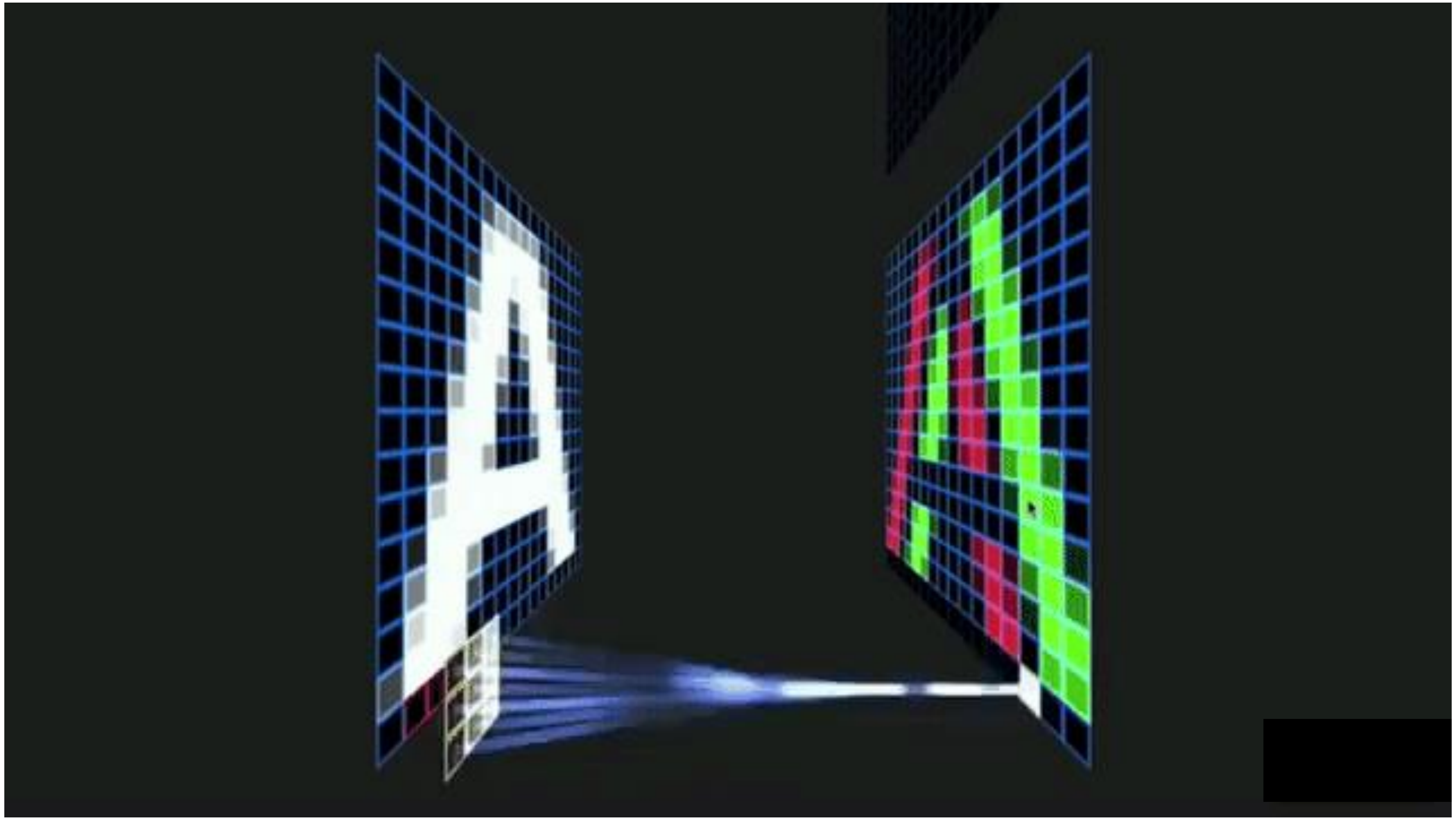
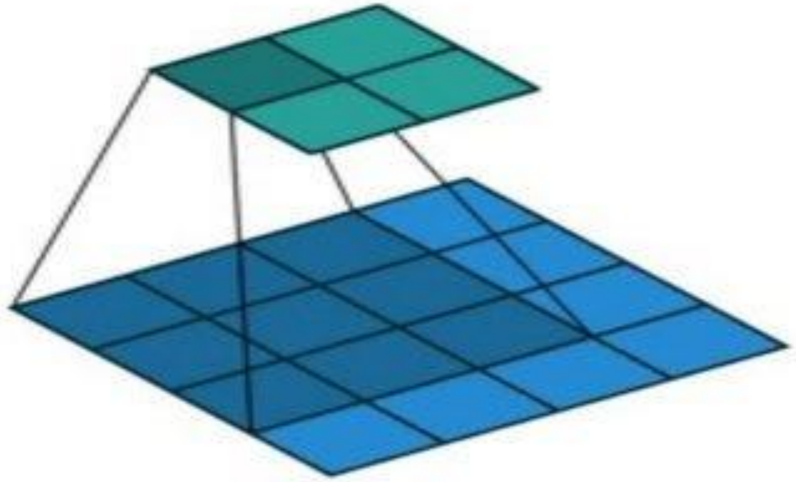
CNN Architectures



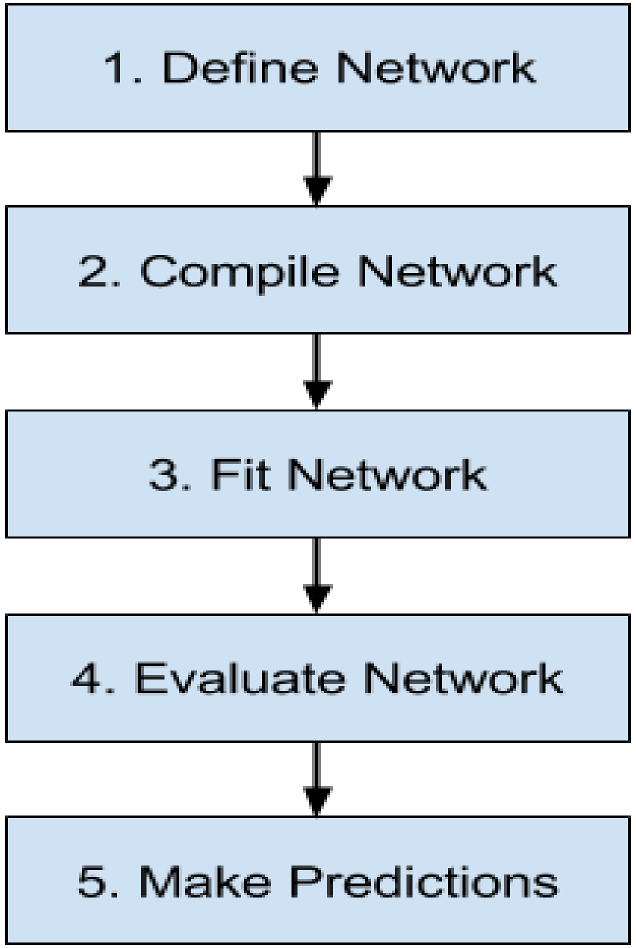
Building a Convolutional Neural Network (CNN) in Keras

- The Keras library in Python makes it pretty simple to build a CNN.
- A convolution multiplies a matrix of pixels with a filter matrix or 'kernel' and sums up the multiplication values. Then the convolution slides over to the next pixel and repeats the same process until all the image pixels have been covered.



Steps to Train and Build a CNN Model

- Import the modules
- Loading the dataset
- Exploratory data analysis
- Data pre-processing
- Building the model
- Compiling the model
- Training the model
- Using our model to make predictions



Steps to Train and Build a CNN Model

For example: The digit identification problem

- Input layer consists of (1, 8, 28) values.
- First layer, Conv2D consists of 32 filters and 'relu' activation function with kernel size, (3,3).
- Second layer, Conv2D consists of 64 filters and 'relu' activation function with kernel size, (3,3).
- Thrid layer, MaxPooling has pool size of (2, 2).
- Fifth layer, Flatten is used to flatten all its input into a single dimension.
- Sixth layer, Dense consists of 128 neurons and 'relu' activation function.
- Seventh layer, Dropout has 0.5 as its value.
- Eighth and final layer consists of 10 neurons and 'softmax' activation function.
- Use categorical_crossentropy as loss function.
- Use Adadelta() as Optimizer.
- Use accuracy as metric.
- Use 128 as batch size.
- Use 20 as epochs.

Steps to Train and Build a CNN Model

- Step 1 – Import the modules

```
import keras from keras.datasets

import mnist from keras.models

 import Sequential from keras.layers

 import Dense, Dropout, Flatten from keras.layers

import Conv2D, MaxPooling2D from keras

 import backend as K import numpy as np
```

Step 2– Load data

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

Steps to Train and Build a CNN Model

Step 3– Process the data

```
img_rows, img_cols = 28, 28
if K.image_data_format() == 'channels_first':
   x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
   x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
   input_shape = (1, img_rows, img_cols)
else:
   x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
   x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
   input_shape = (img_rows, img_cols, 1)
   x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
y_train = keras.utils.to_categorical(y_train, 10)
y_test = keras.utils.to_categorical(y_test, 10)
```

Steps to Train and Build a CNN Model

Step 4 – Create the model

```
model = Sequential()
model.add(Conv2D(32, kernel_size = (3, 3),
   activation = 'relu', input_shape = input_shape))
model.add(Conv2D(64, (3, 3), activation = 'relu'))
model.add(MaxPooling2D(pool_size = (2, 2)))
model.add(Dropout(0.25)) model.add(Flatten())
model.add(Dense(128, activation = 'relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation = 'softmax'))
```

Step 5 – Compile the model

```
model.compile(loss = keras.losses.categorical_crossentropy,
                optimizer = keras.optimizers.Adadelta(), metrics = ['accuracy'])
```

Step 6 – Train the model

```
model.fit(   x_train, y_train,    batch_size = 128,   epochs = 12,
   verbose = 1,    validation_data = (x_test, y_test) )
```

Steps to Train and Build a CNN Model

Step 7 – Evaluate the model

```
score = model.evaluate(x_test, y_test, verbose = 0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

Step 8 – Predict

```
pred = model.predict(x_test)

pred = np.argmax(pred, axis = 1)[:5]

label = np.argmax(y_test,axis = 1)[:5]

print(pred)

print(label)
```

CNN Architectures

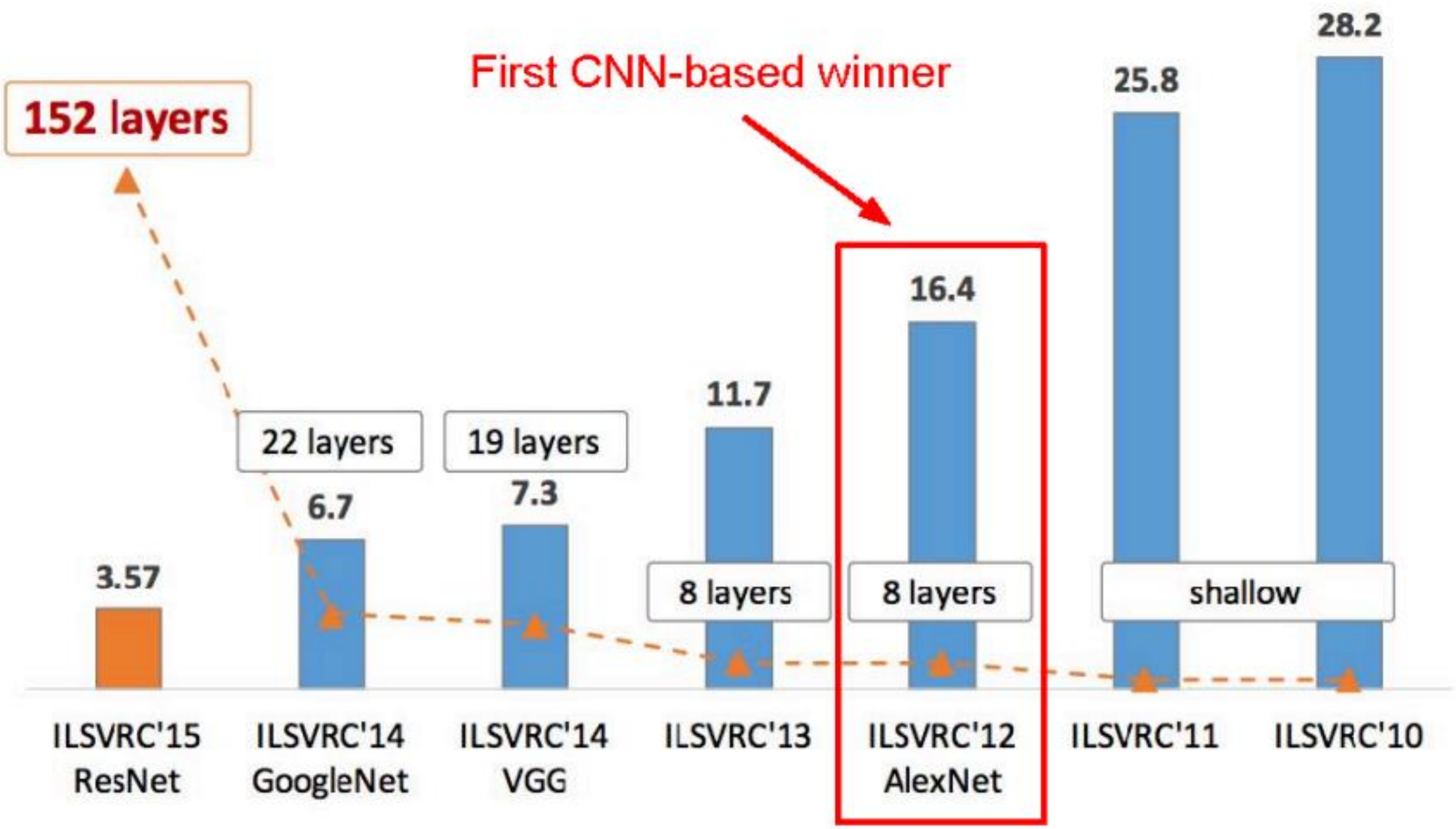
- The ImageNet project is a large visual database designed for use in visual object recognition software research.
- The ImageNet project runs an annual software contest, the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), where software programs compete to correctly classify and detect objects and scenes.

- LeNet-5 (1998)
- AlexNet(2012)
- ZefNet (2013)
- Visual geometry group (VGG) (2014)
- GoogLeNet (2014)
- Highway network (2015)

- ResNet (2015)
- DenseNet (2017)
- ResNext (2016 Runnerup)
- WideResNet (2016)
- Pyramidal Net (2017)
- Xception (2017)

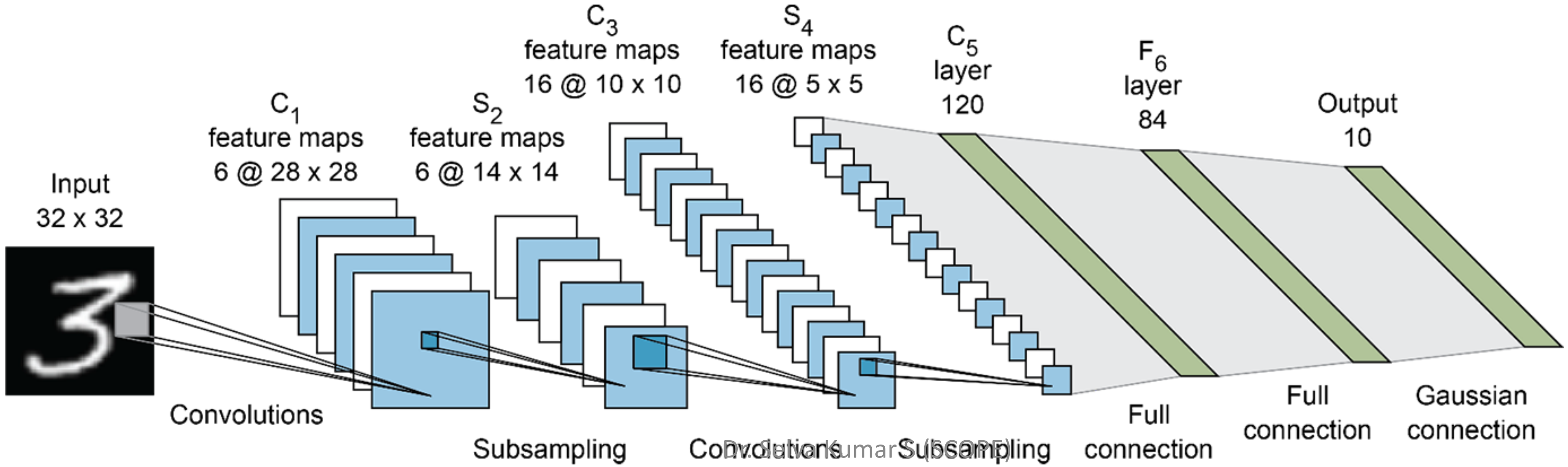
2023-
BASIC-L (Lion, fine-tuned)

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) Winners



LeNet-5 (1998)

- LeNet-5, a pioneering 7-level convolutional network by LeCun et al in 1998, that classifies digits, was applied by several banks to recognize hand-written numbers on checks (cheques) digitized in 32x32 pixel greyscale input images.
- The ability to process higher-resolution images requires larger and more convolutional layers, so this technique is constrained by the availability of computing resources.



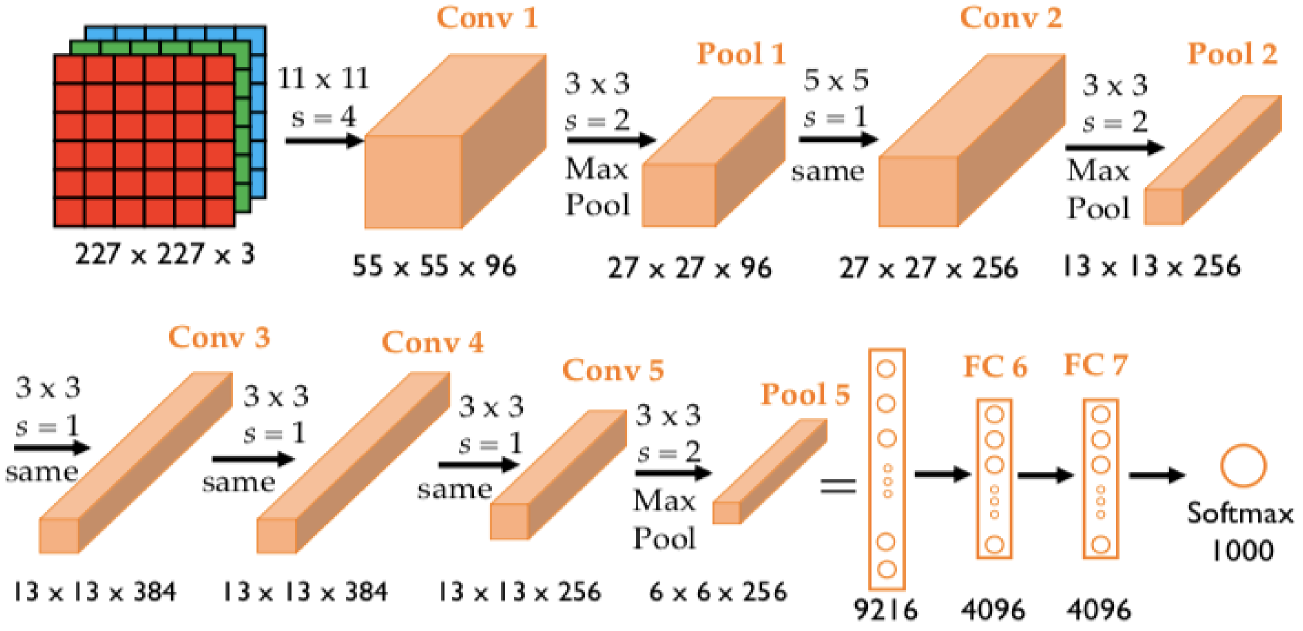
AlexNet (2012)

- AlexNet is one of the most popular neural network architectures to date.
- It was proposed by Alex Krizhevsky.
- Te learning ability of the deep CNN was limited at this time due to hardware restrictions.
- To overcome these hardware limitations, two GPUs (NVIDIA GTX 580) were used in parallel to train AlexNet.
- The network had a very similar architecture as LeNet but deeper, with more filters per layer, and with stacked convolutional layers.
- It consisted 11x11, 5x5,3x3, convolutions, max pooling, dropout, data augmentation, ReLU activations, SGD with momentum.

AlexNet (2012) Cont'd

Details/Retrospectives

- The first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- 7 CNN ensemble



- Input: 227x227x3 images (224x224 before padding)
- First layer: 96 11x11 filters applied at stride 4
- Output volume size?

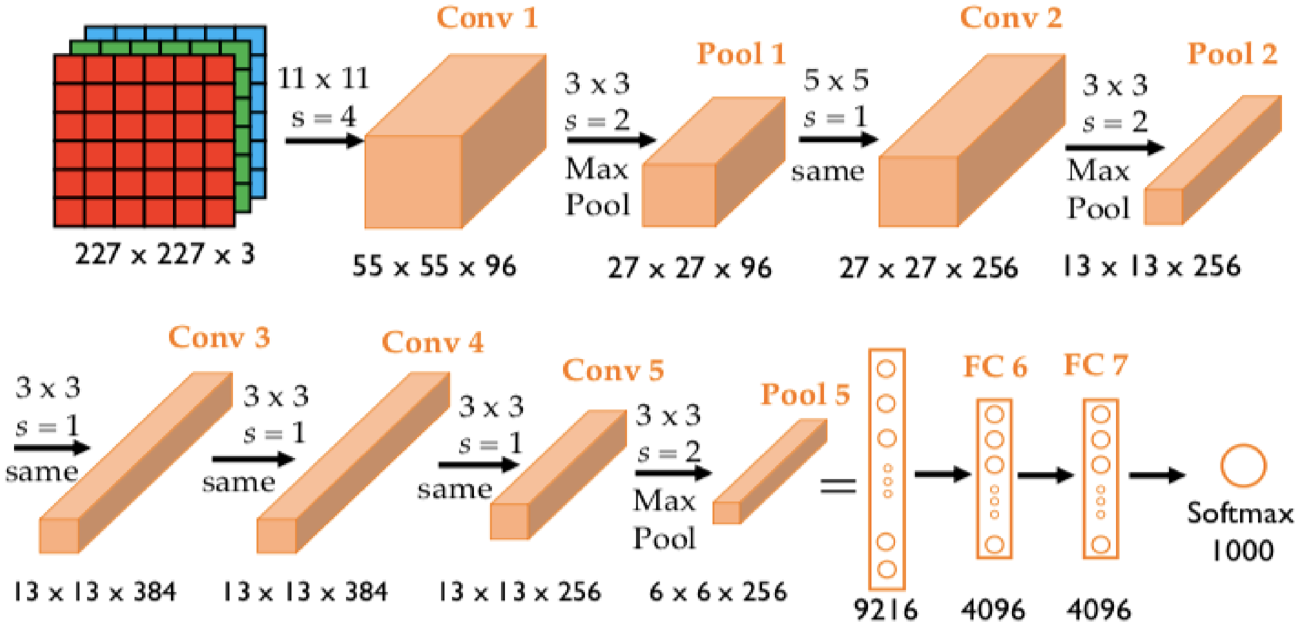
  (N-F)/s+1 = (227-11)/4+1 = 55 -> [55x55x96]
- Number of parameters in this layer?

  (11*11*3)*96 = 35K

AlexNet (2012) Cont'd

Details/Retrospectives

- The first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- 7 CNN ensemble



- Input: 227x227x3 images (224x224 before padding)
- After CONV1: 55x55x96
- Second layer: 3x3 filters applied at stride 2
- Output volume size?
- (N-F)/s+1 = (55-3)/2+1 = 27 -> [27x27x96]
- Number of parameters in this layer?
- 0!

AlexNet (2012) Cont'd

- Trained on GTX 580 GPU with only 3 GB of memory.
- Network spread across 2 GPUs, half the neurons (feature maps) on each GPU.
- CONV1, CONV2,CONV4,CONV5:

  - Connections only with feature maps on the same GPU.
- CONV3, FC6,FC7,FC8:

  - Connections with all feature maps in the preceding layer, communication across GPUs.

AlexNet was the coming out party for CNNs in the computer vision community. This was the first time a model performed so well on a historically difficult ImageNet dataset.

Visual Geometry Group (VGG) Net

- The VGG architecture is the basis of object recognition models.
- Different variants of VGG exists based on the number of layers.
- VGG-16 or VGG-19 consisting of 16 and 19 convolutional layers respectively.
- VGGNet with16 layers is called as VGG16, which is a CNN model proposed by A. Zisserman and K. Simonyan from the University of Oxford.
- It replaces the large sized kernels with several 3×3 kernels one after the other. It provides significant improvements over AlexNet.
- VGG19 model (VGGNet-19) is the same as the VGG16 except that it supports 19 layers. The "16" and "19" stands for the number of weight layers (convolutional layers and FC Layers) in the model.
- VGG19 has three more convolutional layers than VGG16.

Visual Geometry Group (VGG) Net

Input:

- The VGGNet receives an image as an input size of 224×224.
- The designers of the model cropped out the center 224×224 patch in each image to keep the size of the input image consistent.

Convolutional Layers:

- VGG's convolutional layers leverage a minimal receptive field, i.e., 3×3, the smallest possible size that still captures up/down and left/right.
- Moreover, there are also 1×1 convolution filters acting as a linear transformation of the input. This is followed by a ReLU (rectified linear unit), which is a huge innovation from AlexNet that reduces training time.

Visual Geometry Group (VGG) Net

Convolutional Layers Cont'd

- ReLU is a piecewise linear function that will output the input if positive; otherwise, the output is zero.
- The convolution stride is fixed at 1 pixel to keep the spatial resolution preserved after convolution (stride is the number of pixel shifts over the input matrix).

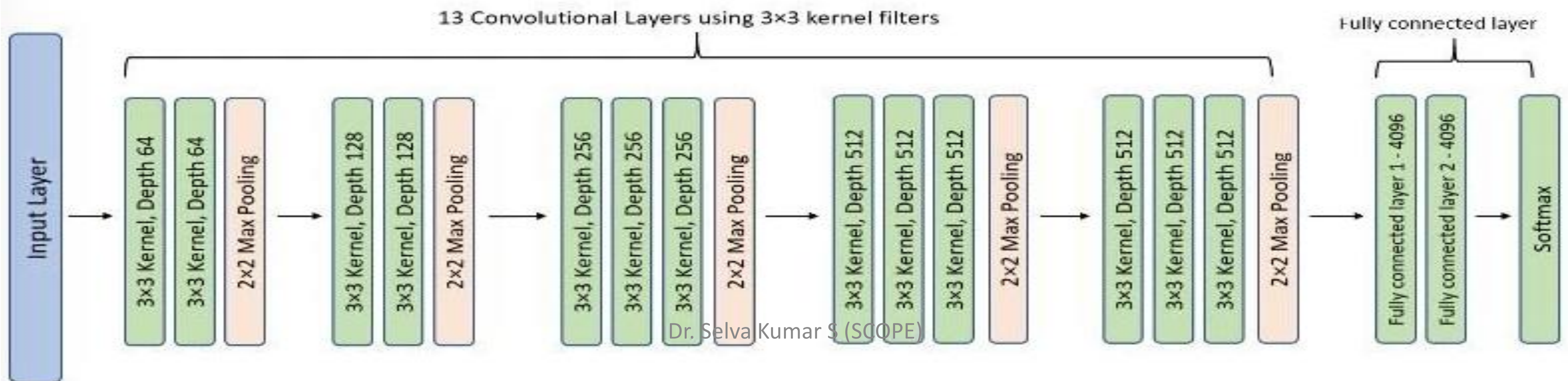
Hidden Layers:

- All the hidden layers in the VGG network use ReLU. VGG does not usually leverage Local Response Normalization (LRN) as it increases memory consumption and training time.
- Moreover, it makes no improvements to overall accuracy.

Visual Geometry Group (VGG) Net

Fully-Connected Layers:

- The VGGNet has **three** fully connected layers.
- Out of the three layers, the first two have 4096 channels each, and the third has 1000 channels, 1 for each class in ILSVRC classification.
- VGG16 is a pretty extensive network and has a total of around 138 million parameters.
- There are a few convolution layers followed by a pooling layer that reduces the height and the width.



VGG16 (Visual Geometry Group) Net Architecture

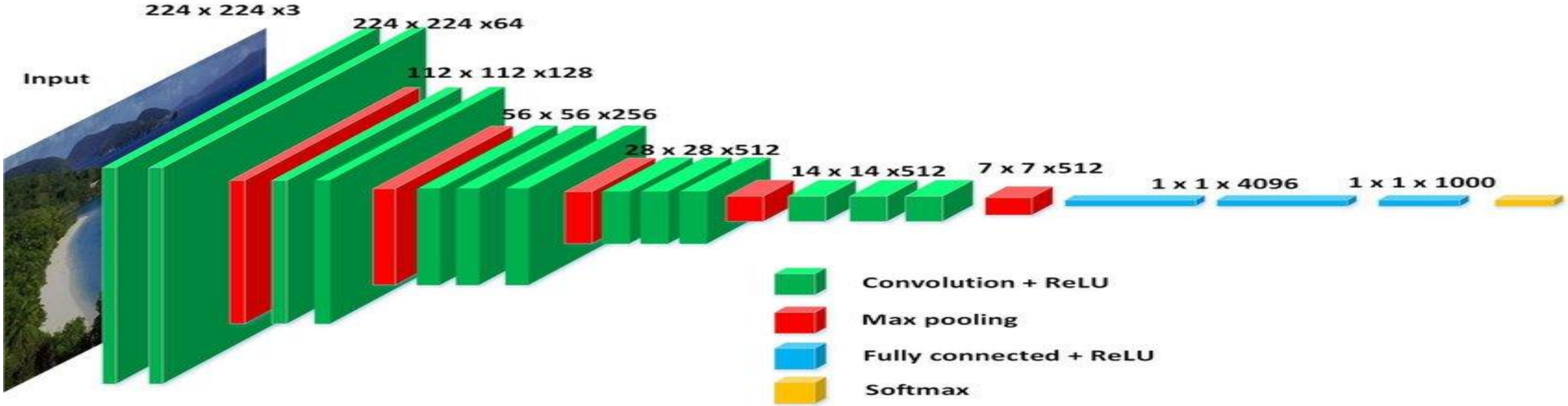
Filters

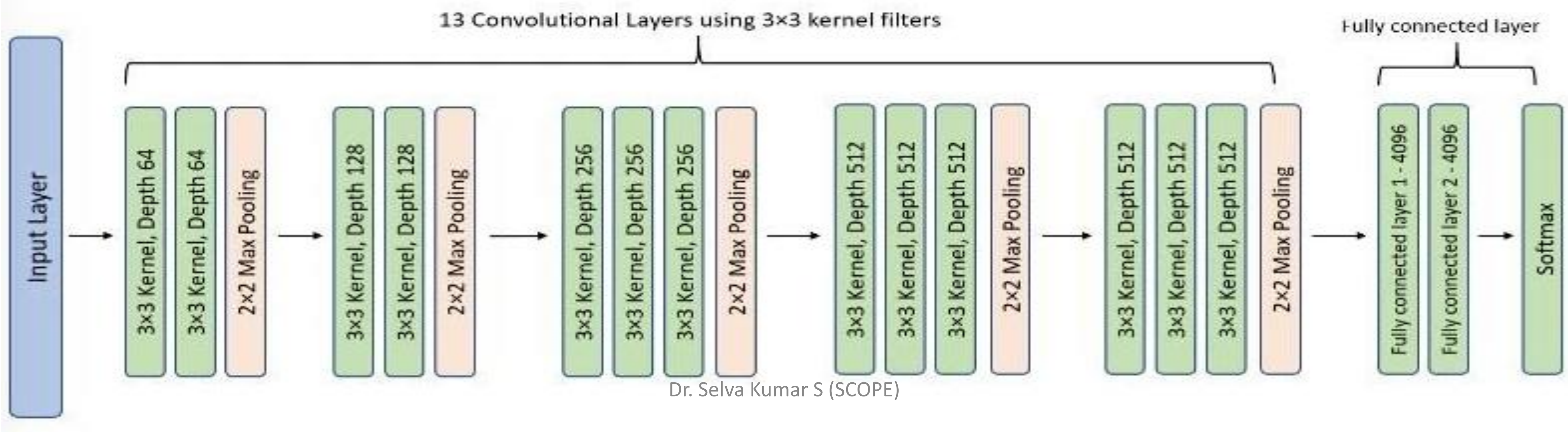
- Number of filters that we can use, around 64 filters are available that we can double to about 128 and then to 256 filters. In the last layers, we can use 512 filters.

Complexity and challenges

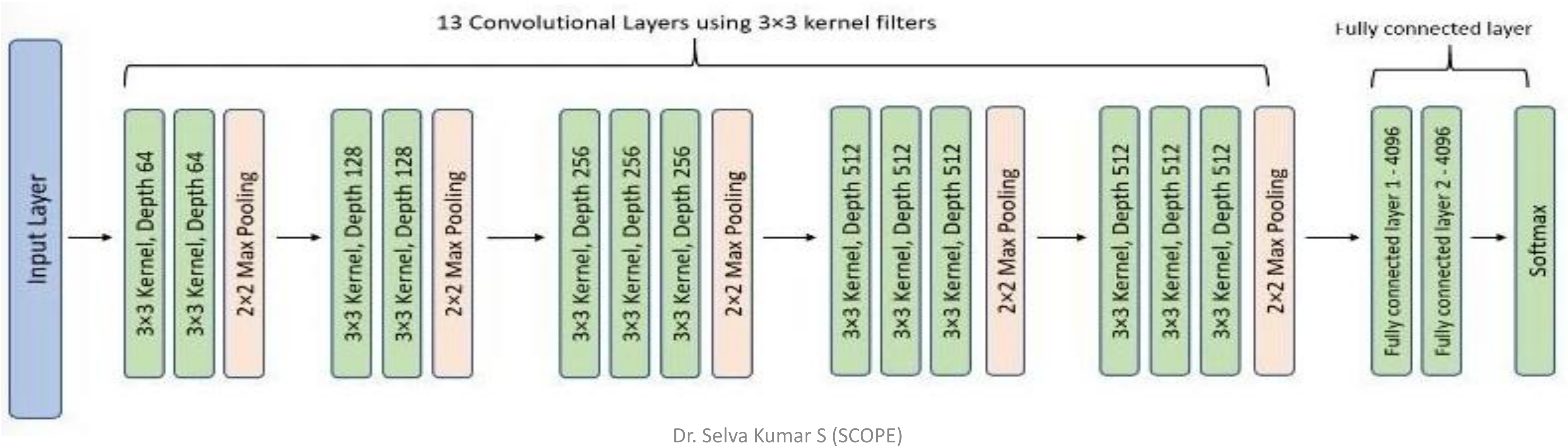
- The number of filters that we can use doubles on every step or through every stack of the convolution layer. This is a major principle used to design the architecture of the VGG16 network.
- One of the crucial downsides of the VGG16 network is that it is a huge network, which means that it takes more time to train its parameters. Because of its depth and number of fully connected layers, the VGG16 model is more than 533MB.
- This makes implementing a VGG network a time-consuming task.
- The VGG16 model is larger network architecture than GoogLeNet and SqueezeNet.

Visual Geometry Group (VGG) Net





Visual Geometry Group (VGG) Net



GoogLeNet : Going deeper with Convolutions

- Google proposed a deep Convolution Neural Network named Inception that achieved top results for classification and detection in ILSVRC 2014.
- "Going deeper with convolutions" is actually inspired by an internet meme: 'We need to go deeper'.

- Also significantly deeper than AlexNet
- x12 less parameters than AlexNet
- Focused on computational efficiency

GoogLeNet : Going deeper with Convolutions Cont'd

Problems Inception v1 is trying to solve

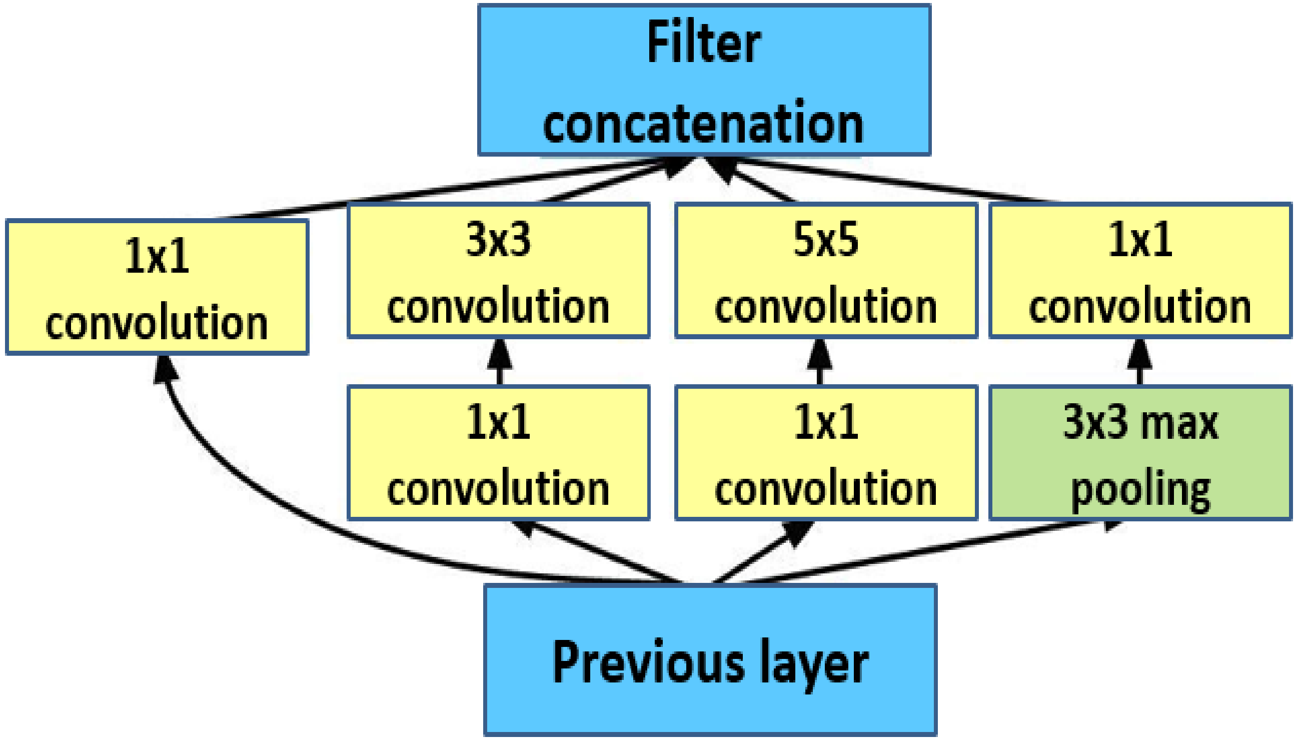
- The important parts in the image can have large variations in size.
- For instance, the image of an object can be in various positions and some pictures are zoomed in and others may get zoomed out.
- Because of such variation in images, choosing the right kernel size for performing convolution operations becomes very difficult.
- We require a larger kernel to extract information of an object that is distributed more in the picture and a smaller kernel is preferred to extract information of an image that is distributed less in the picture.

- One of the major approaches to increasing the performance of neural networks is by increasing its size. This includes increasing its depth and also its size.
- Bigger size of neural networks corresponds to a larger number of parameters, which makes the network more prone to overfitting, especially when labeled training examples are limited.
- Another drawback of increased network size is the increased use of computational resources.

GoogLeNet : Going Deeper With Convolutions Cont'd

Inception Module

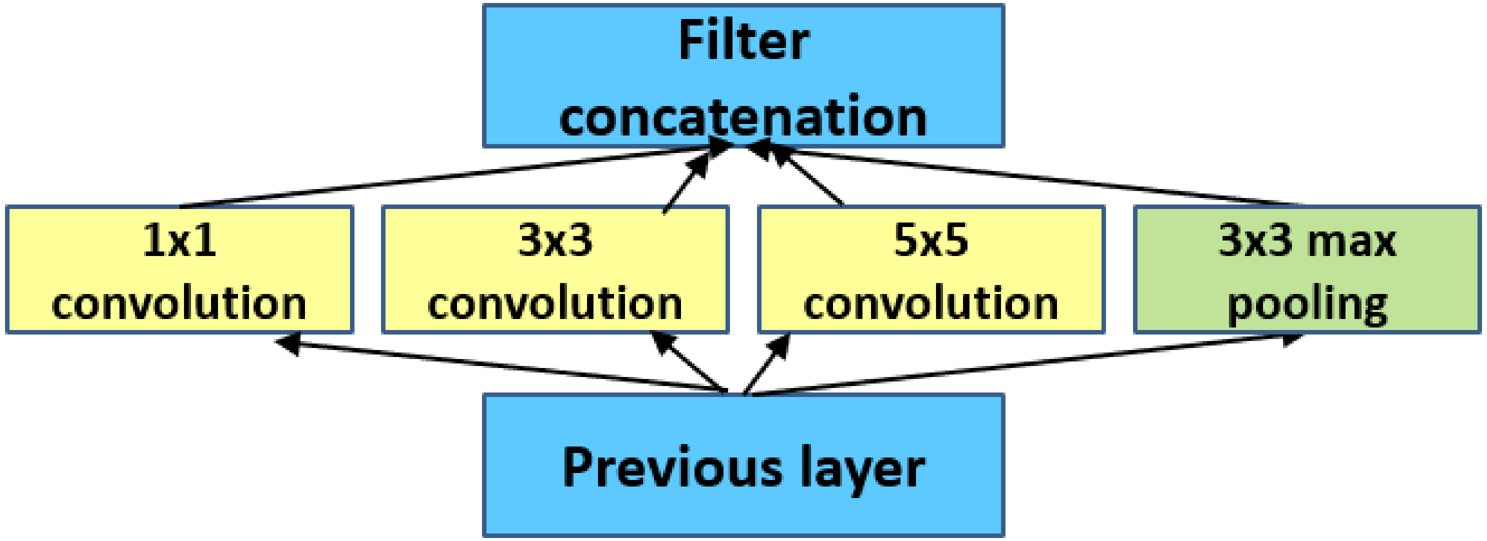
- A 'wider' network rather than 'deeper'
- Design a good local network topology (network within a network) and then stack these modules on top of each other.



GoogLeNet : Going Deeper With Convolutions Cont'd

Naïve Inception Model

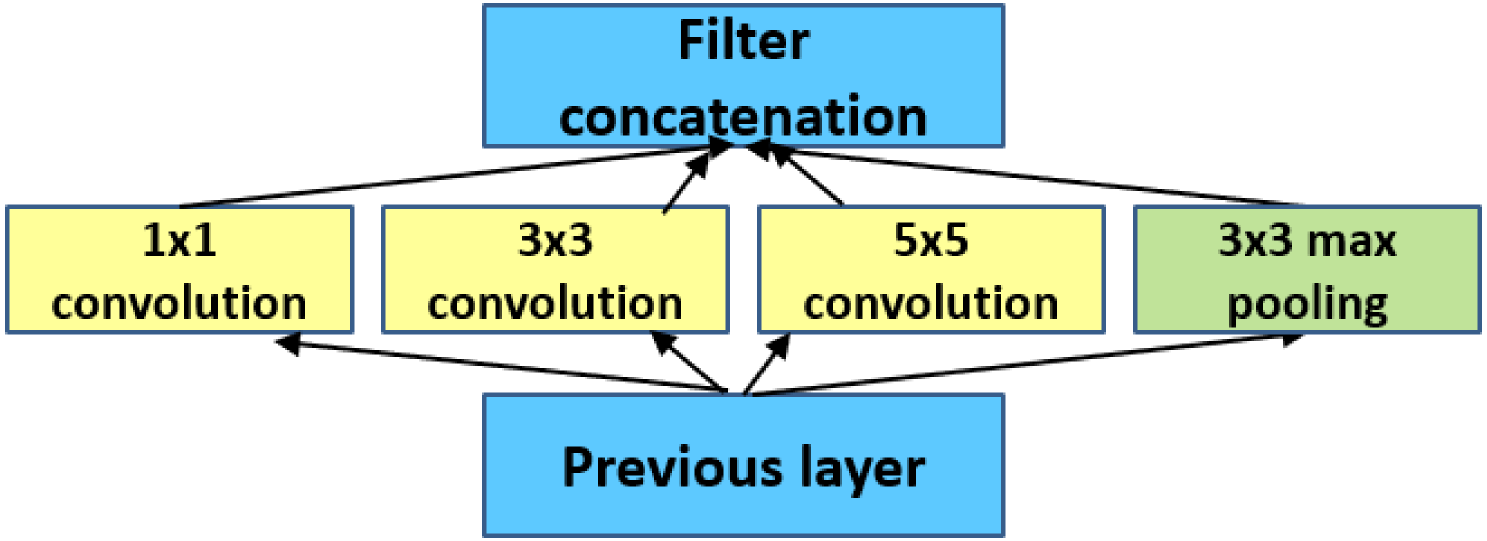
- Apply parallel filter operations on the input :
  - Multiple receptive field sizes for convolution (1x1, 3x3, 5x5)
  - Pooling operation (3x3)
- Concatenate all filter outputs together depth-wise



GoogLeNet : Going Deeper With Convolutions Cont'd

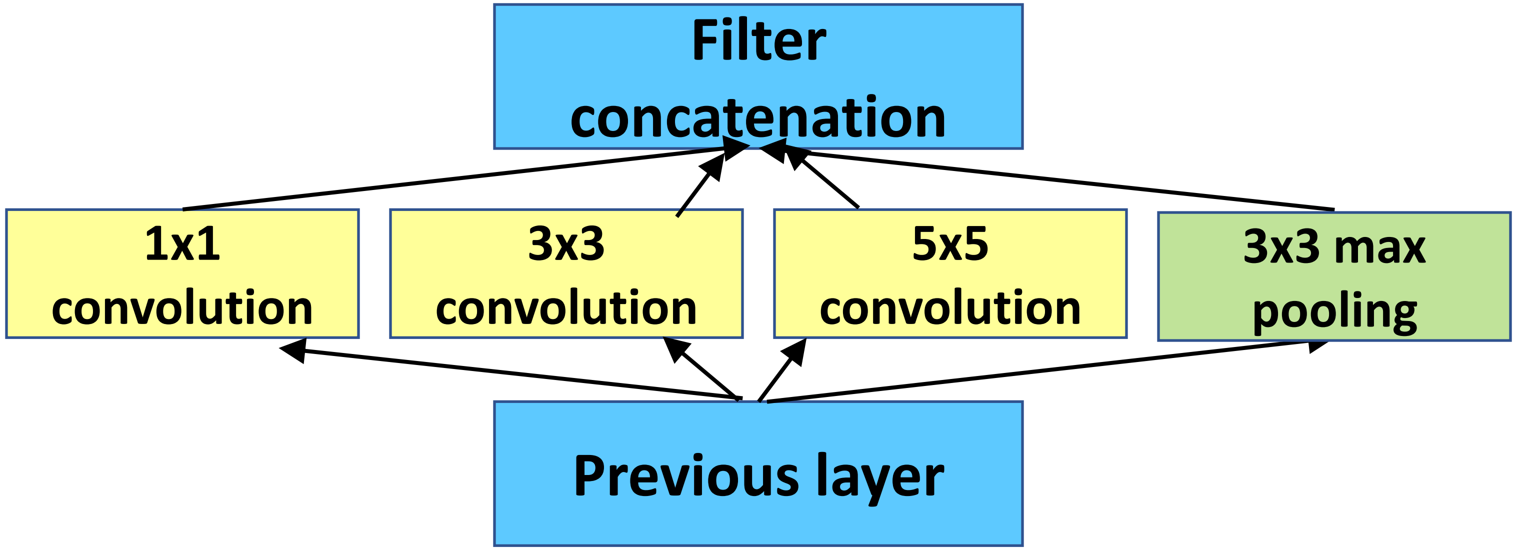
Naïve Inception Model

- Apply parallel filter operations on the input :
  - Multiple receptive field sizes for convolution (1x1, 3x3, 5x5)
  - Pooling operation (3x3)
- Concatenate all filter outputs together depth-wise



GoogLeNet : Going Deeper With Convolutions Cont'd

- **What's the problem with this?**
  High computational complexity



GoogleNet

- **Output volume sizes:**

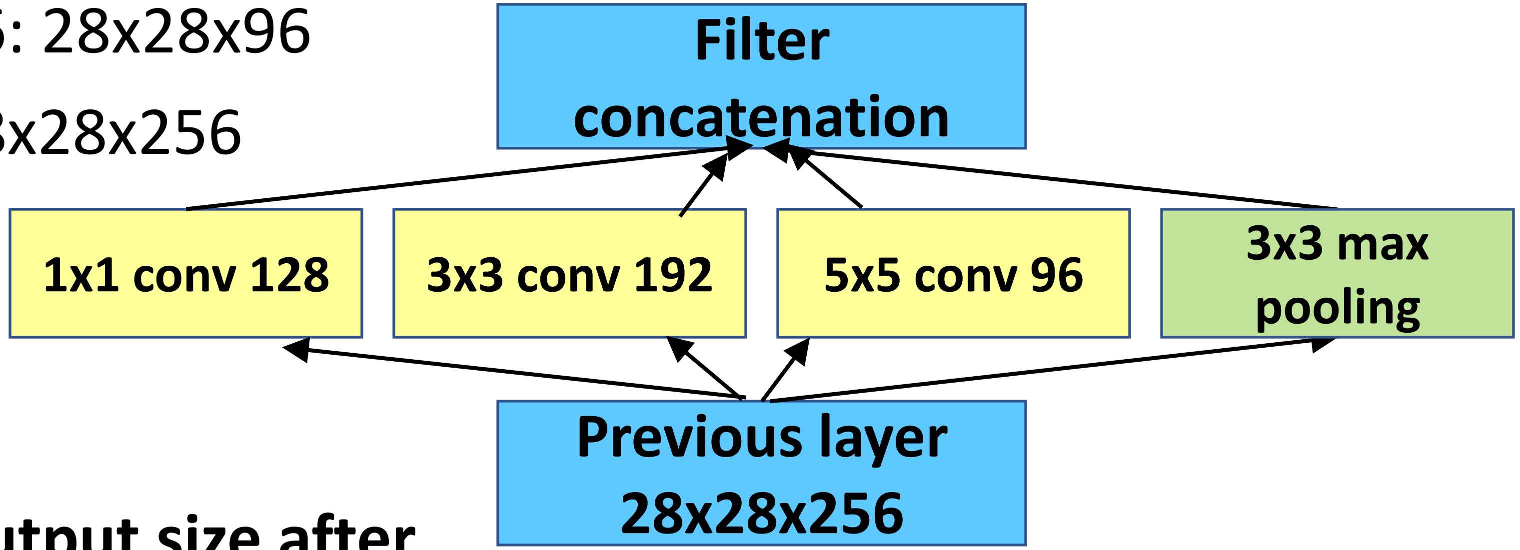
1x1 conv, 128: 28x28x128

3x3 conv, 192: 28x28x192

5x5 conv, 96: 28x28x96

3x3 pool: 28x28x256

Example:



- **What is output size after filter concatenation?**

28x28x(128+192+96+256) = 28x28x672

[Szegedy et al., 2014]

GoogLeNet : Going Deeper With Convolutions Cont'd

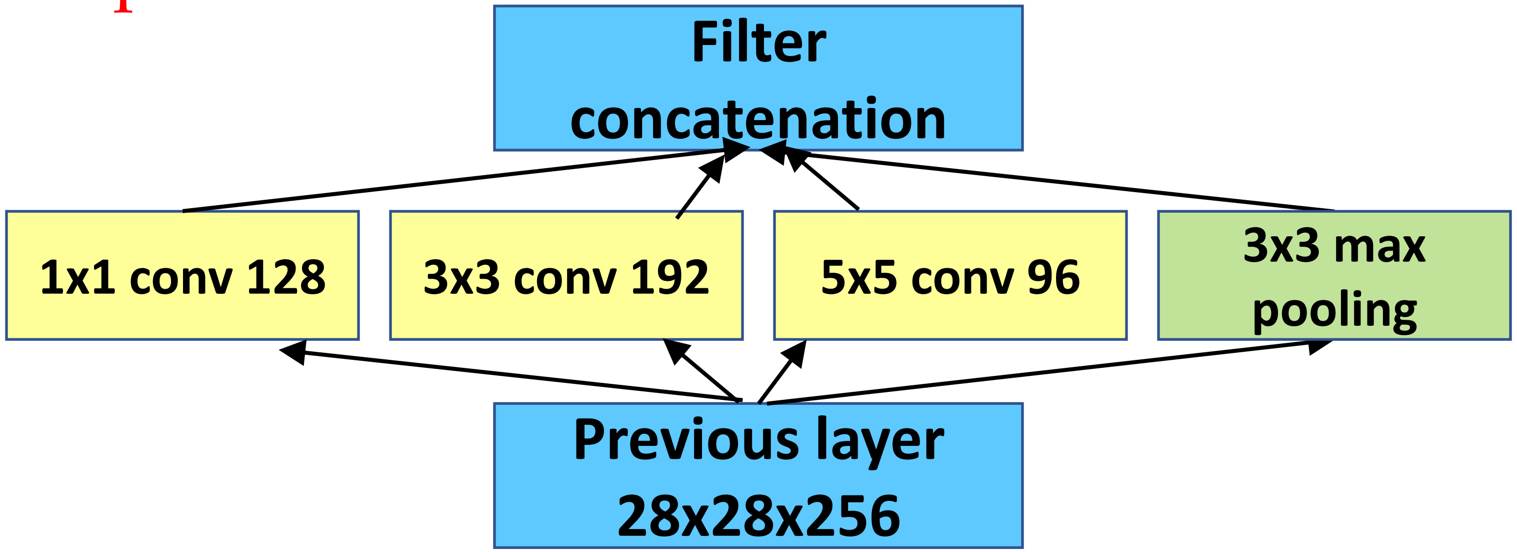
- **Number of convolution operations:**

1x1 conv, 128: 28x28x128x1x1x256

3x3 conv, 192: 28x28x192x3x3x256

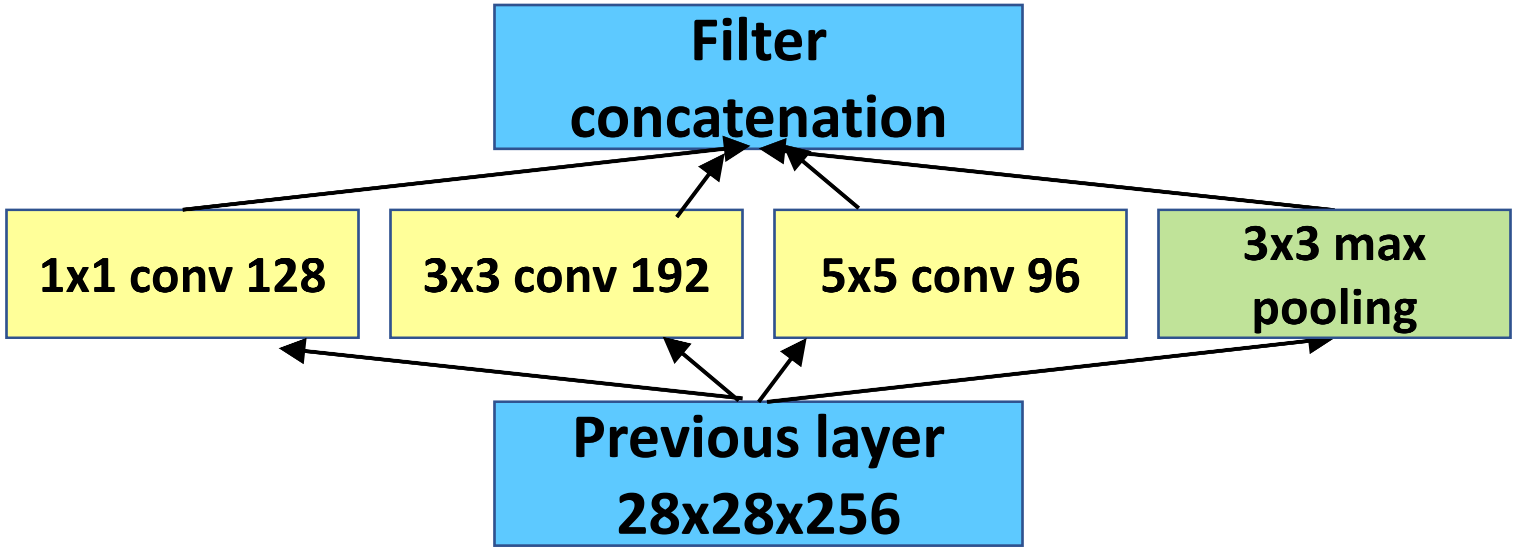
5x5 conv, 96: 28x28x96x5x5x256

Total: 854M ops



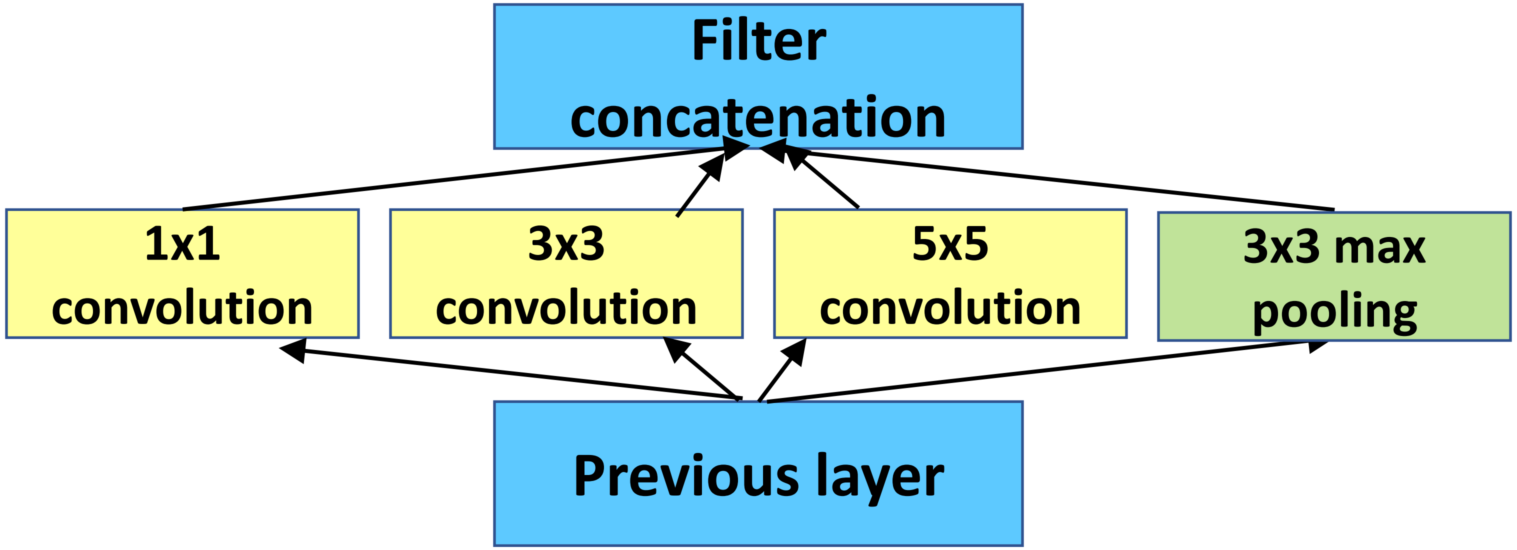
GoogLeNet : Going Deeper With Convolutions Cont'd

- Very expensive compute!
- Pooling layer also preserves feature depth, which means total depth after concatenation can only grow at every layer.



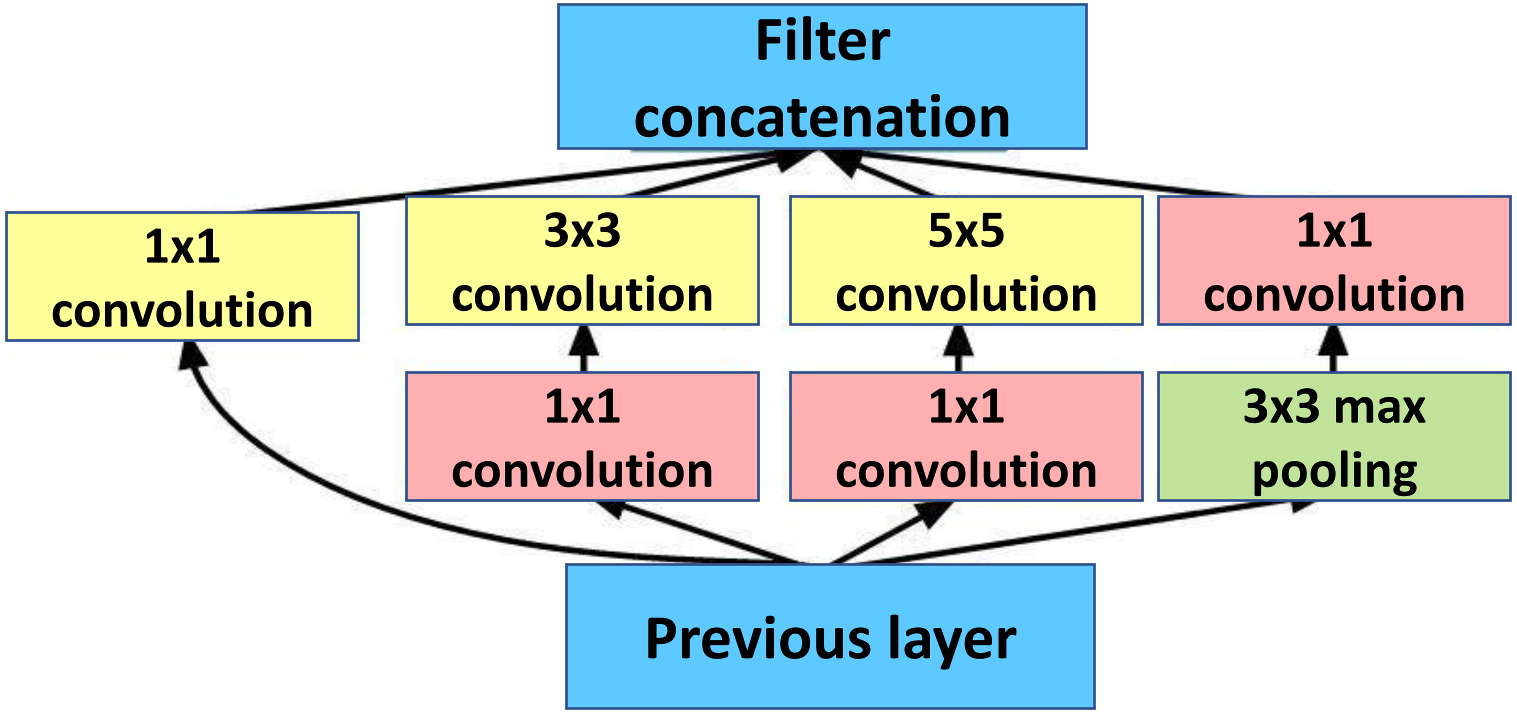
GoogLeNet : Going Deeper With Convolutions Cont’d

- **Solution:** “bottleneck” layers that use 1x1 convolutions to reduce feature depth (from previous hour).



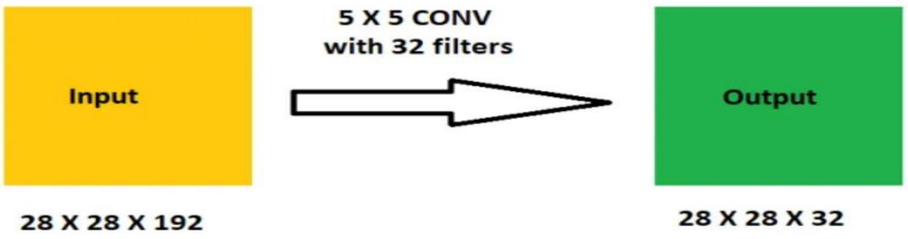
GoogLeNet : Going Deeper With Convolutions Cont'd

Solution: "bottleneck" layers that use 1x1 convolutions to reduce feature depth (from previous hour).

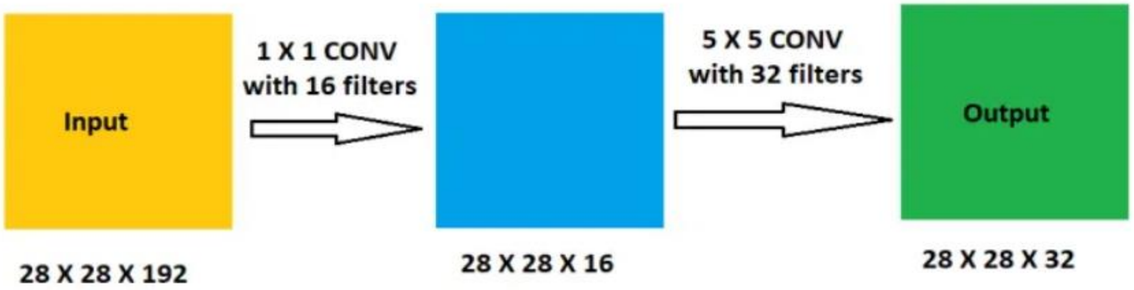


GoogLeNet : Going Deeper With Convolutions Cont'd

- Dimensionality reductions by adding 1×1 convolutions before 3×3 and 5×5 convolutions.



- Computation for the convolution operation is:
- $(5^2)(192)(32)(28^2)$ = 120,422,400 operations



Number of operations for above convolution becomes

$(1^2)(192)(16)(28^2)$ = 2,408,448 operations for the 1 × 1 convolution and,

$(5^2)(16)(32)(28^2)$ = 10,035,200 operations for the 5 × 5 convolution.

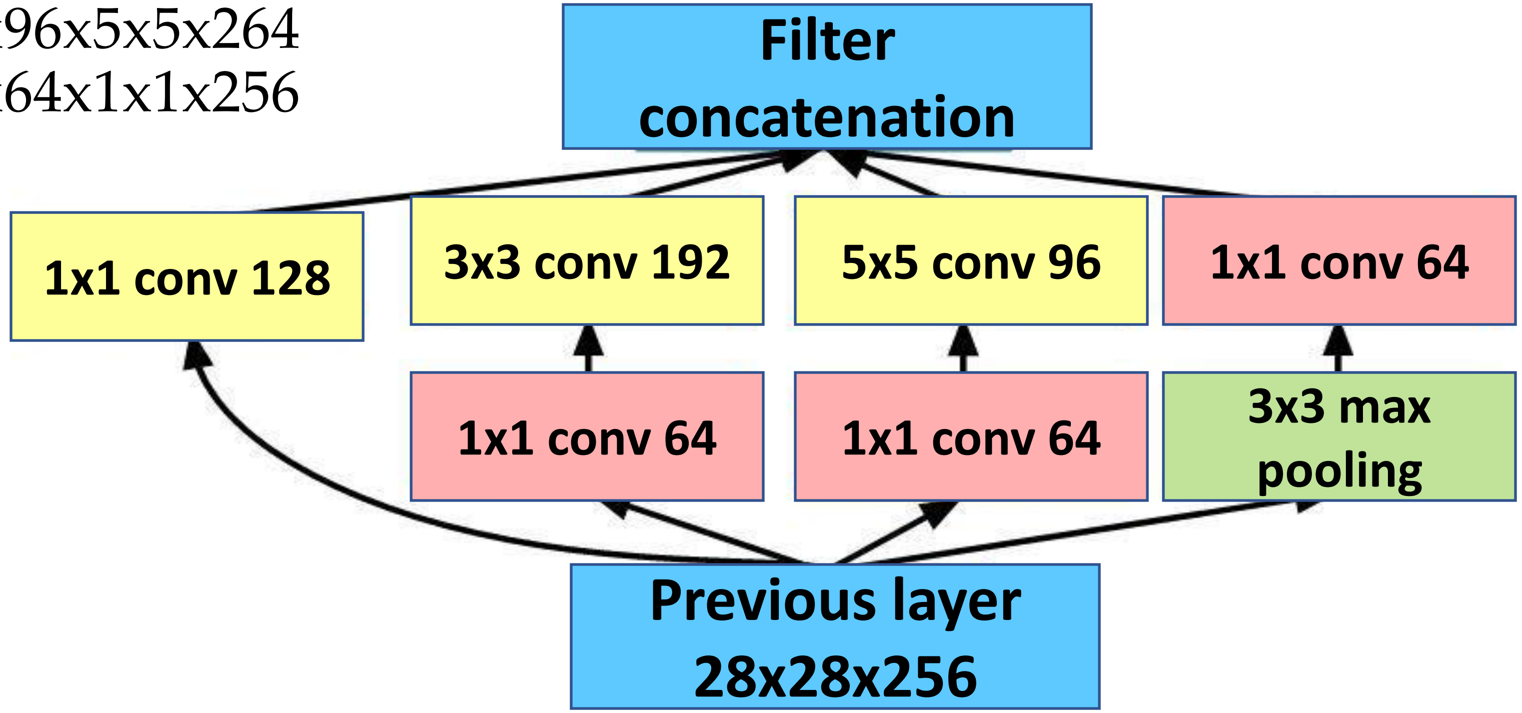
In total, there will be 2,408,448 + 10,035,200 = 12,443,648 operations.

There is a large amount of reduction in computation.

GoogLeNet : Going Deeper With Convolutions Cont'd

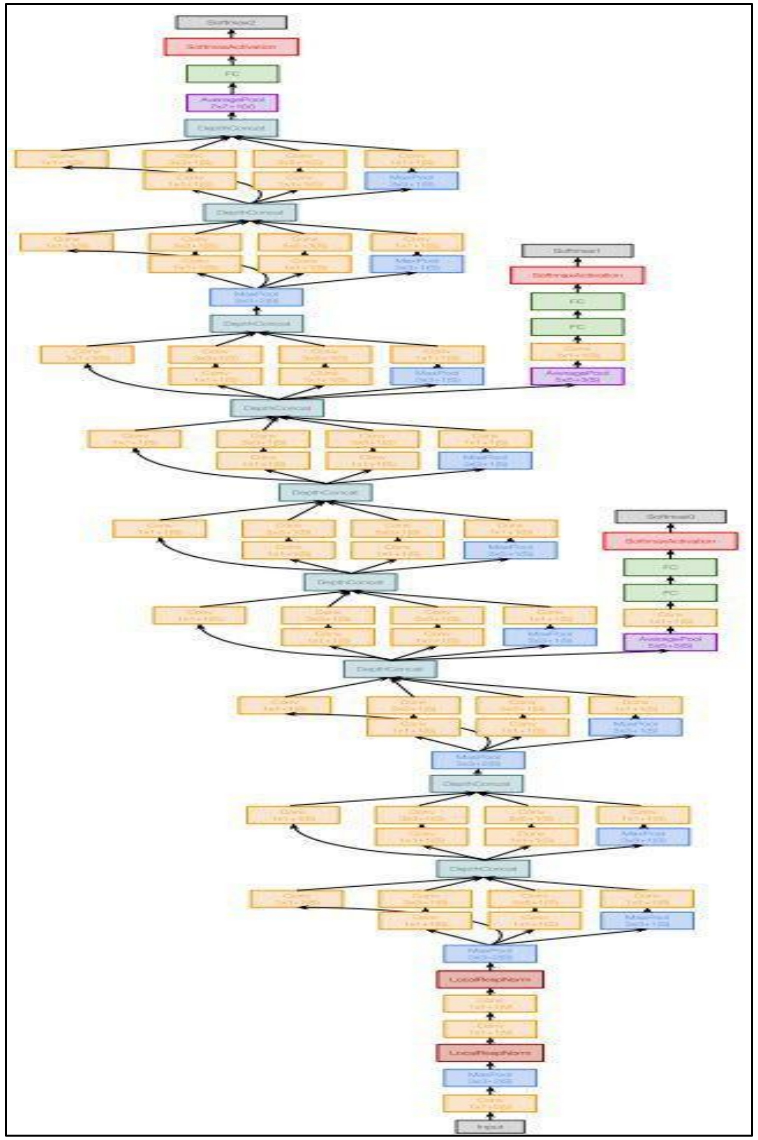
- **Number of convolution operations:**
  1x1 conv, 64: 28x28x64x1x1x256
  1x1 conv, 64: 28x28x64x1x1x256
  1x1 conv, 128: 28x28x128x1x1x256
  3x3 conv, 192: 28x28x192x3x3x64
  5x5 conv, 96: 28x28x96x5x5x264
  1x1 conv, 64: 28x28x64x1x1x256
  Total: 353M ops

After applying dimensionality reduction, our inception module becomes:



- Compared to 854M ops for naive version

GoogLeNet : Going Deeper With Convolutions Cont’d



Details/Retrospectives :

- Deeper networks, with computational efficiency
- 22 layers
- Efficient “Inception” module
- No FC layers
- 12x less params than AlexNet
- ILSVRC’14 classification winner (6.7% top 5 error)

GoogLeNet : Going Deeper With Convolutions Cont'd

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

GoogLeNet : Going Deeper With Convolutions Cont'd

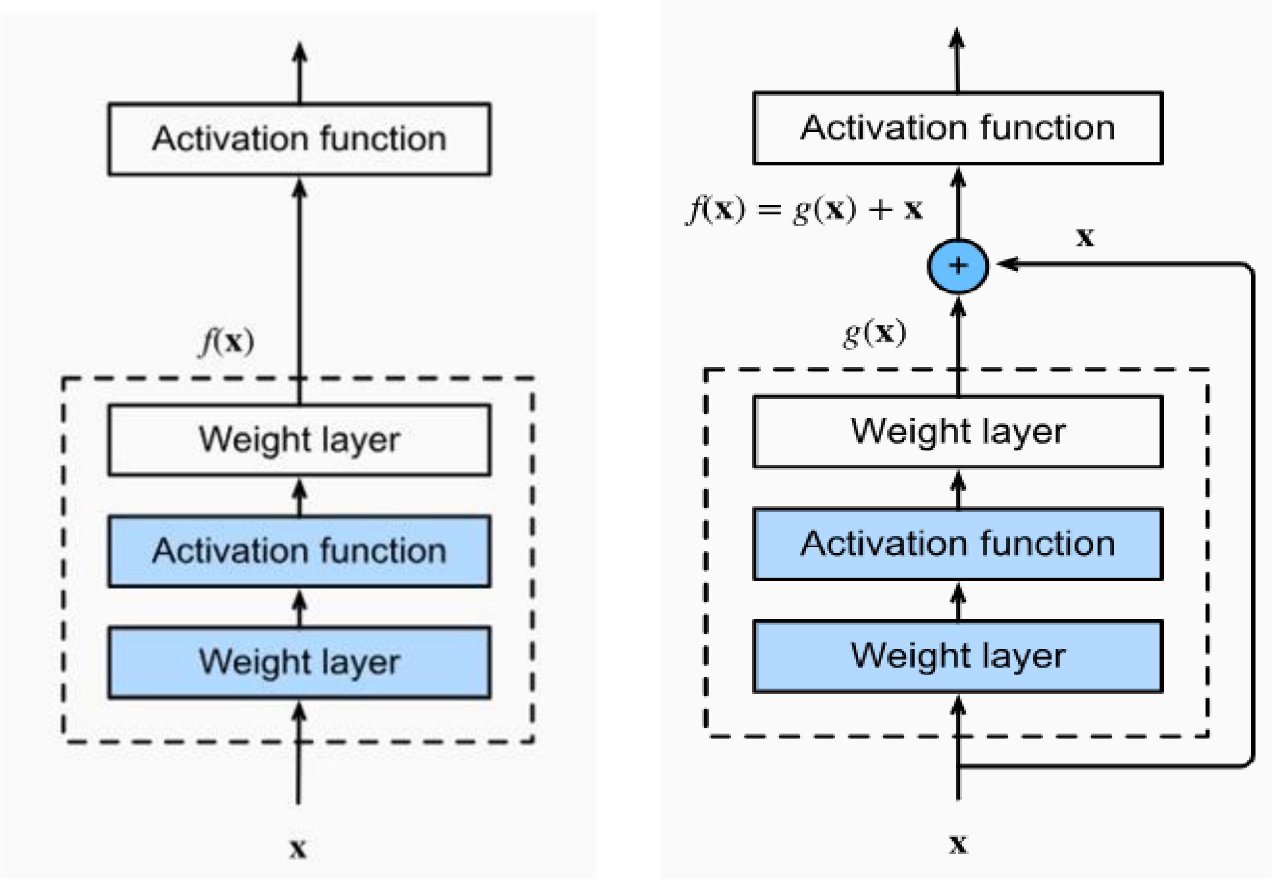
- Introduced the idea that CNN layers **didn't always have to be stacked up sequentially**.
- Coming up with the Inception module, the authors showed that a creative structuring of layers can lead to improved performance and **computationally efficiency**.

ResNet (Residual Network)

- Residual connections were introduced in 2015 by Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun.
- This architecture introduced the concept called Residual Blocks. That solves the problem of the vanishing/exploding gradient.
- In this network, use a technique called skip connections.
- The skip connection connects activations of a layer to further layers by skipping some layers in between. This forms a residual block. Results are made by stacking these residual blocks together.
- The approach behind this network is instead of layers learning the underlying mapping, to allow the network to fit the residual mapping. So, instead of say H(x), initial mapping, let the network fit.

ResNet (Residual Network)

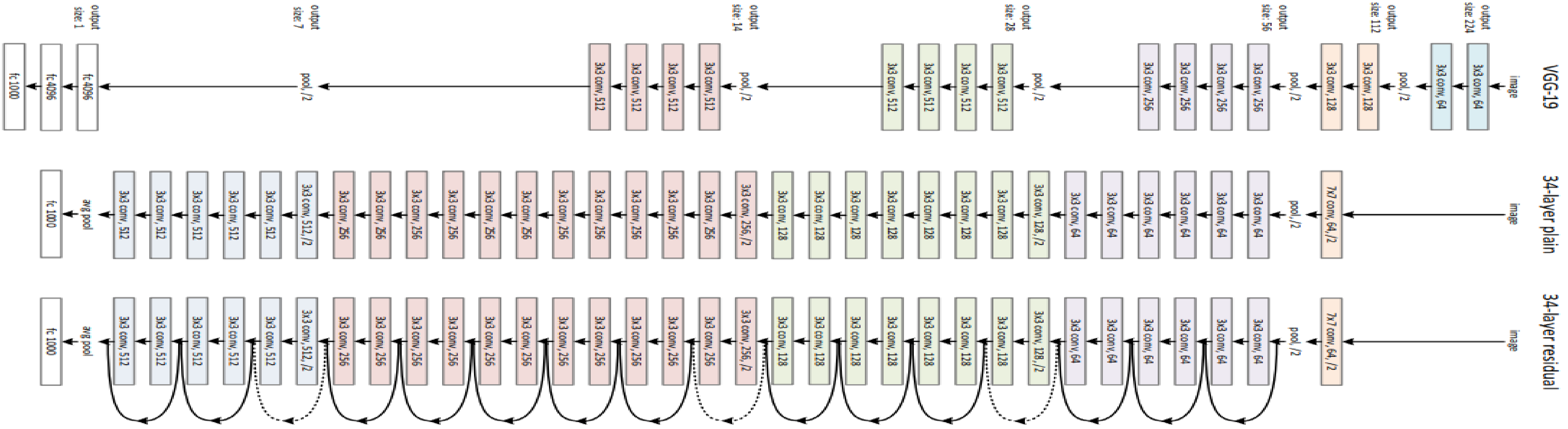
$F(x) := H(x) - x$ which gives $H(x) := F(x) + x$.



The advantage of adding this type of skip connection is that if any layer hurt the performance of architecture then it will be skipped by regularization

ResNet (Residual Network)

- Network Architecture: This network uses a 34-layer plain network architecture inspired by VGG-19 in which then the shortcut connection is added. These shortcut connections then convert the architecture into a residual network.



ResNet (Residual Network)

- Even though ResNet is much deeper than VGG16 and VGG19, the model size is actually substantially smaller due to the usage of global average pooling rather than fully-connected layers — this reduces the model size down to 102MB for ResNet50.

Variants of ResNets

ResNet-18
ResNet-50
ResNet-101
ResNet-152
ResNet-1000

Summary

Comparison					
Network	Year	Salient Feature	top5 accuracy	Parameters	FLOP
AlexNet	2012	Deeper	84.70%	62M	1.5B
VGGNet	2014	Fixed-size kernels	92.30%	138M	19.6B
Inception	2014	Wider - Parallel kernels	93.30%	6.4M	2B
ResNet-152	2015	Shortcut connections	95.51%	60.3M	11B