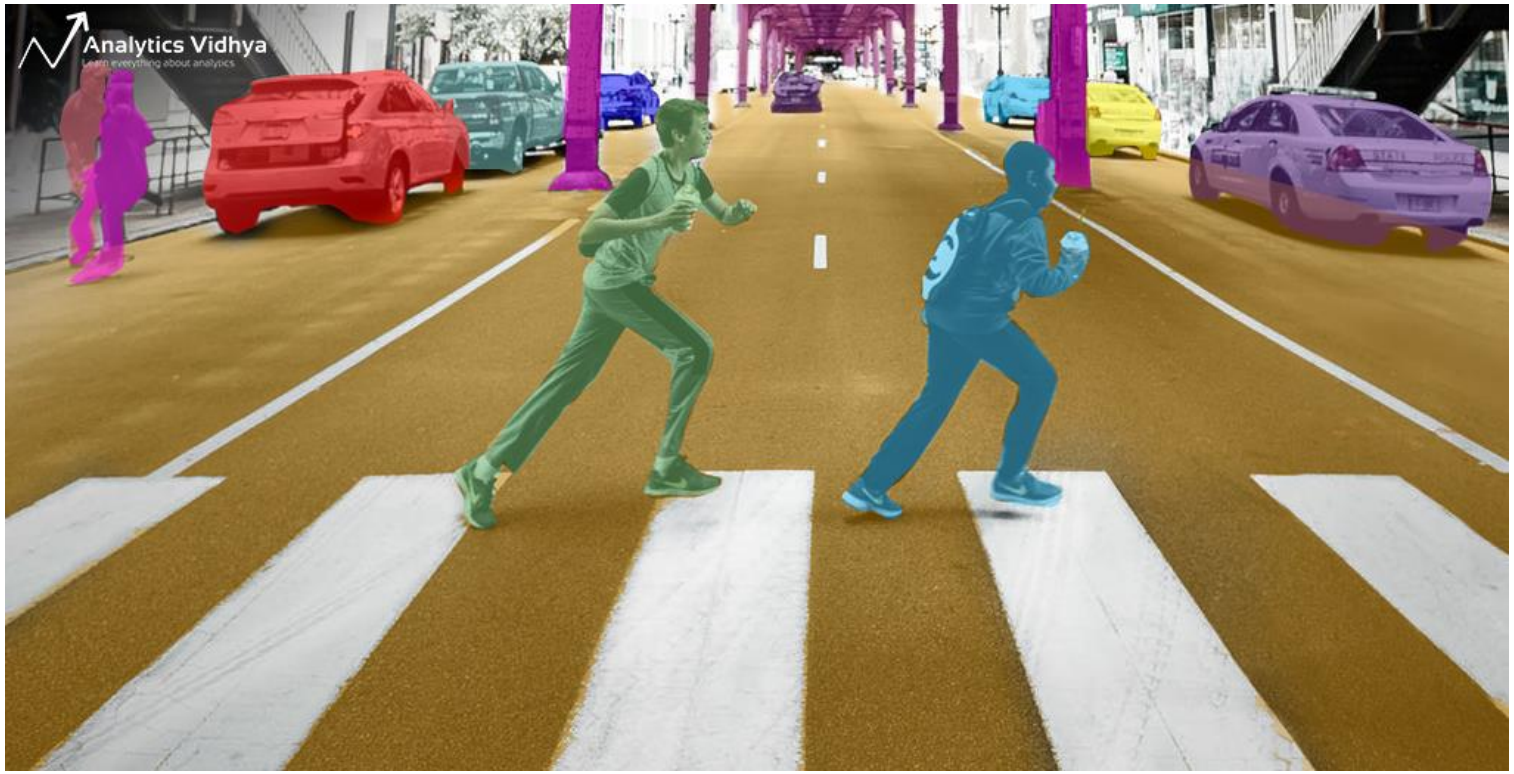
Image Segmentation

Agenda

IMAGE SEGMENTATION



What it comes to our mind when we are attempting cross the road ?

We typically look left and right, take stock of the vehicles on the road, and make our decision. Our brain is able to analyze, in a matter of milliseconds, what kind of vehicle (car, bus, truck, auto, etc.) is coming towards us. Can machines do that?

Yes we are able to build computer vision models that can detect objects, determine their shape, predict the direction the objects will go in, and many other things. You might have guessed it – that's the powerful technology behind self-driving cars!

WHAT IS IMAGE SEGMENTATION

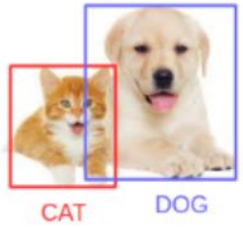
Let's understand image segmentation using a simple example. Consider the below image:

There's only one object here – a dog. We can **build a straightforward cat-dog classifier model and predict that there's a dog in the given image**. But what if we have both a cat and a dog in a single image?

We can train a multi-label classifier, in that instance. Now, there's another caveat **– we won't know the location of either animal/object in the image.**

That's where image **localization comes into the picture** . It helps us to identify the location of a single object in the given image. In case we have multiple objects present, we then rely on the concept of object detection (OD) .

Image Localization



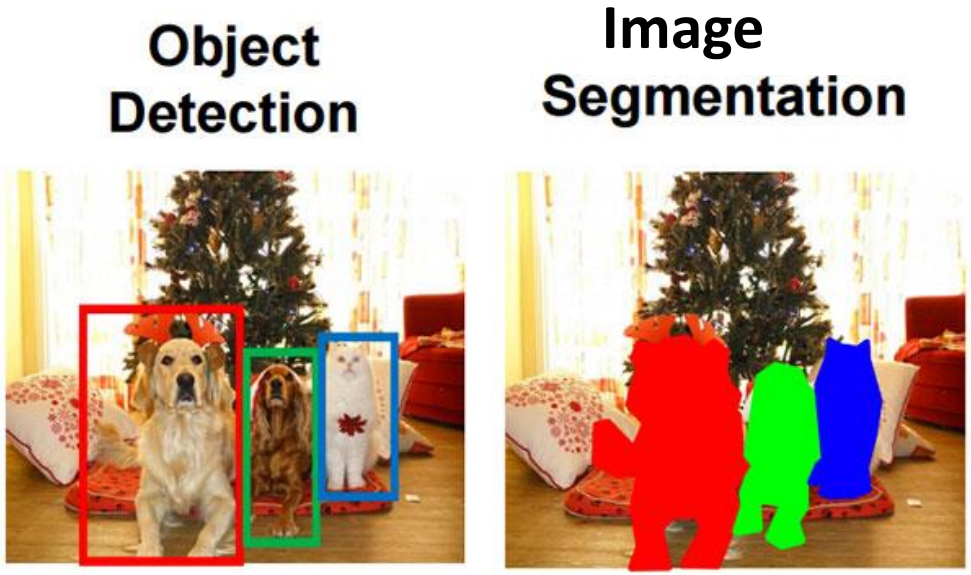
Object Detection

Before detecting the objects and even before classifying the image, we need to understand what the image consists of. Enter – Image Segmentation.

There were various object Detection Algorithms (RCNN,FAST RCNN and Yolo)

OBJECT DETECTION VS IMAGE SEGMENTATION

An image is a collection or set of different pixels. **We group together the pixels that have similar attributes using image segmentation.** Take a moment to go through the below visual (it'll give you a practical idea of image segmentation):



Why do we need to go this deep? Can't all image processing tasks be solved using simple **bounding box coordinates**? **Let's take a real-world example to answer this pertinent question in following slides**

- **Object detection** builds a **bounding box corresponding to each class** in the image. But it tells **us nothing about the shape of the object**. We only get the set of bounding box coordinates. We want to get more information – this is too vague for our purposes.

- **Image segmentation creates a pixel-wise mask for each object in the image**. This technique gives us a far more granular understanding of the object(s) in the image.

DIFFERENT TYPES OF IMAGE SEGMENTATION

We can broadly divide **image segmentation techniques into two types**. Consider the below images:

Semantic Segmentation

Instance Segmentation

Identify the difference between these two? Both the images are using image segmentation to identify and locate the people present.

In image 1, every pixel belongs to a particular class (**either background or person).** Also, all the pixels belonging to a particular class are represented by the same color (background as black and person as pink). **This is an example of semantic segmentation**

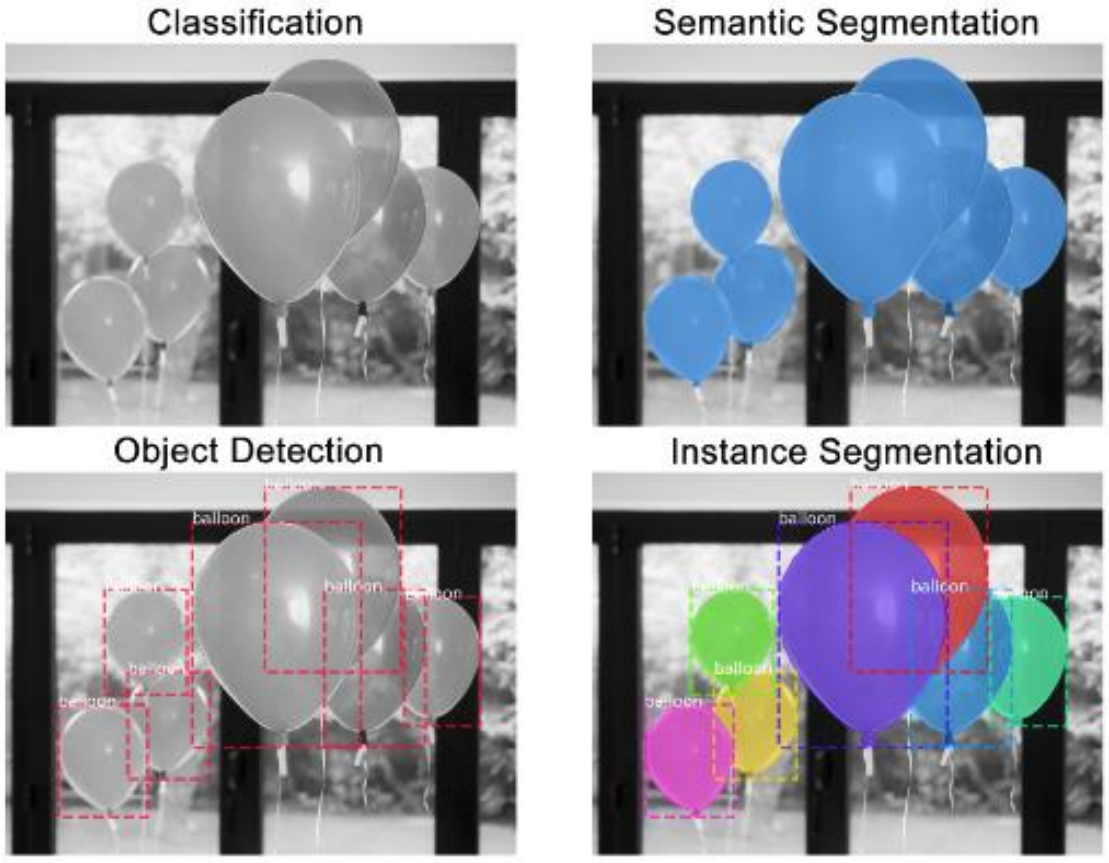
Image 2 has also assigned a particular class to each pixel of the image. However, different objects of the same class have different colors (Person 1 as red, Person 2 as green, background as black, etc.). **This is an example of instance segmentation**

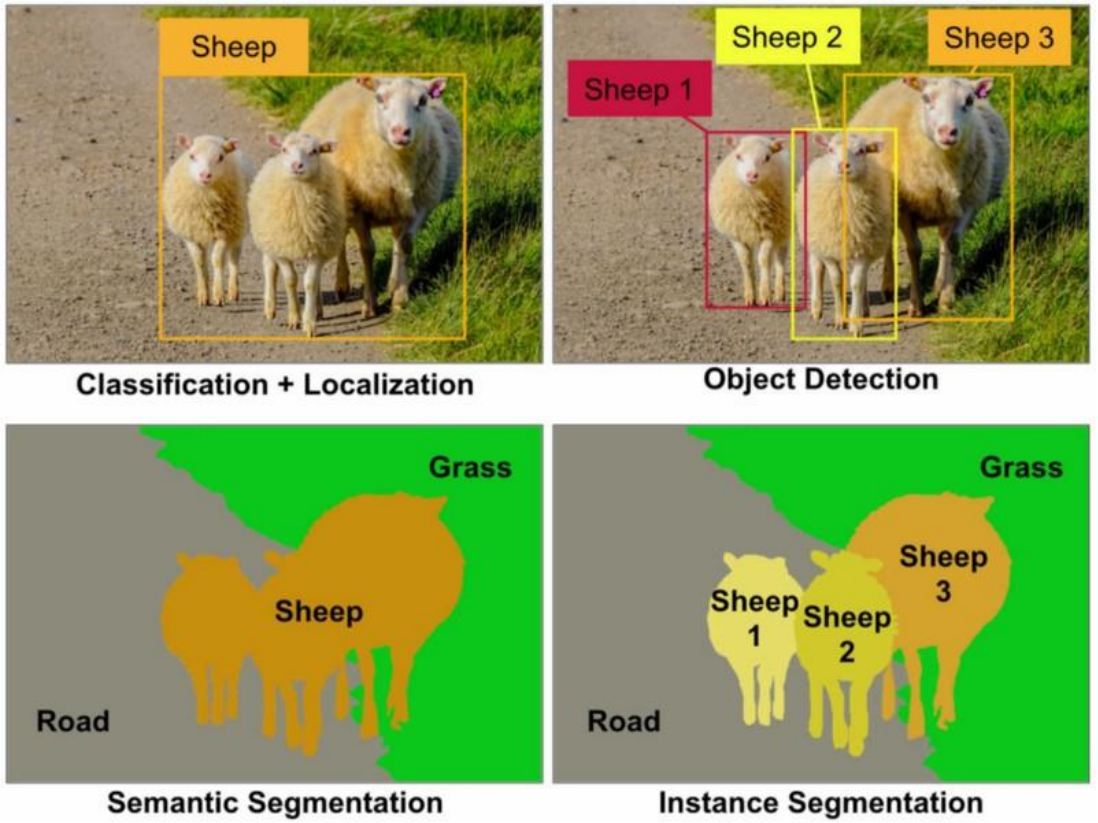
WHAT IS SEMANTIC SEGMENTATION

Semantic segmentation is the task of classifying each and every pixel in an image into a class as shown in the image below. Here we can see that all persons are red, the road is purple, the vehicles are blue, street signs are yellow etc.

SEMANTIC SEGMENTATION VS INSTANCE SEGMENTATION

Semantic segmentation is different from instance segmentation which is that different objects of the same class will have different labels as in person1, person2 and hence different colors. The picture below very crisply illustrates the **difference between instance and semantic segmentation also classification and object detection**





AUTONOMOUS DRIVING

Semantic segmentation is used to identify lanes, vehicles, people and other objects of interest. The resultant is used to make intelligent decisions to guide the vehicle properly.

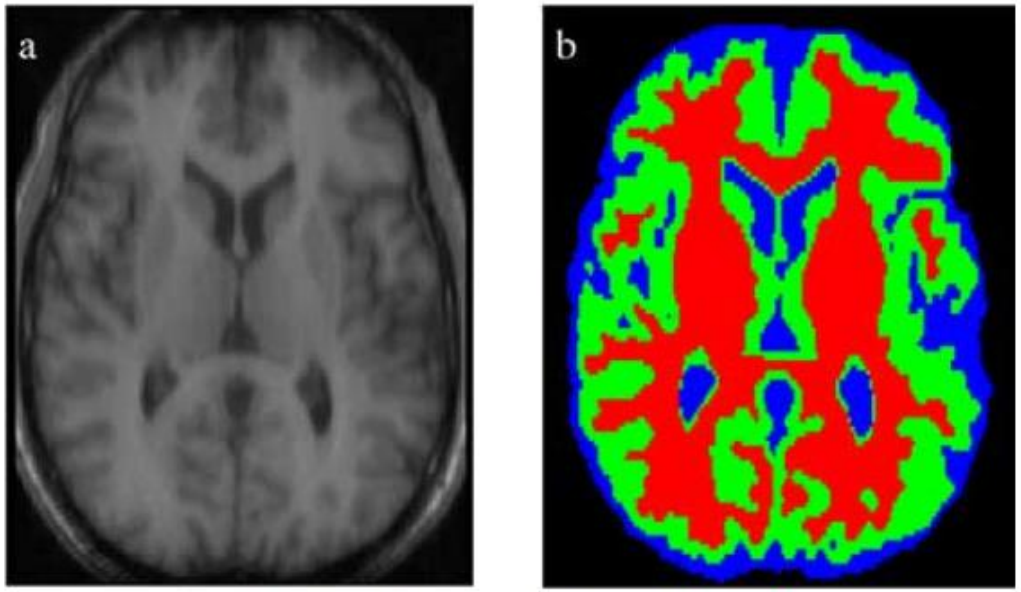
Why do we need this granularity of understanding pixel by pixel location?

May need to know exactly where another car is on the road or the location of a human crossing the road

One constraint on autonomous vehicles is that performance must be real time. A solution to the above problem is to integrate a GPU locally along with the vehicle. To enhance performance of the above solution, lighter (low parameters) neural networks can be used or techniques to fit neural networks on the edge can be implemented.

MEDICAL IMAGE SEGMENTATION

Semantic Segmentation is used to identify salient elements in medical scans. It is especially useful to identify abnormalities such as tumors. The accuracy and low recall of algorithms are of high importance for these applications.



Segmentation of medical scans. (Source)

Why do we need this granularity of understanding pixel by pixel location?

It maybe important in this case to know the exact extent of damage or abnormalities

We can also automate less critical operations such as estimating the volume of organs from 3D semantically segmented scans.

SATELLITE (OR AERIAL) IMAGE PROCESSING

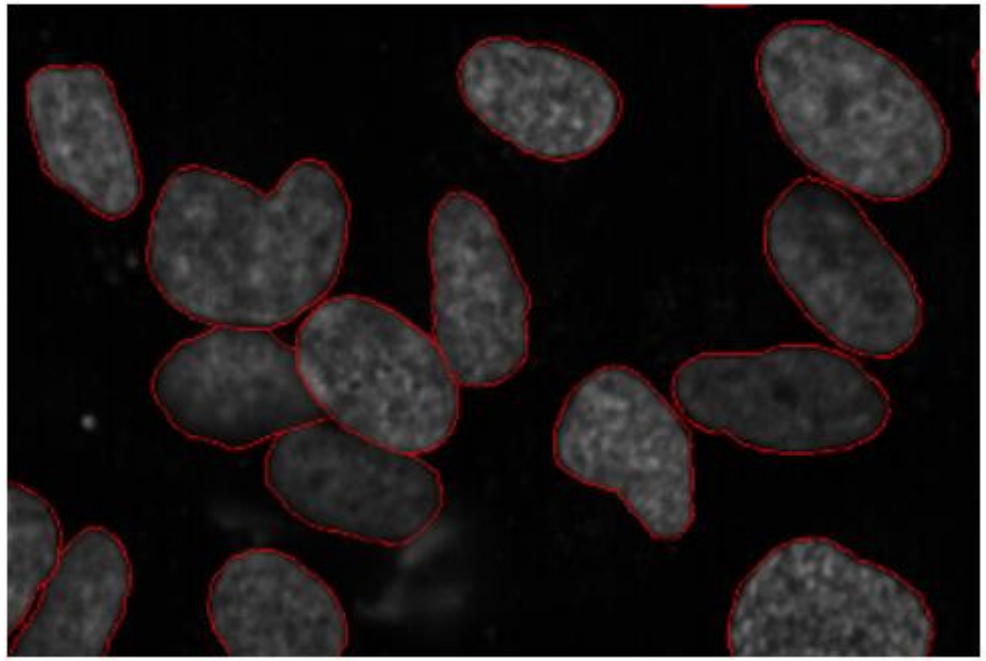
Semantic Segmentation is used to identify types of land from satellite imagery. Typical use cases involve segmenting water bodies to provide accurate map information. Other advanced uses cases involve mapping roads, identifying types of crops, identifying free parking space and so on.

Semantic segmentation of satellite/aerial images. (Source)

Deep Learning greatly enhanced and simplified Semantic Segmentation algorithms and paved the way for greater adoption in real-life applications.

The shape of the cancerous cells plays a vital role in determining the severity of the cancer. You might have put the pieces together – object detection will not be very useful here. We will only generate bounding boxes which will not help us in identifying the shape of the cells.

Image Segmentation techniques make a MASSIVE impact here. They help us approach this problem in a more granular manner and get more meaningful results. A win-win for everyone in the healthcare industry.



Source: Wikipedia

Here, we can clearly see the shapes of all the cancerous cells. There are many other applications where Image segmentation is transforming industries:

SEMANTIC LABELS

Representing the task

Simply, our goal is to take either a RGB color image ($height \times width \times 3$) or a grayscale image ($height \times width \times 1$) and output a segmentation map where each pixel contains a class label represented as an integer ($height \times width \times 1$).

segmented

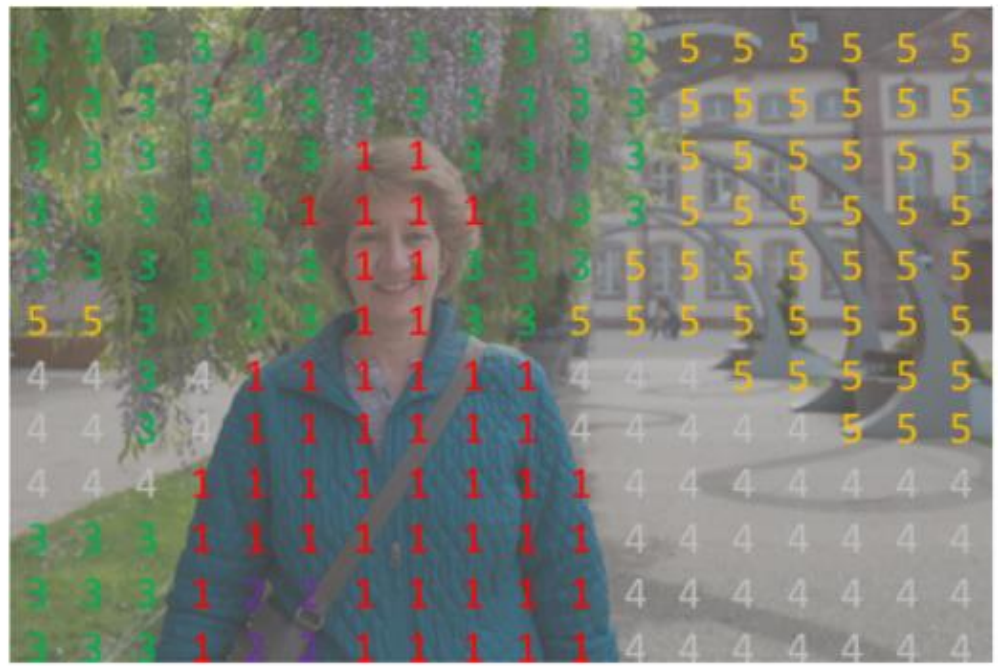
1: Person
2: Purse
3: Plants/Grass
4: Sidewalk
5: Building/Structures

3	3	3	3	3	3	3	3	3	3	3	3	5	5	5	5	5	5
3	3	3	3	3	3	3	3	3	3	3	3	5	5	5	5	5	5
3	3	3	3	3	3	1	1	3	3	3	3	5	5	5	5	5	5
3	3	3	3	3	1	1	1	1	3	3	3	5	5	5	5	5	5
3	3	3	3	3	3	1	1	3	3	3	5	5	5	5	5	5	5
5	5	3	3	3	3	1	1	3	3	5	5	5	5	5	5	5	5
4	4	3	4	1	1	1	1	1	1	4	4	4	5	5	5	5	5
4	4	3	4	1	1	1	1	1	1	4	4	4	4	4	5	5	5
4	4	4	1	1	1	1	1	1	1	1	4	4	4	4	4	4	4
3	3	3	1	1	1	1	1	1	1	1	4	4	4	4	4	4	4
3	3	3	1	2	2	1	1	1	1	1	4	4	4	4	4	4	4
3	3	3	1	2	2	1	1	1	1	1	4	4	4	4	4	4	4

Input

Semantic Labels

We can easily inspect a target by overlaying it onto the observation.



When we overlay a *single channel* of our target (or prediction), we refer to this as a **mask** which illuminates the regions of an image where a specific class is present.

DEEP LEARNING MODEL ARCHITECTURES FOR SEMANTIC SEGMENTATION

1. **Fully Convolutional Network (FCN)**

2. **SEG-Net**

3. **U-Net**

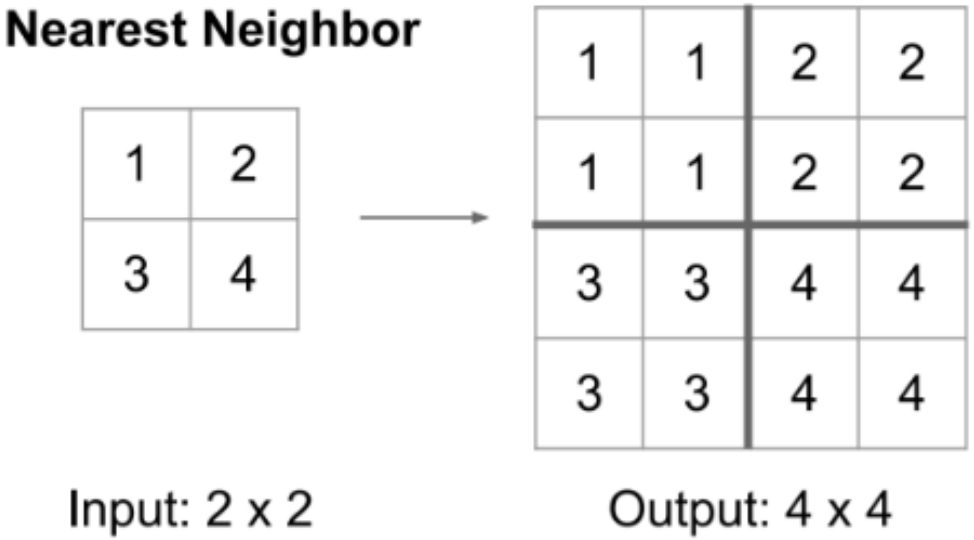
4. **Mask RCNN**

UnPooling

How do we increase layer size to reach the dimensions of the original input? One way we can up sample is by unpooling.

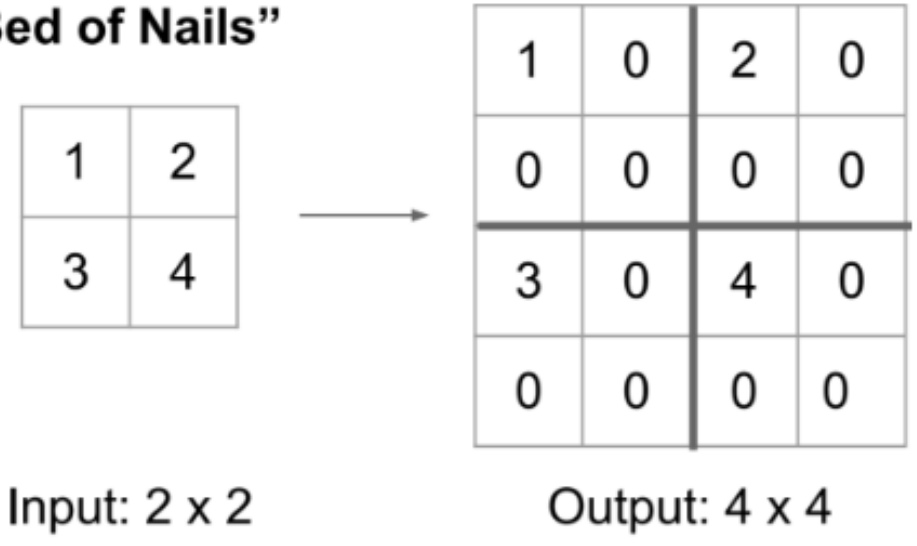
Replicate

Nearest Neighbor



Use once

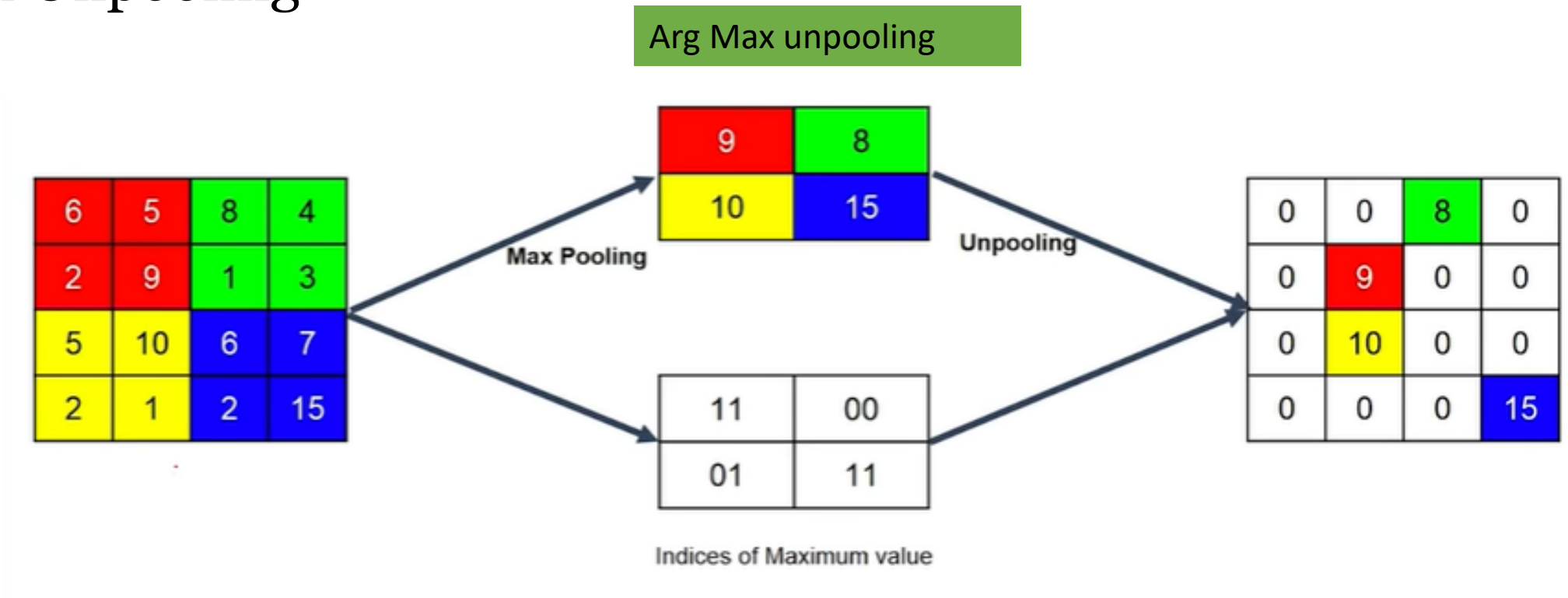
“Bed of Nails”



There are multiple approaches to unpooling. One approach is “Nearest Neighbor", we simply repeat every element. “Bed of Nails" unpooling simply places the value in a particular position in the output, filling the rest with zeros. The above example places the input values in the upper left corner.

UnPooling

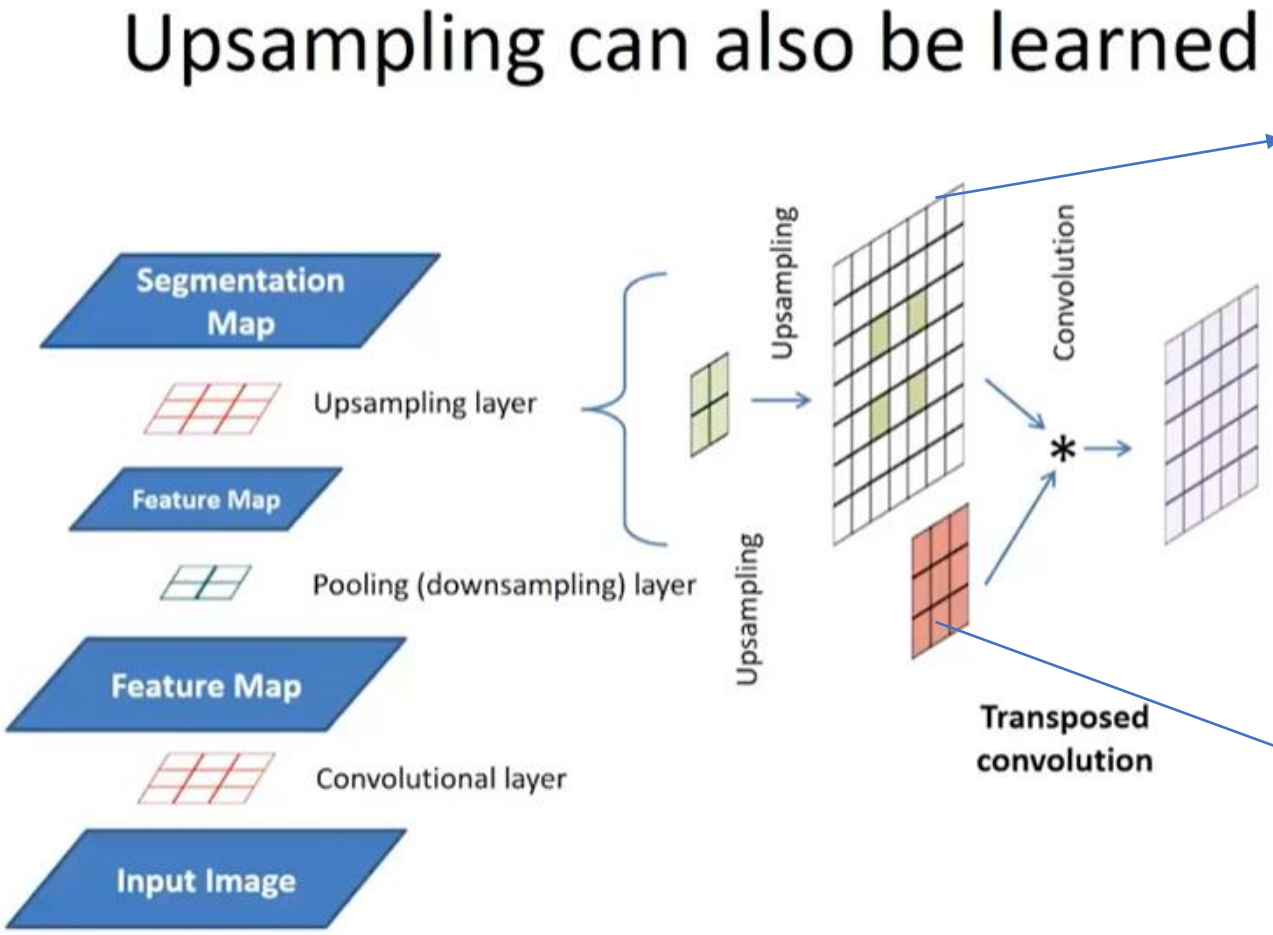
Max Unpooling



Max Unpooling is a smarter “bed of nails" method. Rather than a predetermined, fixed location for the “nails", we use the position of the maximum elements from the corresponding max pooling layer earlier in the network.

Learn a Filter for Up sampling

Transposed Convolution, Fractionally Strided Convolution or Deconvolution



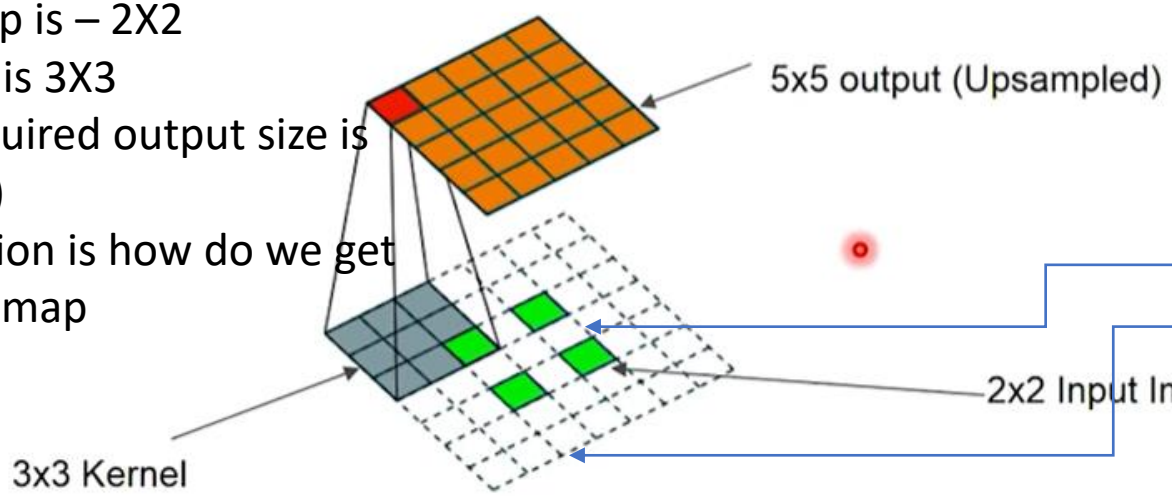
Insert required padding columns and fill with zeros

Takes some random weights to get the feature map
These weights gets learned
We learn these weights with backpropagation

TRANSPOSED CONVOLUTION, FRACTIONALLY STRIDED CONVOLUTION OR DECONVOLUTION

- Convolution operation that aid in increasing the size of the output feature map
- Used in Encoder-Decoder Networks to increase the spatial dimension of the feature map
- In transposed convolution reconstructs the original spatial resolution and performs a convolution.
- The input image is appropriately padded before convolution operation.

- Lets assume that input size of feature map is – 2X2
- Kernel size is 3X3
- Output required output size is MXM (5X5)
- Now question is how do we get 5X5 future map



- Lets assume that input size of image is– 5X5
- Kernel size is 3X3
- Output size =5-3+1=3X3, we do see 2X2 output
- To get the 2X2 output we might need to apply stride=2 and zero padding, i.e ((5-3)/2)+1=2X2
- If we want to reverse this operation we need to insert s-1 padding rows and columns in 2X2 matrix and apply S padding

- Do the convolution with stride=1

Input to the transposed convolution size is 2X2

SEMANTIC SEGMENTATION - FULLY CONVOLUTIONAL NETWORK (FCN)

- One of the simplest and popular architecture used for semantic segmentation is the **Fully Convolutional Network (FCN).**

- First down sample the input image to a smaller size (while gaining more channels) through a series of convolutions. This set of convolutions is typically called the **Encoder**.

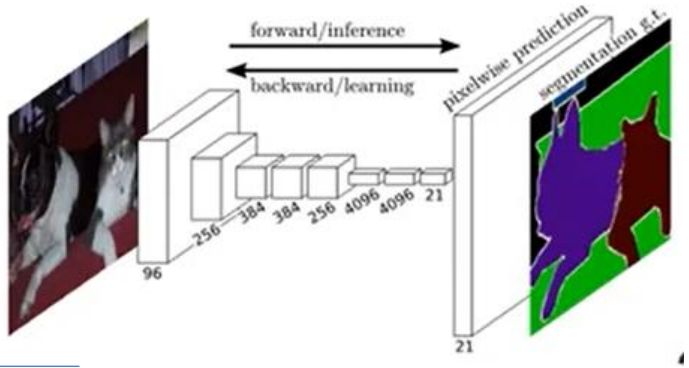
- The encoded output is then up sampled either through series of transpose-convolutions. This set of transposed-convolutions is typically called the **decoder**.

- These models typically **don't have any fully connected layers**. The goal of down sampling steps is to capture **semantic/contextual information** while the goal of up sampling is to **recover spatial information.**

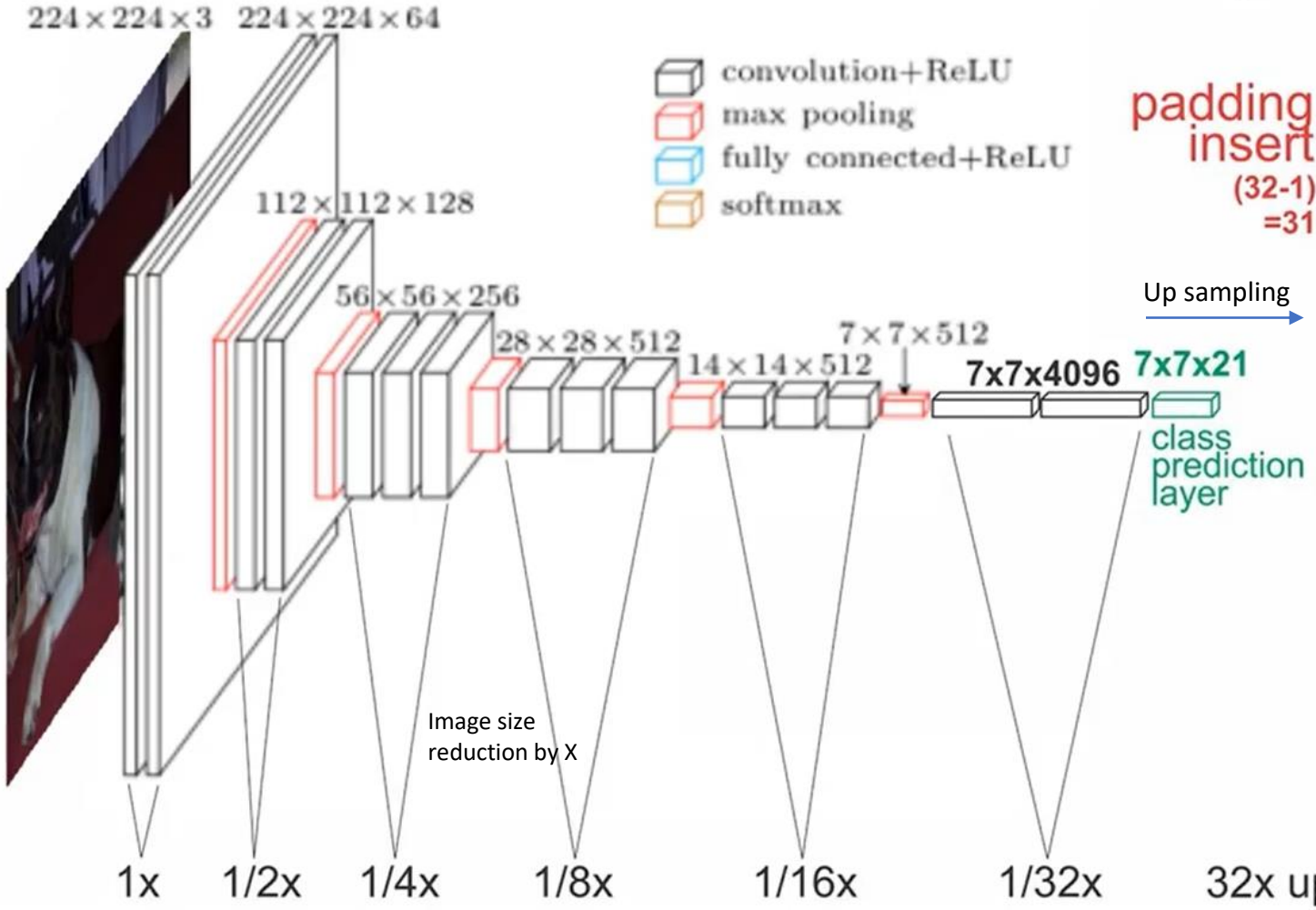
- To fully recover the fine grained spatial information lost in down sampling, **skip connections are used**.
  - **A skip connection is a connection** that bypasses at least one layer. Here it is used to pass information from the down sampling step to the up sampling step. Merging features from various resolution levels helps combining context information with spatial information.

SEMANTIC SEGMENTATION - FULLY CONVOLUTIONAL NETWORK (FCN)

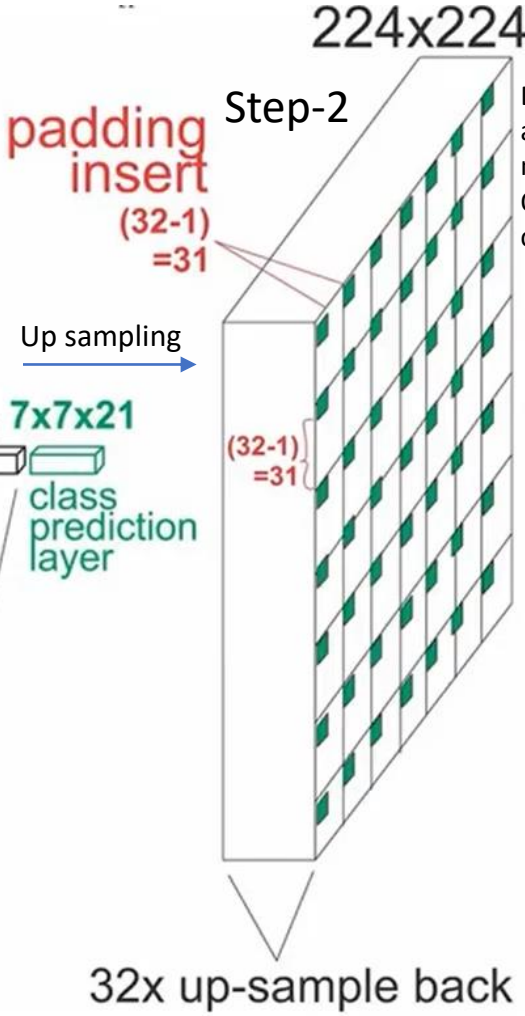
Lets assume that we have 21 classes



Step-1

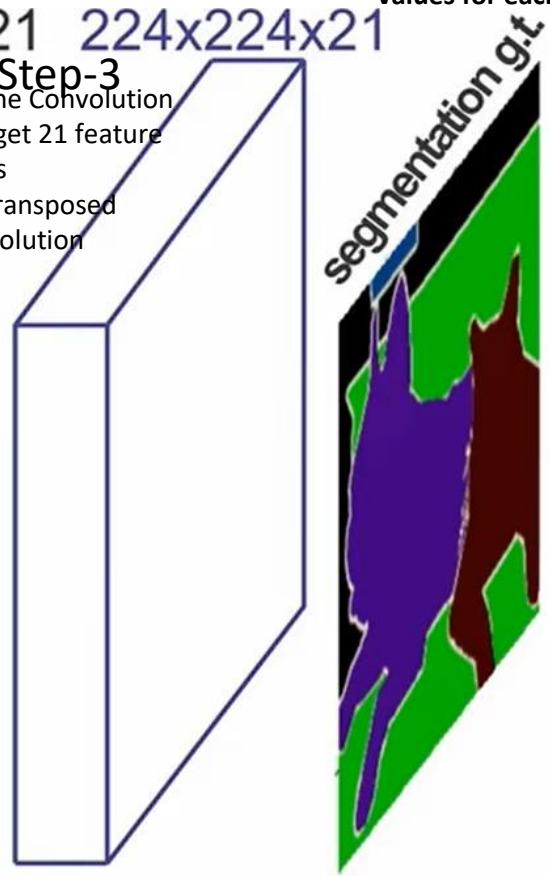


Step-2



Step-3

Do the Convolution and get 21 feature maps
Or Transposed convolution

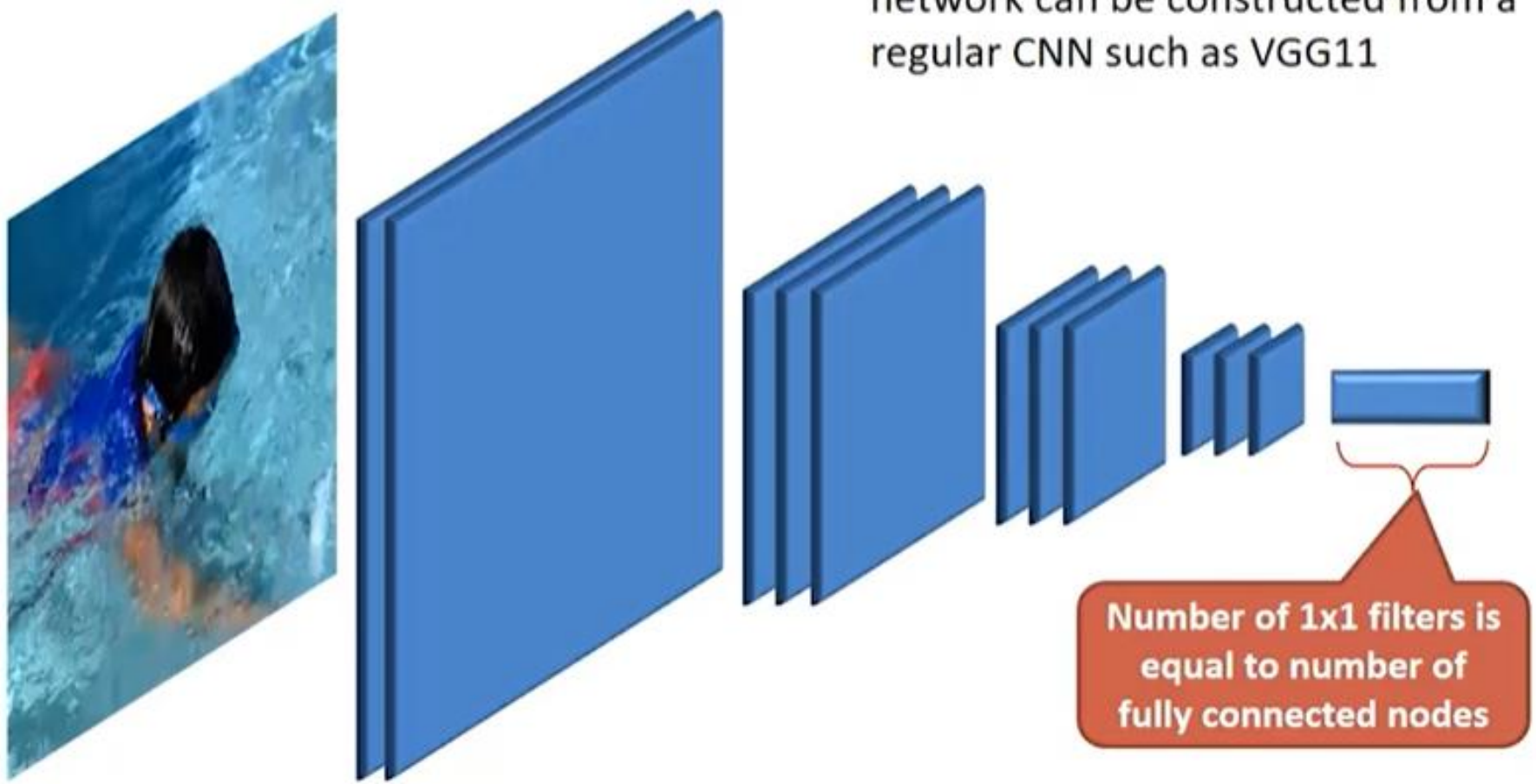


Step-4

Apply Soft max function for each cell to get probability values for each pixel

Using 1x1 convolutions is equivalent to having a fully connected layer

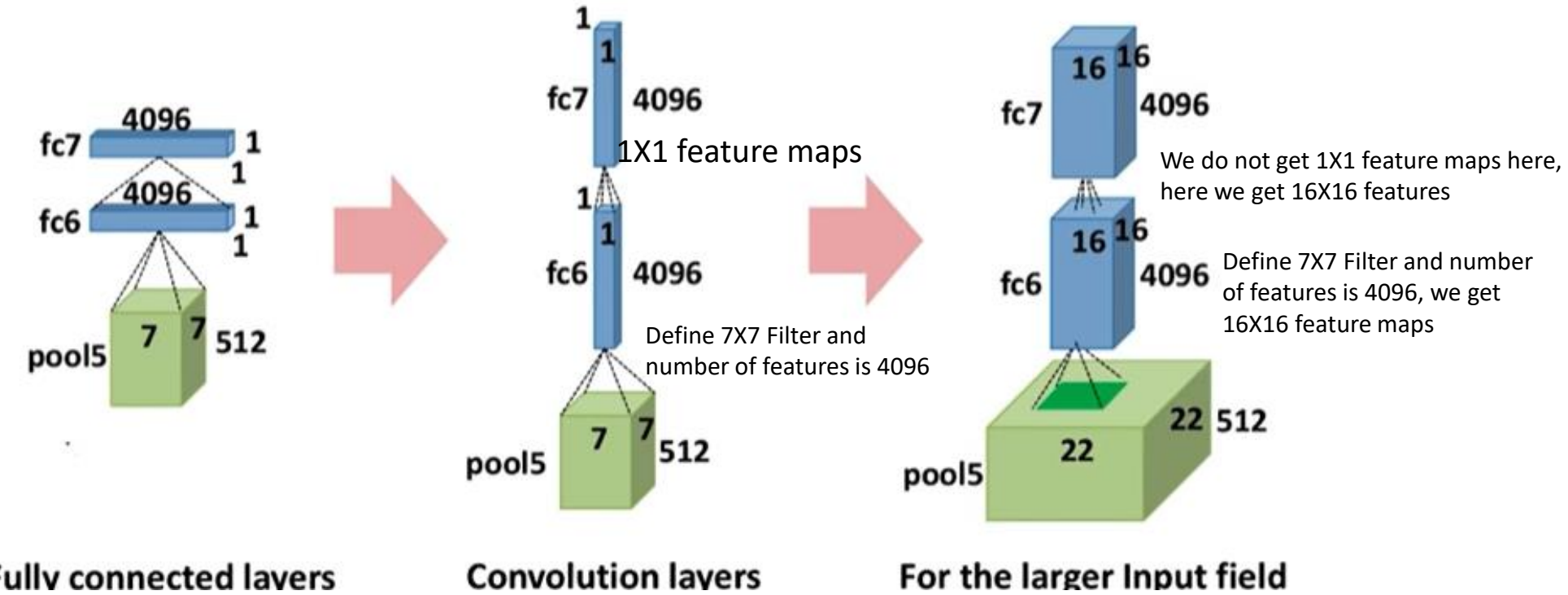
- This way, a fully convolutional network can be constructed from a regular CNN such as VGG11



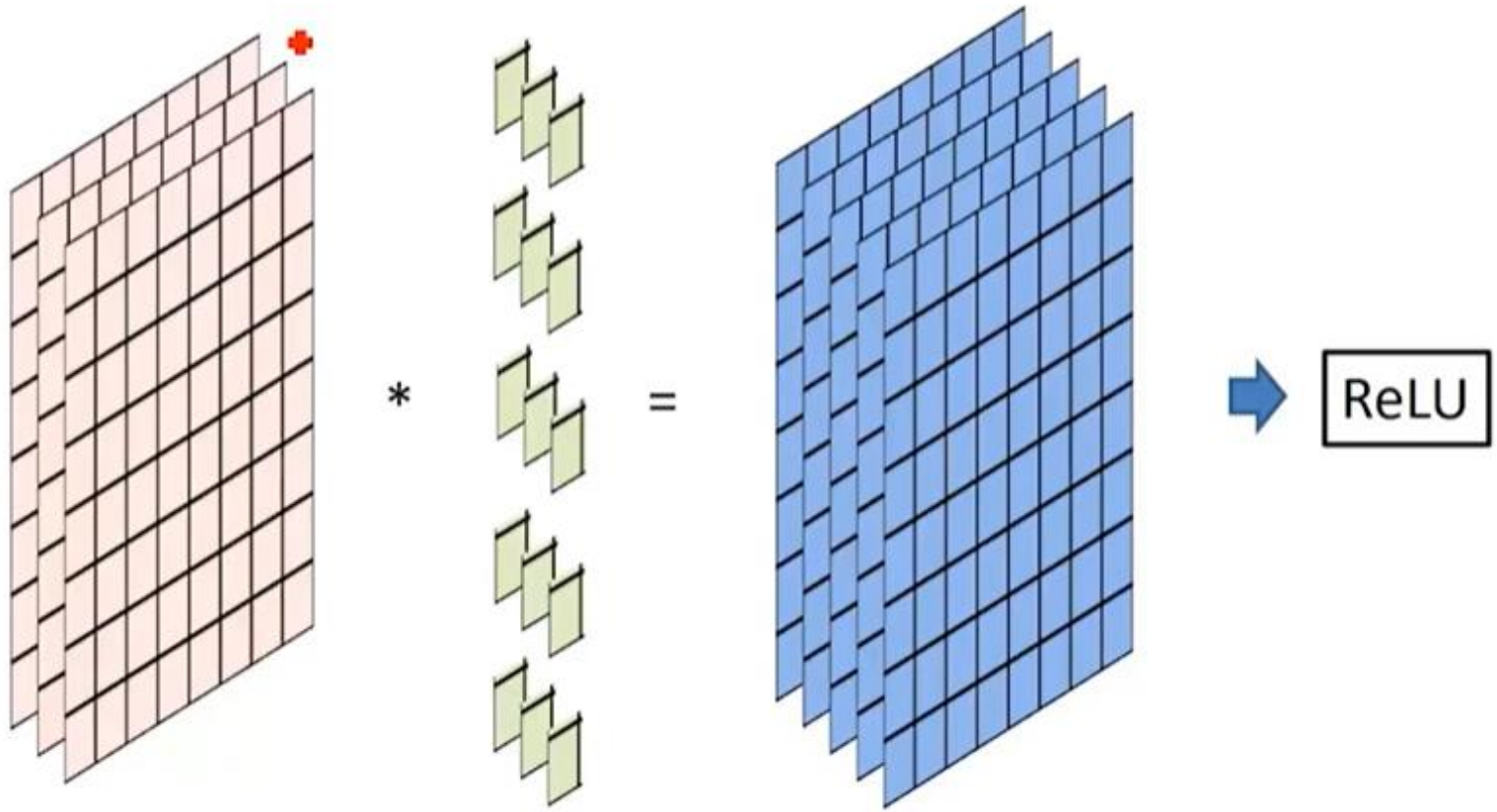
Emulate Fully connected layer

Fully Connected to Fully Convolutional Network

- Fully connected layers = Convolutions with kernels that cover their entire input regions
- Doing so casts them into fully convolutional networks that take input of arbitrary size and output classification maps



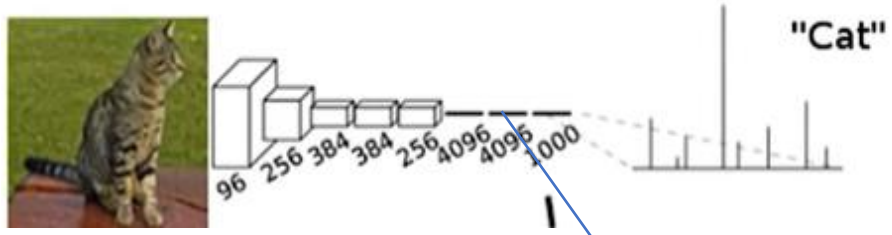
1x1 convolutions can also be used to change the number of feature maps



1X1 convolutions gives required number of channels

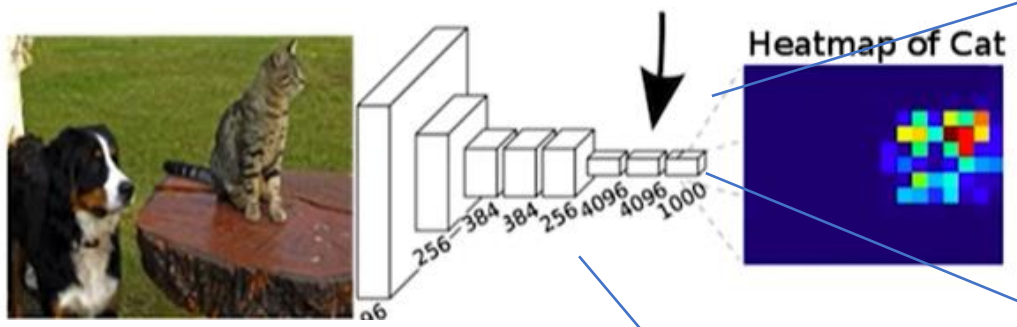
INTERPRETING FULLY CONNECTED LAYERS AS CONVOLUTIIONALIZATION

Image recognition network:



Fully connected layer

In this case assume that our image net have 1000 classes, we directly get 1000 probability scores from fully connected layer, which ever the probability score is high we assign that label



- It gives the 1000 feature maps(activation maps or score maps, each map contains the probability assigned to 1000 class)
- One particular feature map, which gives probability score for being any of the 1000 categories.
- Take one feature map and generate heat map of probability of every pixel being cat in the feature map

Use these 1000 score maps as input for up sampling to get the final probability of each pixel at size of the actual image

From 256 feature map we can get 4096 feature map and from there we again do the convolution to get 1000 output feature maps as we have 1000 classes, if we apply the SoftMax function each feature map represents the probability of category

SEMANTIC SEGMENTATION - FULLY CONVOLUTIONAL NETWORK (FCN)

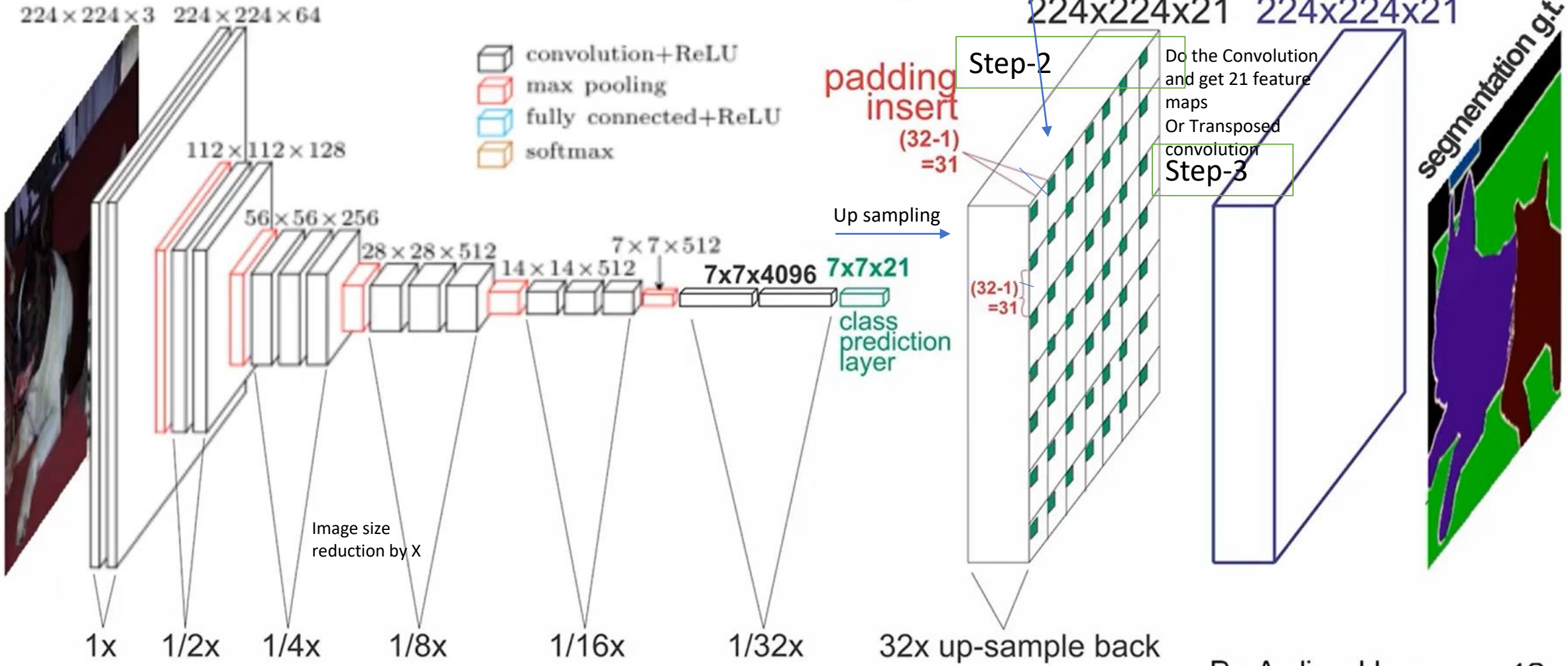
Lets assume that we have 21 classes

32x times up sample which indicates back by inserting 31 padding columns and rows, to fill these cells we can use bilinear approximation or nearest neighbors ..

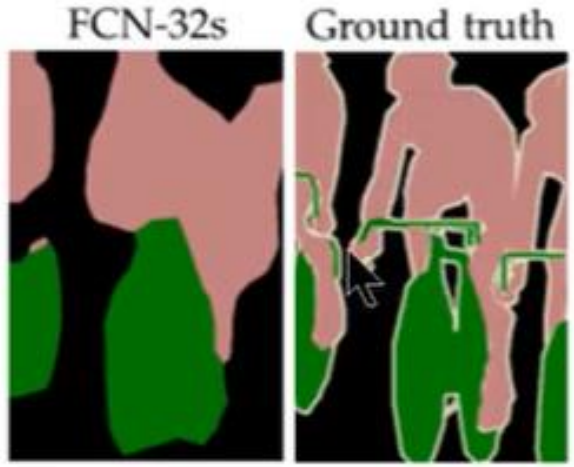
Step-4

Apply Soft max function for each cell to get probability values for each pixel

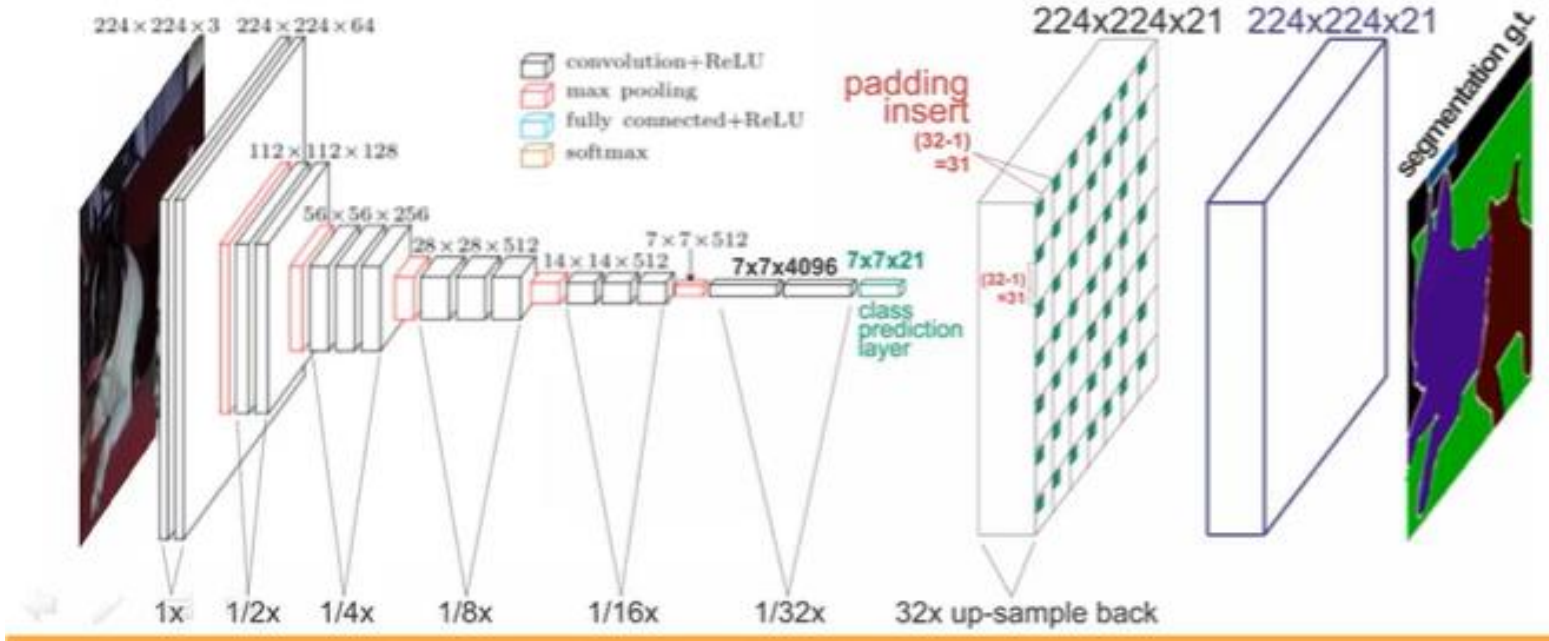
Step-1



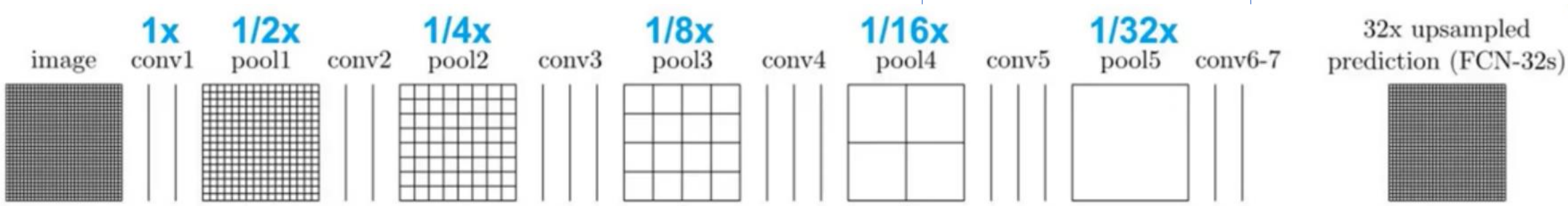
FCN- OTHER VARIATIONS



FCN 32 up sample back alone is not that great to do the task



These tow have different dimensions hence need a up sampling to concatenate them



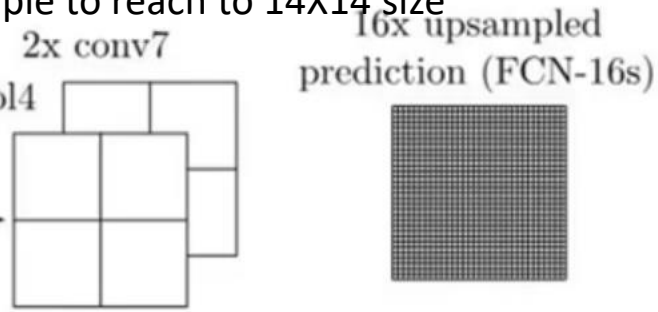
FCN

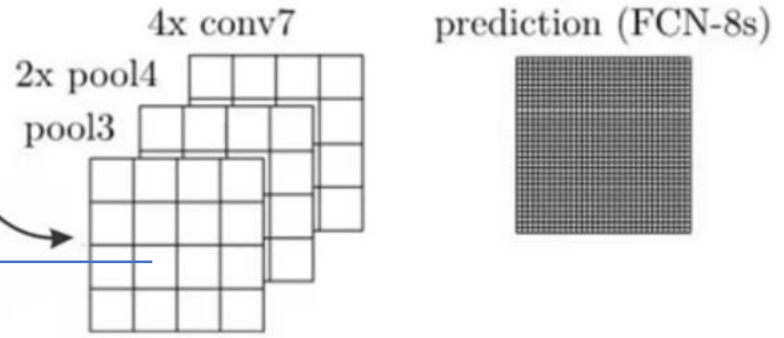
a. FCN-32s

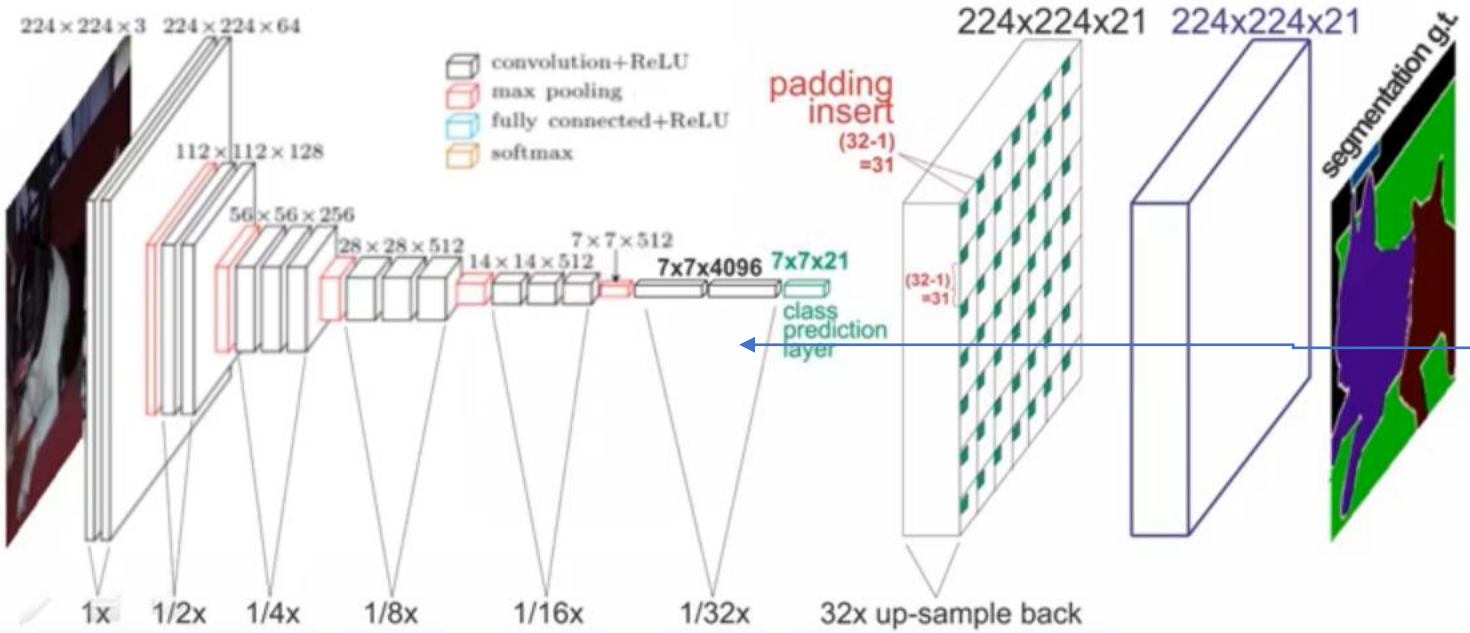
b. FCN-16s

c. FCN-8s

2 times up sample to reach to 14X14 size

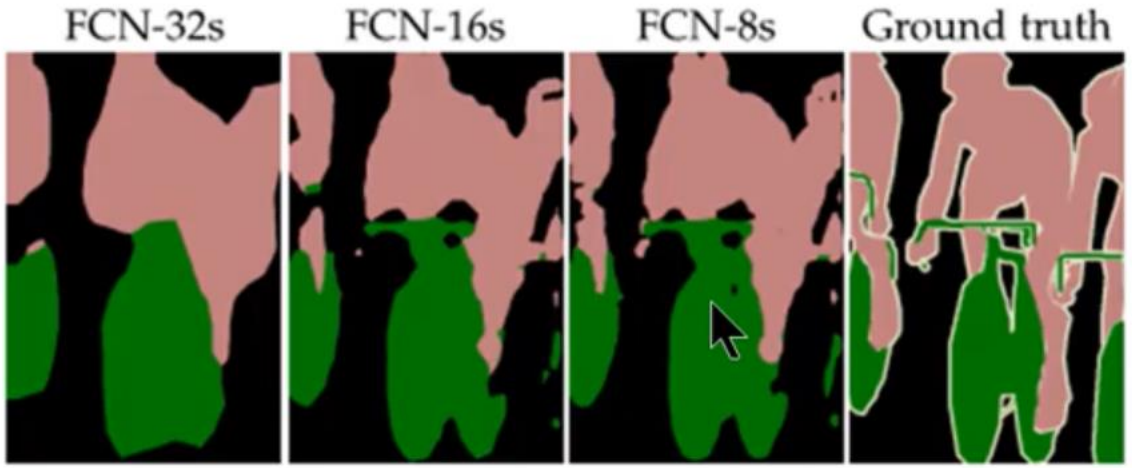






Use 1X1 convolution to concatenate or simply add them, then add these to prediction layer to get the class probabilities

FCN- OTHER VARIATIONS

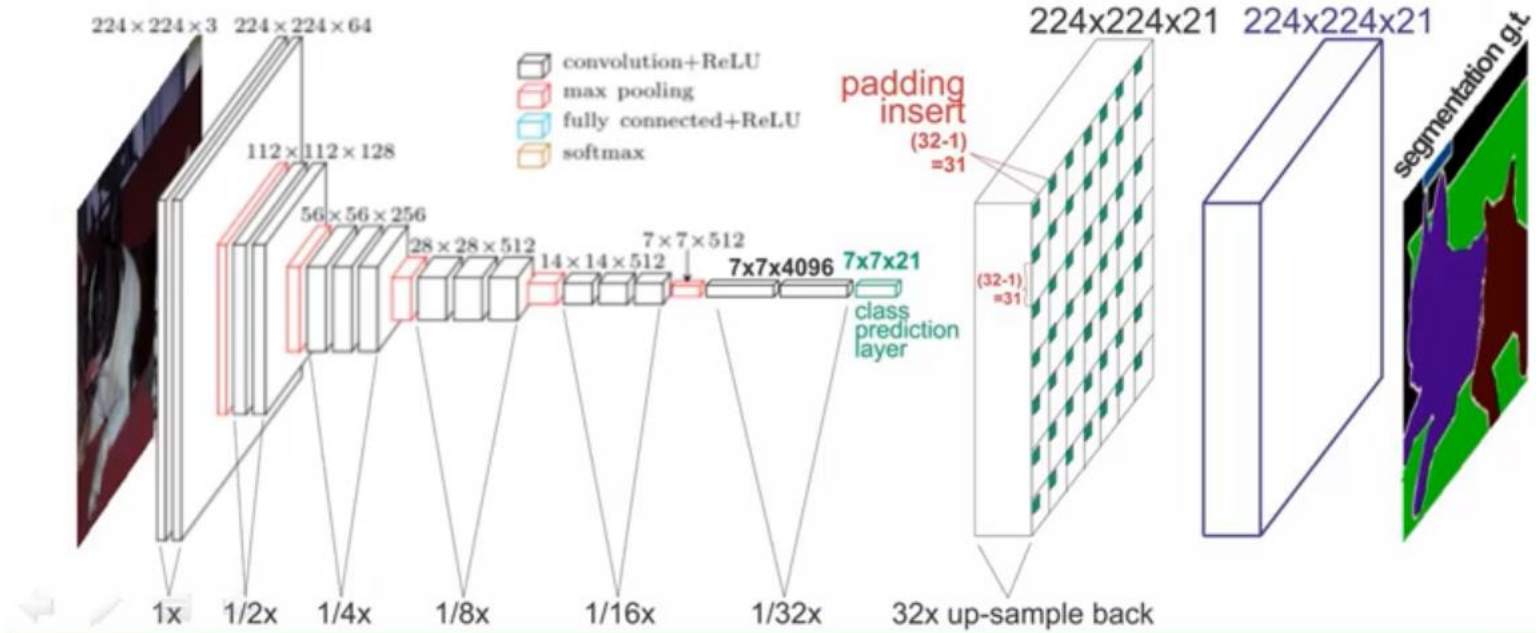


FCN

a. FCN-32s

b. FCN-16s

c. FCN-8s



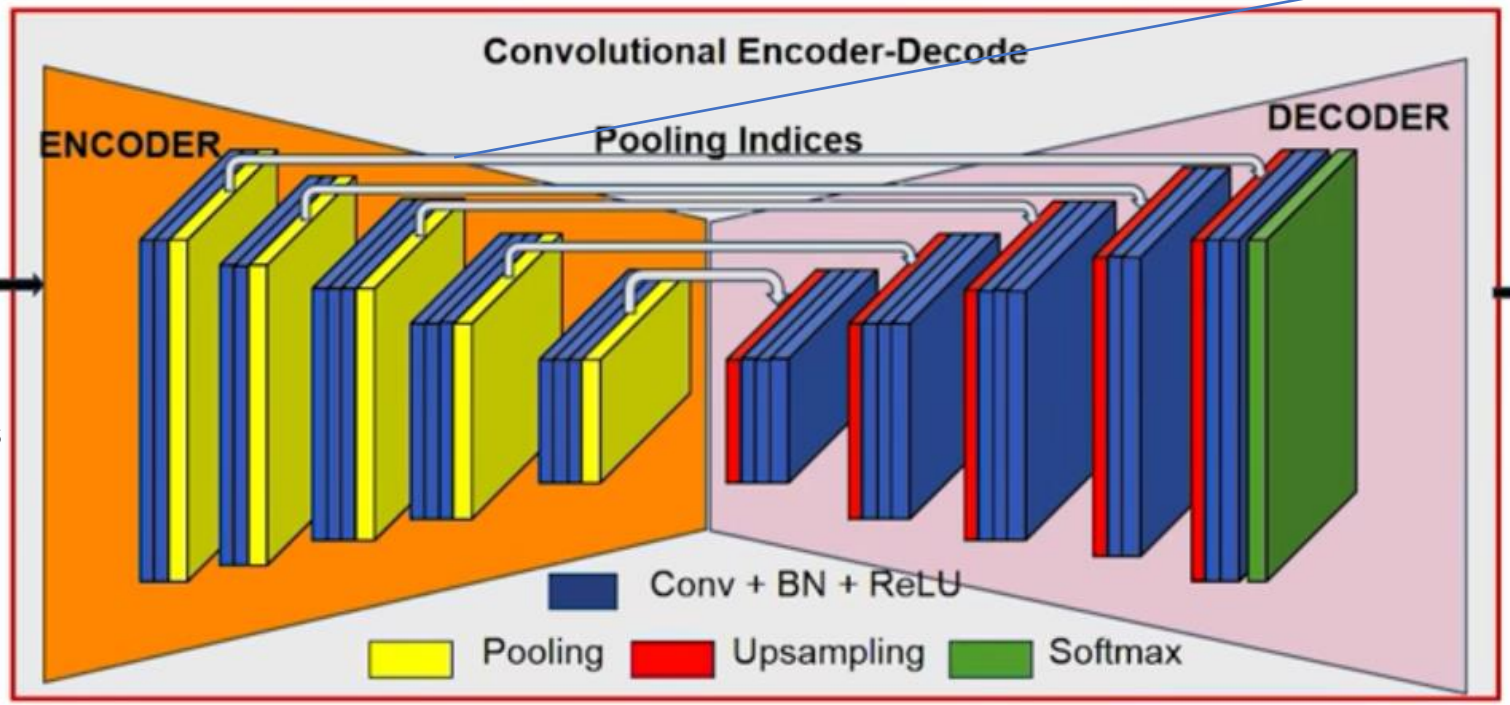
SKIP CONNECTIONS

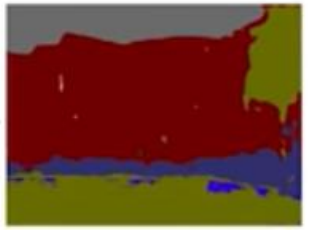
- “Fully Convolutional Networks for Semantic Segmentation" by Long et al. introduced the idea of skip connections into FCNs to improve segmentation accuracy. (It also popularized FCNs as a method for semantic segmentation).
- Upsampling using transposed convolutions or unpooling loses information, and thus produces rough segmentation. Skip connections allow us to produce finer segmentation by using layers with finer information.
- Skip connections combine the coarse final layer with finer, earlier layers to provide local predictions that “respect" global positions. Refer to the figure below for a diagram of the skip connection architecture.

SEMANTIC SEGMENTATION – SEGNET (A DEEP CONVOLUTIONAL ENCODER-DECODER ARCHITECTURE FOR IMAGE SEGMENTATION)

SegNet has an **encoder** network and a corresponding **decoder** network, followed by a final pixelwise classification layer.

Takes the inputs from earlier layers and pass it to decoder layer called skip connections





•At the encoder, convolutions and max pooling are performed.
•There are 13 convolutional layers from VGG-16. (The original fully connected layers are discarded.)
•While doing 2×2 max pooling, the corresponding max pooling indices (locations) are stored.

•At the decoder, upsampling and convolutions are performed. At the end, there is softmax classifier for each pixel.
•During upsampling, the max pooling indices at the corresponding encoder layer are recalled to upsample as shown above.
•Finally, a K-class softmax classifier is used to predict the class for each pixel.

1. Eliminates the need for learning to up-sample
2. Keeps high frequency details intact by using unpooling method

We see a symmetry between Encoder and decoder convolution and pooling layers

Badrinarayanan et al., "Segnet: A deep convolutional encoder-decoder architecture for image segmentation.", CVPR, 2015

SEMANTIC SEGMENTATION – SEGNET

2. Differences from DeconvNet and U-Net

DeconvNet and U-Net have similar structures as SegNet.

2.1. Differences from DeconvNet

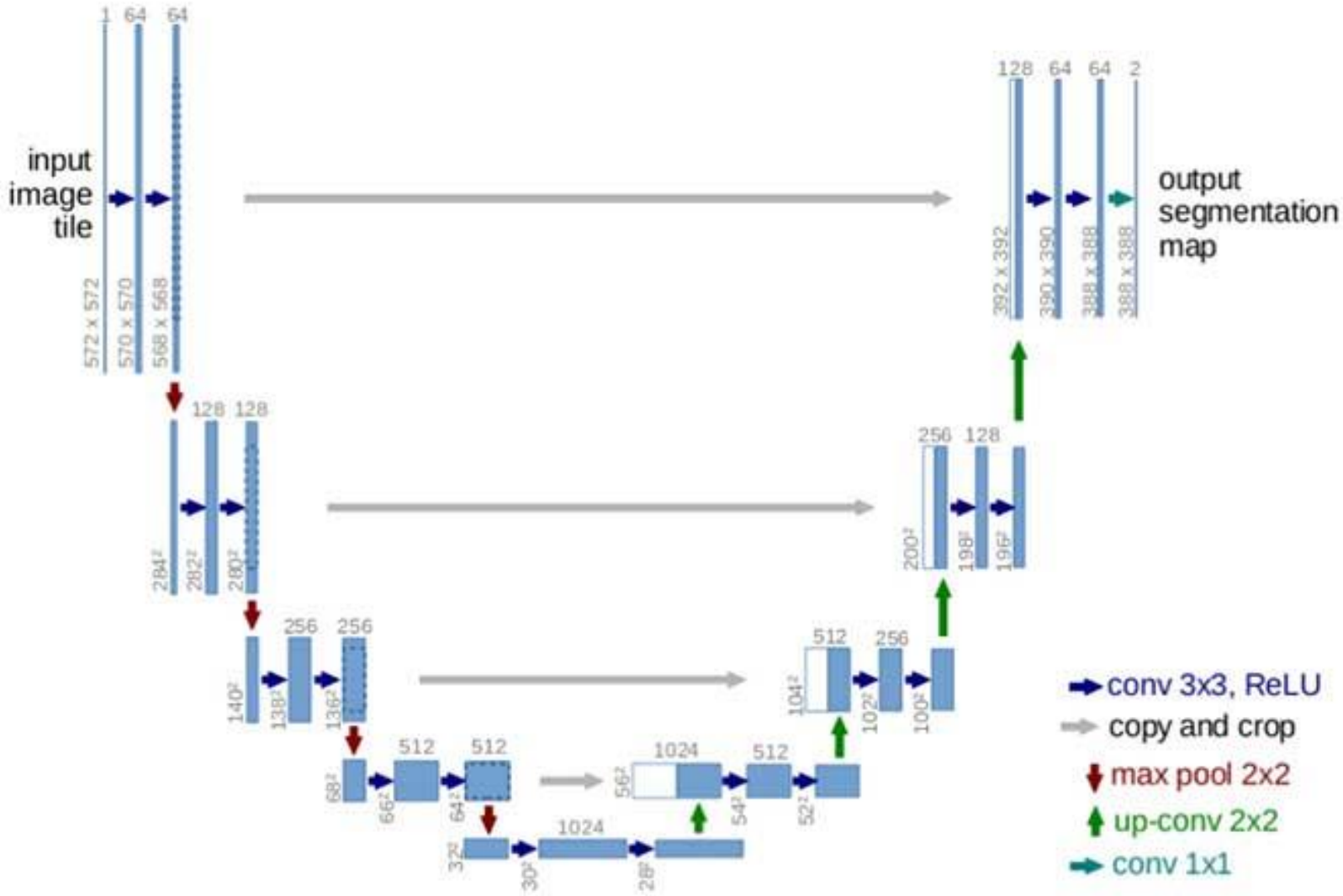
- Similar upsampling approach called unpooling is used.
- However, there are fully-connected layers which make the model larger.

2.2. Differences from U-Net

- It is used for biomedical image segmentation.
- Instead of using pooling indices, the entire feature maps are transfer from encoder to decoder, then with concatenation to perform convolution.
- This makes the model larger and need more memory.

U-NET

The U-Net is an upgrade to the simple FCN architecture. It has skip connections from the output of convolution blocks to the corresponding input of the transposed-convolution block at the same level.



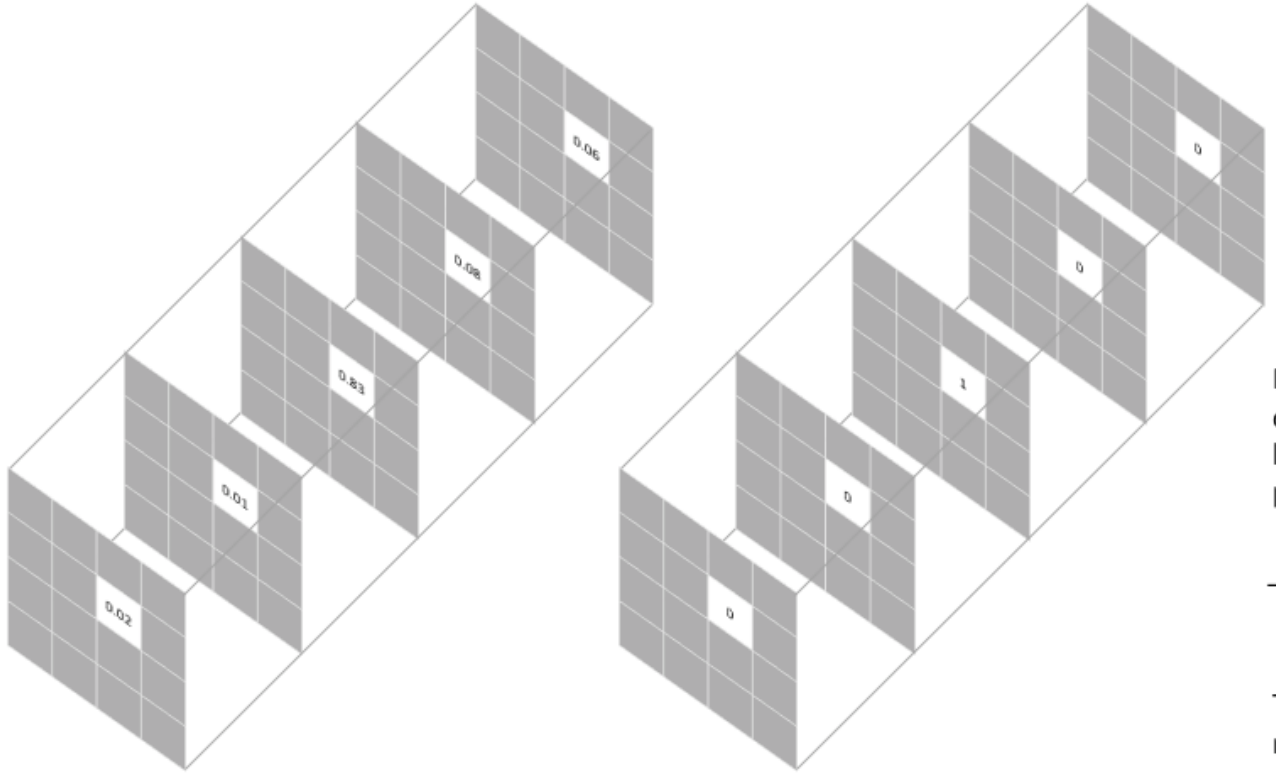
- This skip connections allows gradients to flow better and provides information from multiple scales of the image size.
- Information from larger scales (upper layers) can help the model classify better.
- Information from smaller scales (deeper layers) can help the model segment/localize better.

U-NET

- The U-Net architecture is built upon the Fully Convolutional Network (FCN) and modified in a way that it yields better segmentation in medical imaging.
- Compared to FCN-8, the two main differences are:
- (1) U-net is symmetric and
- (2) the skip connections between the downsampling path and the upsampling path apply a concatenation operator instead of a sum.
- These skip connections intend to provide local information to the global information while upsampling. Because of its symmetry, the network has a large number of feature maps in the upsampling path, which allows to transfer information. B
- The U-Net owes its name to its symmetric shape, which is different from other FCN variants.

DEFINING A LOSS FUNCTION

- The most commonly used loss function for the task of image segmentation is a **pixel-wise cross entropy loss**. This loss examines *each pixel individually*, comparing the class predictions (depth-wise pixel vector) to our one-hot encoded target vector.



Prediction for a selected pixel

Target for the corresponding pixel

Pixel-wise loss is calculated as the log loss, summed over all possible classes

$$-\sum_{classes} y_{true} \log\left(y_{pred}\right)$$

This scoring is repeated over all **pixels** and averaged

Because the cross entropy loss evaluates the class predictions for each pixel vector individually and then averages over all pixels, we're essentially asserting equal learning to each pixel in the image. This can be a problem if your various classes have unbalanced representation in the image, as training can be dominated by the most prevalent class

DEFINING A LOSS FUNCTION – DICE COEFFICIENT

- Another popular loss function for image segmentation tasks is based on the Dice coefficient, which is essentially a measure of overlap between two samples. This measure ranges from 0 to 1 where a Dice coefficient of 1 denotes perfect and complete overlap. The Dice coefficient was originally developed for binary data, and can be calculated as:

$$Dice = \frac{2\,|A \cap B|}{|A| + |B|}$$

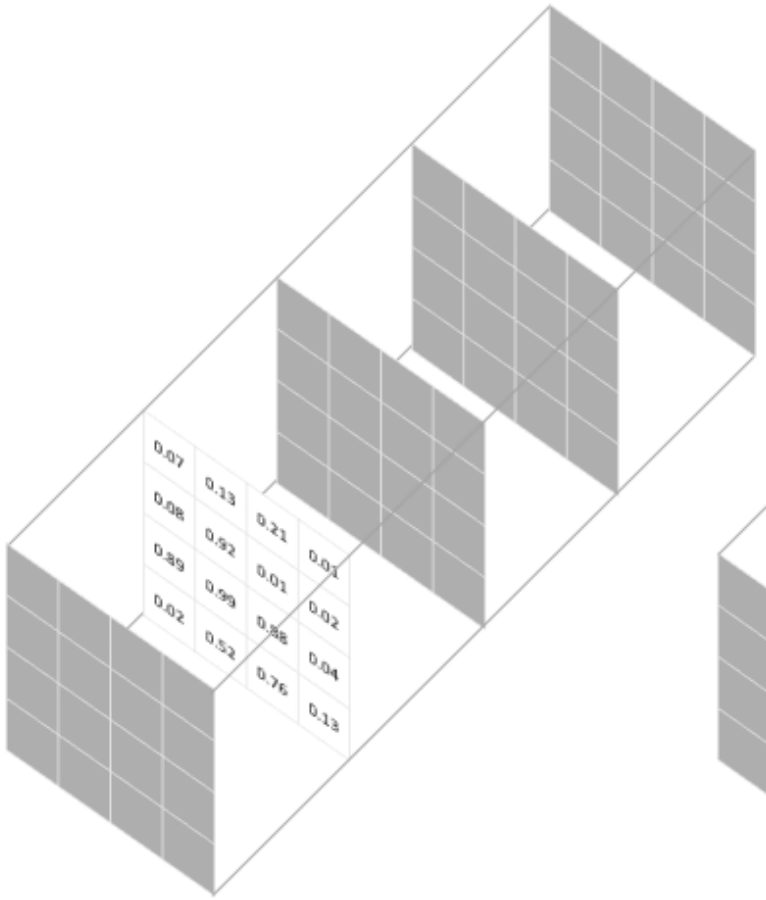
$$|A \cap B| = \begin{bmatrix} 0.01 & 0.03 & 0.02 & 0.02 \\ 0.05 & 0.12 & 0.09 & 0.07 \\ 0.89 & 0.85 & 0.88 & 0.91 \\ 0.99 & 0.97 & 0.95 & 0.97 \end{bmatrix} * \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \xrightarrow{\text{element-wise multiply}} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0.89 & 0.85 & 0.88 & 0.91 \\ 0.99 & 0.97 & 0.95 & 0.97 \end{bmatrix} \xrightarrow{\text{sum}} 7.41$$

prediction target

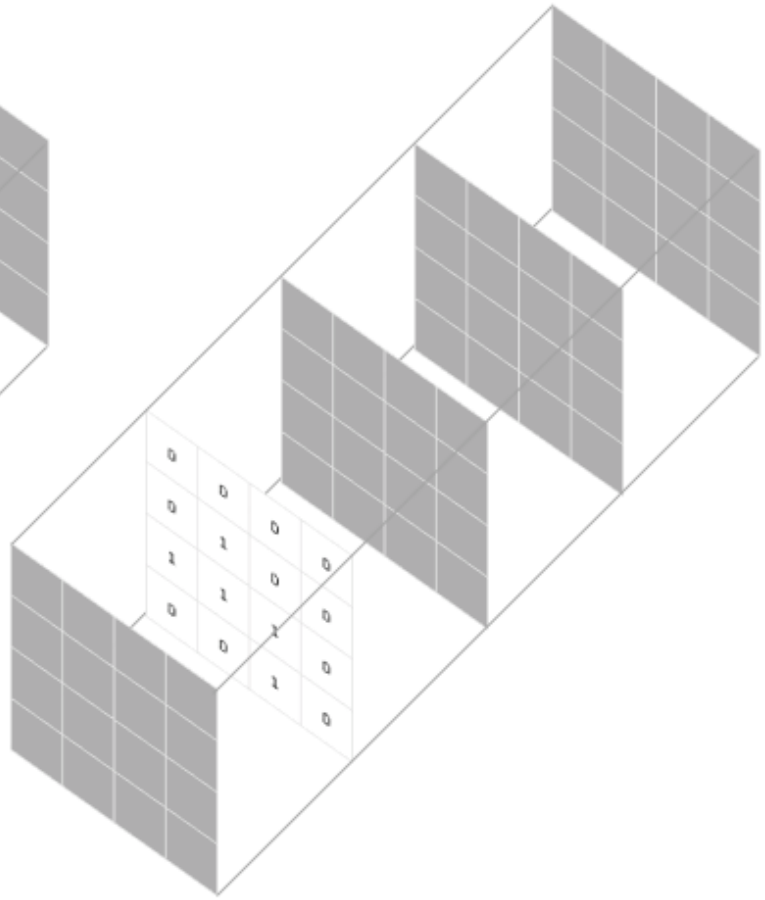
$$|A| = \begin{bmatrix} 0.01 & 0.03 & 0.02 & 0.02 \\ 0.05 & 0.12 & 0.09 & 0.07 \\ 0.89 & 0.85 & 0.88 & 0.91 \\ 0.99 & 0.97 & 0.95 & 0.97 \end{bmatrix}^{2\ (optional)} \xrightarrow{\text{sum}} 7.82$$

$$|B| = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}^{2\ (optional)} \xrightarrow{\text{sum}} 8$$

DICE



Prediction for a selected class



Target for the corresponding class

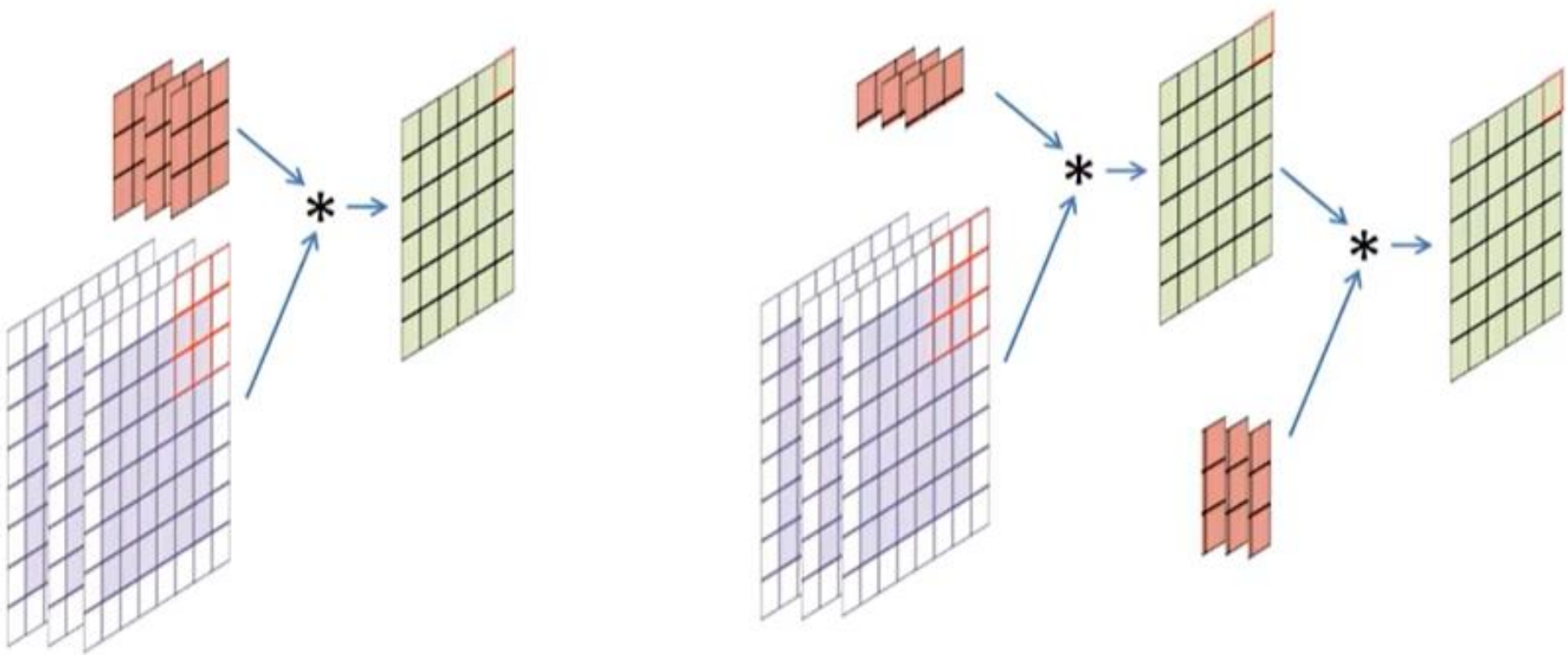
Soft Dice coefficient is calculated for each class mask

$$1 - \frac{2\sum_{pixels} y_{true} y_{pred}}{\sum_{pixels} y_{true}^2 + \sum_{pixels} y_{pred}^2}$$

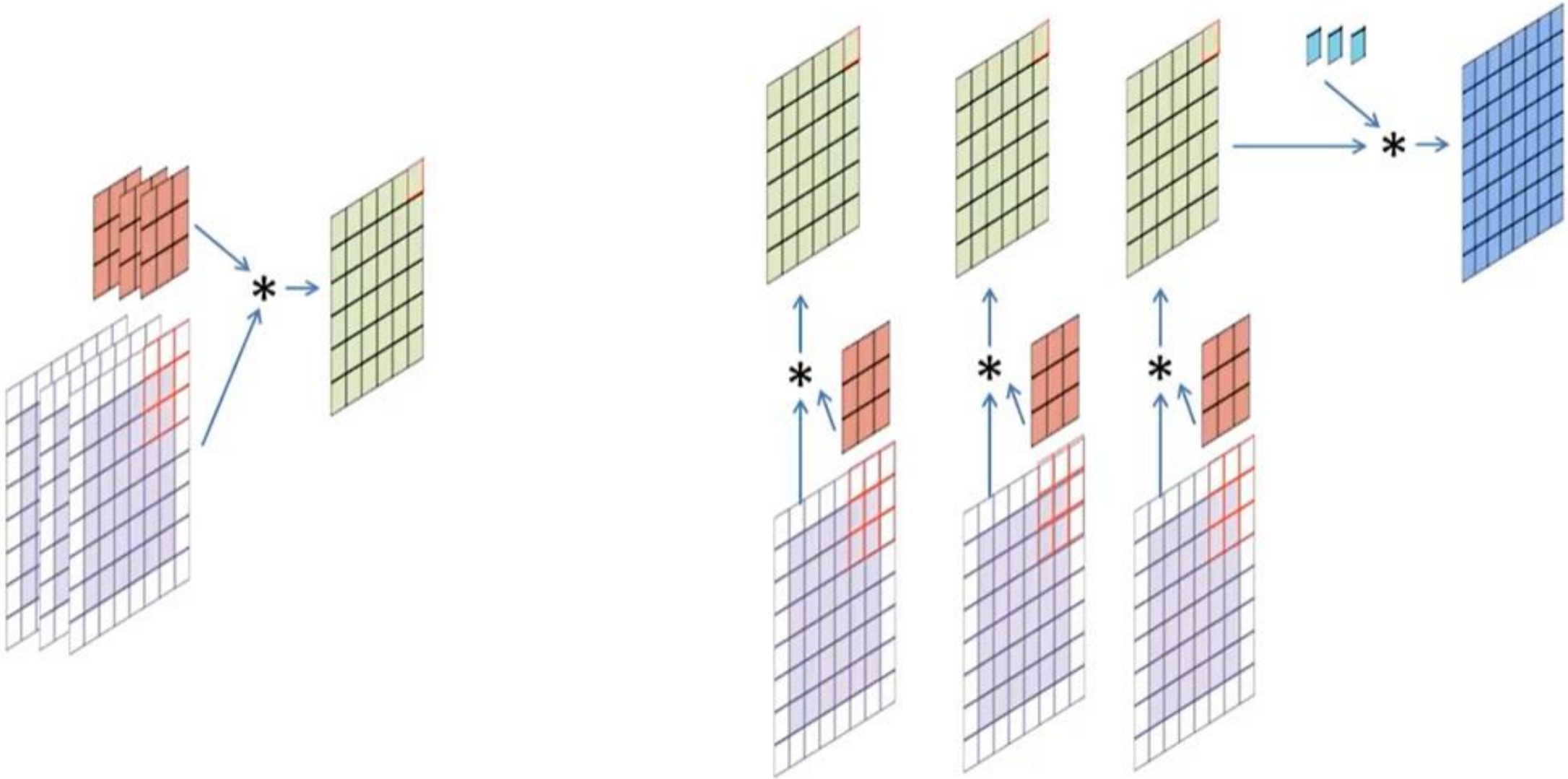
This scoring is repeated over all **classes** and averaged

In order to formulate a loss function which can be minimized, we'll simply use 1−Dice. This loss function is known as the soft Dice loss because we directly use the predicted probabilities instead of thresholding and converting them into a binary mask.

Separable convolutions

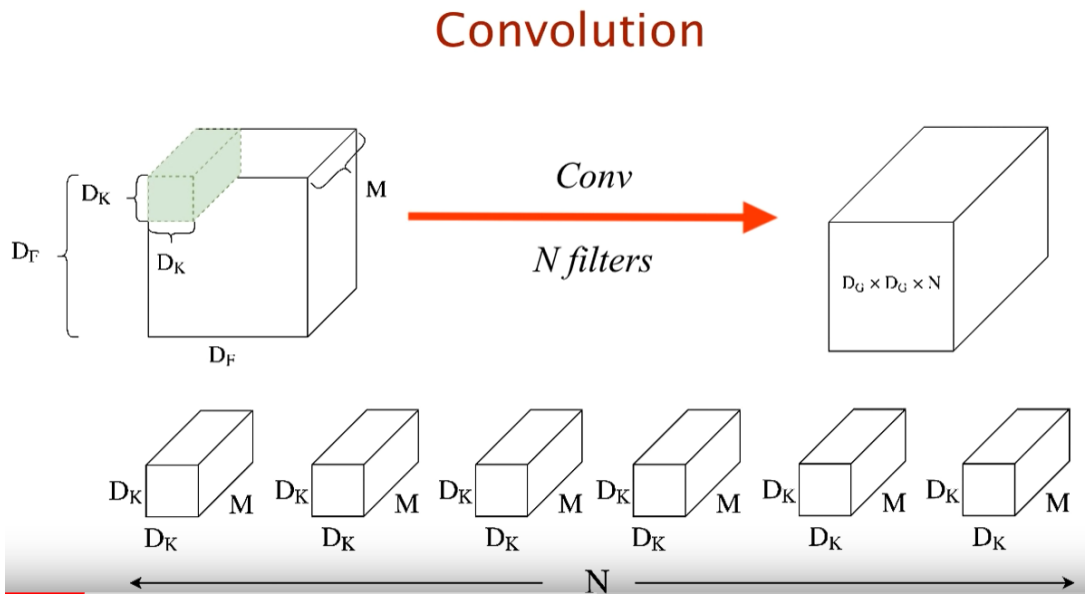


MobileNet filters each feature map separately



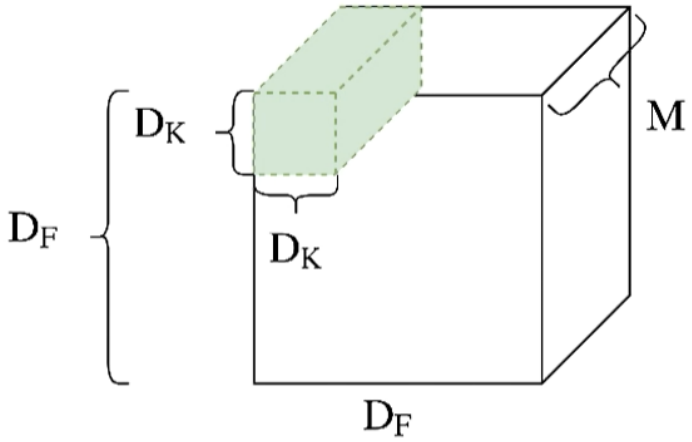
"MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications" by Andrew G. Howard Menglong Zhu Bo Chen Dmitry Kalenichenko Weijun Wang Tobias Weyand Marco Andreetto Hartwig Adam, Google , 2017

Multiplication operations in normal convolution operation



M is input channels
DFXDF – Width and height of the image
DKXDK is kernel size
DGXDG is feature maps we have N features

In order to extract N feature maps it does many Multiplication operations, we also know that multiplication operation is expensive relatively addition operation, below are the number of multiplication operations it does for N kernels



$$\text{Mults once} = D_K^2 \times M$$

For each filter here we have DKxDK and we have M channels

$$\text{Mults per Kernel} = D_G^2 \times D_K^2 \times M$$

DG times convolve operations per each kernel

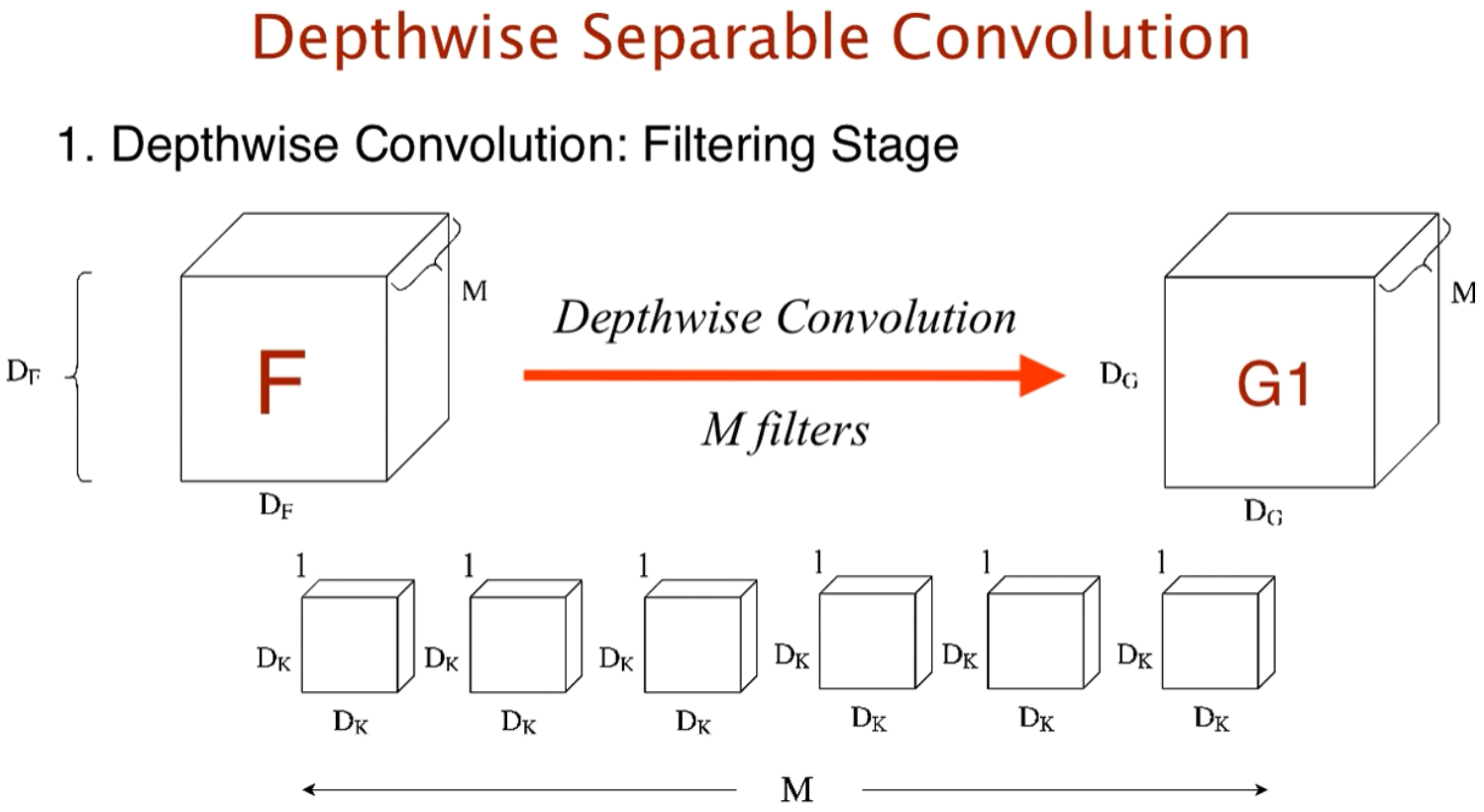
$$\text{Mults N Kernels} = N \times D_G^2 \times D_K^2 \times M$$

We have such N Features

MobileNet Architecture

MobileNet is an architecture which **is more suitable for mobile and embedded based vision applications** where there is lack of compute power. This architecture was proposed by Google.

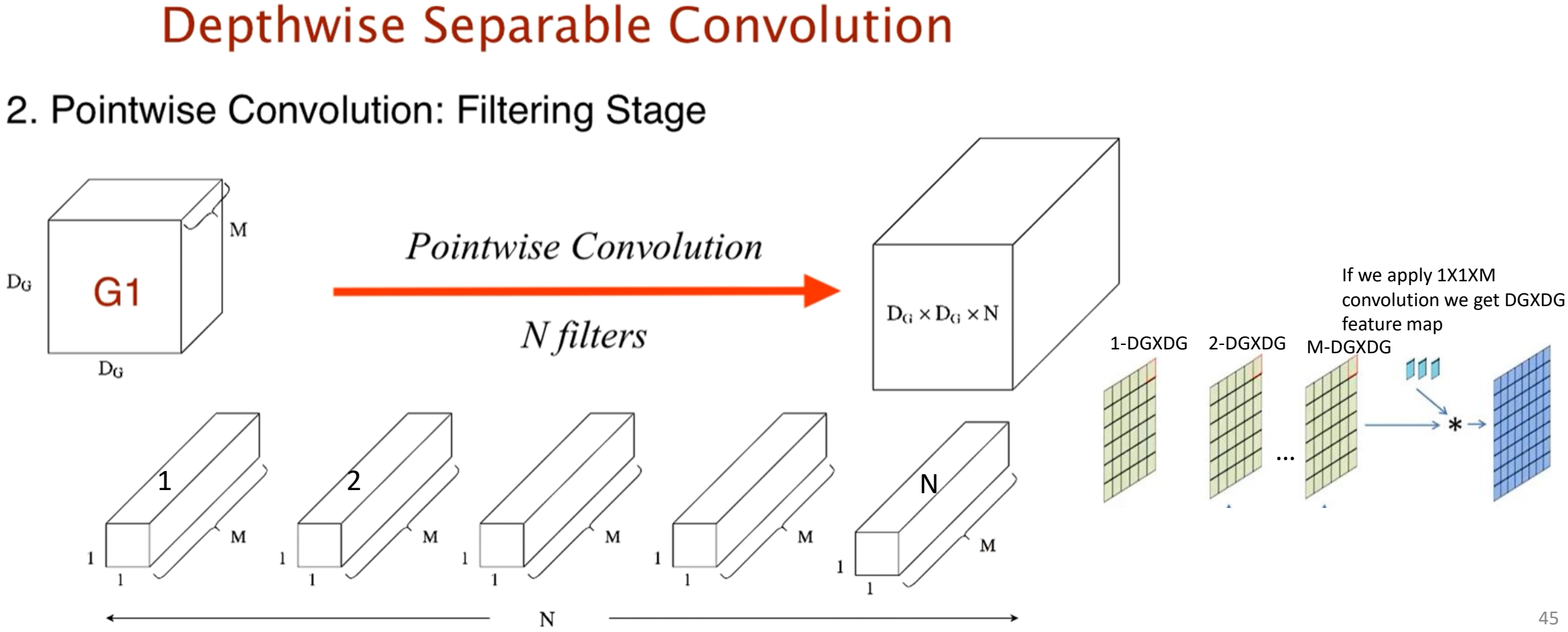
- This architecture uses **depthwise separable convolutions** which significantly **reduces the number of parameters** when compared to the network with normal convolutions with the **same depth** in the networks. This results in **light weight deep neural networks**.
- The normal convolution is replaced by **depthwise convolution followed by pointwise convolution** which is called as **depthwise separable convolution**.



M is input channels
DFXDF – Width and height of the image
DKXDK is kernel size we apply this filter for each channel or filter of the image separately
DGXDGXM feature maps

MobileNet Architecture

First step is depth wise convolution now this is succeeded by point wise convolution which involves performing linear combination of each of these layers , here input is volume of DGXDGXM and filter KPC have **1X1XM**, this is basically 1X1 convolution operation of M layers. **Output of each filter is DGXDGX1 assuming that we have N such Filters we get DGXDGXN**



Multiplication operations in MobileNet

At Stage - 1

$$\text{Mults once} = D_K^2$$

$$\text{Mults 1 Channel} = D_G^2 \times D_K^2$$

$$\text{DC Mults} = M \times D_G^2 \times D_K^2$$

At Stage - 2

$$\text{Mults once} = M$$

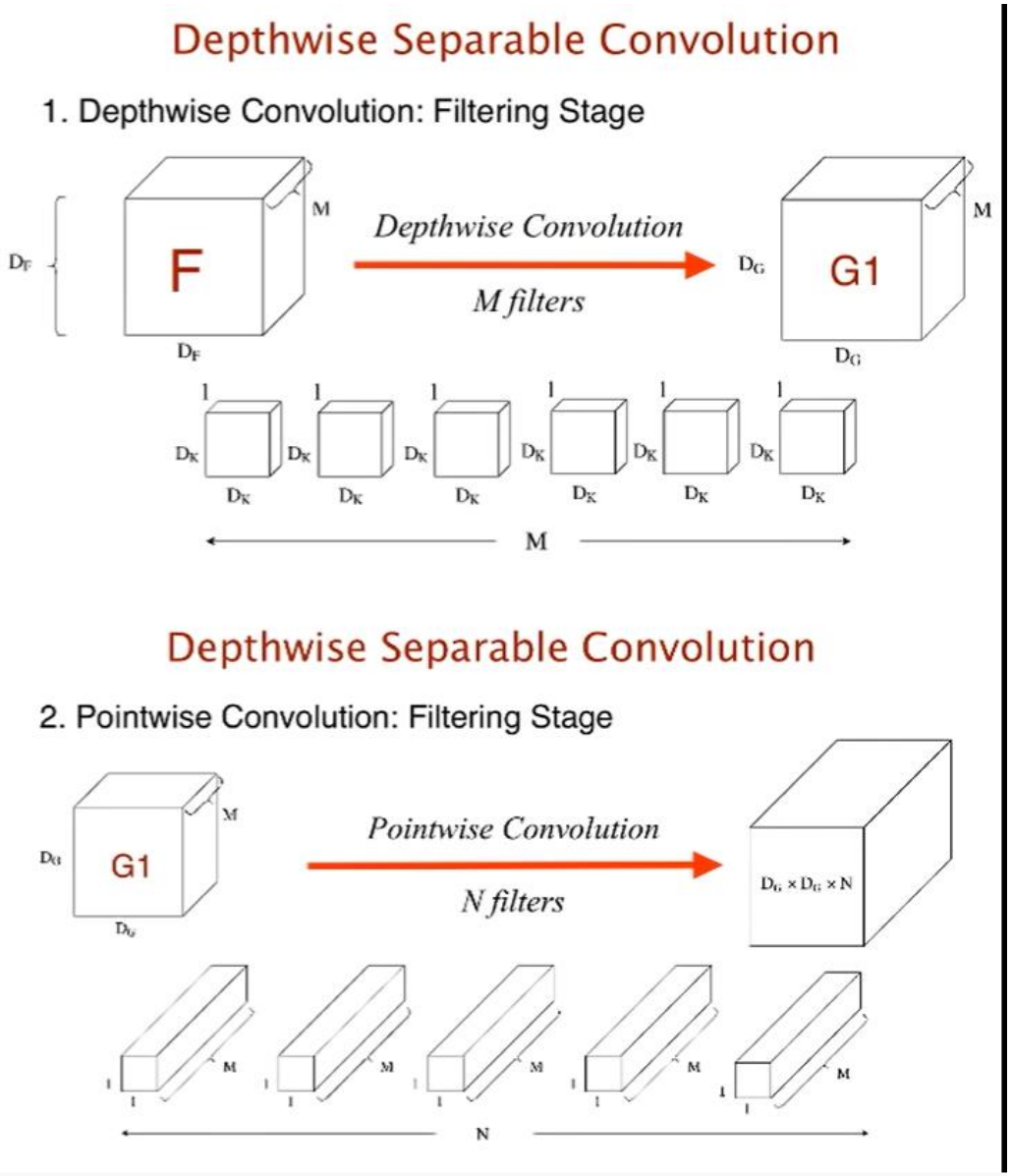
$$\text{Mults 1 Kernel} = D_G \times D_G \times M$$

$$\text{PC Mults} = N \times D_G \times D_G \times M$$

$$\text{Total} = \text{DC Mults} + \text{PC Mults}$$

$$M \times D_G^2 \times D_K^2 + N \times D_G^2 \times M$$

$$M \times D_G^2 (D_K^2 + N)$$



Comparison Standard Vs. Depthwise

$$\frac{No.Mults\ in\ Depthwise\ Separable\ Conv}{No.Mults\ in\ Standard\ Conv} = \frac{M \times D_G^2\ (D_K^2 + N)}{N \times D_G \times D_G \times D_K \times D_K \times M}$$

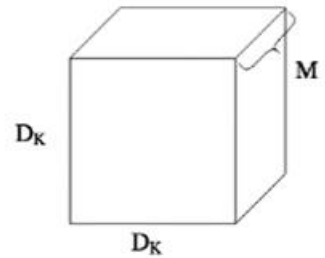
$$\frac{No.\ Mults\ in\ Depthwise\ Separable\ Conv}{No.\ Mults\ in\ Standard\ Conv} = \frac{D_K^2 + N}{(D_K^2 \times N)} = \frac{1}{N} + \frac{1}{D_K^2}$$

$$N = 1{,}024 \qquad D_K = 3$$

$$\frac{No.\ Mults\ in\ Depthwise\ Separable\ Conv}{No.\ Mults\ in\ Standard\ Conv} = \frac{1}{1024} + \frac{1}{3^2} = 0.112$$

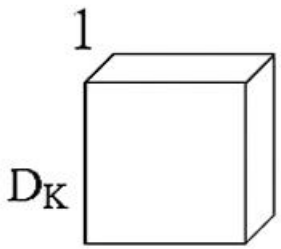
If we taken N is 1024 filters and kernel is 3X3, Which clearly indicates that standard convolution is 9 times more than the depthwise convolution we can save lot of compute power

Comparison Standard Vs. Depthwise



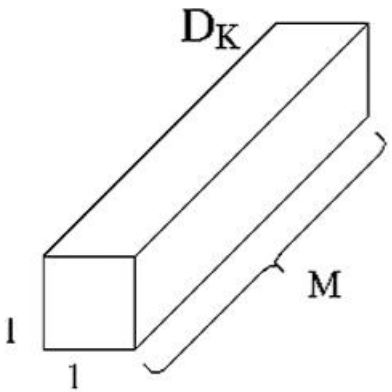
Param 1 Kernel = $D_K^2 \times M$

Param N Kernels = $N \times M \times D_K^2$



Param 1 Kernel = D_K^2

Param M Kernels = $M \times D_K^2$



Param 1 Kernel = M

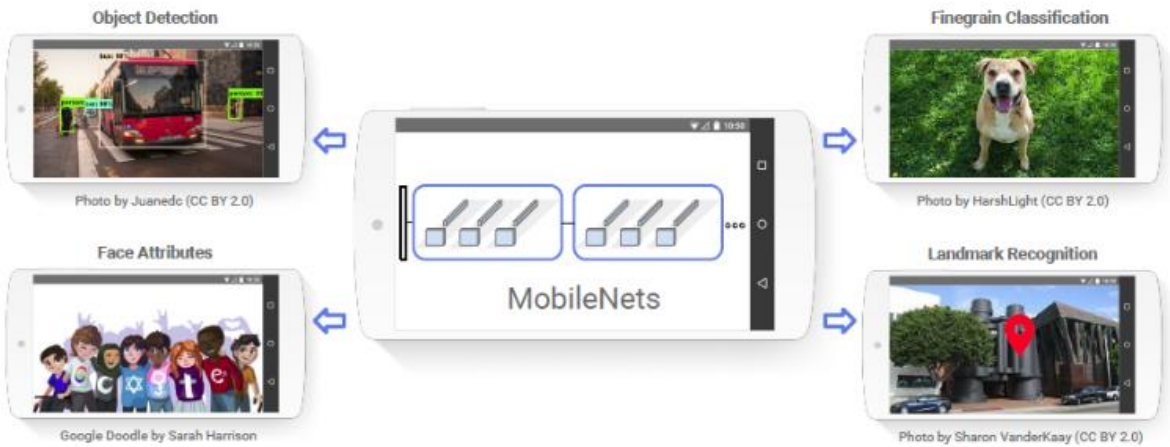
Param N Kernels = $N \times M$

$M(D_K^2 + N)$

$$\frac{No.params\ in\ Depthwise\ Separable\ Conv}{No.params\ in\ Standard\ Conv} = \frac{M \times (D_K^2 + N)}{N \times D_K^2 \times M} = \frac{1}{N} + \frac{1}{D_K^2}$$

In this story, **MobileNetV1** from Google is reviewed. **Depthwise Separable Convolution** is used to reduce the model size and complexity. It is **particularly useful for mobile and embedded vision applications**.

- **Smaller model size**: Fewer number of parameters
- **Smaller complexity**: Fewer Multiplications and Additions (Multi-Adds)



When MobileNets Applied to Real Life

Table 4. Depthwise Separable vs Full Convolution MobileNet

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
Conv MobileNet	71.7%	4866	29.3
MobileNet	70.6%	569	4.2

Things to Remember

1. Depthwise Sep. Conv. reduces computation time, parameters

2. Depthwise Sep. Conv. = Depthwise Conv + Poinwise Conv

3. Used in recent architectures (MultiModel Nets, Xception, MobileNets).

Two parameters are introduced so that MobileNet can be tuned easily: **Width Multiplier α** and **Resolution Multiplier ρ**.

3. Width Multiplier α for Thinner Models

#of Channels

Width Multiplier α is introduced to **control the input width of a layer**, which makes M become αM. And the depthwise separable convolution cost become:

$$D_K \cdot D_K \cdot \alpha M \cdot D_F \cdot D_F + \alpha M \cdot \alpha N \cdot D_F \cdot D_F$$

Depthwise Separable Convolution Cost with Width Multiplier α

where α is between 0 to 1, with typical settings of 1, 0.75, 0.5 and 0.25. When α=1, it is the baseline MobileNet. And the computational cost and the number of parameters can be reduced quadratically by roughly α^2.

Table 6. MobileNet Width Multiplier

Width Multiplier	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
0.75 MobileNet-224	68.4%	325	2.6
0.5 MobileNet-224	63.7%	149	1.3
0.25 MobileNet-224	50.6%	41	0.5

Different Values of Width Multiplier α

Accuracy drops off smoothly from α=1 to 0.5 until α=0.25 which is too small.

4. Resolution Multiplier ρ for Reduced Representation

Width and height of the image

Resolution Multiplier ρ is introduced to **control the input image resolution** of the network, with Resolution Multiplier ρ, the cost become:

$$D_K \cdot D_K \cdot \alpha M \cdot \rho D_F \cdot \rho D_F + \alpha M \cdot \alpha N \cdot \rho D_F \cdot \rho D_F$$

Depthwise Separable Convolution Cost with Both Width Multiplier α and Resolution Multiplier ρ

where ρ is between 0 to 1. And the input resolution is 224, 192, 160, and 128. When ρ=1, it is the baseline MobileNet.

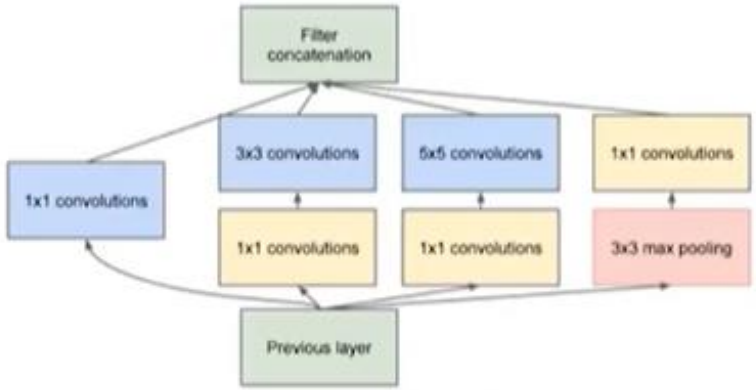
Table 7. MobileNet Resolution

Resolution	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
1.0 MobileNet-192	69.1%	418	4.2
1.0 MobileNet-160	67.2%	290	4.2
1.0 MobileNet-128	64.4%	186	4.2

Different Values of Resolution Multiplier ρ

Accuracy drops off smoothly across resolution from 224 to 128.

Inception uses multiple sized convolution filters



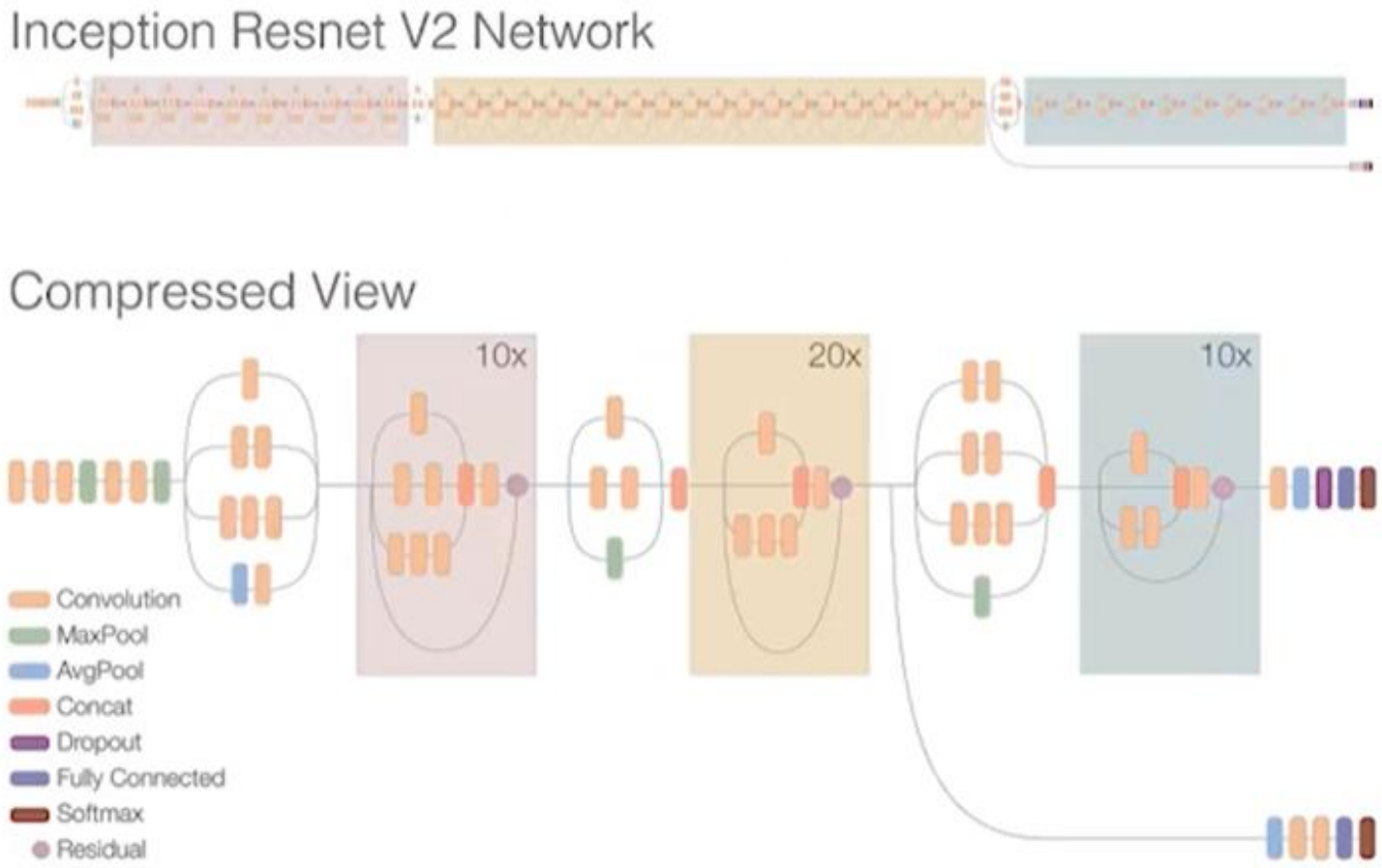


Image source: https://ai.googleblog.com/2016/08/improving-inception-and-image.html

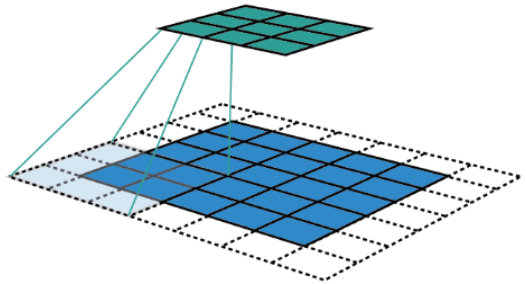
APPENDIX

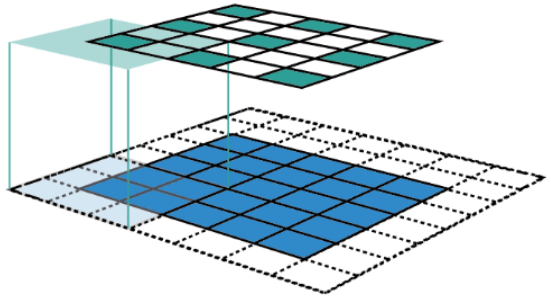
RECEPTIVE FIELD

A essential design choice of any CNN architecture is that the input sizes grow smaller and smaller from the start to the end of the network, while the number of channels grow deeper. This is done through strides or pooling layers.

- **Locality determines what inputs from the previous layer the outputs get to see.**
- **The receptive field determines what area of the *original input* to the entire network the output gets to see.**

Figure 1 shows some receptive field examples. By applying a convolution C with kernel size ***k* = 3x3**, padding size ***p* = 1x1**, stride ***s* = 2x2** on an input map **5x5**, we will get an output feature map **3x3** (green map). Applying the same convolution on top of the 3x3 feature map, we will get a **2x2** feature map (orange map).



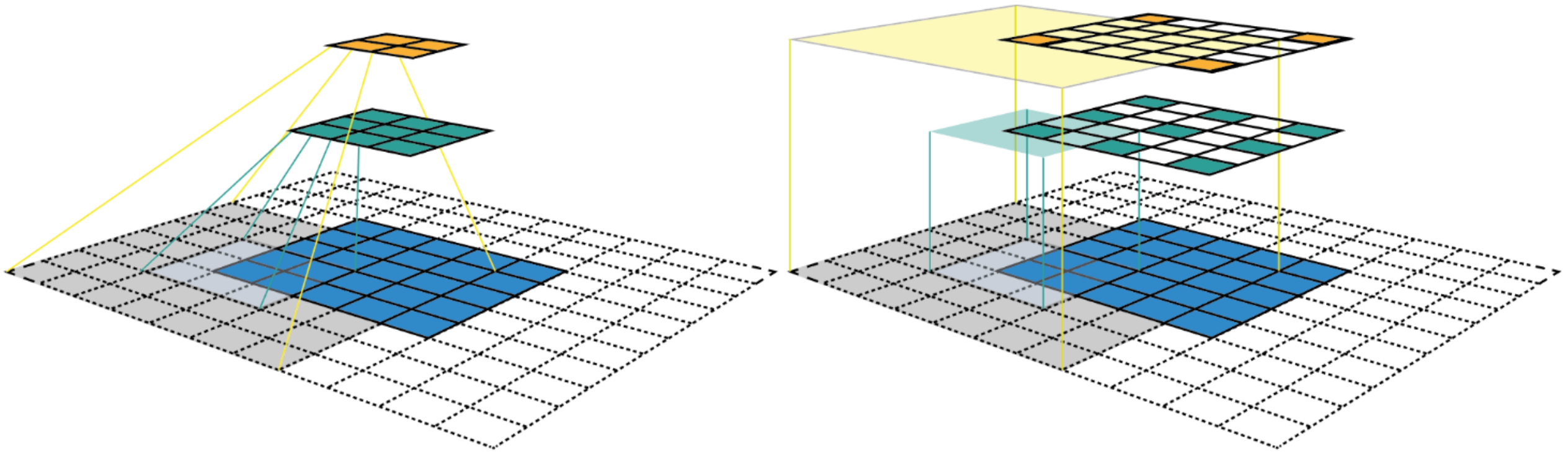


This figure shows a common way to visualize a CNN
Only looking at the feature map, we do not know where a feature is looking at (the center location of its receptive field) and how big is that region (its receptive field size).

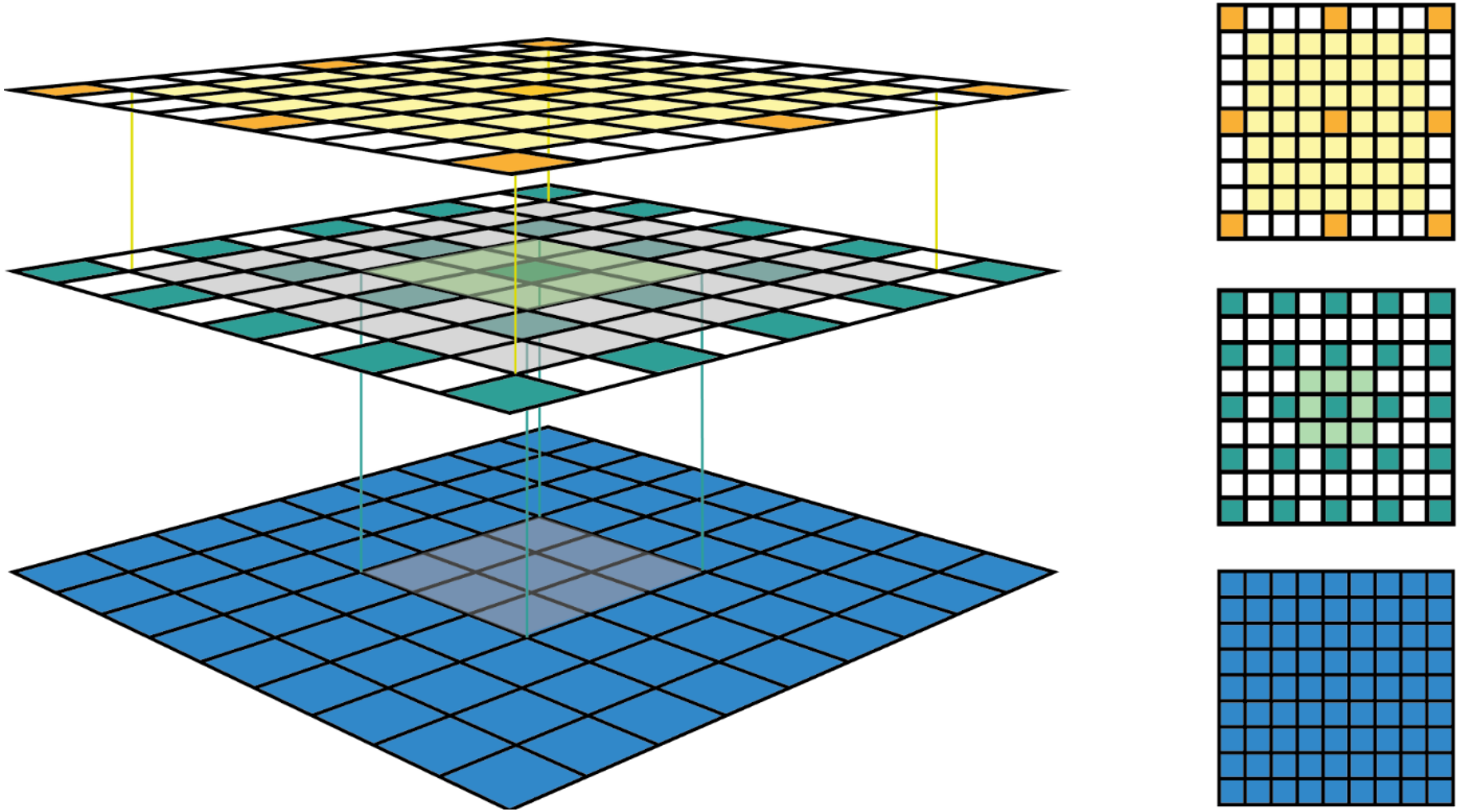
In fixed-sized CNN feature map visualization, where the size of each feature map is fixed, and the feature is located at the center of its receptive field.
Each feature is then marked at the center of its receptive field location. Because all features in a feature map have the same receptive field size, we can simply draw a bounding box around one feature to represent its receptive field size.

RECEPTIVE FIELD

Figure 1 shows some receptive field examples. By applying a convolution C with kernel size ***k* = 3x3**, padding size ***p* = 1x1**, stride ***s* = 2x2** on an input map **5x5**, we will get an output feature map **3x3** (green map). Applying the same convolution on top of the 3x3 feature map, we will get a **2x2** feature map (orange map).



Another example using the same convolution but applied on a bigger input map — 7x7. We can either plot the fixed-sized CNN feature maps in 3D (Left) or in 2D (Right). Notice that the size of the receptive field in below **escalates very quickly to the point that the receptive field of the center feature of the second feature layer covers almost the whole input map**. This is an important insight which was used to improve the design of a deep CNN.



Atrous (dilated) convolutions can increase the receptive field without increasing the number of weights