corners of the mouth, and corners of the nose etc. in a face recognition problem. This can help a lot in the application like detecting the pose of the face.

Semantic Segmentation

- Want to label each pixel in the image with a category label. If there are two objects of the same class is intersected, we won't be able to separate them.

Instance Segmentation

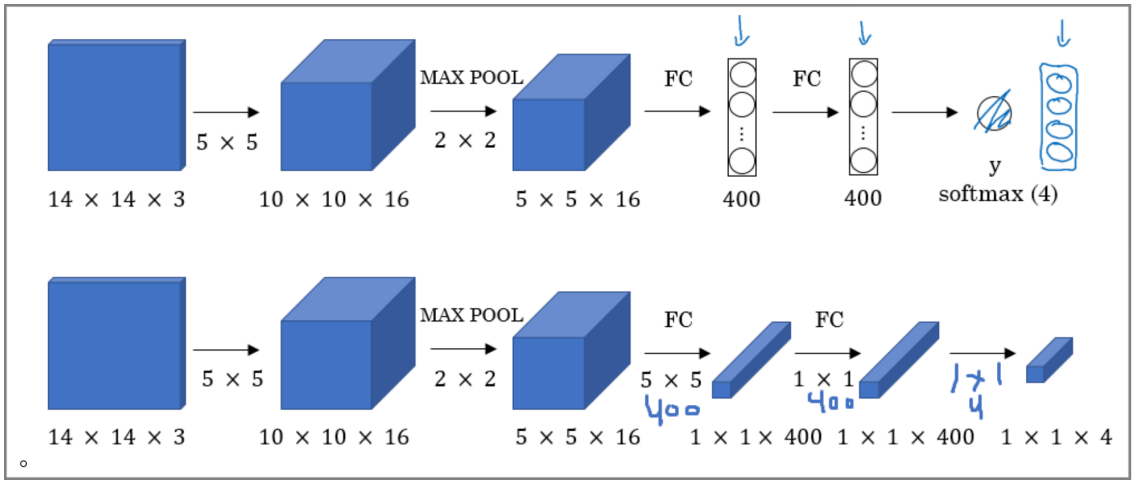
- Want to label each pixel by distinguishing every instance.



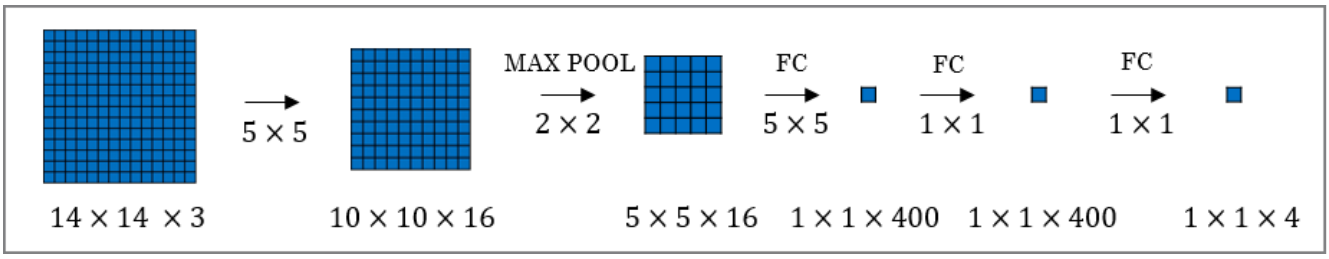
Semantic Segmentation Instance Segmentation

Object Detection Using Sliding Windows

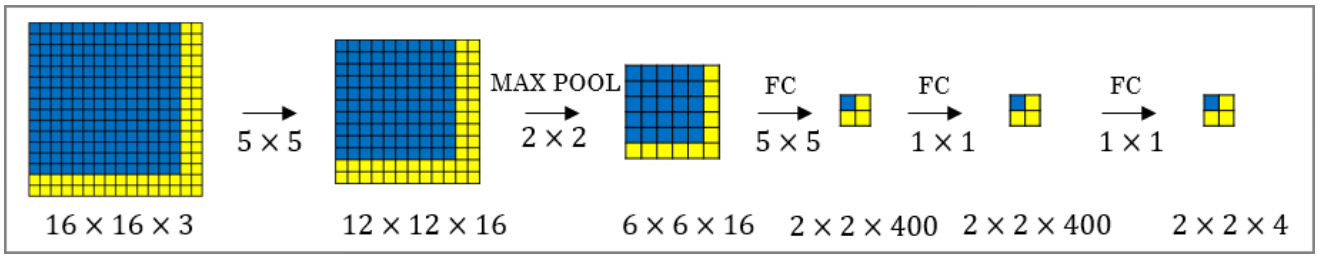
- **<u>Sliding windows detection algorithm</u>**
  - Decide a rectangle size
  - Split your image into rectangles of the size you picked. Every region should be covered. You can use some strides.
  - For each rectangle, feed the image into the Conv net and decide if it is a car or not.
  - Pick larger/smaller rectangles and repeat the process.
  - Store the rectangle that contains the cars.
  - If two or more rectangles intersects, choose the rectangle with the best accuracy.
- Disadvantage 1: heavy computation
  - Turn FC layer into Convolutional layer using 1×1 convolutions.
- Disadvantage 2: inaccurate position of the returned rectangle



Reduce computation by convolutional implementation



Conv Net



16×16 Input Image resulting in 4 sliding windows

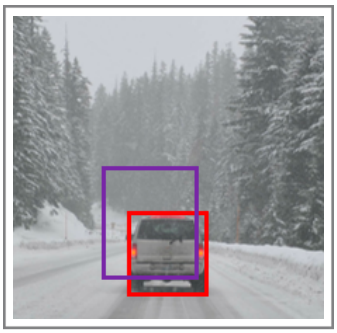
You Only Look Once (YOLO)

- Say you have an image of 100×100.
  - Place a 3×3 grid on the image. For more smother results, you should use 19×19 for the 100×100 input.
  - Apply the classification and localization algorithm we discussed in a previous section to each grid. bx and by would represent the center point of the object in each grid and will be relative to the box, ranging between 0 and 1. bh and bw will represent the height and width of the object which can be greater than 1.0.
  - Do everything at once with the convolution sliding window. If Y shape is 1×8 as we discussed before, then the output of the 100×100 image should be $3 \times 3 \times 8$ which corresponds to 9 cell results.
  - Merge the results using predicted localization mid point.
- Pros: great speed and high accuracy with a single CNN network for both classification and localizing the object using bounding boxes.
- Cons: if have found more than one object in one grid box. Not good at detecting smaller objects.

Bounding Box Prediction

YOLO Optmization

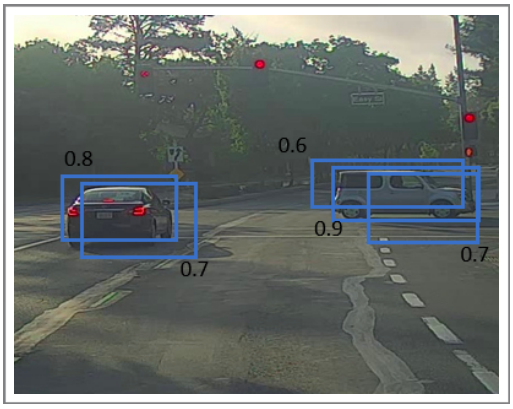
- **Intersection Over Union**



IOU

- Computes size of intersection and divide it by the union. More generally, IoU is a measure of the overlap between two bounding boxes.
- If $IOU = \frac{\text{intersection area}}{\text{union area}} > 0.5$, then it's good.
- The higher the IOU, the more accurate the algorithm is.

- **Non-max Suppression**
  - One of the problems we have addressed in YOLO is that it can detect an object multiple times. Non-max Suppression ensures that YOLO detects the object just once.



Non-max Suppression

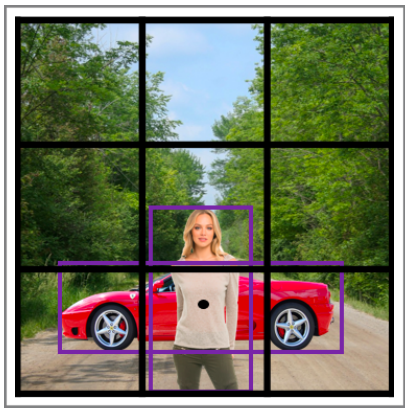
- Assume that we are targeting one class as an output class.
- Y shape should be $[p, bx, by, bh, bw]$ where p is the probability if that object occurs.
- Discard all boxes with $p < 0.6$
- While there are any remaining boxes, pick the box with the largest p and discard any remaining box with $IOU > 0.5$ with a box output in the previous step (any box with high overlap).

  - If there are multiple class/object types c you want to detect, you should run the non-max suprression c times, once for every output class.

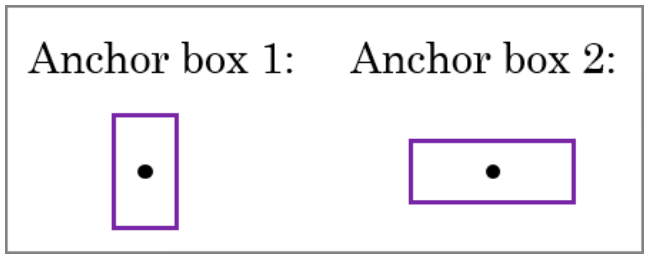
- **Anchor Boxes**
  - In YOLO, a grid only detects one object. What if a grid cell wants to detect multiple object?
  - Previously, each object in training image is assigned to grid cell that contains that object's midpoint.
  - With two anchor boxes, each object in training image is assigned to grid cell that contains that object's midpoint and anchor box for the grid cell with highest IOU. You

have to check where your object should be based on its rectangle closest to whicn anchor box.

- To use two anchor boxes, simply repeat another anchor.
  $y = [p, bx, by, bh, bw, c1, c2, c3, p, bx, by, bh, bw, c1, c2, c3]$



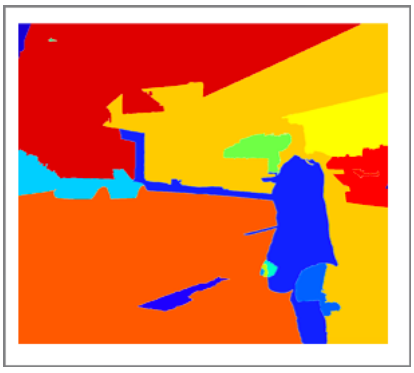
Anchor Boxes



Two anchor boxes shape

- You may have two or more anchor boxes but you should know their shapes. People used to just chose them by hand, maybe 5 or 10 anchor box shapes that spans a variety of shapes that cover the types of objects you seem to detect frequently. You may also use a k-means algorithm on your dataset to specify that.

Region Proposals (R-CNN: Regions with Conv Nets)



Segmentation Algorithm

-R-CNN tries to pick a few windows and run a Conv Net on top of them unlike YOLO, which processes a lot of areas where no objects are present. Such algorithm is called segmentation algorithm.

- If, for example, the segmentation algorithm produces 2000 blob, then we should run our CNN on top of these blobs.

- There has been a lot of work regarding R-CNN trying to make it faster. However, most of the implementation of faster R-CNN are still slower than YOLO.
- **R-FCN** is similar to Faster R-CNN, but more efficient.
- Other algorithms that use one shot to get the output includes **SSD and MultiBox**.