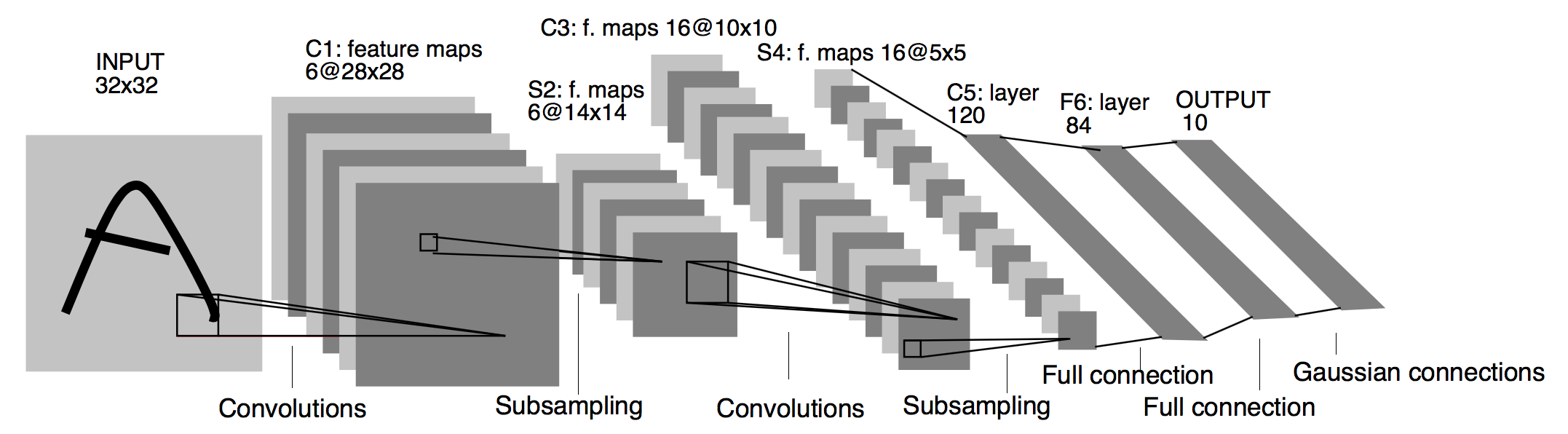
Comparison between different models

**LeNet 5(1998)**



LeNet-5 is a very simple network.

* It only has 7 layers
* There are 3 convolutional layers (C1, C3 and C5)
* 2 sub-sampling (pooling) layers (S2 and S4)
  + 1 fully connected layer (F6)
  + output layer
* Convolutional layers use 5 by 5 convolutions with stride 1. Sub-sampling layers are 2 by 2 average pooling layers.
* Tanh sigmoid activations are used throughout the network. ( tanh function is [-1,1] ,Having stronger gradients: )

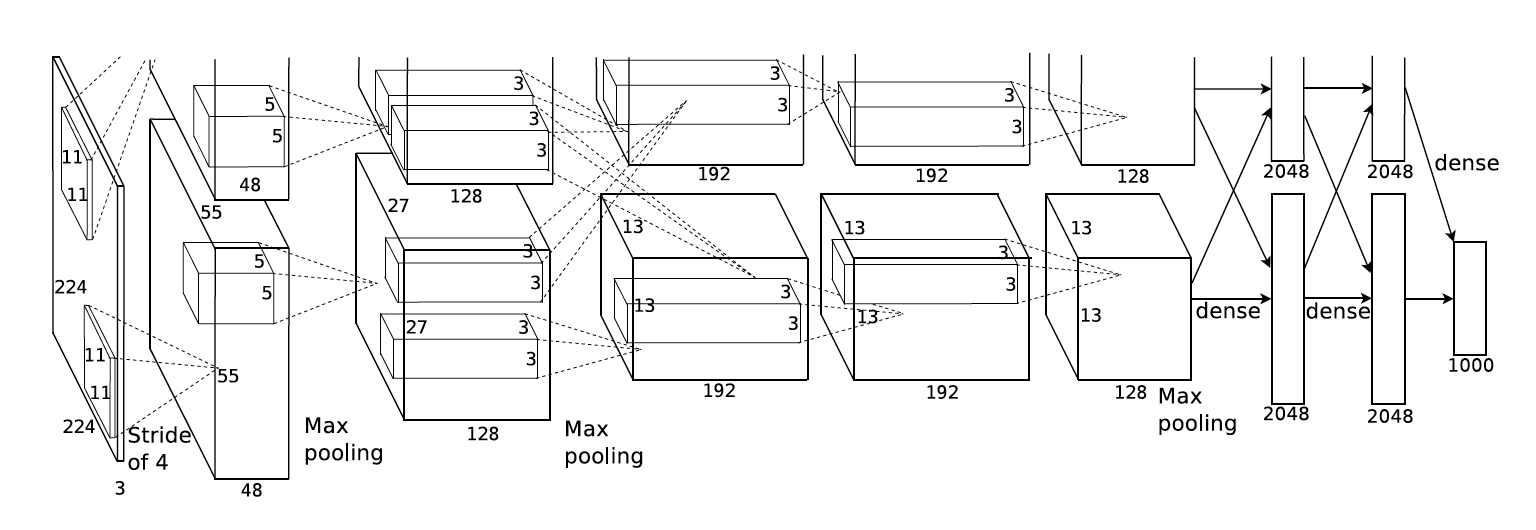
Input 32x32

Output 10x1

### 

### 

### **AlexNet (2012)**



* Use Relu instead of Tanh to add non-linearity. It accelerates the speed by 6 times at the same accuracy.
* Use dropout instead of regularisation to deal with overfitting. However the training time is doubled with the dropout rate of 0.5.
* Overlap pooling to reduce the size of network. It reduces the top-1 and top-5 error rates by 0.4% and 0.3%, repectively.

https://airtable.com/shrArXKRCau4KhAwZ/tbloN5WYjFYpKGUEt?blocks=hide

It contains

* 5 convolutional layers
* 3 fully connected layers.
* Relu is applied after very convolutional and fully connected

layer.

* Dropout is applied before the first and the second fully connected year.
* The image size in the following architecutre chart should be 227 \* 227 instead of 224 \* 224, as it is pointed out by Andrei Karpathy in his famous CS231n Course.
* More insterestingly, the input size is 224 \* 224 with 2 padding in the pytorch torch vision. The output width and height should be (224–11+4)/4 + 1=55.25! The explanation here is pytorch Conv2d apply floor operator to the above result, and therefore the last one padding is ignored.

It consisted

11x11, 5x5,3x3 -convolutions

max pooling

Dropout

data augmentation

ReLU activations

SGD with momentum.

It attached ReLU activations after every convolutional and fully-connected layer.

### 

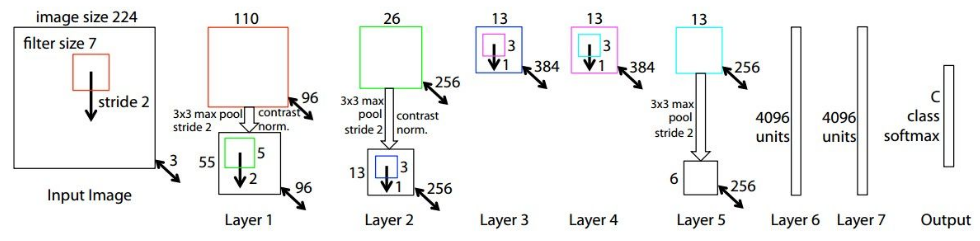
### 

### 

### 

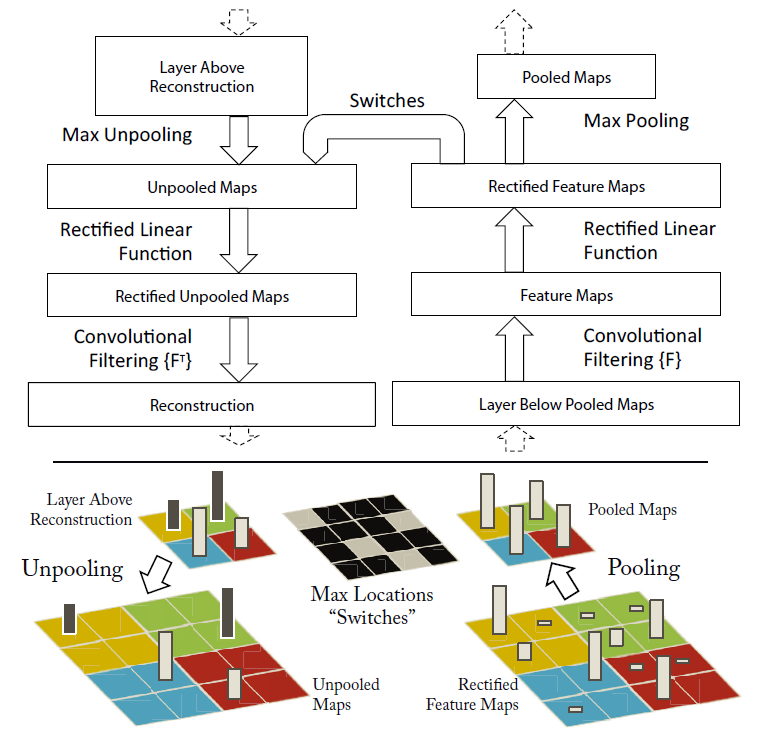
### 

### **ZFNet(2013)**



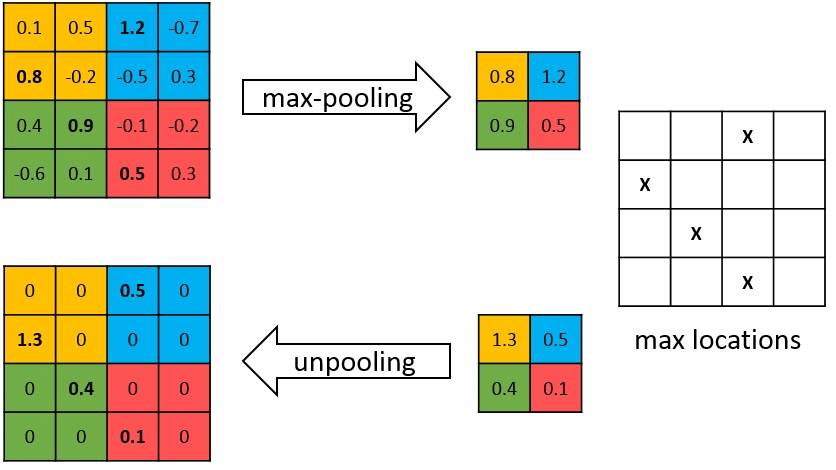
Improvement over AlexNet

Deconvnet Techniques for Visualization



Convolution -> Rectification -> Pooling

Unpooling

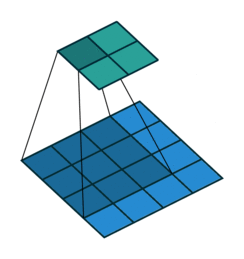


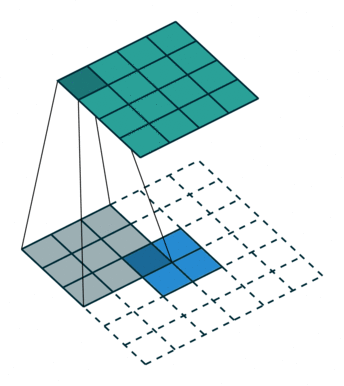
Max pooling operation is non-invertible, however we can obtain an approximate inverse by recording the locations of the maxima within each pooling region, as in the figure above.

Rectification (Activation Function)

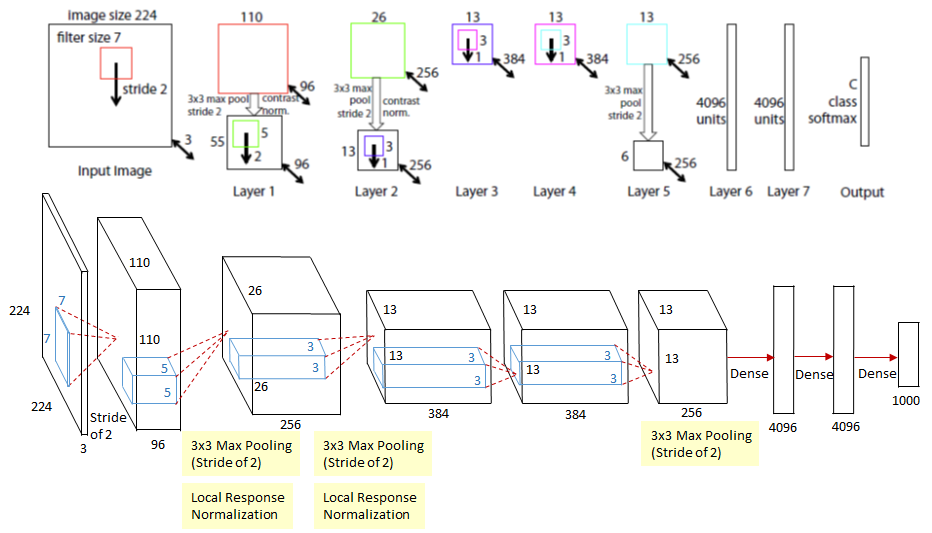
Since ReLU is used as the activation function, and ReLU is to keep all values positive while make negative values become zero. In the reverse operation, we just need to perform ReLU again.

Deconv

**Conv (Blue is input, cyan is output)**

**Deconv (Blue is input, cyan is output)**

### Modifications of AlexNet Based on Visualization Results



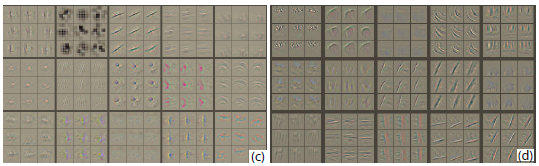
### ZFNet is redrawn as the same style of AlexNet for the ease of comparison. To solve the two problems observed in layer 1 and layer 2, ZFNet makes two changes.

### Reduced the 1st layer filter size from 11x11 to 7x7.

### Made the 1st layer stride of the convolution 2, rather than 4.

### 

Layer 1: (a) More mid-frequencies in ZFNet, (b) Extremely low and high frequencies in AlexNet

****

Layer 2: (c) Aliasing artifacts in AlexNet and (d) much cleaner features in ZFNet

### 

### **GoogleNet/Inception(2014)**

It contains **1×1 Convolution** at the middle of the network.

And **global average pooling** is used at the end of the network instead of using fully connected layers.

Another technique, called **inception module**, is to have different sizes/types of convolutions for the same input and stacking all the outputs.

* The 1×1 Convolution
* Inception Module
* Global Average Pooling
* Overall Architecture
* Auxiliary Classifiers for Training
* Testing Details

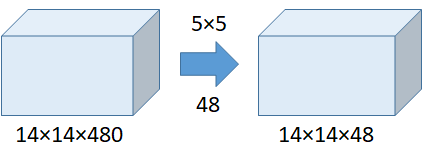
The 1×1 Convolution

1×1 convolution is used with ReLU.

1×1 convolution is used as a dimension reduction module to reduce the computation.

By reducing the computation bottleneck, depth and width can be increased.

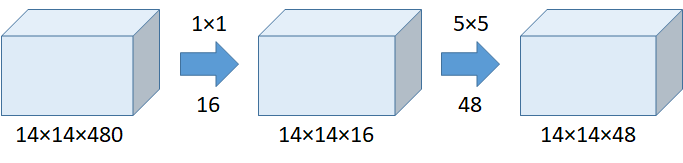
Suppose we need to perform 5×5 convolution without the use of 1×1 convolution as below:



Without the Use of 1×1 Convolution

Number of operations = (14×14×48)×(5×5×480) = 112.9M

With the use of 1×1 convolution:



With the Use of 1×1 Convolution

Number of operations for 1×1 = (14×14×16)×(1×1×480) = 1.5M

**Number of operations for 5×5 = (14×14×48)×(5×5×16) = 3.8M**

**Total number of operations = 1.5M + 3.8M = 5.3M which is much much smaller than 112.9M !!!!!!!!!!!!!!!**

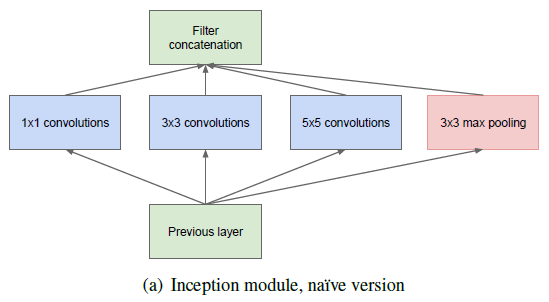
Indeed, the above example is the calculation of **5×5 conv at inception (4a).**

(We may think that, when dimension is reduced, actually we are working on the mapping from high dimension to low dimension in a non-linearity way. In contrast, for PCA, it performs linear dimension reduction.)

Thus, **inception module can be built without increasing the number of operations** **largely!**

Inception Module

The inception module (naive version, without 1×1 convolution) is as below:

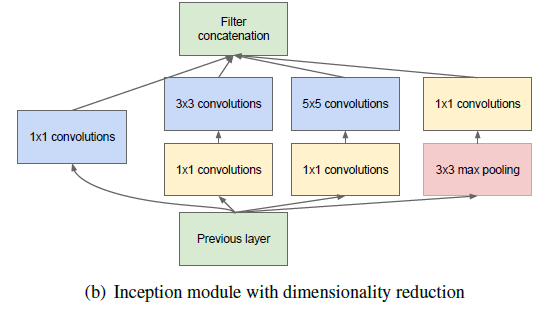


**Inception Module (Without 1×1 Convolution)**

Previously, such as AlexNet, and VGGNet, conv size is fixed for each layer.

Now, 1×1 conv, 3×3 conv, 5×5 conv, and 3×3 max pooling are done altogether for the previous input, and stack together again at output. When image’s coming in, let the network choose the right path.

However, without the 1×1 convolution as above, we can imagine how large the number of operation is!



**Inception Module (With 1×1 Convolution)**

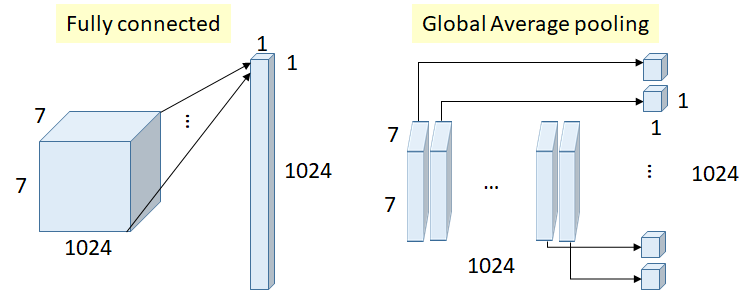
**Thus, 1×1 convolution is inserted into the inception module for dimension reduction!**

### 

### 

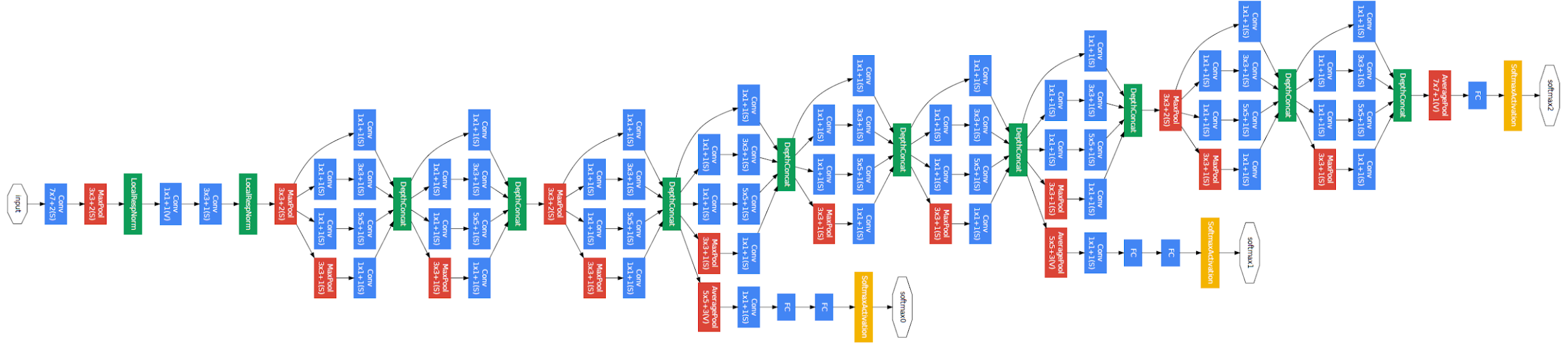
### 

### Global Average Pooling



**Fully Connected Layer VS Global Average Pooling**

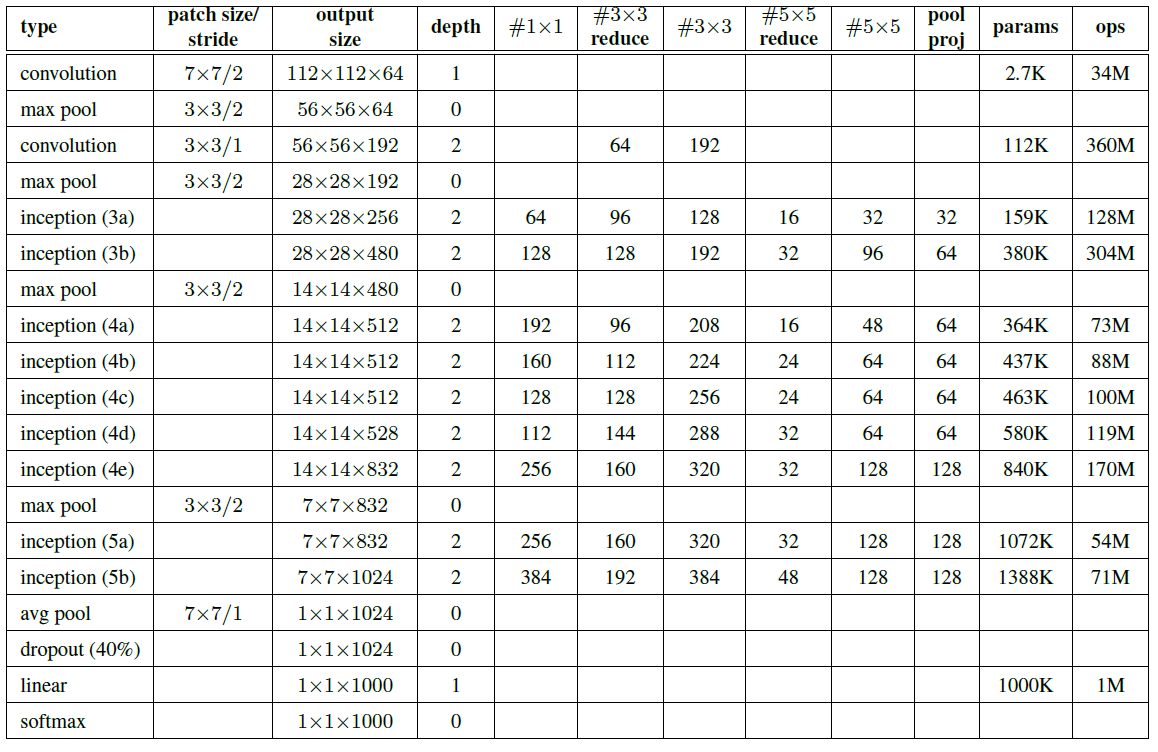
* Previously, fully connected (FC) layers are used at the end of network, such as in AlexNet.
* All inputs are connected to each output.
* Number of weights (connections) above = 7×7×1024×1024 = 51.3M
* In GoogLeNet, global average pooling is used nearly at the end of network by averaging each feature map from 7×7 to 1×1, as in the figure above.
* Number of weights = 0
* And authors found that a move from FC layers to average pooling improved the top-1 accuracy by about 0.6%.
* This is the idea which can be less prone to overfitting.



**There are 22 layers in total**

It is already a very deep model compared with previous AlexNet, ZFNet and VGGNet.And we can see that **there are numerous inception modules connected together to go deeper.** (There are some intermediate softmax branches at the middle, we will describe about them in the next section.)

Below is the details about the parameters if each layer. We actually can extend the example of 1×1 convolution to calculate the number of operations by ourselves. :)



* As we can see there are some intermediate softmax branches at the middle, they are used for training only. These branches are auxiliary classifiers which consist of:
  + 5×5 Average Pooling (Stride 3)
  + 1×1 Conv (128 filters)
  + 1024 FC
  + 1000 FC
  + Softmax
* The loss is added to the total loss, with weight 0.3.
* Authors claim it can be used for combating gradient vanishing problem, also providing regularization.

### 

### 

### 

### 

### 

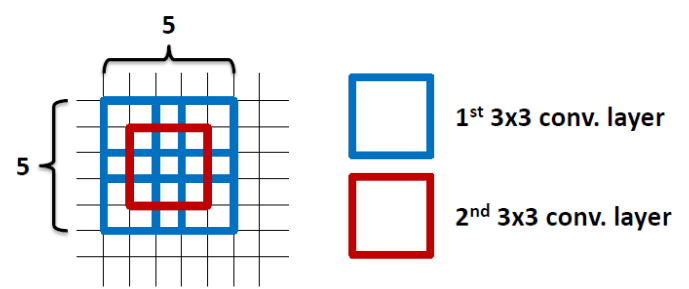
### 

### **VGGNet (2014)**

1. **The Use of 3×3 Filters** instead of large-size filters (such as 11×11, 7×7)
2. **VGG-16** and **VGG-19** based on ablation study   
   (VGG-11, VGG-11 (LRN), VGG-13, VGG-16 (Conv1) are also included.)
3. **Multi-Scale Training**
4. **Multi-Scale Testing**
5. **Dense Testing**
6. **Model Fusion**
7. **Comparison Between VGGNet and GoogLeNet**
8. **Localization Task**

### 

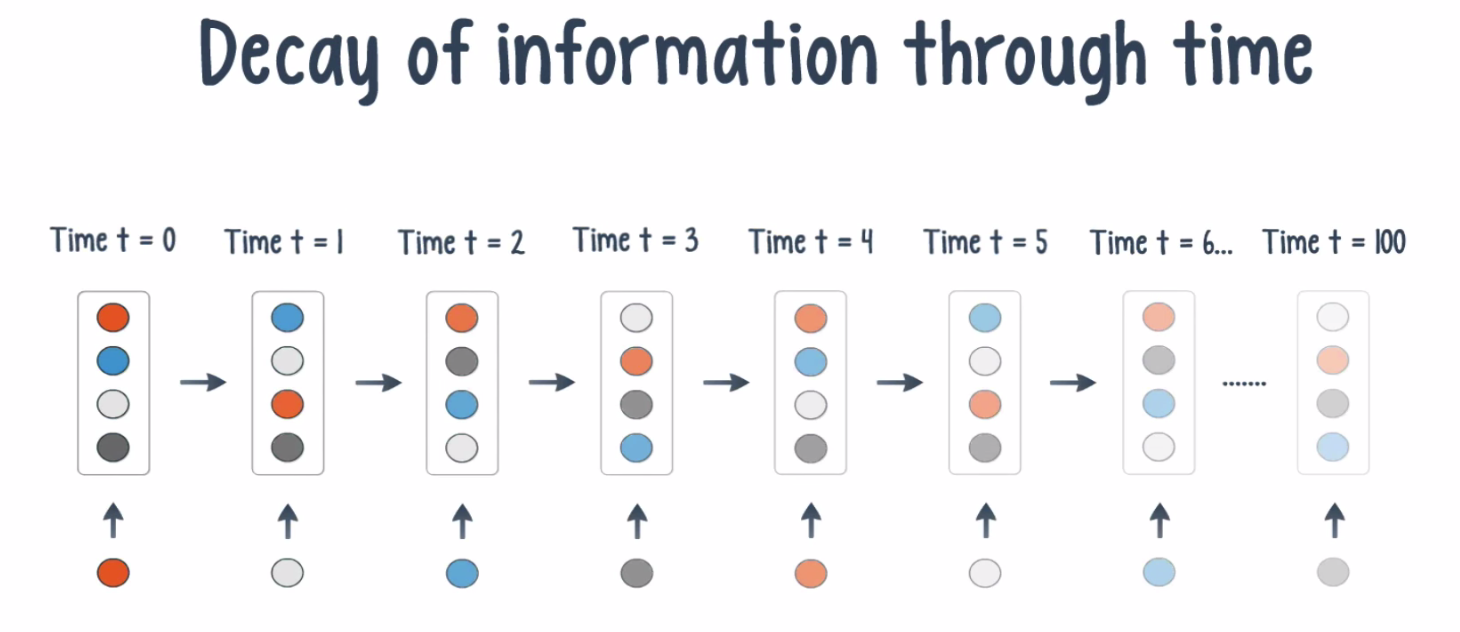
### **The Use of 3×3 Filters**



Another reason is that **the number of parameters are fewer.** Suppose there is only 1 filter per layer, 1 layer at input, and exclude the bias:

* 1 layer of 11×11 filter, number of parameters = 11×11=121
* 5 layer of 3×3 filter, number of parameters = 3×3×5=45
  + Number of parameters is reduced by 63%
* 1 layer of 7×7 filter, number of parameters = 7×7=49
* 3 layers of 3×3 filters, number of parameters = 3×3×3=27
  + Number of parameters is reduced by 45%
* By using 1 layer of 5×5 filter, number of parameters = 5×5=25
* By using 2 layers of 3×3 filters, number of parameters = 3×3+3×3=18
  + Number of parameters is reduced by 28%
* Larger network, hungrier the network for the training images. There are also vanishing gradient problem.
* With fewer parameters to be learnt, it is better for faster convergence, and reduced overfitting problem.

**Vanishing Gradient Problem.**



Vanishing Gradient Problem occurs when we try to train a Neural Network model using Gradient based optimization techniques.

Vanishing Gradient Problem was actually a major problem 10 years back to train a Deep neural Network Model due to the long training process and the degraded accuracy of the Model.

What happens is that as we keep on adding more and more Hidden layers in The model , the learning speed of the next hidden layers in the model keep on getting faster and faster.

Generally, adding more hidden layers tends to make the network able to learn more complex arbitrary functions, and thus do a better job in predicting future outcomes.

This is where **Deep Learning** is making a big difference due to the *thousands and millions of* ***hidden layers*** it has , we can now make sense of highly complicated data such as images , speeches , videos etc and do Speech Recognition and Image Classification , Image Captioning etc.

What is the Vanishing Gradient Problem?

Now when we do Back-propagation i.e moving backward in the Network and calculating gradients of loss(Error) with respect to the weights , the gradients tends to get smaller and smaller as we keep on moving backward in the Network.

This means that the neurons in the Earlier layers learn very slowly as compared to the neurons in the later layers in the Hierarchy.

The Earlier layers in the network are slowest to train.

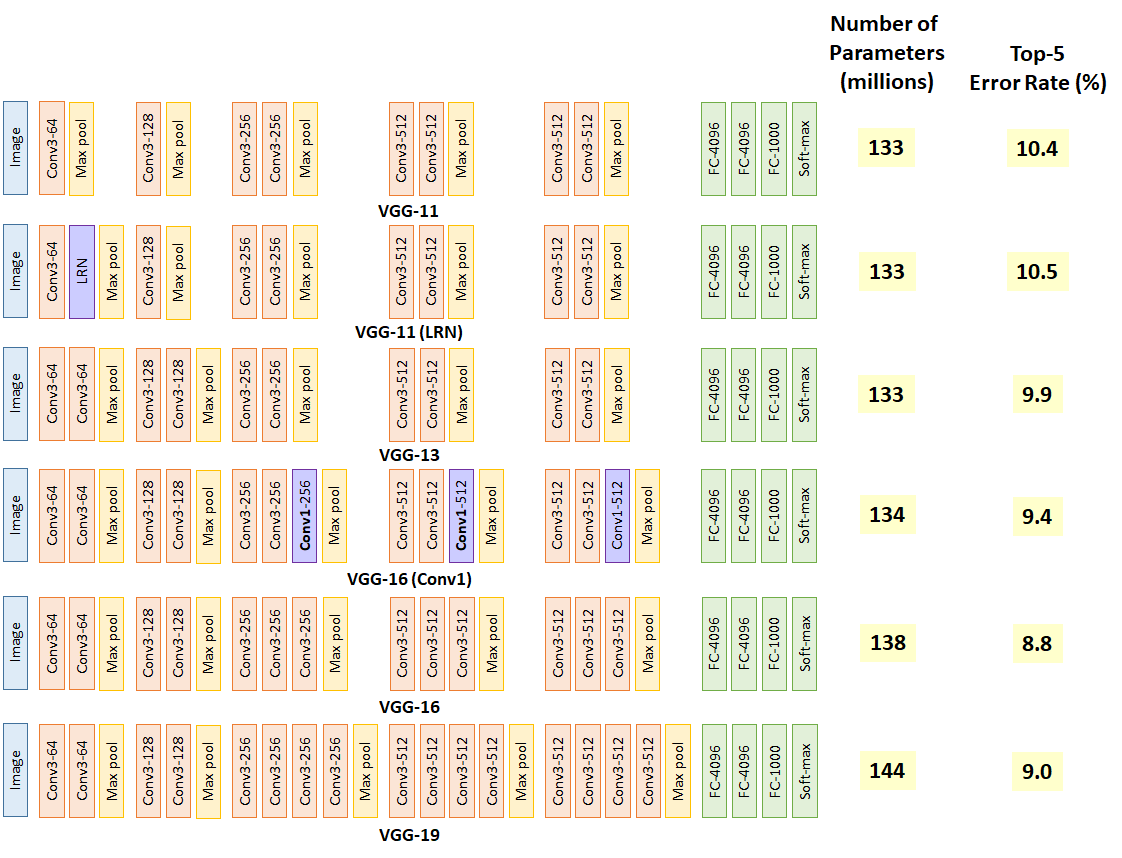
Why Earlier layers of the Network are so important to us?

Earlier layers in the Network are important because they are responsible to learn and detecting the simple patterns and are actually the building blocks of our Network.

IF Training process takes too long and the Prediction Accuracy of the Model will decrease.

Hence This is all Vanishing Gradient problem does to our Neural Network Model. Just think of a Deep Neural Network Model which is highly complicated and has millions of layers in it, how problematic it can be to train such a deep Network and produce good and accurate results.

This is the reason why we do not use Sigmoid and Tanh as Activation functions which causes vanishing Gradient Problems . Hence mostly nowadays we use RELU based activation functions in training a Deep Neural Network Model to avoid such complications and improve the accuracy .



To obtain the optimum deep learning layer structure, ablation study has been done as shown in the above figure.

* VGG-11 already obtains 10.4% error rate, which is similar to that of ZFNet in ILSVRC 2013.
* VGG-11 is set as benchmark.
* VGG-11 (LRN) obtains 10.5% error rate, is the one with additional local response normalization (LRN) operation suggested by AlexNet.
* By comparing VGG-11 and VGG-11 (LRN), the error rate doesn’t improve which means LRN is not useful.
* In fact, LRN is not used any more in later on deep learning network, instead, batch normalization (BN) is used.
* VGG-13 obtains 9.9% error rate, which means the additional conv helps the classification accuracy.
* VGG-16 (Conv1) obtains 9.4% error rate, which means the additional three 1×1 conv layers help the classification accuracy. 1×1 conv actually helps to increase non-linearlity of the decision function. Without changing the dimensions of input and output, 1×1 conv is doing the projection mapping in the same high dimensionality.
* VGG-16 obtains 8.8% error rate which means the deep learning network is still improving by adding number of layers.
* VGG-19 obtains 9.0% error rate which means the deep learning network is NOT improving by adding number of layers. Thus, authors stop adding layers.
* By observing the addition of layers one by one, we can observe that VGG-16 and VGG-19 start converging and the accuracy improvement is slowing down. When people are talking about VGGNet, they usually mention VGG-16 and VGG-19.

### **Multi-Scale Training**

If we only train the network at the same scale, we might miss the detection or have the wrong classification for the objects with other scales.

For multi-scale training, an image is scaled with smaller-size equal to a range from 256 to 512, i.e. S=[256;512], then cropped to 224×224. Therefore, with a range of S, we are inputting different scaled objects into the network for training.

**VGG-13** reduced the error rate from 9.4%/9.3% to **8.8%.**

**VGG-16** reduced the error rate from 8.8%/8.7% to **8.1%.**

**VGG-19** reduced the error rate from 9.0%/8.7% to **8.0%.**

### **Multi-Scale Testing**

Similar to multi-scale training, multi-scale testing can also reduce the error rate since we do not know the size of object in the test image. If we scale the test image to different sizes, we can increase the chance of correct classification.

VGG-13 reduced the error rate from 9.4%/9.3% to 9.2%.

VGG-16 reduced the error rate from 8.8%/8.7% to 8.6%.

VGG-19 reduced the error rate from 9.0%/8.7% to 8.7/8.6%.

By using both multi-scale training and testing, error rate is reduced.

Compared with only multi-scale testing,

VGG-13 reduced the error rate from 9.2%/9.2% to 8.2%,

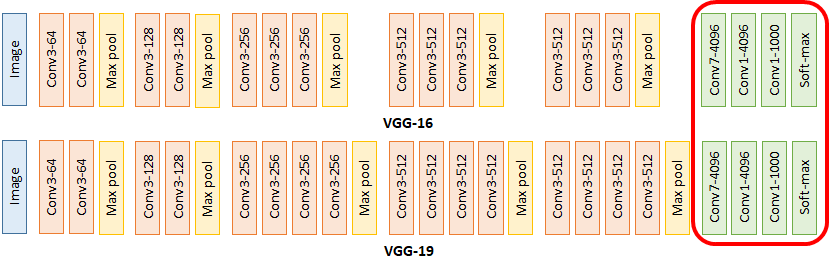
VGG-16 reduced the error rate from 8.6%/8.6% to 7.5%,

VGG-19 reduced the error rate from 8.7%/8.6% to 7.5%

### 

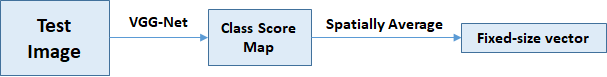
### 

### **Dense (Convolutionalized) Testing**



**VGGNet During Testing**

During testing, in **VGGNet**, the test image is directly go through the VGGNet and obtain a class score map. This class score map is spatially averaged to be a fixed-size vector.



### **ResNet(2015)**