Convolutional Neural Networks

{ وَاتَّقُوا اللَّهَ وَيُعَلِّمُكُمُ اللَّهُ وَاللَّهُ بِكُلِّ شَيْءٍ عَلِيمٌ }

👨🏽‍💻 **github**: elma-dev

🔗 **linkden**: EL MAJJODI Abdeljalil

Computer Vision & DL

- Example Of a Computer Vision Applications:
  - Image Classification
  - Object Detections
    - Detect objects in the Image
  - Neural Style Transfer
    - Change the style of an image using another one.

Problem

⇒ When we talk about images we talk about huge data.

⇒ The challenge facing CV is image can be so large.

- We Need a fast and accurate method to work with this.

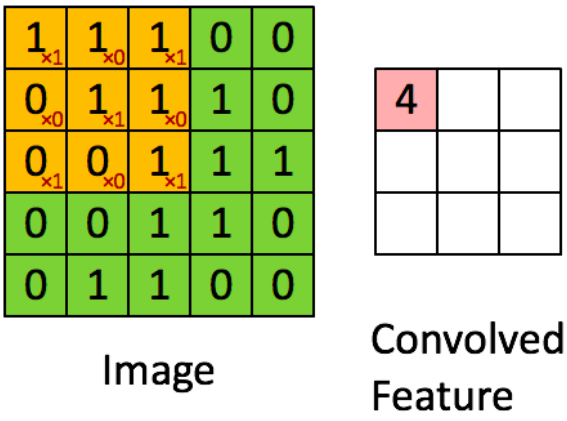
⇒ Let’s do some calculations:

```
image.shape=[1000,1000,3]
⇒ input.shape=1000*1000*3=3M features
⇒ hiddenLayer.shape=[1000]
⇒ W.shape=[hideenLayer,input]=[1000,3M]=3B
```

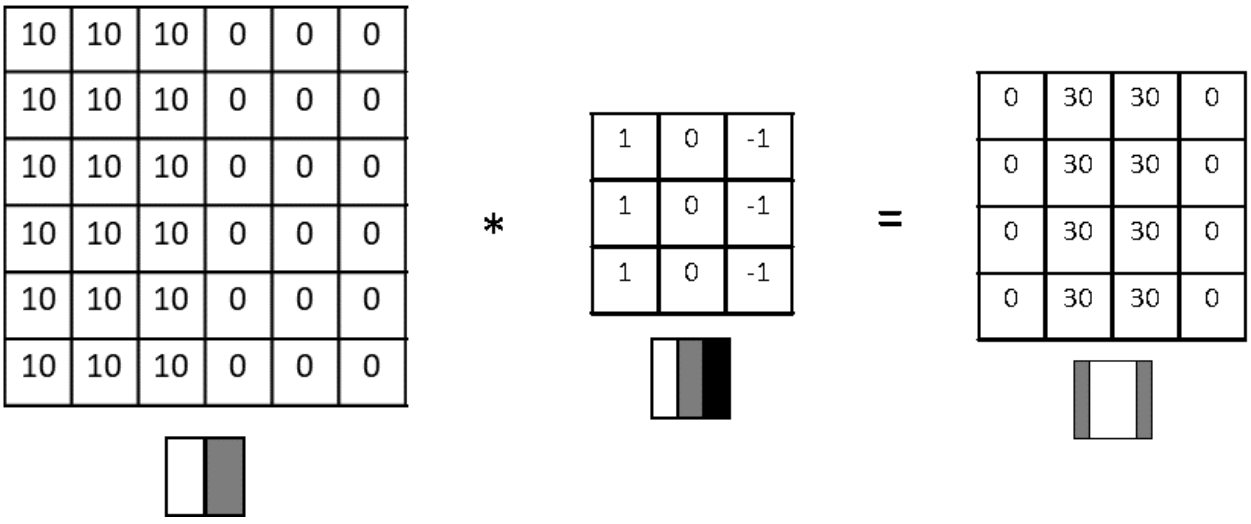
So One of The Solutions to Solve this Problem is Convolution layer instead of the Fully Connected Layer.

Edge Detection

- **Early** layers of CNN might detect edges then the **middle** layers will detect parts of objects and the **later** layers will put the these parts together to produce an output.
- With The Convolution operation we can also detect image edges (H,V,fully).



⇒ An example of convolution operation to detect vertical edges:



→ **6×6 matrix** convolved with **3×3 filter/kernel** gives us a **4×4 matrix.**

```
#Convolution Operation With TF
tf.nn.conv2d
#CO With Keras
tf.keras.Conv2D
```

→ 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 d**ark region followed by a white region** it will **give us negative values**. To solve this we can use the abs function to make it positive.

Find The Filter

⇒ We have various filter for example :

- Horizontal / Vertical / Sobel Filter/ Scharr Filter ...

⇒ With Deep Learning We Can Put The Values of Filter As a Weights And try to Find it.

Filtering General Rule

$$M(n,n) * M(f,f) = M(n-f+1, n-f+1)$$

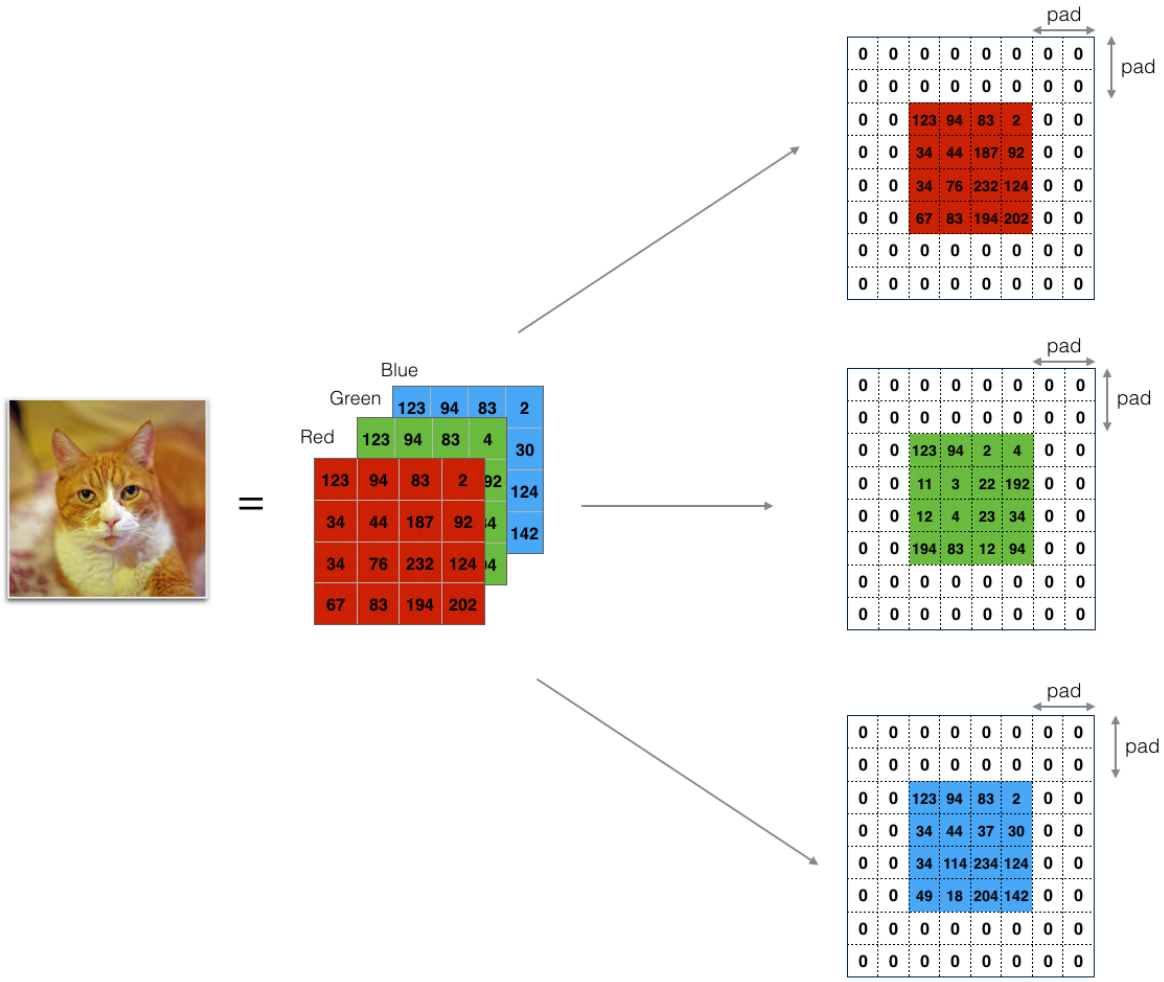
Padding Operation

⇒ As we see after each filtering operation the dim of image decrease. so after applicate filtering multiple times we can't applicate more filtering operation.

> We want to apply convolution operation multiple times, but if the image shrinks we will lose a lot of data on this process. Also the edges pixels are used less than other pixels in an image.

⇒ To avoid this problem we need to use **Padding Operation.**

> With this technic we adding some rows and columns to the original image. **we call P the number of pixels we add in the top, bottom, right and left of image.**



$$M(n,n) * M(f,f) = M(n+2p-f+1, n+2p-f+1)$$

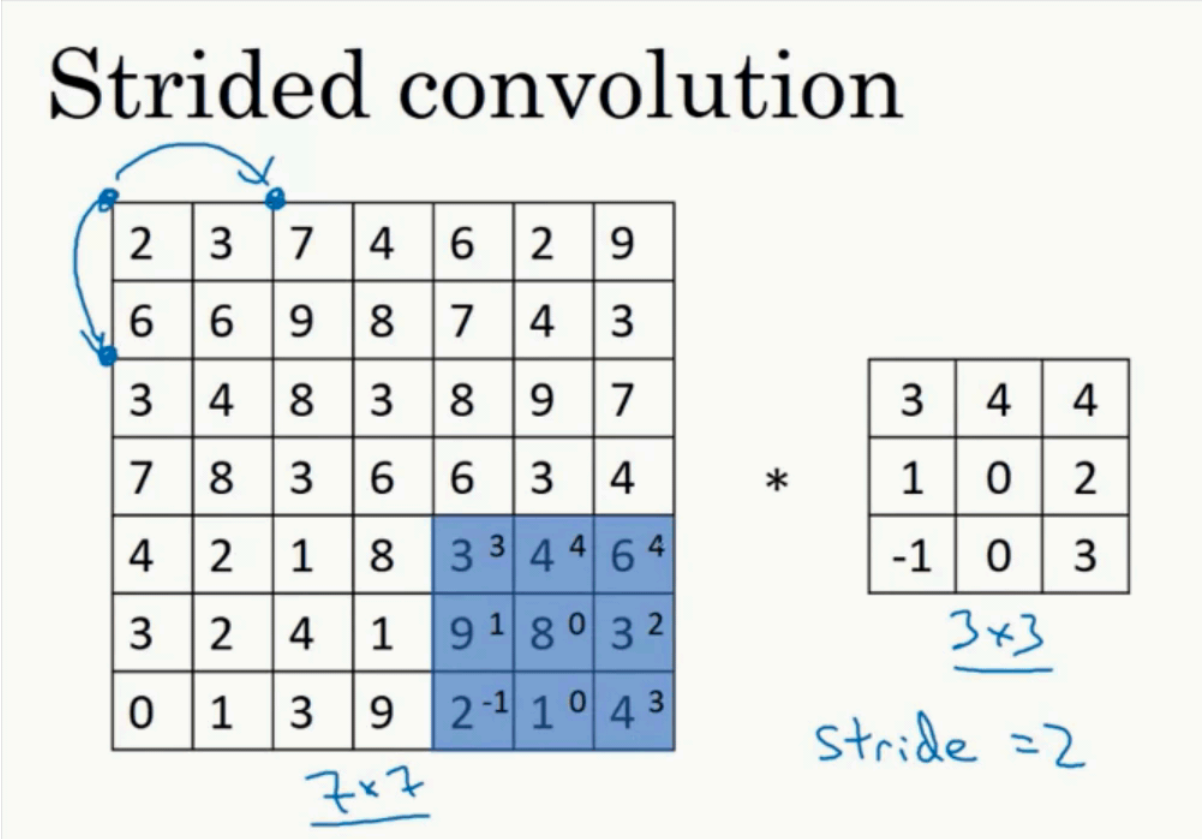
Always:

$$n+2p-f+1 = n => p = (f-1)/2$$

```
import numpy as np
np.pad(image,((1d),(2d),(3d),(4d)),mode="constant",constant_values=(0,0))
```

Strided Convolution

⇒ When Applicate convolution operation we need to know how of pixel (**S**) we will jump when we are convolving filter.



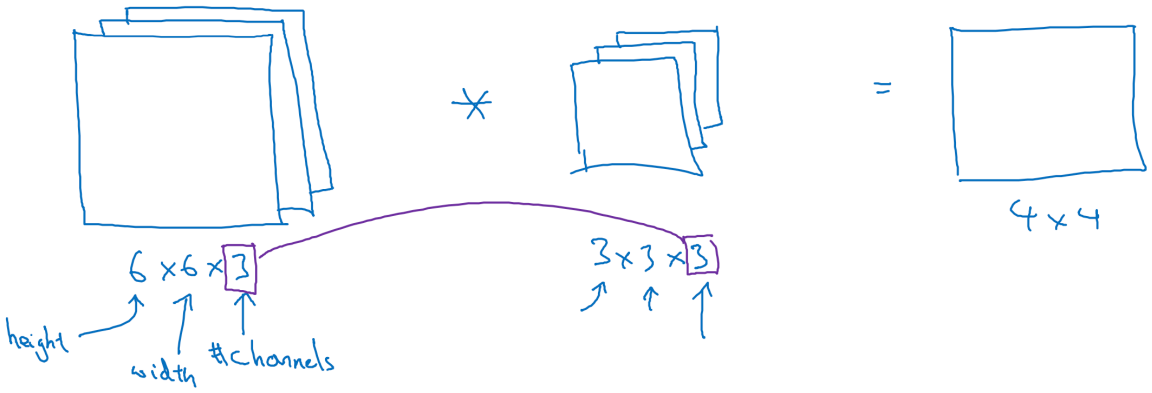
Strided General Rule

$$M(n,n) * M(f,f) = M((n+2p-f+)/s+1)$$

- In case (n+2p-f)/s + 1 is fraction we can take **floor** of this value.

Convolutions over volumes

> When we have image input with the dimension 3D (RGB, height, width, # of channels) or more we should to applicate a filter with the same dimension (height_f, width_f, same # of channels) .

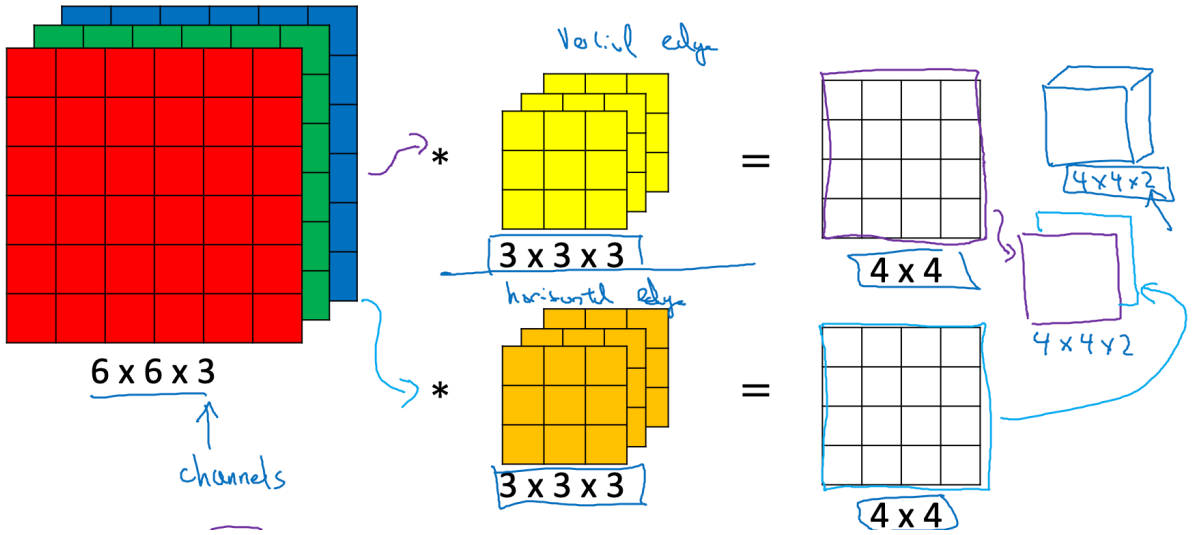


⇒ In this example we have input image with 6*6*3 and filter with (3,3,3) as we see the tow matrix has the same number of channels.

⇒ The output is 2D (4*4)

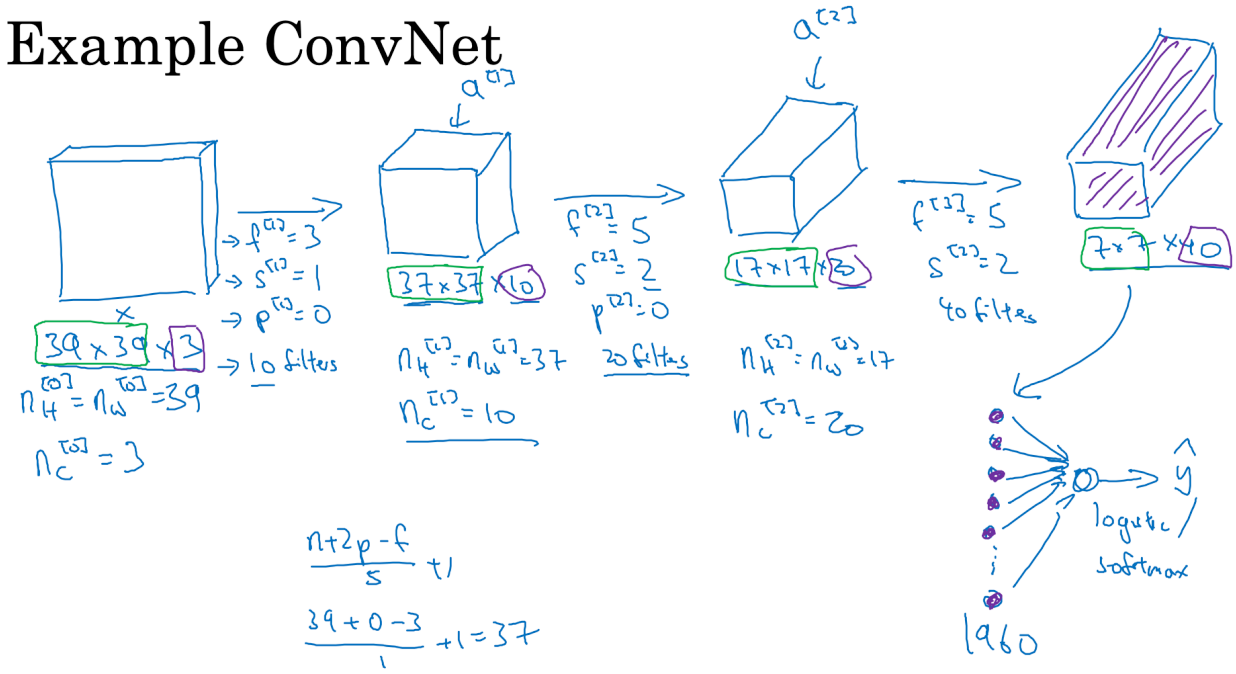
Multiple Filter:

> To detect more informations/features in images you should to use more filters for example (VE filter / HE...)



Hyperparameters
f[l] = filter size

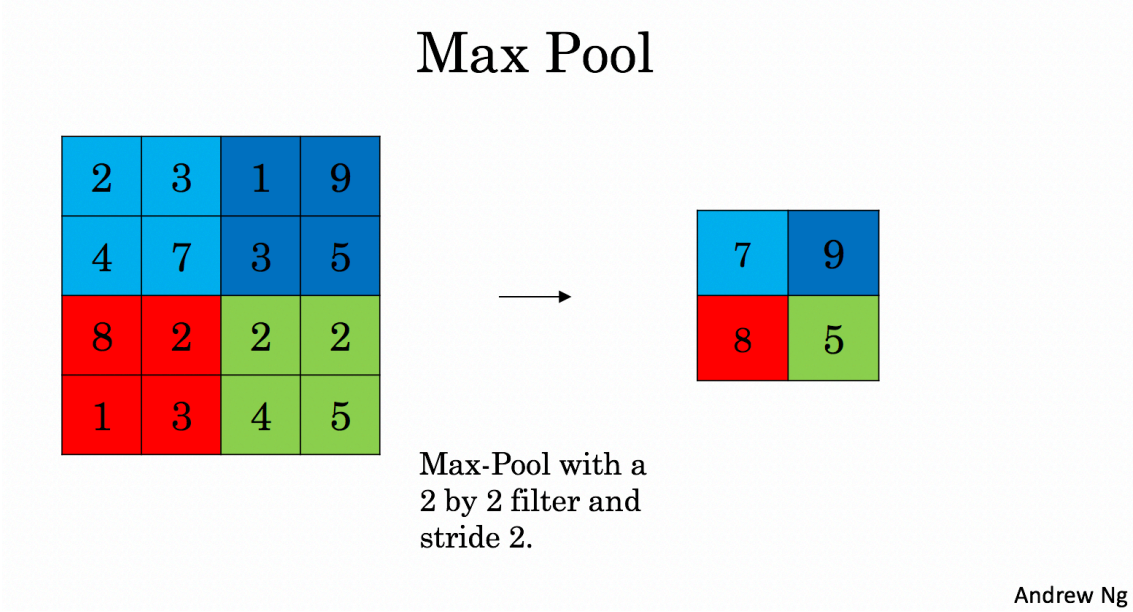
p[l] = padding # Default is zero
s[l] = stride
nc[l] = number of filters
Input: n[l-1] x n[l-1] x nc[l-1] Or nH[l-1] x nW[l-1] x nc[l-1]
Output: n[l] x n[l] x nc[l] Or nH[l] x nW[l] x nc[l]
Where n[l] = (n[l-1] + 2p[l] - f[l] / s[l]) + 1
Each filter is: f[l] x f[l] x nc[l-1]
Activations: a[l] is nH[l] x nW[l] x nc[l]
A[l] is m x nH[l] x nW[l] x nc[l] # In batch or minbatch training
Weights: f[l] * f[l] * nc[l-1] * nc[l]
bias: (1, 1, 1, nc[l])



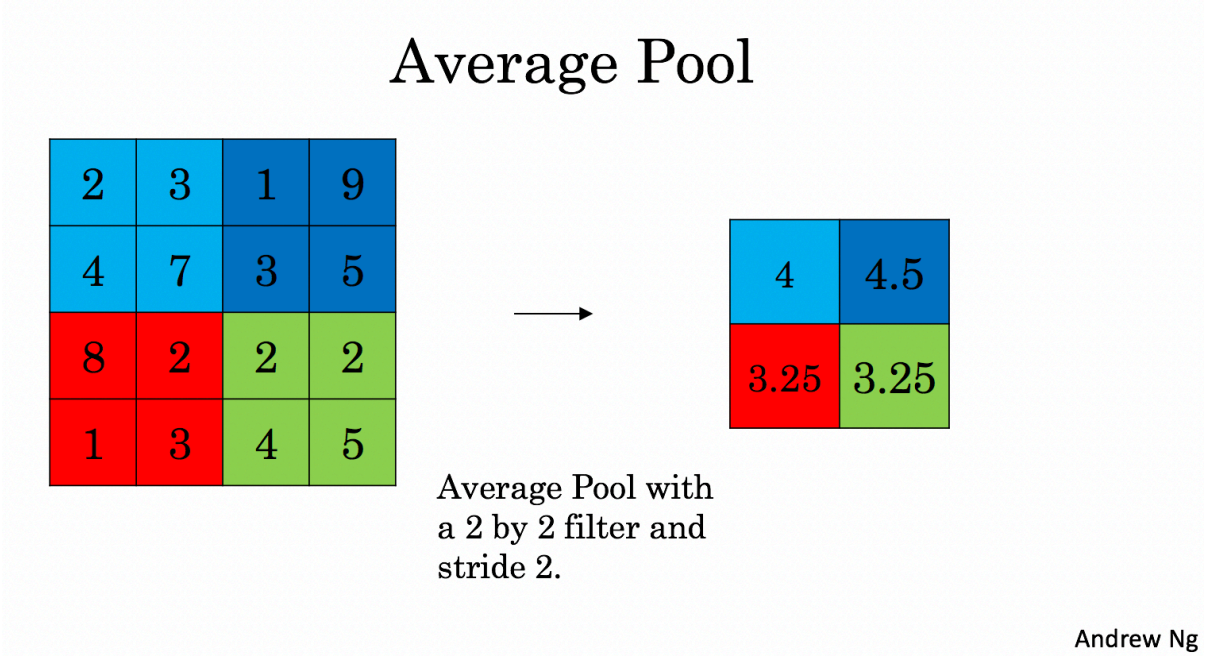
Input Polling

> The pooling (POOL) layer reduces the height and width of the input. It helps reduce computation, as well as helps make feature detectors more invariant to its position in the input. The two types of pooling layers are:

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

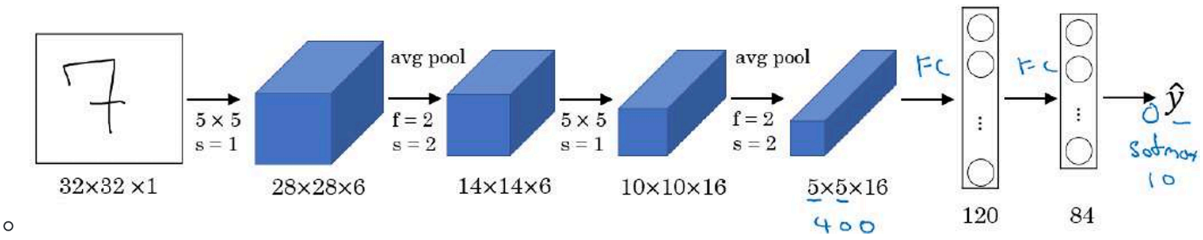


- **Average-pooling layer**: slides an (f,f) window over the input and stores the average value of the window in the output.



⇒ These pooling layers have no parameters for backpropagation to train. However, they have hyperparameters such as the window size *f*. This specifies the height and width of the *f*×*f* window you would compute a *max* or a*verage* over.

Example:

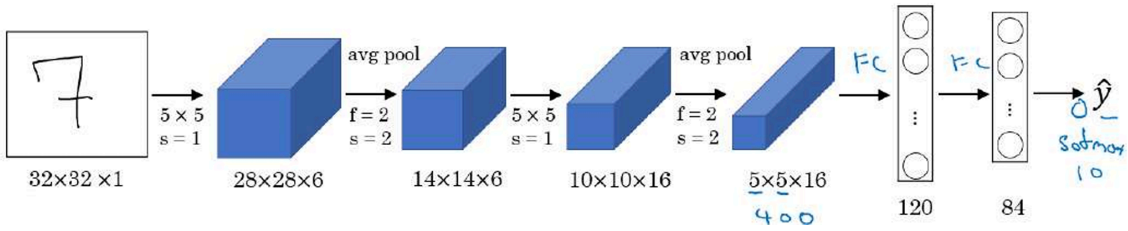


Classic Neural Network

- Some neural networks architecture that works well in some tasks can also work well in other tasks.
- Some Examples of Classical CNN:
  - **LetNet-5**
  - **AlexNet**
  - **VGG**
- **ResNet** is a last winner of ImageNet Competition and it has 150 layers.
- **Inception** Architecture made by Google.

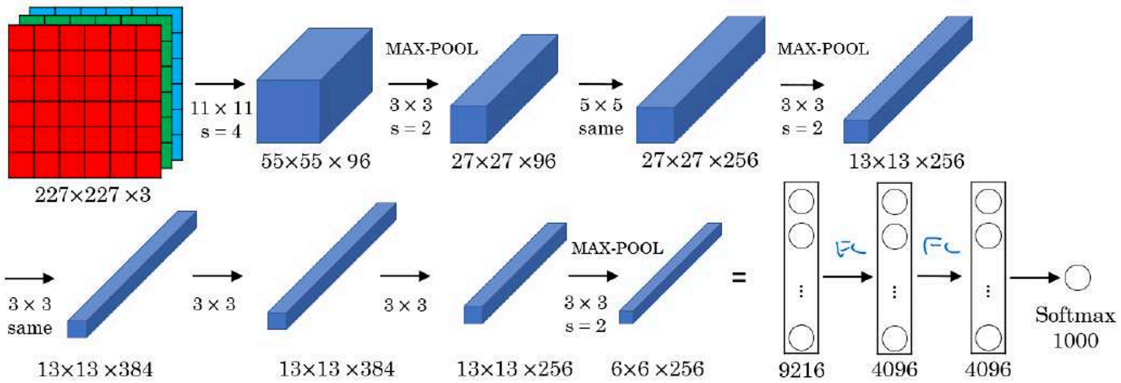
→ In this case we will talk about Calssic Neural Network (LetNet-5, AlexNet, VGG)

▼ Classic Neural Network Architecture LetNet5 (1988)



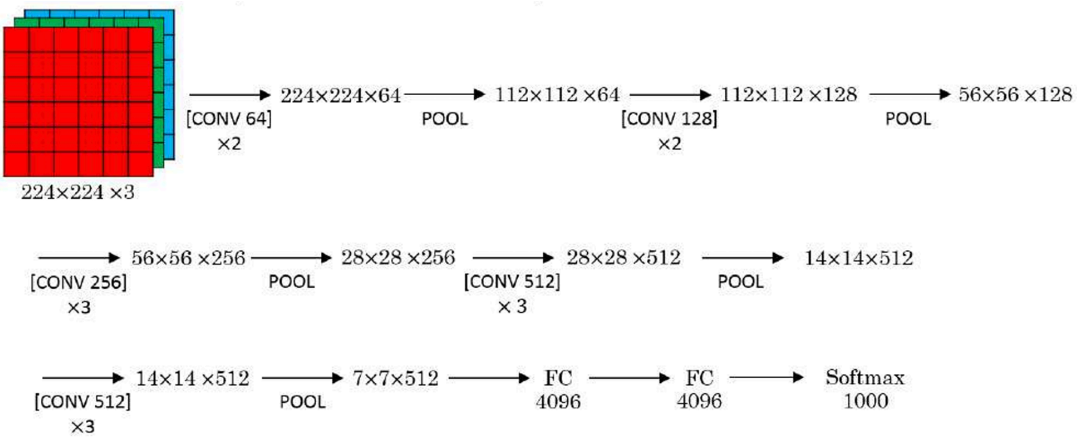
- The Goal of this architect was classify the gray scale image carry a digit.
- it has 60K params

▼ AlexNet(Alex Krizhevsky 2012,8L)



- The goal for the model was the ImageNet challenge which classifies images into 1000 classes.
- Conv ⇒ Max-pool ⇒ Conv ⇒ Max-pool ⇒ Conv ⇒ Conv ⇒ Conv ⇒ Max-pool ⟹ Flatten ⟹ FC ⟹ FC ⟹ Softmax
- Has 60 Million parameter compared to 60k parameter of LeNet-5.
- It used the RELU activation function

▼ VGG-16

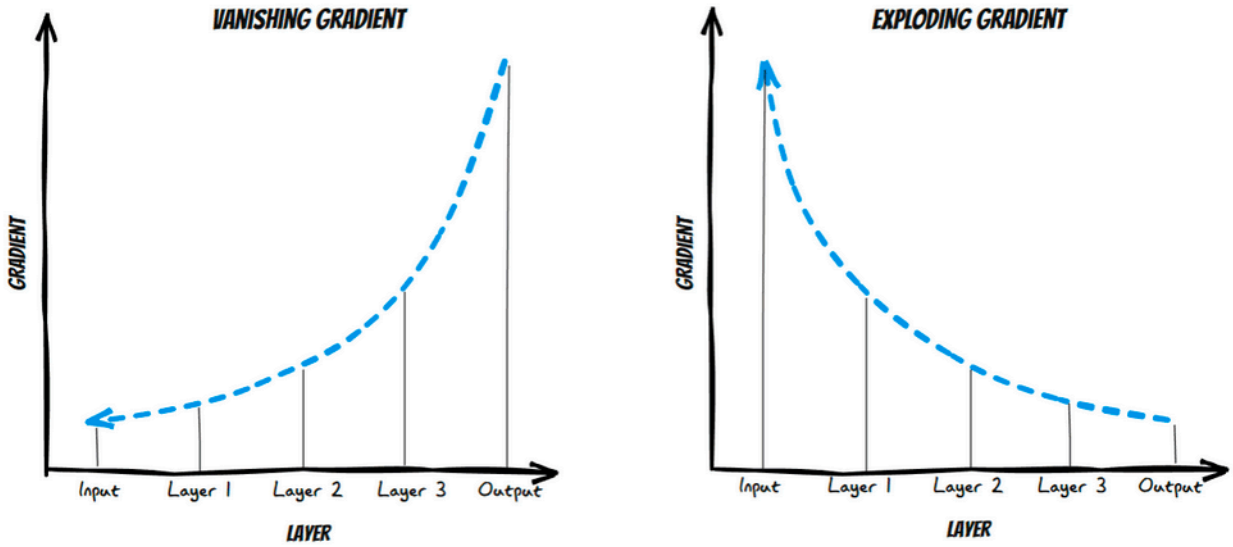


- A modification for AlexNet

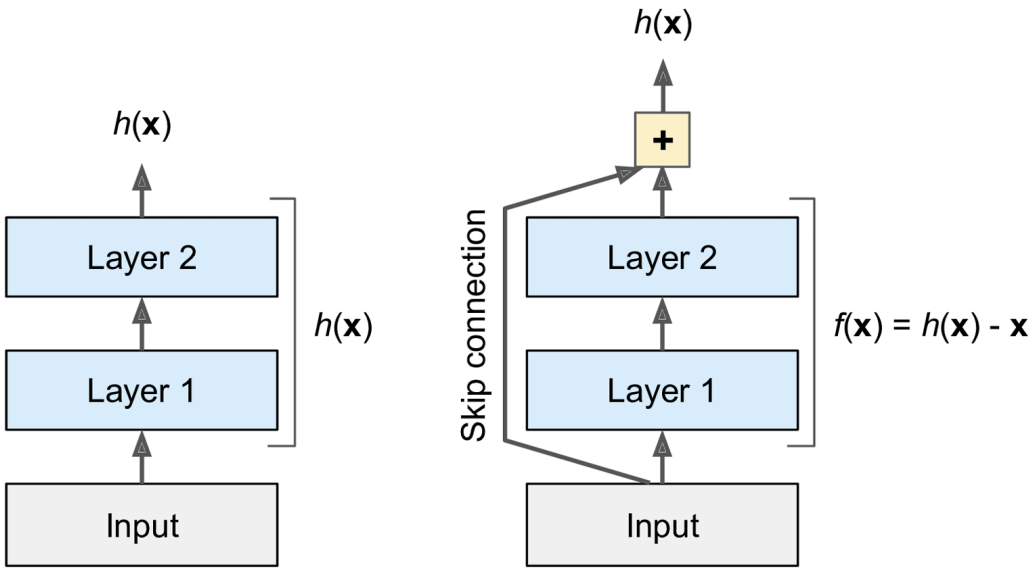
- Focus on having only these blocks:
  - CONV = 3 X 3 filter, s = 1, same
  - MAX-POOL = 2 X 2 , s = 2

Residual Networks (ResNet 152Ls)

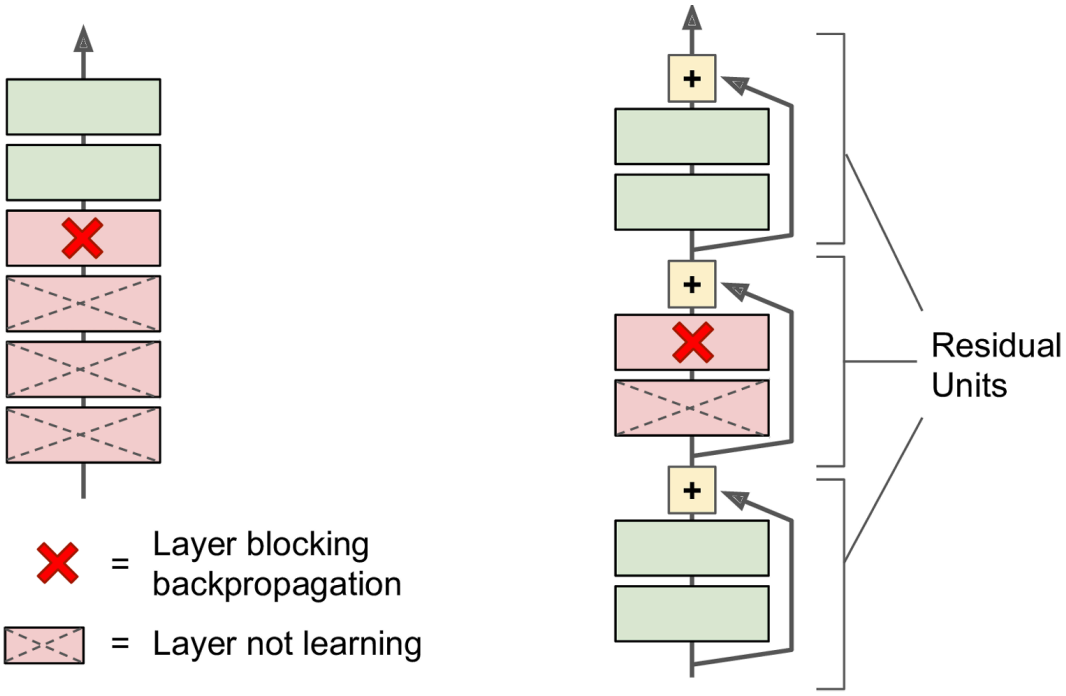
→ Very, very deep NNs are difficult to train because of **vanishing** and **exploding** gradients problems.



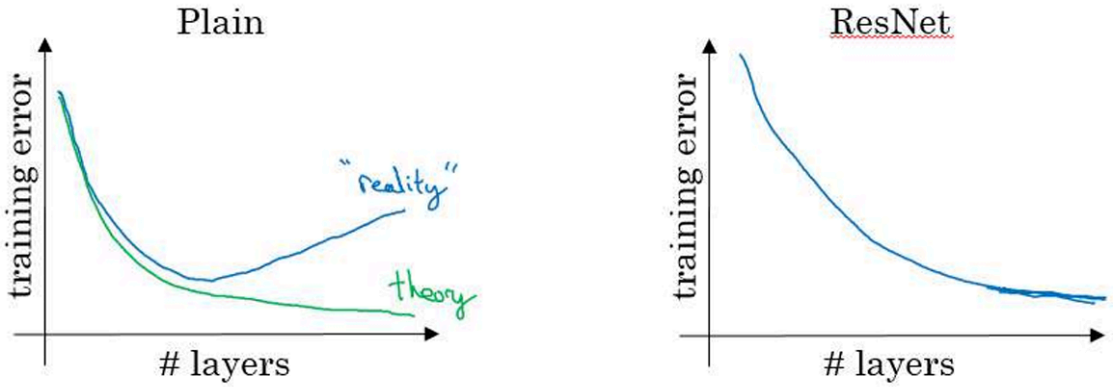
→ To solve this Problem we use **Skip Connection Method**(shortcut connections)



> When you initialize a regular neural network, its weights are close to zero, so the network just outputs values close to zero. If you add a skip connection, the resulting network just outputs a copy of its inputs; in other words, it initially models the identity function. If the target function is fairly close to the identity function (which is often the case), this will speed up training considerably. Moreover, if you add many skip connections, the network can start making progress even if several layers have not started learning yet (see Figure 14-16). Thanks to skip connections, the signal can easily make its way across the whole network. The deep residual network can be seen as a stack of residual units, where each residual unit is a small neural network with a skip connection.

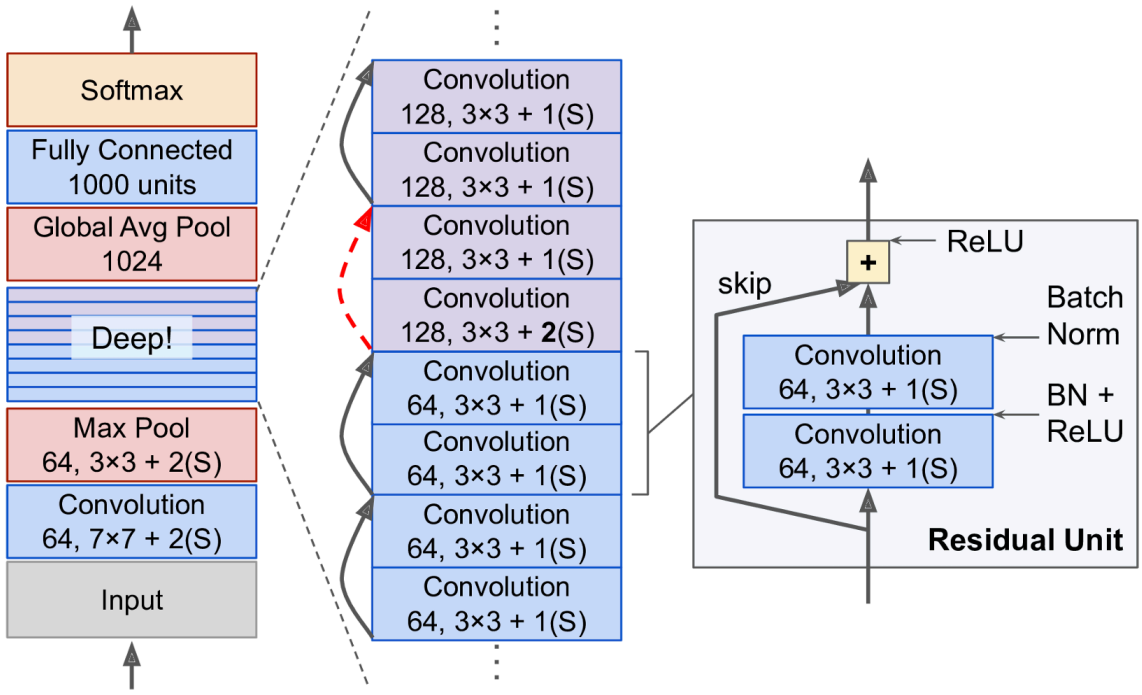


- These networks can go deeper without hurting the performance. In the normal NN - Plain networks - the theory tell us that **if we go deeper we will get a better solution** to our problem, but **because of the vanishing and exploding gradients problems the performance of the network suffers as it goes deeper.** Thanks to Residual Network we can go deeper as we want now.



→ On the left is the normal NN and on the right are the ResNet. As you can see the performance of ResNet increases as the network goes deeper

ResNet Architecture



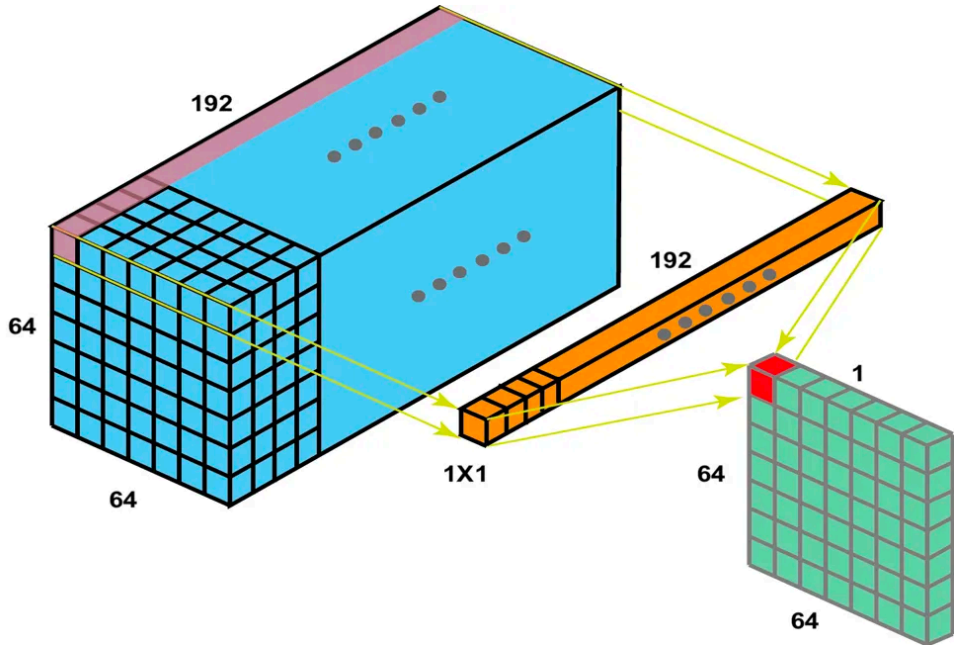
- S : Strided

Network in Network and 1 X 1 convolutions

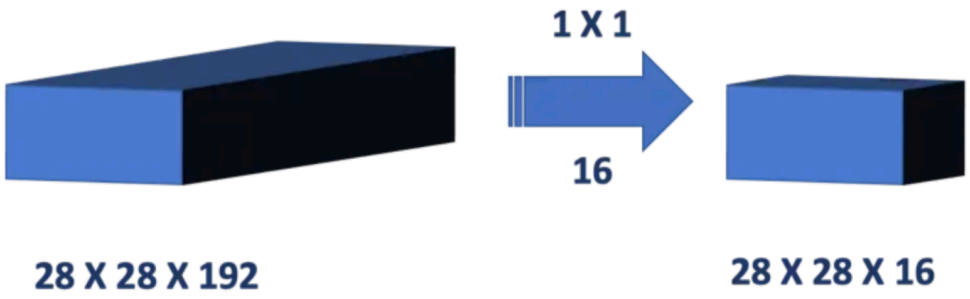
→ This method is very helpful when we want to reduce or augmentation of the number of channels in Image.

⇒ Example:

- **Reduce nbr channels**

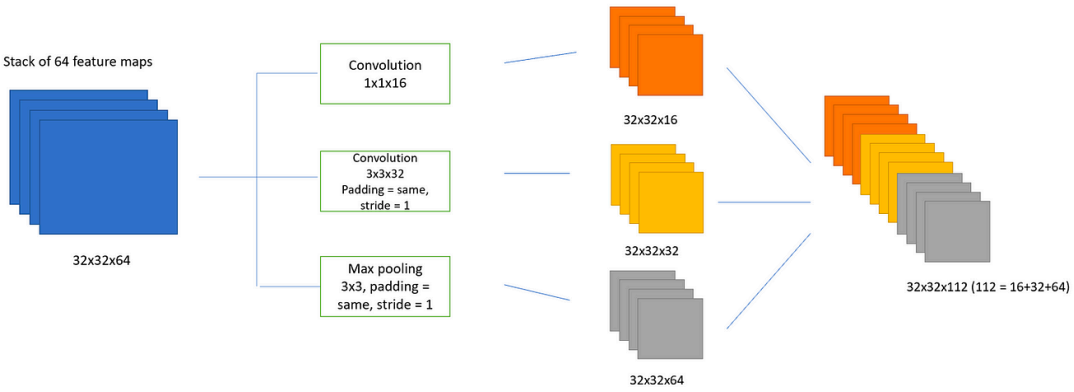


- **Augmention**

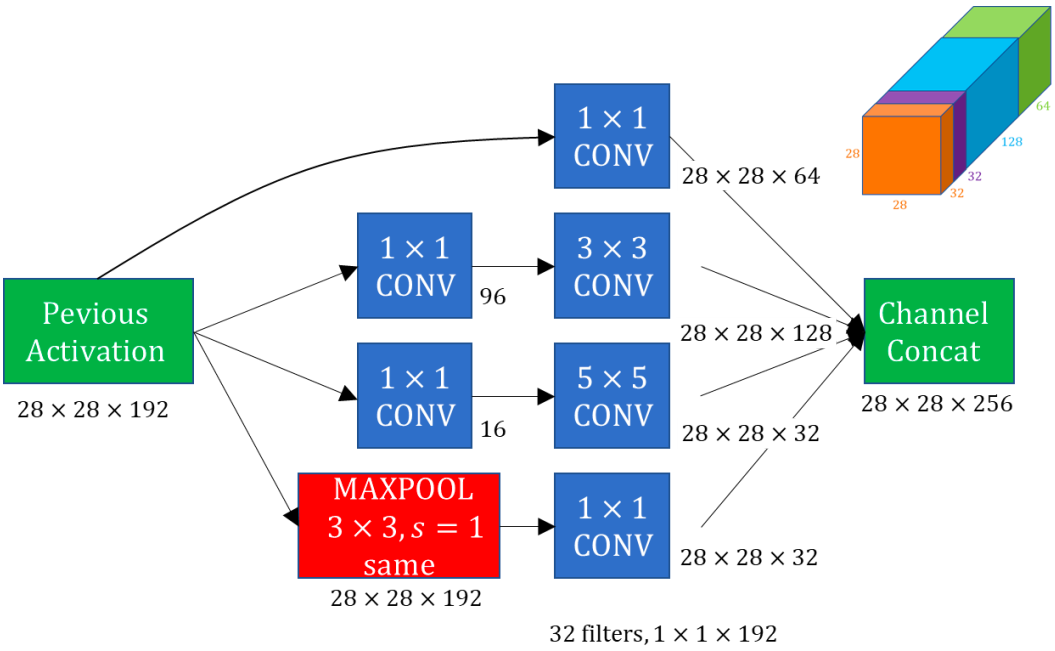


Inception Network (GoogleNN 2014)

> Inception schema came with new type of schema that's can gave the ability to use multiple convolutional layers of different filter sizes (e.g., 1×1, 3×3, 5×5) and pooling operations in the same layer and finaly combine the resulte as a one .

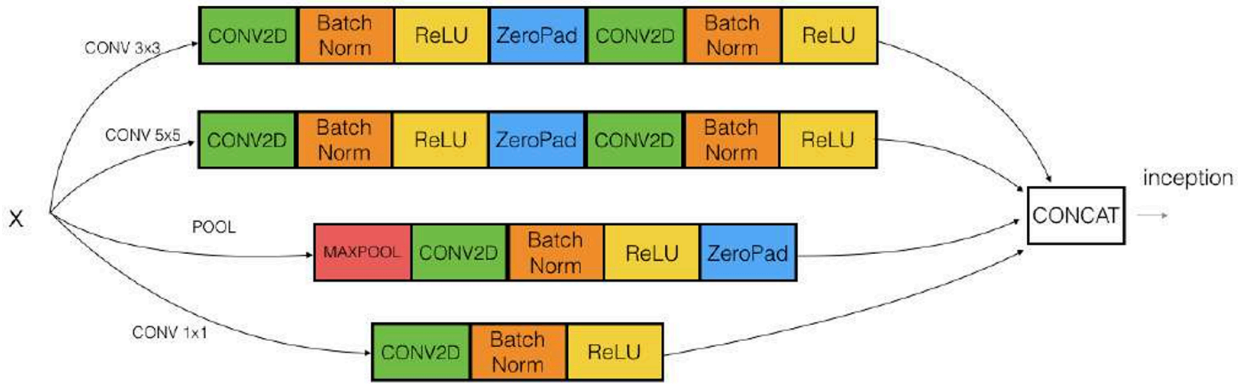


⇒ With this module we have alot of calculation and to avoid this we add a layer before called **Bottleneck**



- It turns out that the 1 × 1 Conv won't hurt the performance.
- calculation before 120Mil after 12M

Example in keras:



Final Schema GoogleNet:

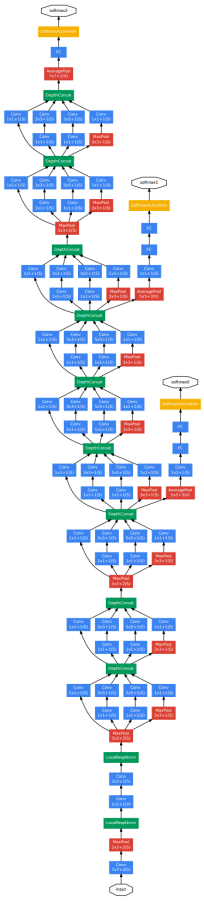


Figure 3: GoogLeNet network with all the bells and whistles

- Concatenation of inception blocs.

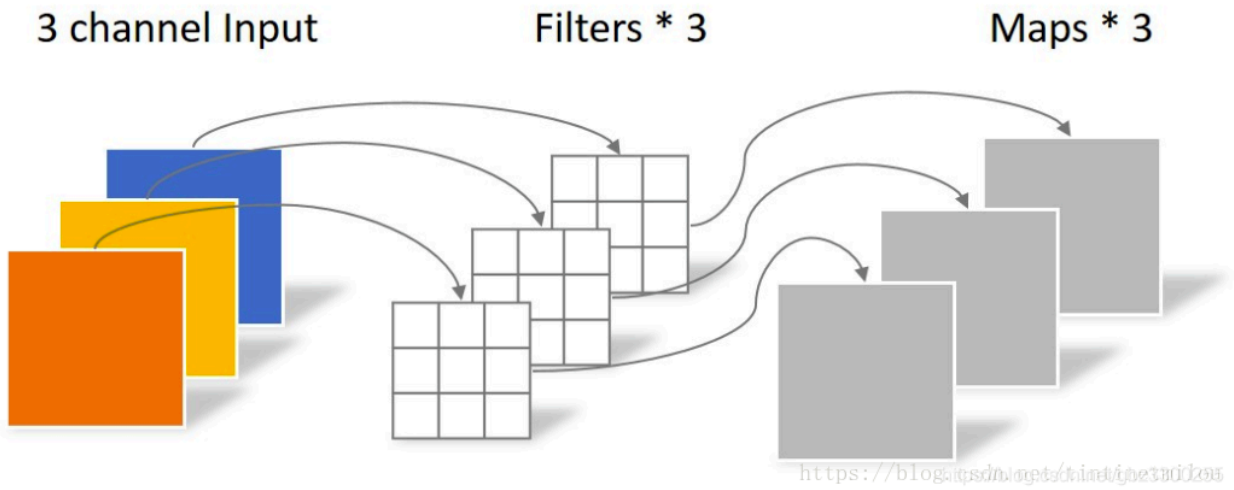
- Some times a Max-Pool block is used before the inception module to reduce the dimensions of the inputs.
- There are a 3 Sofmax branches at different positions to push the network toward its goal. and helps to ensure that the intermediate features are good enough to the network to learn and it turns out that softmax0 and sofmax1 gives regularization effect.
- Since the development of the Inception module, the authors and the others have built another versions of this network. Like inception v2, v3, and v4. Also there is a network that has used the inception module and the ResNet together.

MobileNet(2017)

> Another foundational convolutional neural network architecture used for computer vision. Using MobileNets will allow you to build and deploy new networks that work even in low compute environment, such as a mobile phone.

⇒ These models are specifically engineered to have a small memory footprint and low computational requirements while still achieving good performance in various computer vision tasks, such as image classification and object detection

Depthwise Convolution Operation



⇒ When we have input image (eg 6,6,3) in depthwise operation we should to have a filter without depth (3,3) and number of filter equal number of inputChannels .

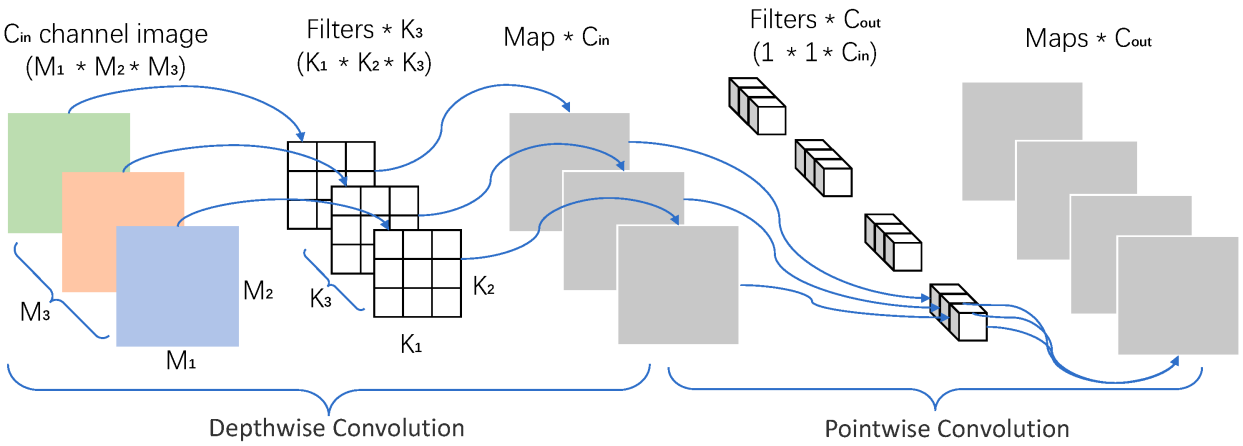
⇒ Each filter applicated in each depth for example filter1 → channel1, filter2→ channel2, filter3→channel3 .

⇒ Finally we combine the results

Pointwise Convolution Operation

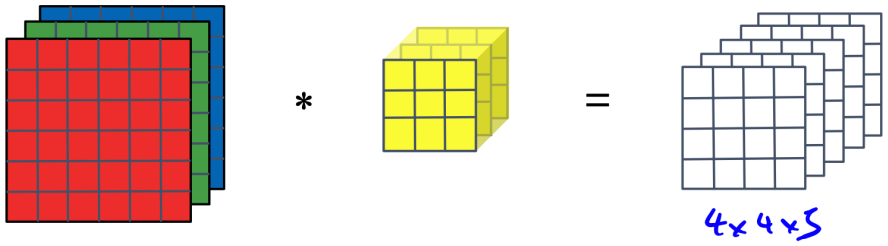
⇒ We need this operation just to increase number of channel in out results

⇒ App(1,1,nbrChannels) * nbrChannelWeNeed(nbrFilters)

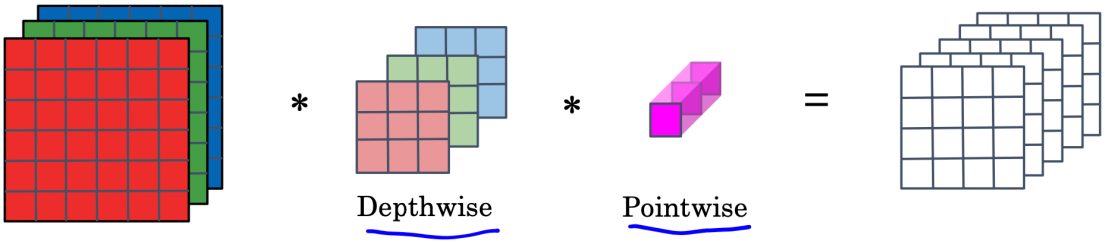


Normal Conv VS Depthwise Conv

Normal Convolution



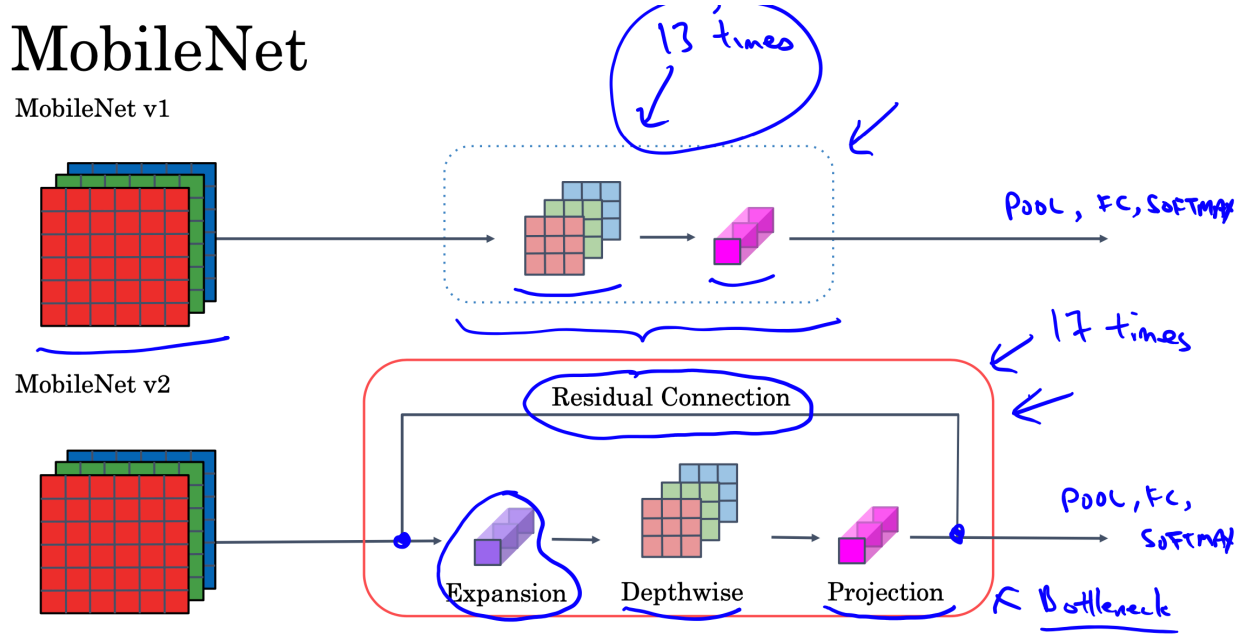
Depthwise Separable Convolution



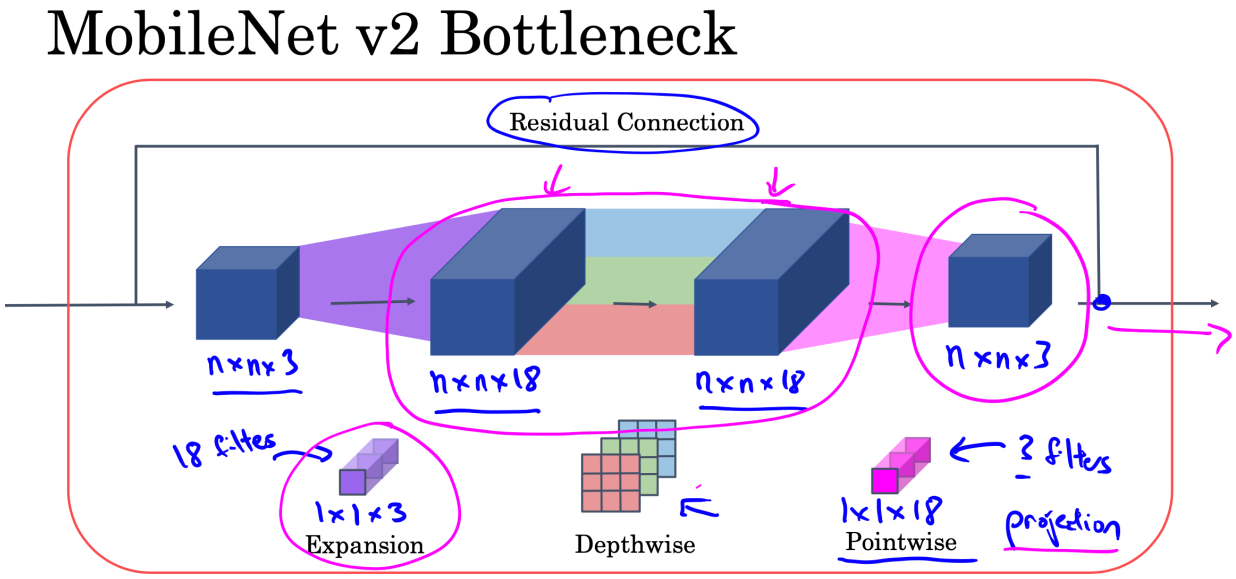
With Depthwise Seperale Convolution we reduce number of calculation 31% than Normal Convolution.

MobileNet Architecture

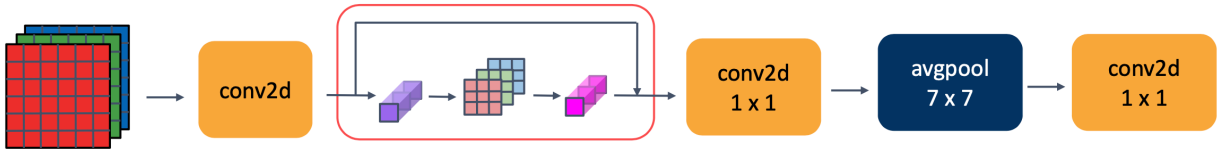
> In mobileNet Architecture we just try to apply Depthwise Separable Convolution multiple times (e.g 13). after we add a Fully Connected layers (FC,SoftMax)



- As we see in MobNetV2 there is also a **residual connection**, and before depthwise operation we have another opeartion called **Expansion** this operation came to increase the number of channels of input image it's the **opposite of Projection operation** .



Full Architect V2:



Example :

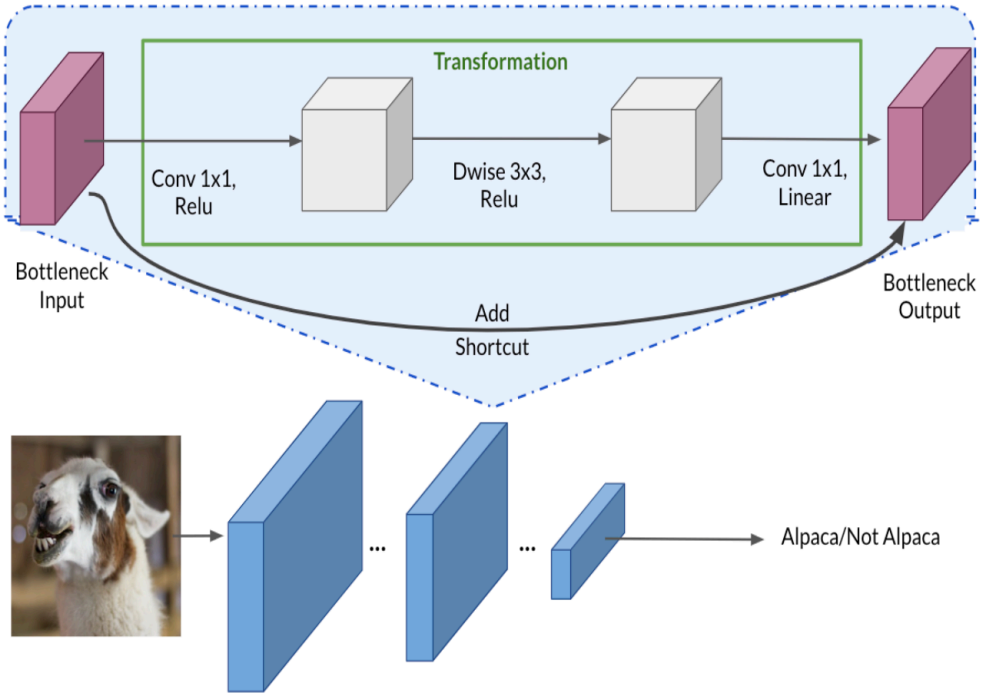


Figure 1 : MobileNetV2 Architecture
This diagram was inspired by the original seen here.

Practical advice for using ConvNets

> In this section we talk a little about trics could be helpful to use convNet architects in our projects .

▼ **Using Open Source Implementation**

- If you see a research paper and you want to build over it, the first thing you should do is to look for an open source implementation for this paper.

- Some advantage of doing this is that you might download the network implementation along with its parameters/weights. The author might have used multiple GPUs and spent some weeks to reach this result and its right in front of you after you download it.

▼ **Transfer Learning**

- If you are using a specific NN architecture that has been trained before, you can use this **pretrained parameters/weights instead of random initialization to solve your problem**.
- The pretrained models might have trained on a large datasets like **ImageNet**, **Ms COCO**, or **pascal** and took a lot of time to learn those parameters/weights with optimized hyperparameters. This can save you a lot of time.

Example

> Lets say you have a cat classification problem which contains 3 classes **Tigger**, **Misty** and **neither**. and you don't have **much a lot of data** to train a NN on these images.

⇒ Andrew Ng Recommds this :

- Download a good NN with its weights.
- Remove the softmax activation layer and put your own one
- Make the network learn only the new layer while other layer weights are fixed/frozen (trainable = 0 or freeze = 0).

⇒ **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 of your images and save them to disk**. And then use these representation to a **shallow NN network**. This can save you the time needed to run an image through all the layers.

⇒ **Another example:** If you have enough data, you can fine tune all the layers in your pretrained network but don't random initialize the parameters, leave

the learned parameters as it is and learn from there.

▼ **Data Augmentation**

What you should remember:

- When calling image_data_set_from_directory(), specify the train/val subsets and match the seeds to prevent overlap
- Use prefetch() to prevent memory bottlenecks when reading from disk
- Give your model more to learn from with simple data augmentations like rotation and flipping.
- When using a pretrained model, it's best to reuse the weights it was trained on.

What you should remember:

- To adapt the classifier to new data: Delete the top layer, add a new classification layer, and train only on that layer
- When freezing layers, avoid keeping track of statistics (like in the batch normalization layer)
- Fine-tune the final layers of your model to capture high-level details near the end of the network and potentially improve accuracy

---

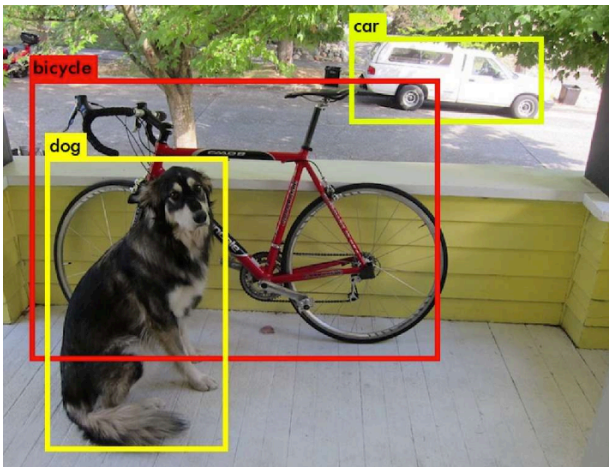
Object Detection

Object Localization

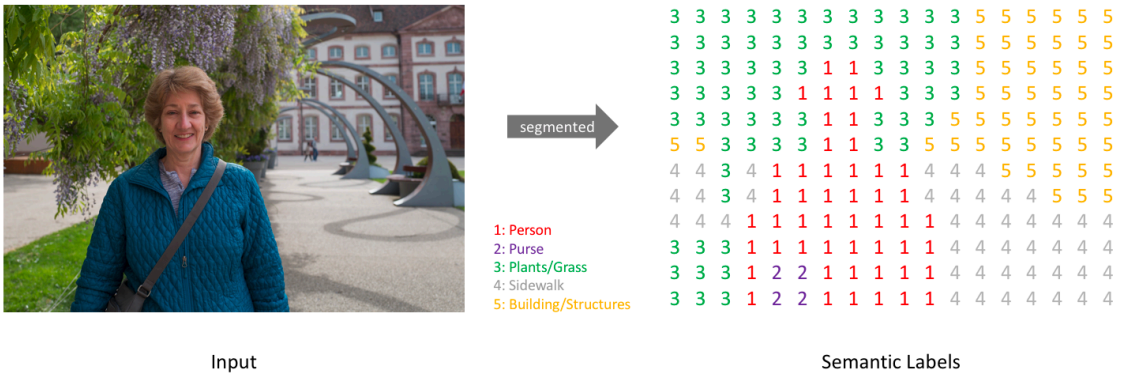
> Previously in image classification we try to classify an image and say the object in the image is class x, and we don’t care about the place of the object in the image. here the object localization came we try to classify the image and also find the place of the object in the image and that’s callled

“**Classification with localization**”. all we need is just tech the model this things.

- There is also another things call “**Object Detection**” with it we tech the model to detect all object in the image that’s belong to a specific classes and give their location.



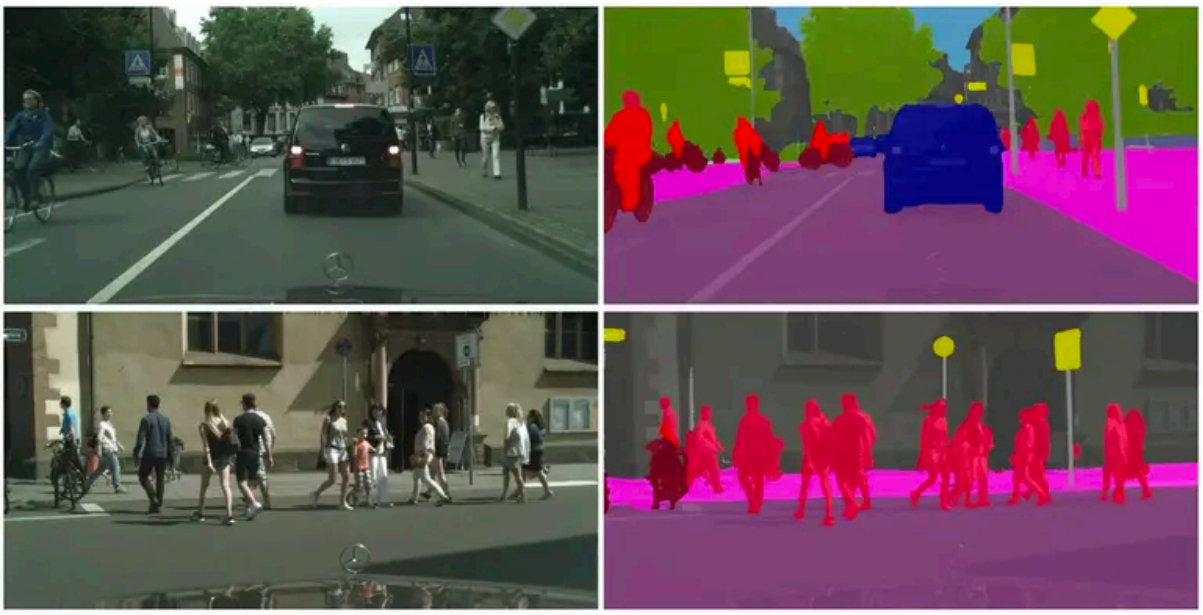
- **Semantic Segmentation :**



Input

Semantic Labels

  - label each pixel in the input with a category label. segmentation only care about pixels, it detect no object just pixels.



⇒ To make classification with localization we use a **Conv Net** with a **softmax** attached to the end of it and a four numbers **bx, by, bh,** and bw

> to tell you the location of the class in the image. **The dataset should contain this four numbers with the class too.**

```
Y=[
   Pc,  # Probablity of an object is presented
   bx,  # Bounding box
   by,  # Bounding box
   bh,  # Bounding box
   bw,  # Bounding box
   c1,  # The classes
   c2,
   ...
]
#example:
Y=[1,0,0,100,100,0,1,0]
```

⇒ **Loss Function** :

$$L(y, y') = (y1' - y1)^2 + (y2' - y2)^2 + (y1' - y1)^2 + ...$$

Landmark Detection

- With Landmark Detection we get some point in our image and those point have a signification for example nose place eyes place etc... this tech can help in a lot of application like detecting the pose of the face.,

⇒ The outputs show like this:

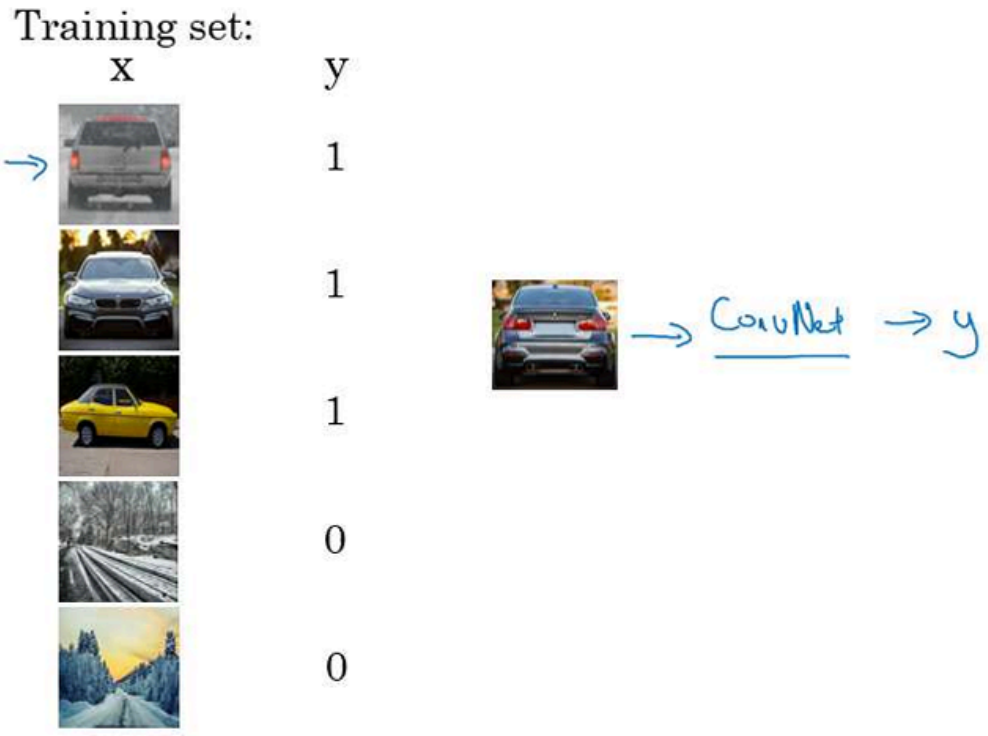
```
Y=[
   ThereIsAface,  #Proba of the face is presented 0 or 1
   l1x,
   l1y,
   ...,
   l64x,
   l64y
]
```

Object Detection

▼ Slide Window Technic

- Using Sliding window technic we can detect an object in image.
- To do this operation we should to follow this steps:
  - train a ConvNet from data contain object which want to detect

    ⇒ For example we want to detect car in any image, then we should to train a convnet in a data set contain image of cars and the out is car or not



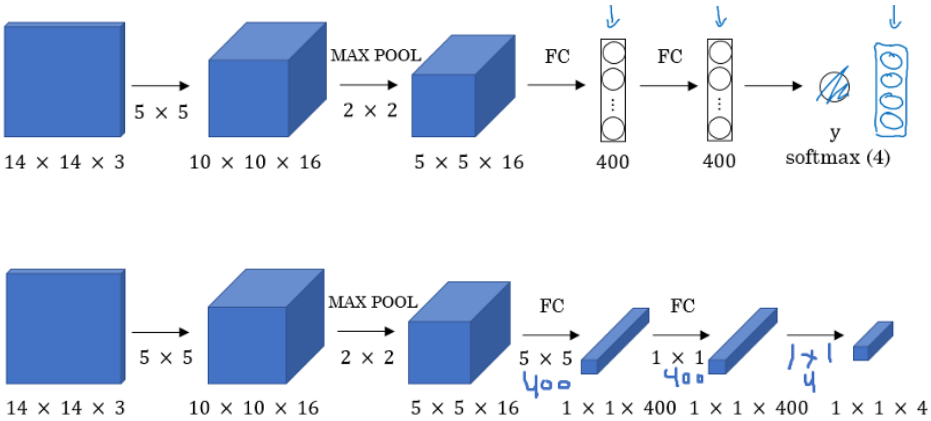
- After to detect an object with a model we train early in the image we use a slide window with this rules:
  - Decide a rectangle size.
  - Split your image into rectangles of the size you picked. Each region should be covered. You can use some strides.
  - For each rectangle feed the image into the Conv net and decide if its a car or not.
  - Pick larger/smaller rectangles and repeat the process from 2 to 3.
  - Store the rectangles that contains the cars
  - If two or more rectangles intersects choose the rectangle with the best accuracy.

⇒ Slide Window tech is very slow and expensive because of a lot of calculations.

⇒ To solve the problem we use Sliding window with a **Convolutional Approach**

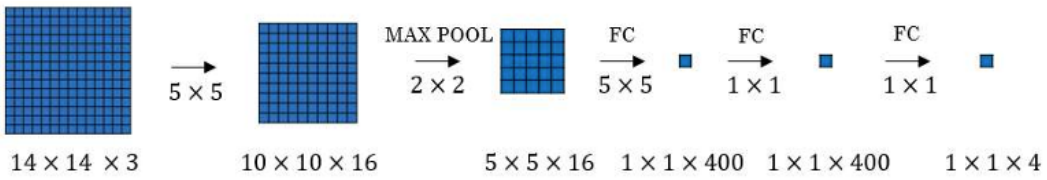
▼ Convolutional Implementation of Sliding Window

- **Transfer Fully connected layer To Convolution Layer**



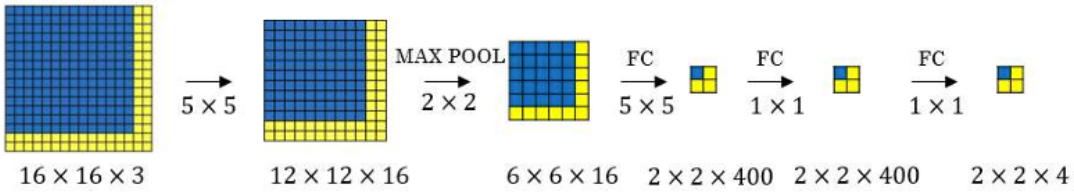
⇒ in the last layer we show softmax layer turned to convLayer 1*1*4

- **<u>Convolution implementation of sliding windows</u>**
  - In the first let say we trained a convNet like this:



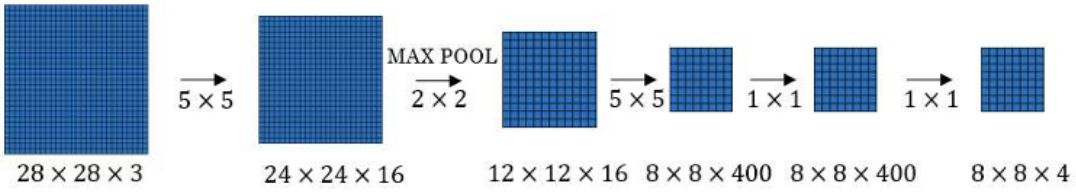
  - For example when we have image input of size 16*16*3, in first implementation of sliding window we slide 14*14*3 **4 times** in this image.

    ⇒ To do this in 1 operation we just put the image 16*16*3 and using the same convolution layer we trained .

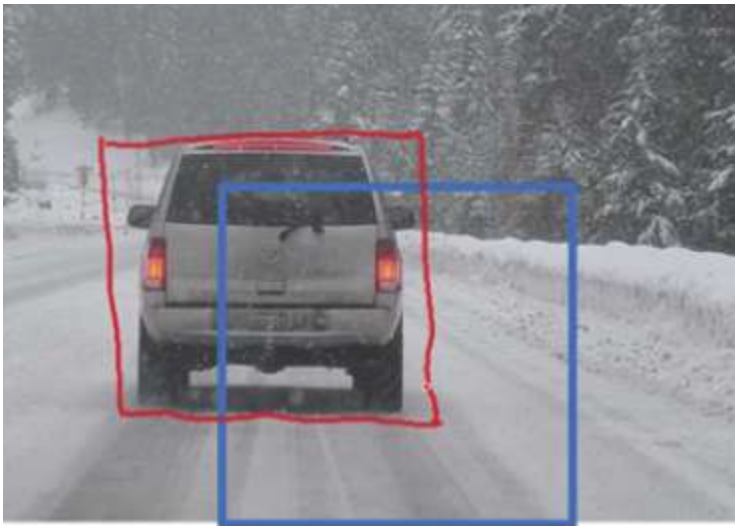


    - The left cell of the result "The blue one" will represent the first sliding window of the normal implementation. The other cells will represent the others.

- It’s more efficient because it now shares the computations of the four times needed.
- Other example:



⇒ The weakness of the algorithm is that the position of the rectangle wont be so accurate . Maybe none of the rectangles is exactly on the object you want to recognize.



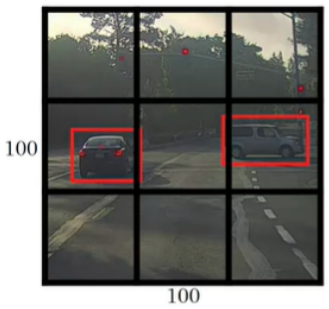
False Bounding Box

▼ YOLO (You Only Look Once)

⇒ YOLO idea:

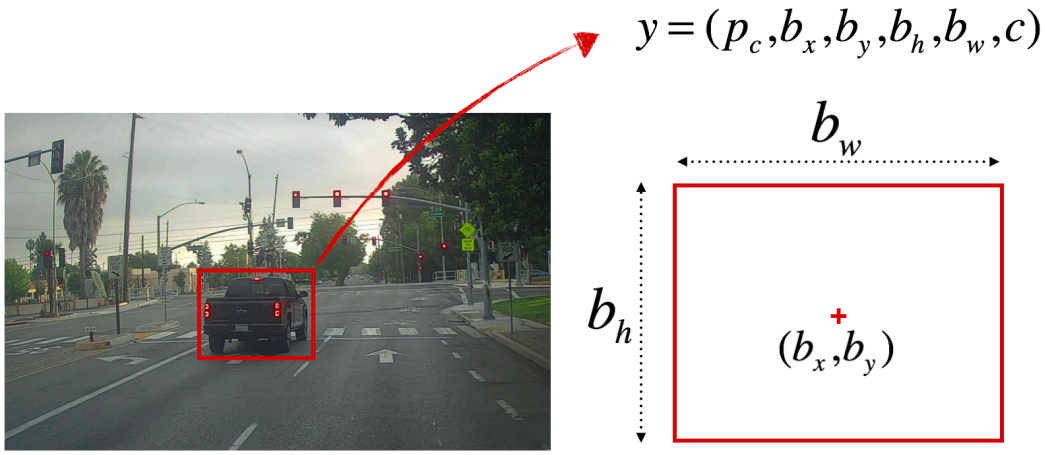
- Split the image to **s*s** grids.

YOLO algorithm



- each grid hold informations about object

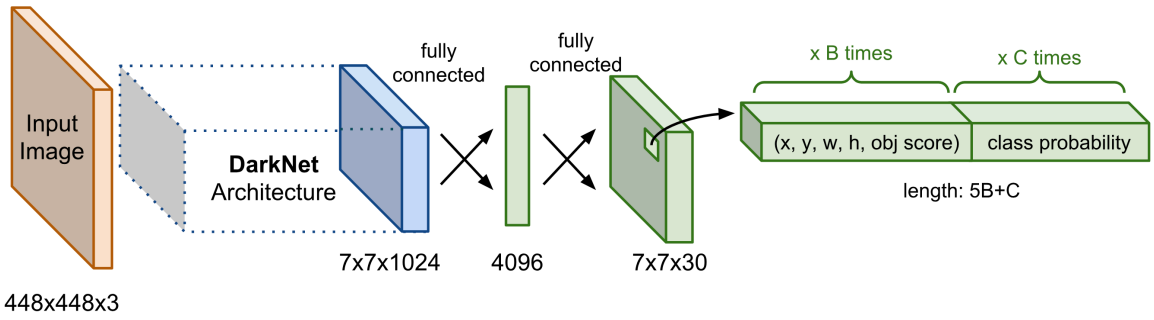
$$Y = [Pc, bx, by, bh, bw, c1, c2, ...]$$



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

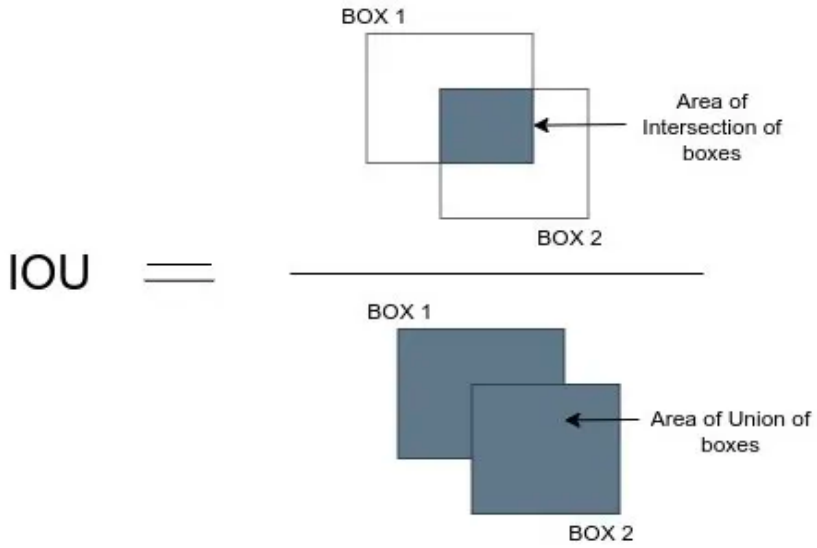
$c = 3$: class of the object being detected (here 3 for “car”)

  - `bx` and `by` will r***epresent the center point of the object in each grid*** and will be relative to the box so the range is between 0 and 1.
  - `bh` and `bw` will **represent the height and width of the object** which can be greater than 1.0 but still a floating point value.
- Do everything at once with the convolution sliding window.
- if the shape of the target Y is 1×8 then the output of the image of shape 100×100 and gridsShape=3×3 is 3×3×8



▼ Intersection Over Union

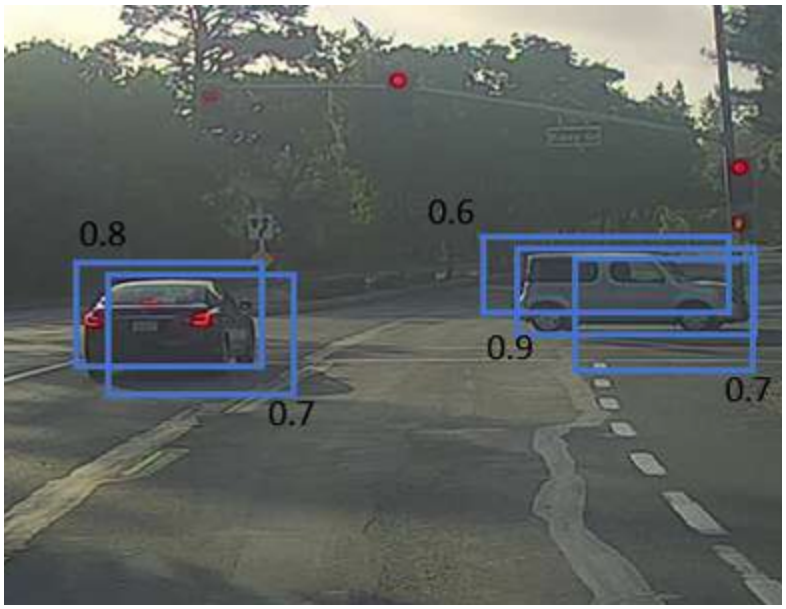
⇒ To evaluate the object detection algorithm we use Intersection Over Union metric.



- We use IOU between the **Target box** and the **Predict box** to evaluate how much the box is right (better).
- If `IOU >=0.5` then its good. The best answer will be 1.

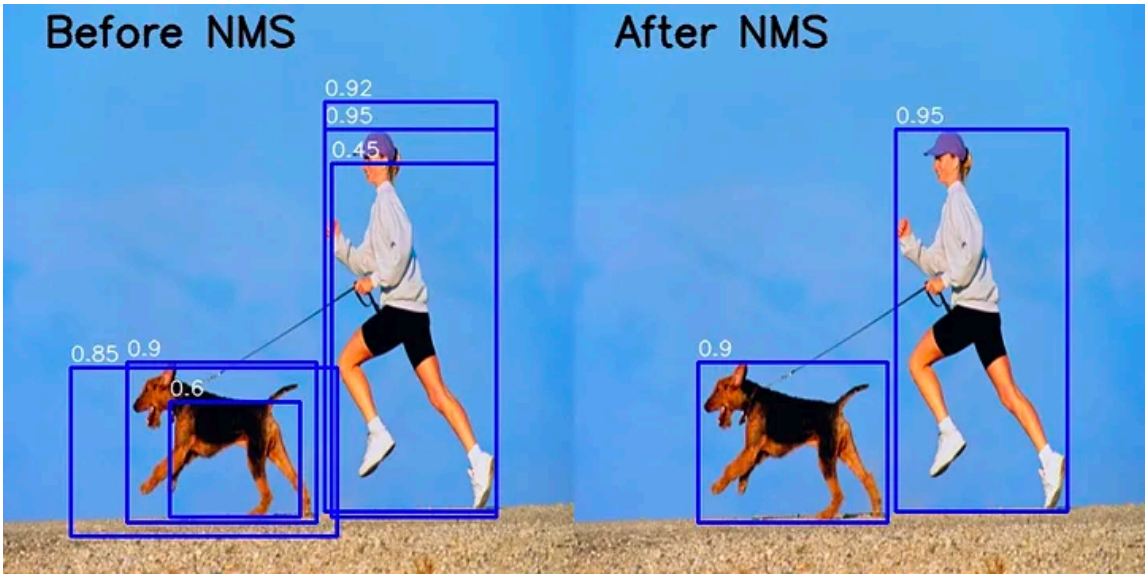
▼ Non max-suppression

→ The Problem of YOLO is can detect object multiple times.



⇒ To solve this problem we use Non Max-Suppression technic.

- This tech could us to eliminate duplicate detections and select the most relevant bounding boxes that correspond to the detected objects.



1-Define a value for Confidence_Threshold, and IOU_Threshold.
2-Sort the bounding boxes in a descending order of confidence.
3-Remove boxes that have a confidence < Confidence_Threshold
4-Loop over all the remaining boxes, starting first with the box that has highes

5-Calculate the IOU of the current box, with every remaining box that belongs
6-If the IOU of the 2 boxes > IOU_Threshold, remove the box with a lower con
7-Repeat this operation until we have gone through all the boxes in the list.

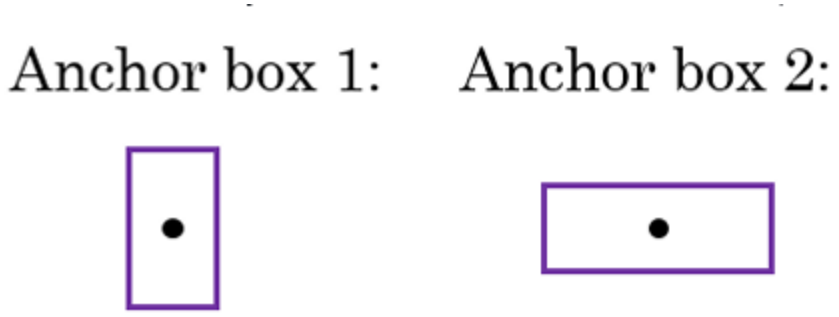
```python
def nms(boxes, conf_threshold=0.7, iou_threshold=0.4):
    bbox_list_thresholded = []
    bbox_list_new = []
    # Stage 1: (Sort boxes, and filter out boxes with low confidence)
    boxes_sorted = sorted(boxes, reverse=True, key = lambda x : x[5])
    for box in boxes_sorted:
        if box[5] > conf_threshold:
            bbox_list_thresholded.append(box)
        else:
            pass
    #Stage 2: (Loop over all boxes, and remove boxes with high IOU)
    while len(bbox_list_thresholded) > 0:
        current_box = bbox_list_thresholded.pop(0)
        bbox_list_new.append(current_box)
        for box in bbox_list_thresholded:
            if current_box[4] == box[4]:
                iou = IOU(current_box[:4], box[:4])
                if iou > iou_threshold:
                    bbox_list_thresholded.remove(box)

    return bbox_list_new
```

▼ Anchor Box

⇒ When a grid cell has more than object we use Anchor Box as solution.

⇒ The Idea of Anchor box is just increase the number of elements in target.



Y=[Pc,bx,by,bh,bw,c1,c2,c3 ,Pc,bx,by,bh,bw,c1,c2,c3 **]**