Famous CNN Architectures and CNN Visualization

Giacomo Boracchi

giacomo.boracchi@polimi.it

Teaching Rooms have changed on 8-9/11

Wednesday 8 November: 16:15/18:15 aula 2.0.2

Thursday 9 November: 14:15/16:15 aula Rogers

A few popular architectures

AlexNet

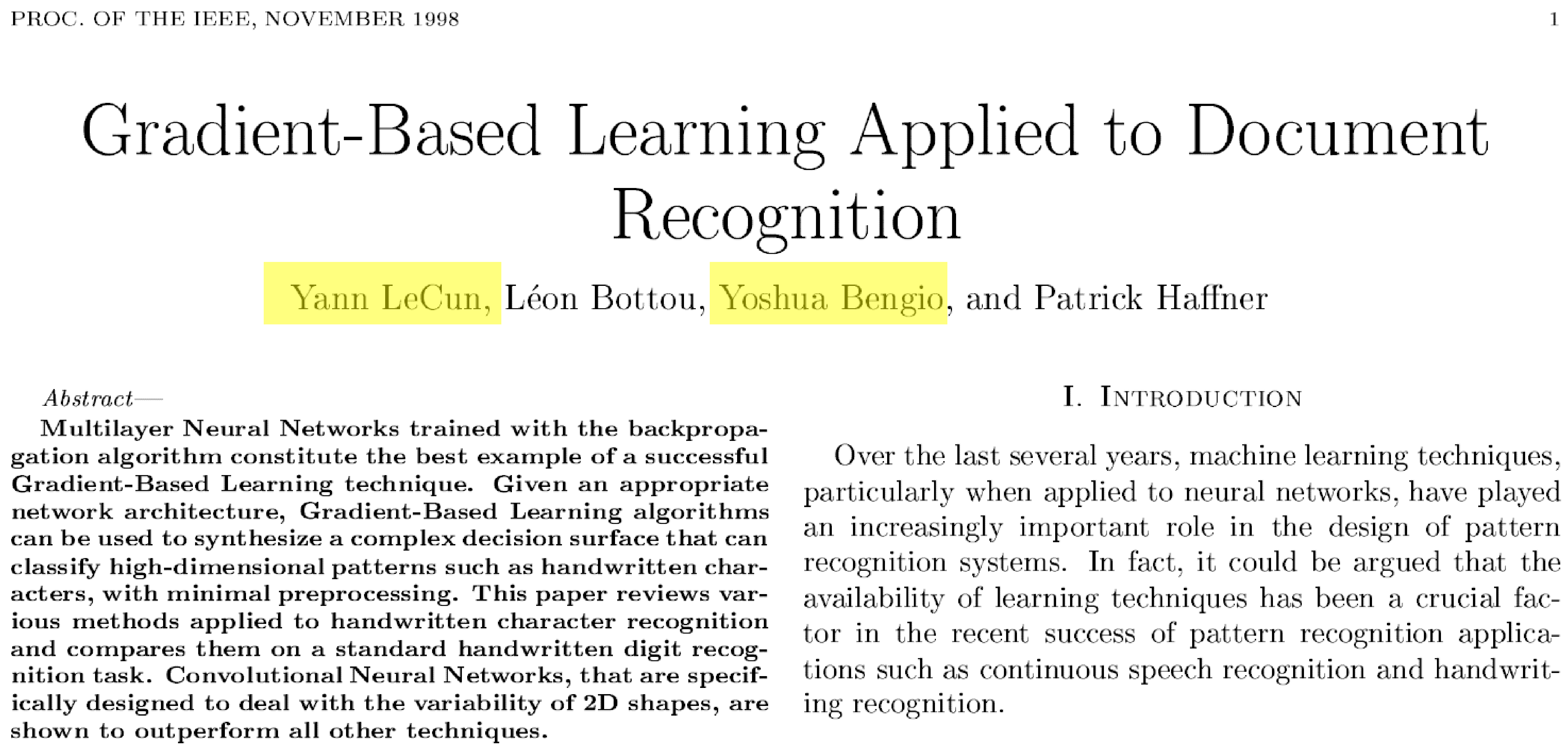
VGG

Networks In Networks (and GAP)

Inception

Resnet

The First CNN



PROC. OF THE IEEE, NOVEMBER 1998 1

Gradient-Based Learning Applied to Document Recognition

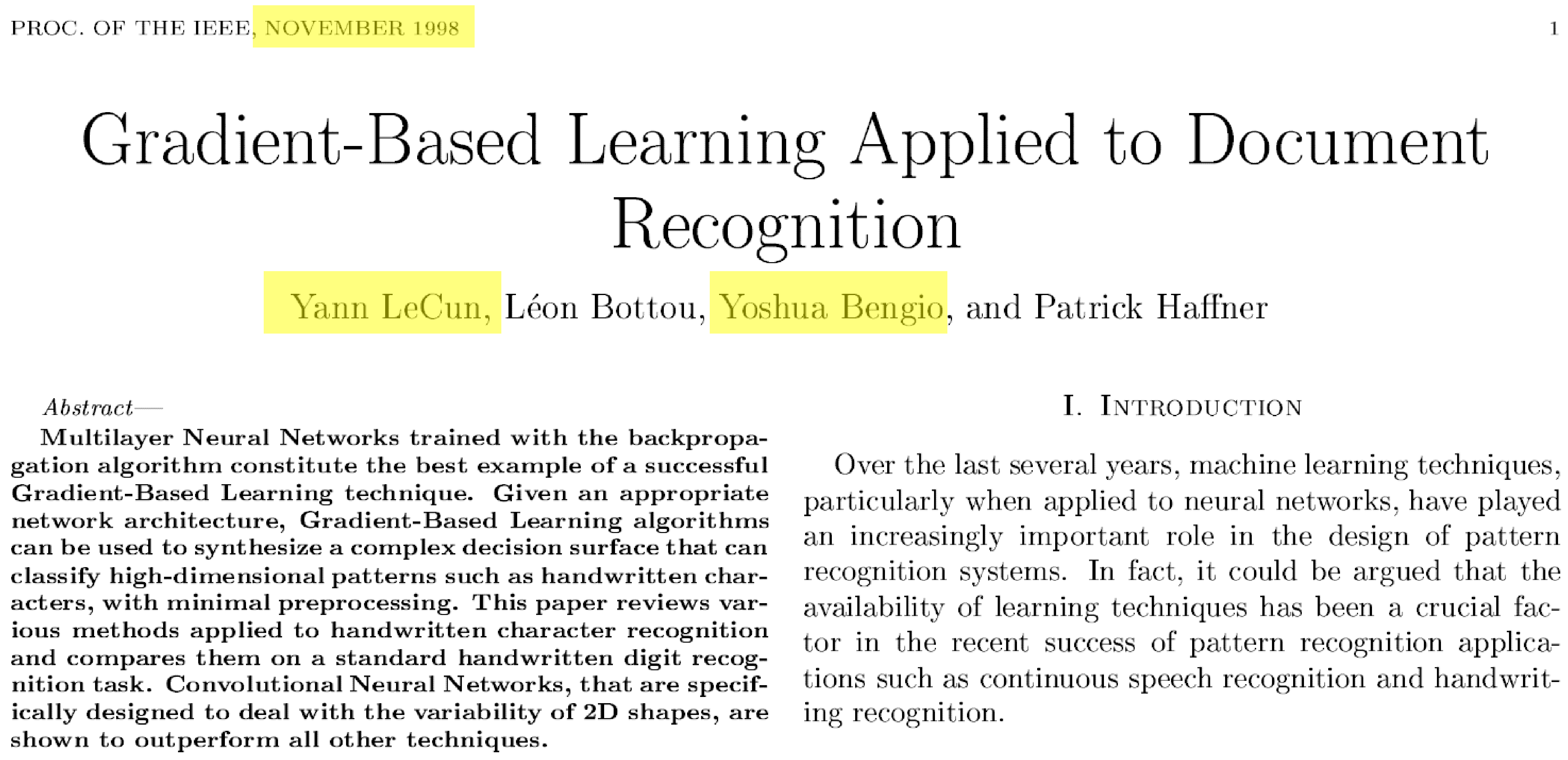
Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner

Abstract—**Multilayer Neural Networks trained with the backpropagation algorithm constitute the best example of a successful Gradient-Based Learning technique. Given an appropriate network architecture, Gradient-Based Learning algorithms can be used to synthesize a complex decision surface that can classify high-dimensional patterns such as handwritten characters, with minimal preprocessing. This paper reviews various methods applied to handwritten character recognition and compares them on a standard handwritten digit recognition task. Convolutional Neural Networks, that are specifically designed to deal with the variability of 2D shapes, are shown to outperform all other techniques.**

I. Introduction

Over the last several years, machine learning techniques, particularly when applied to neural networks, have played an increasingly important role in the design of pattern recognition systems. In fact, it could be argued that the availability of learning techniques has been a crucial factor in the recent success of pattern recognition applications such as continuous speech recognition and handwriting recognition.

LeCun, Yann, et al. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE* 86.11 (1998)


PROC. OF THE IEEE, NOVEMBER 1998 1

Gradient-Based Learning Applied to Document Recognition

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner

Abstract—

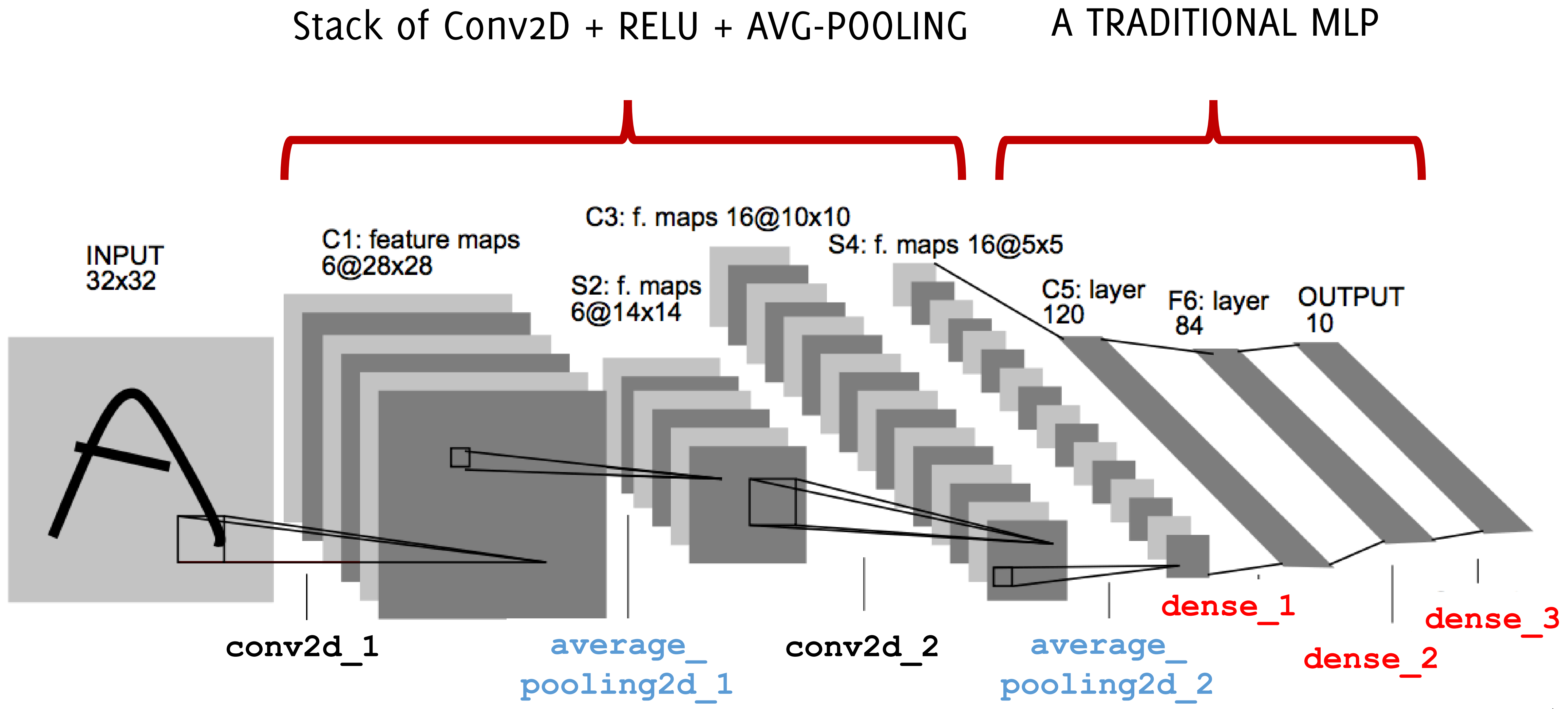
Multilayer Neural Networks trained with the backpropagation algorithm constitute the best example of a successful Gradient-Based Learning technique. Given an appropriate network architecture, Gradient-Based Learning algorithms can be used to synthesize a complex decision surface that can classify high-dimensional patterns such as handwritten characters, with minimal preprocessing. This paper reviews various methods applied to handwritten character recognition and compares them on a standard handwritten digit recognition task. Convolutional Neural Networks, that are specifically designed to deal with the variability of 2D shapes, are shown to outperform all other techniques.

I. Introduction

Over the last several years, machine learning techniques, particularly when applied to neural networks, have played an increasingly important role in the design of pattern recognition systems. In fact, it could be argued that the availability of learning techniques has been a crucial factor in the recent success of pattern recognition applications such as continuous speech recognition and handwriting recognition.

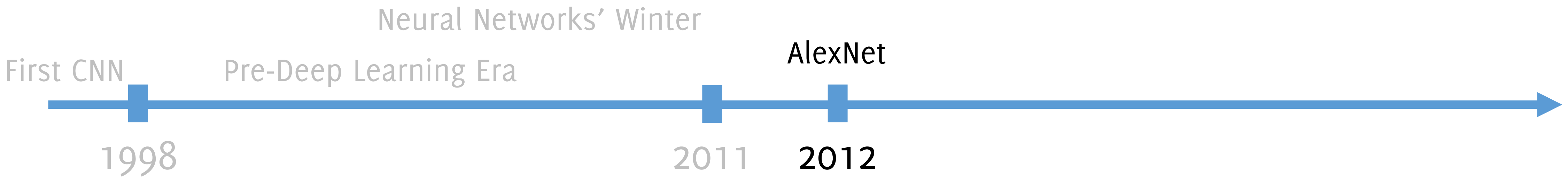
LeCun, Yann, et al. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE* 86.11 (1998)

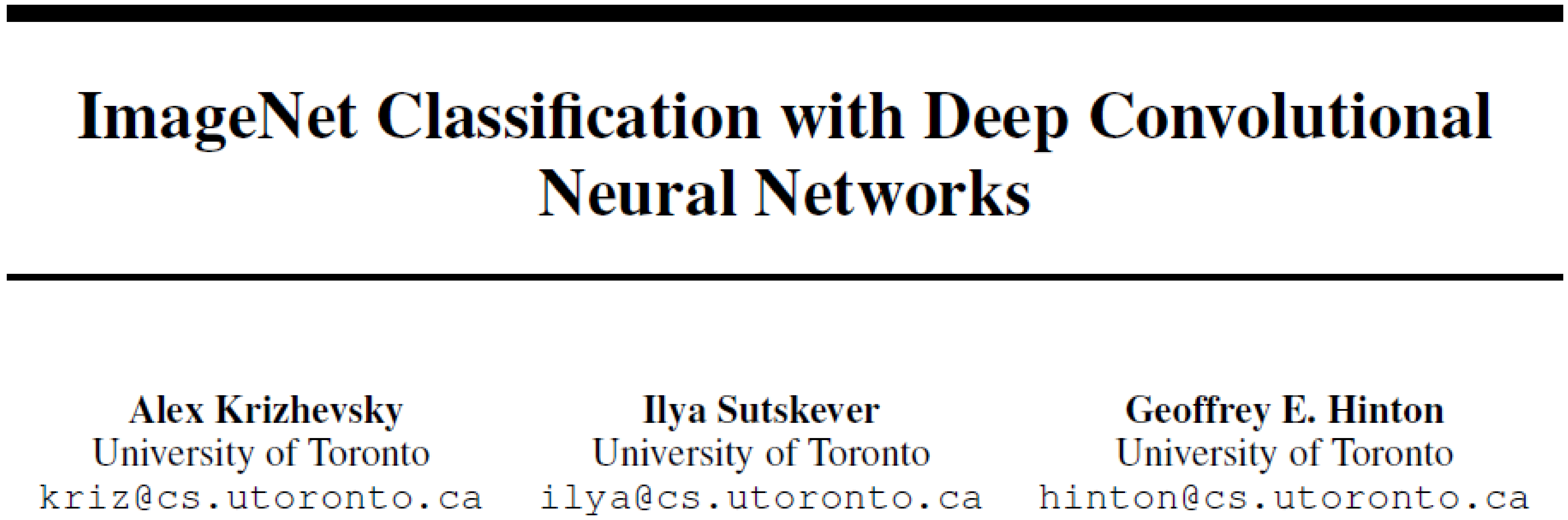
LeNet-5 (1998)



LeCun, Yann, et al. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE* 86.11 (1998) :hi

Award Winning CNNs





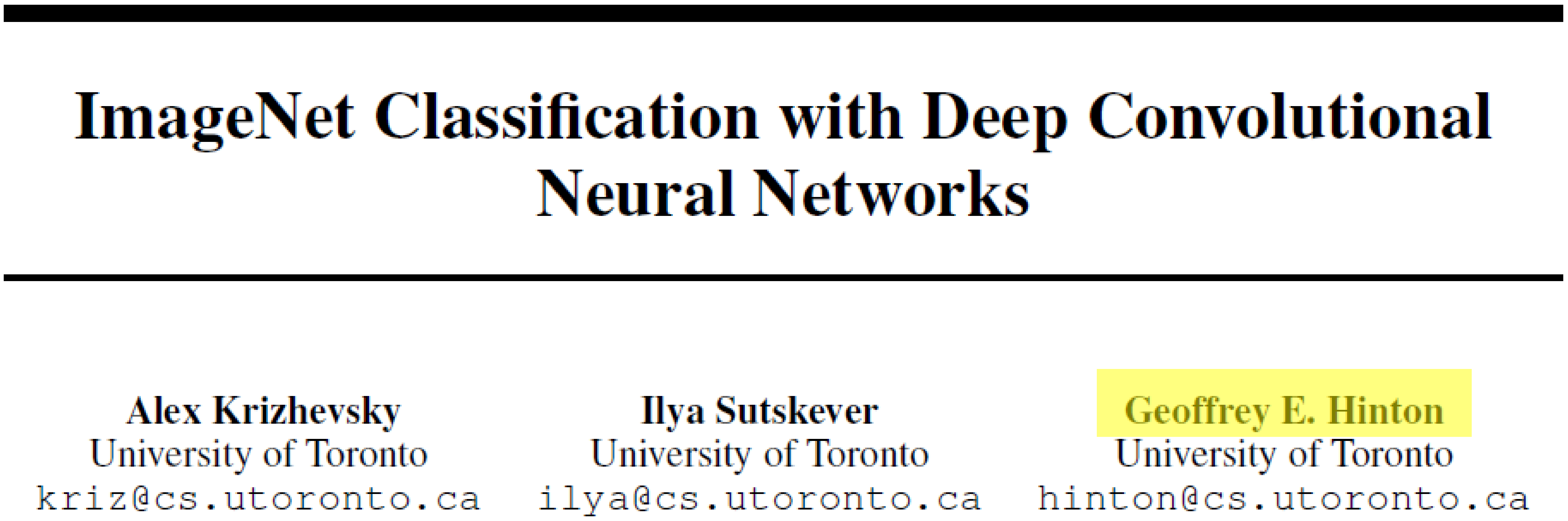
ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." NIPS 2012.



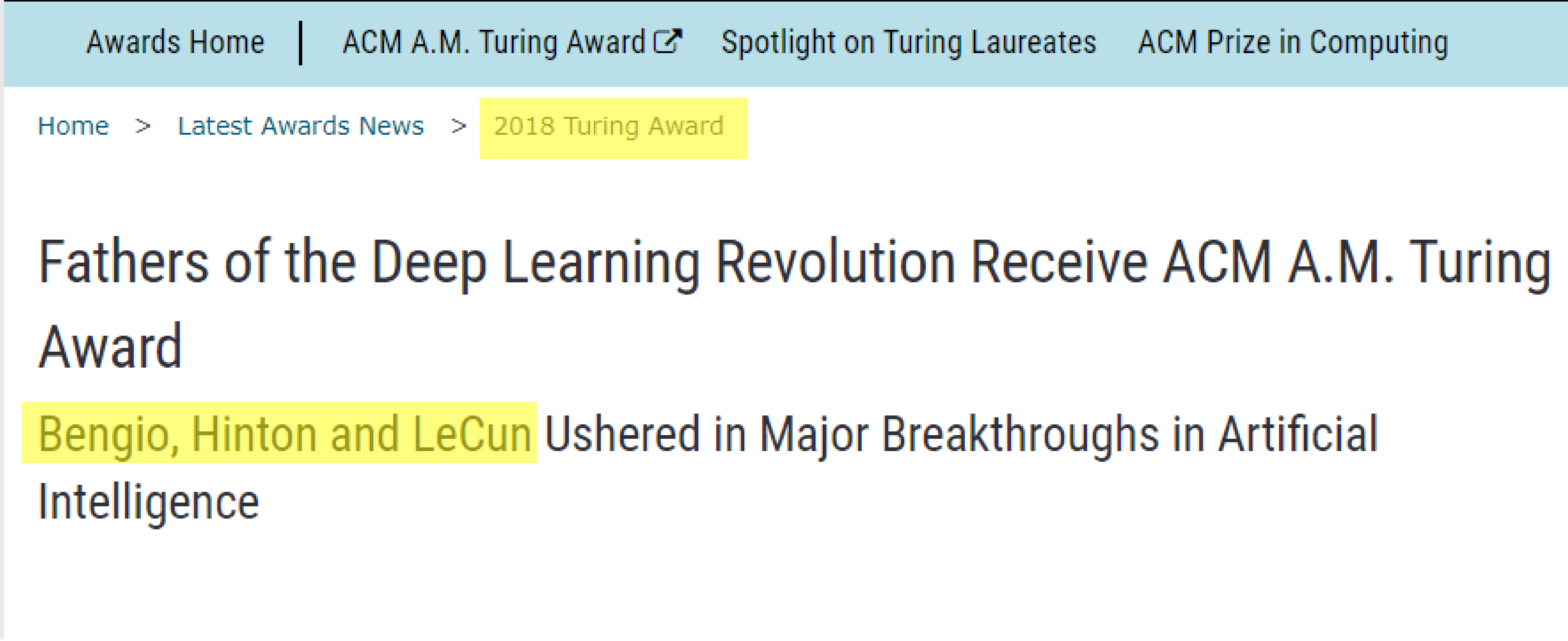
ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." NIPS 2012.


Awards Home
ACM A.M. Turing Award
Spotlight on Turing Laureates
ACM Prize in Computing
Home > Latest Awards News > 2018 Turing Award

Fathers of the Deep Learning Revolution Receive ACM A.M. Turing Award

Bengio, Hinton and LeCun Ushered in Major Breakthroughs in Artificial Intelligence

https://awards.acm.org/about/2018-turing

AlexNet (2012)

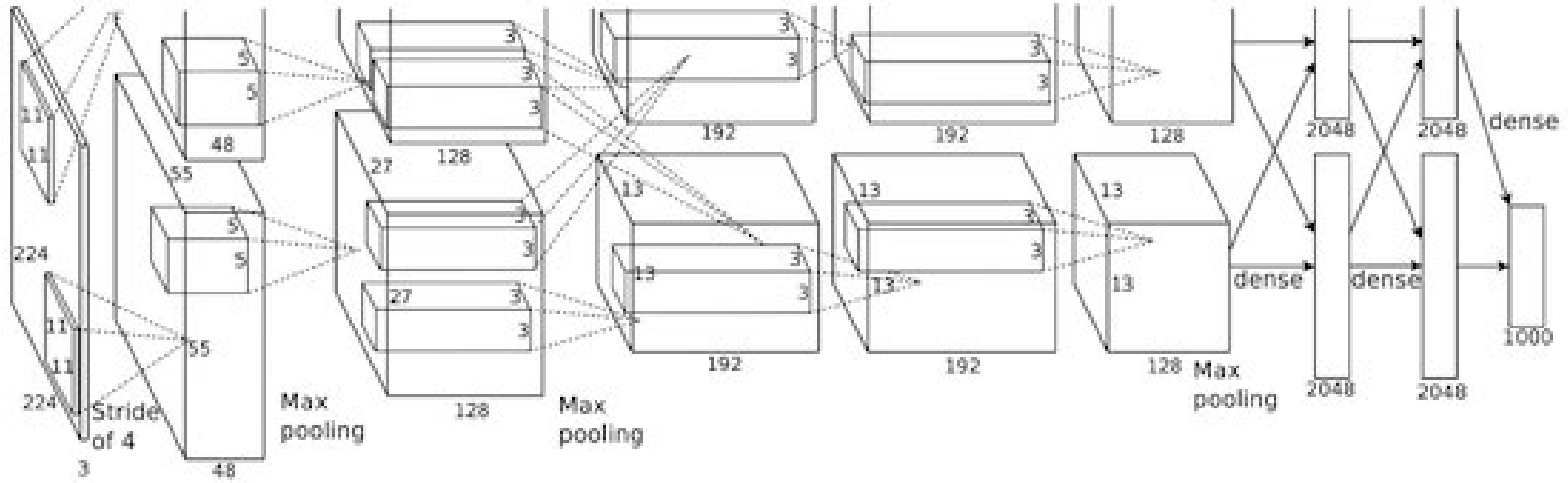
Developed by Alex Krizhevsky et al. in 2012 and won Imagenet competition

Architecture is quite similar to LeNet-5:

- 5 convolutional layers (rather large filters, 11x11, 5x5),
- 3 MLP

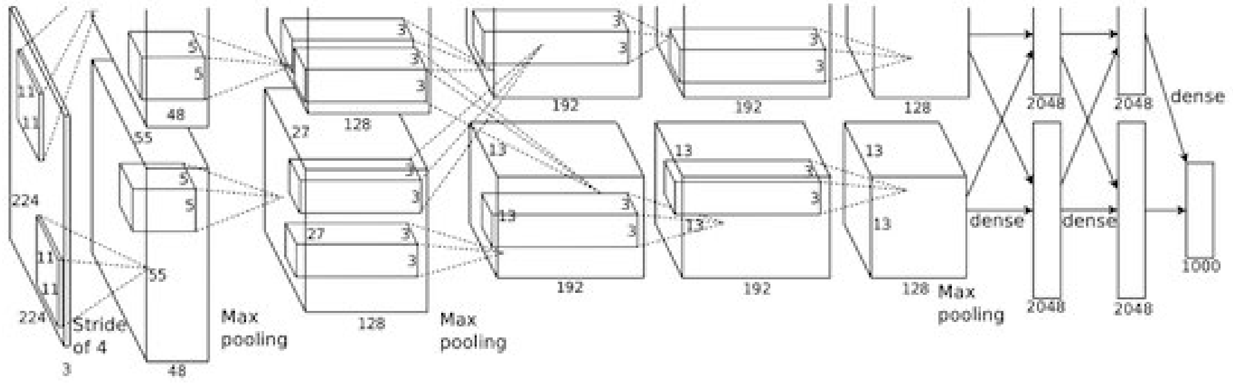
Input size 224 × 224 x 3 (the paper says 227 x 227 x 3)

Parameters: 60 million [**Conv**: 3.7million (6%), **FC**: 58.6 million (94%)]



Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." NIPS 2012.

AlexNet (2012)



To counteract overfitting, they introduce:

- RELU (also faster than tanh)
- Dropout (0.5), weight decay and norm layers (not used anymore)
- Maxpooling

The first conv layer has 96 **11x 11 filters, stride 4.**

The output are **two volumes of 55 x 55 x 48 separated over two GTX 580 GPUs** (1.5GB each GPU, 90 epochs, 5/6 days to train).

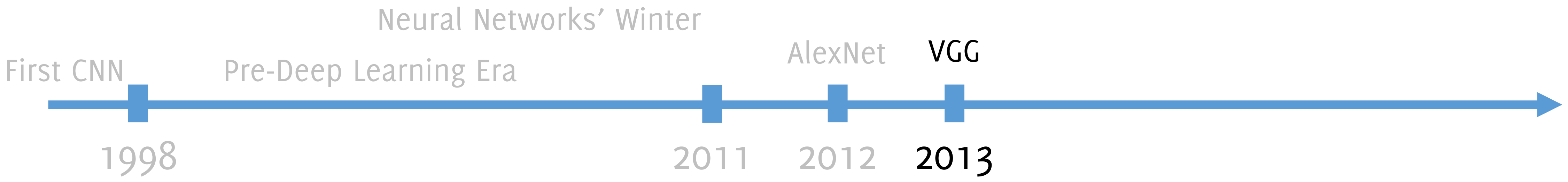
Most **connections are among** feature maps **of the same GPU**, which will be mixed at the last layer.

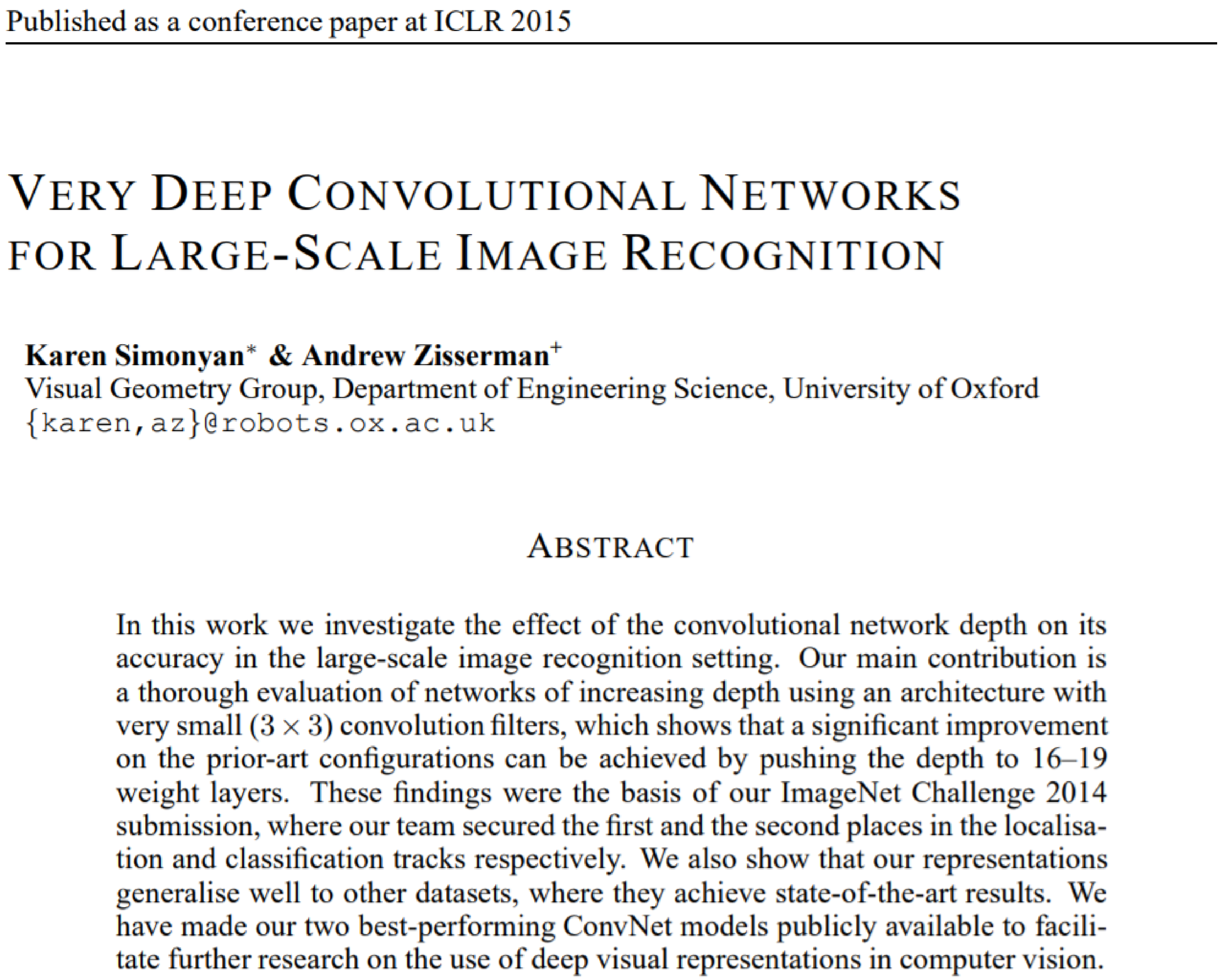
Won the ImageNet challenge in 2012

At the end they also trained an **ensemble of 7 models** to drop error: 18.2%->15.4%

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." NIPS 2012.

VGG: going deeper!





VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION

Karen Simonyan[*] **& Andrew Zisserman**[+]
Visual Geometry Group, Department of Engineering Science, University of Oxford
{karen,az}@robots.ox.ac.uk

ABSTRACT

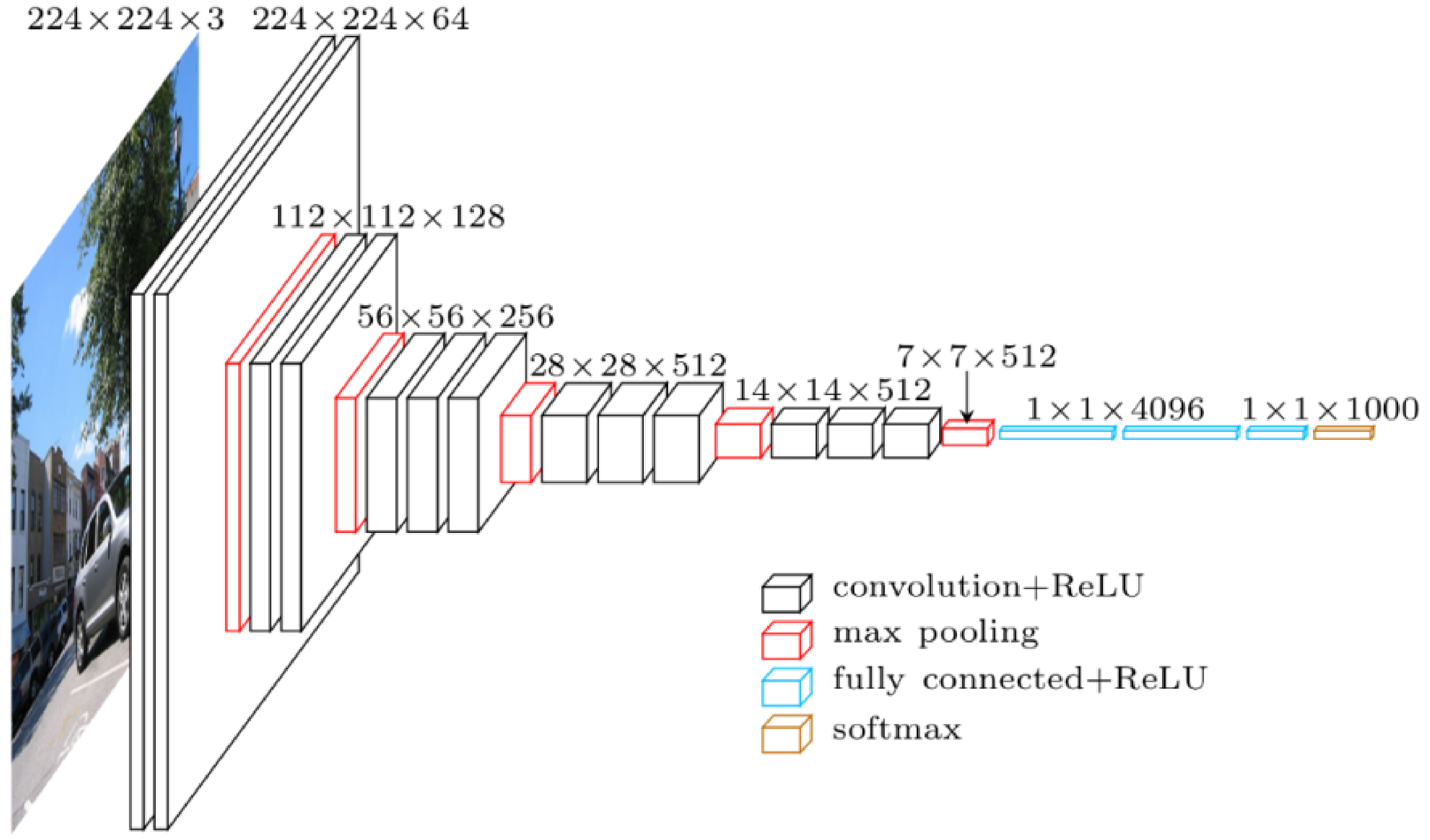
In this work we investigate the effect of the convolutional network depth on its accuracy in the large-scale image recognition setting. Our main contribution is a thorough evaluation of networks of increasing depth using an architecture with very small (3×3) convolution filters, which shows that a significant improvement on the prior-art configurations can be achieved by pushing the depth to 16–19 weight layers. These findings were the basis of our ImageNet Challenge 2014 submission, where our team secured the first and the second places in the localisation and classification tracks respectively. We also show that our representations generalise well to other datasets, where they achieve state-of-the-art results. We have made our two best-performing ConvNet models publicly available to facilitate further research on the use of deep visual representations in computer vision.

Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." ICLR (2015)

VGG16 (2014)

The VGG16, introduced in 2014 is a deeper variant of the AlexNet convolutional structure. Smaller filters are used and the network is deeper

Parameters: 138 million [Conv: 11%, FC: 89%]



Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." ICLR (2015)

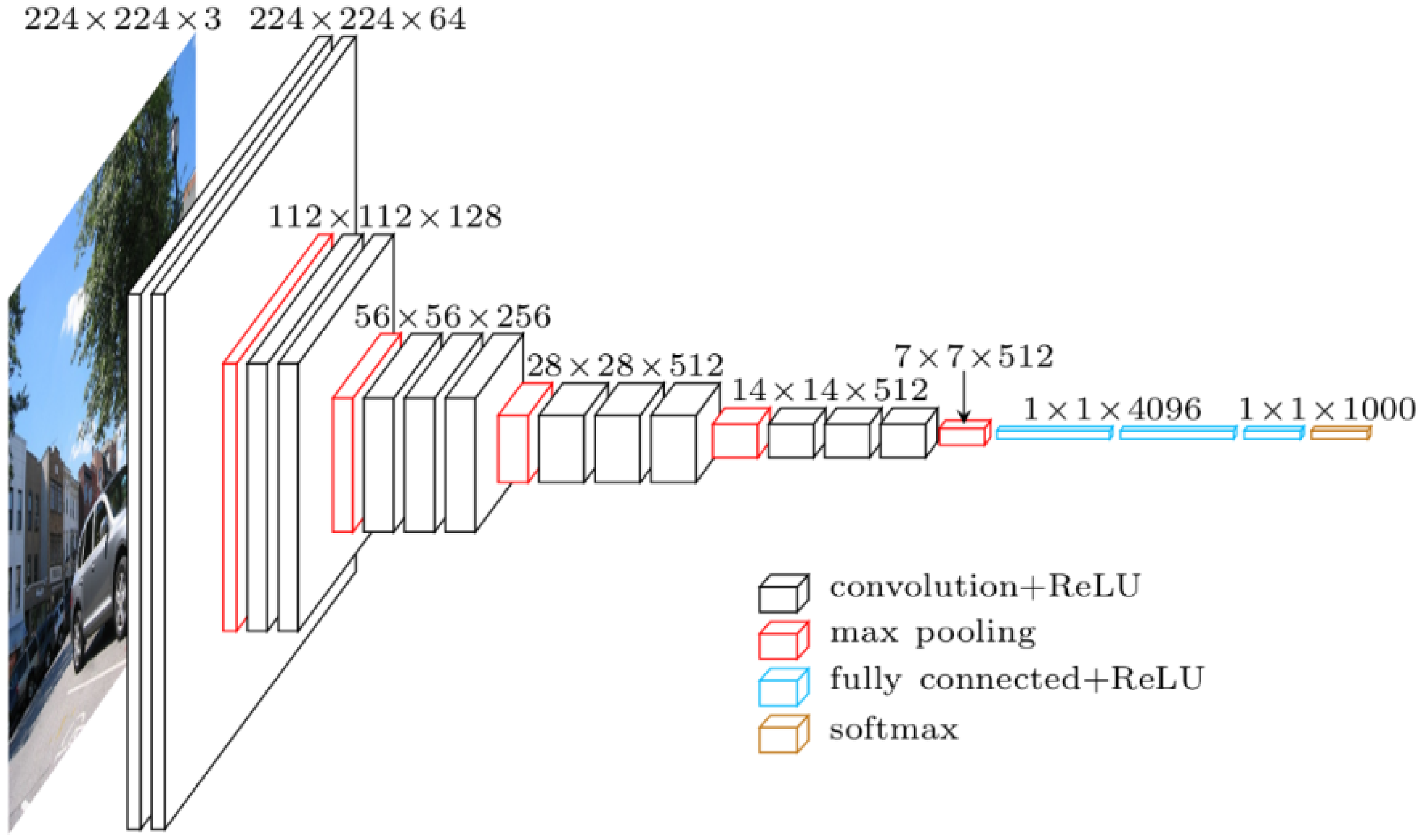
VGG16 (2014)

The VGG16, introduced in 2014 is a deeper variant of the AlexNet convolutional structure. Smaller filters are used and the network is deeper

Parameters: 138 million [Conv: 11%, FC: 89%]

These architecture **won the first place places (localization) and the second place (classification) tracks in ImageNet Challenge 2014**

Input size 224 × 224 x 3



VGG16 (2014): Smaller Filter, Deeper Network

The paper actually present a thorough **study on the role of network depth**.

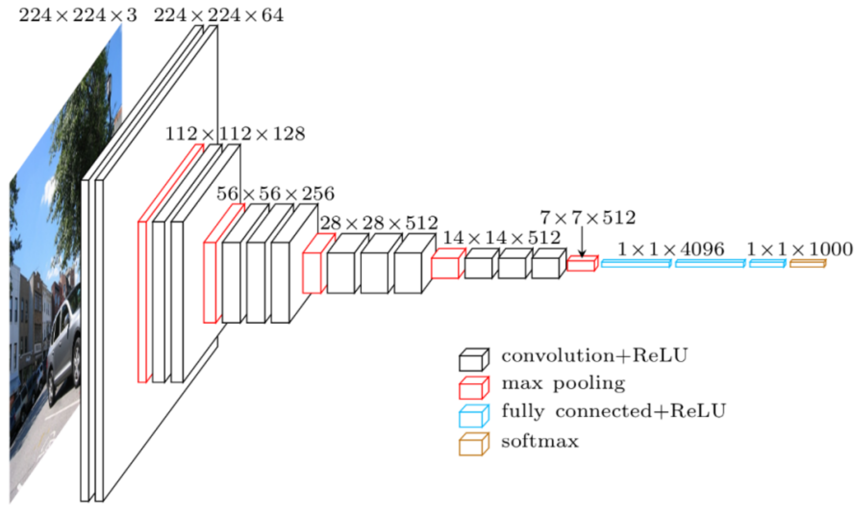
[…]Fix other parameters of the architecture, and steadily increase the depth of the network by adding more convolutional layers, which is feasible due ***to the use of very small (3. × 3) convolution filters in all layers.***

Idea: Multiple 3×3 convolution in a sequence achieve large receptive fields with:

- less parameters
- more nonlinearities

than larger filters in a single layer

	3 layers 3x3	**1 layer 7x7**
Receptive field	7x7	7x7
Nr of filter weights	3 x 3 x 3 = 27	49
Nr of nonlinearities	3	1



VGG16

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 224, 224, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
                                 […]
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
                                 […]
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
_________________________________________________________________
predictions (Dense)          (None, 1000)              4097000
=================================================================
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
_________________________________________________________________
```

VGG16

```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 224, 224, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
                                 […]
```

```
Layer (type)                 Output Shape              Param #
=================================================================
                                 […]
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
_________________________________________________________________
predictions (Dense)          (None, 1000)              4097000
=================================================================
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
_________________________________________________________________
```

Many convolutional blocks without maxpooling

VGG16

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 224, 224, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
                                [...]
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
                                [...]
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
_________________________________________________________________
predictions (Dense)          (None, 1000)              4097000
=================================================================
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
_________________________________________________________________
```

Most parameters in FC layers : **123,642,856**

G. Boracchi

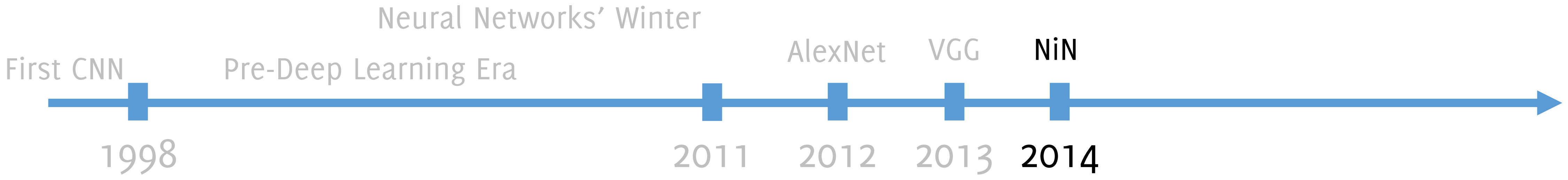
VGG16

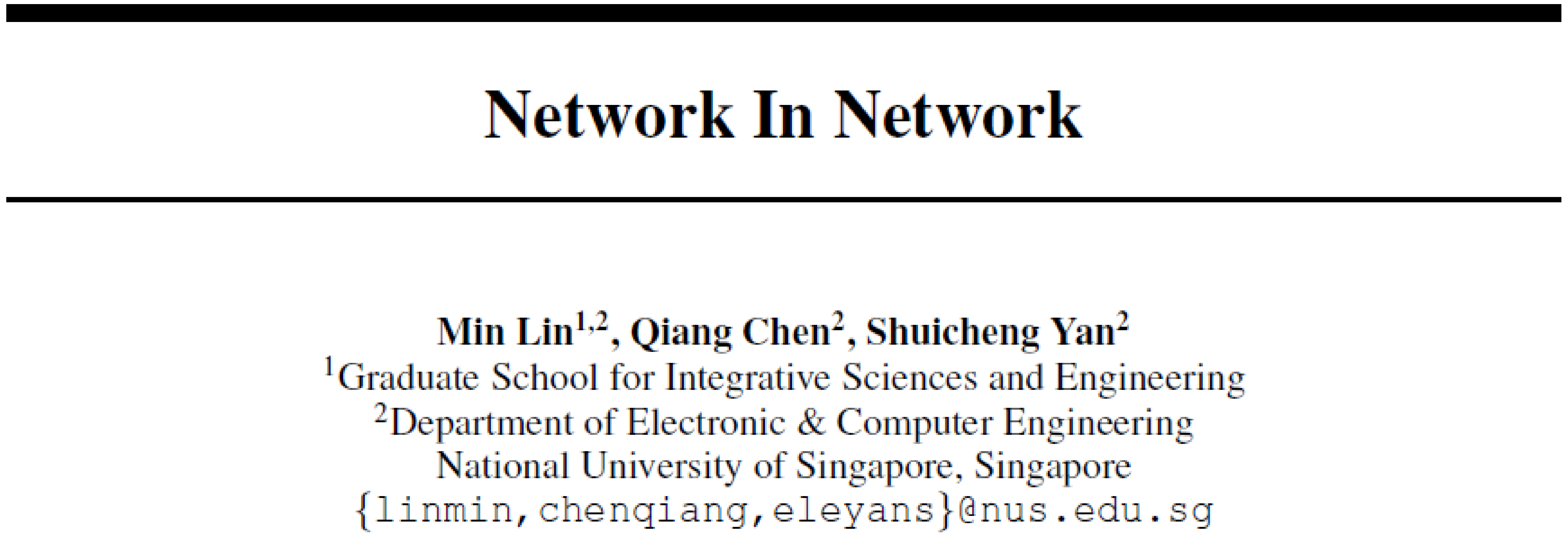
```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 224, 224, 3)       0

block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792

block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928

block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0

block2_conv1 (Conv2D)

block2_conv2 (Conv2D)

block2_pool (MaxPooling2D)

block3_conv1 (Conv2D)

block3_conv2 (Conv2D)

block3_conv3 (Conv2D)

block3_pool (MaxPooling2D)

block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160

block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808

block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
                      […]
```

```
Layer (type)                 Output Shape              Param #
=================================================================
                             […]

block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0

block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808

block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808

                              , 14, 512)               2359808

                               7, 512)                 0

                               088)                    0

                               96)                     102764544

                               96)                     16781312

                               00)                     4097000
                               ==============================

Trainable params: 138,357,544
Non-trainable params: 0
```

High memory request, about 100MB per image (224 × 224 × 3) to be stored in all the activation maps, only for the forward pass. During training, with the backward pass it's about twice

Networks in Networks





Network In Network

Min Lin[1,2], Qiang Chen[2], Shuicheng Yan[2]
[1]Graduate School for Integrative Sciences and Engineering
[2]Department of Electronic & Computer Engineering
National University of Singapore, Singapore
`{linmin,chenqiang,eleyans}@nus.edu.sg`

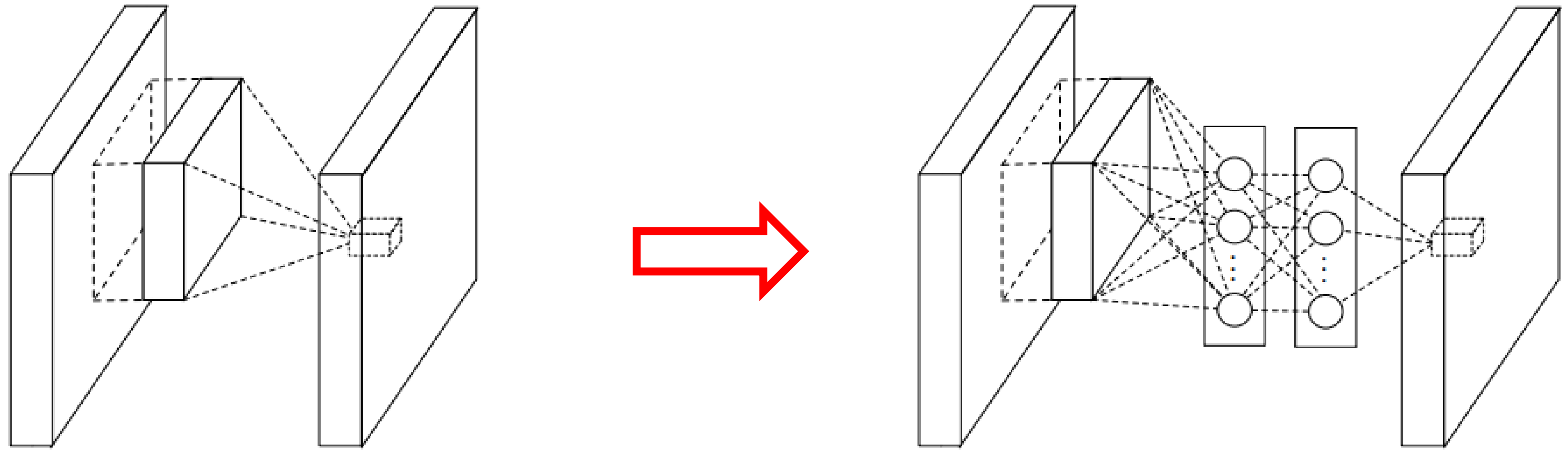
Lin, Min, Qiang Chen, and Shuicheng Yan. "Network in network." *ICLR 2014*

Network in Network

Mlpconv layers: instead of conv layers, use **a sequence of FC + RELU**

- Uses a stack of FC layers followed by RELU **in a sliding manner on the entire image. This corresponds to MLP networks used convolutionally**

Each layer features a **more powerful functional approximation** than a convolutional layer which is just linear + RELU



(a) Linear convolution layer (b) Mlpconv layer

Network in Network

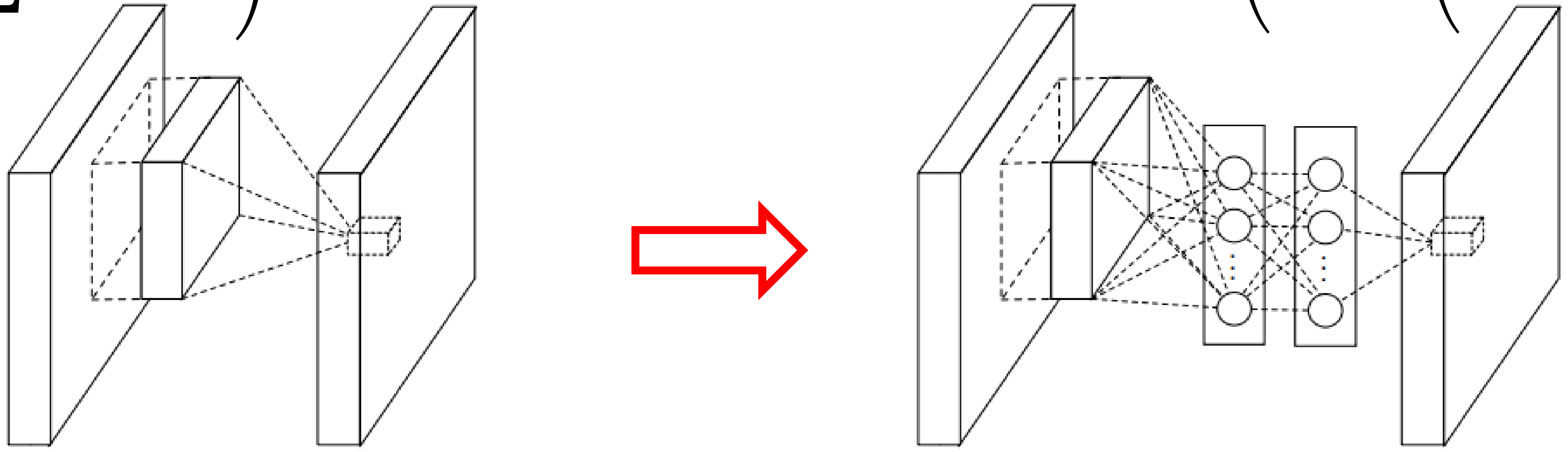
Mlpconv layers: instead of conv layers, use **a sequence of FC + RELU**

- Uses a stack of FC layers followed by RELU **in a sliding manner on the entire image. This corresponds to MLP networks used convolutionally**

Each layer features a **more powerful functional approximation** than a convolutional layer which is just linear + RELU

$$f = ReLU\left(\sum w_i x_i + b\right)$$

$$f = ReLU\left(\sum w_i^1 \left(ReLU\left(\sum w_i^2 x_i + b^2\right)\right) + b^1\right)$$



(a) Linear convolution layer

(b) Mlpconv layer

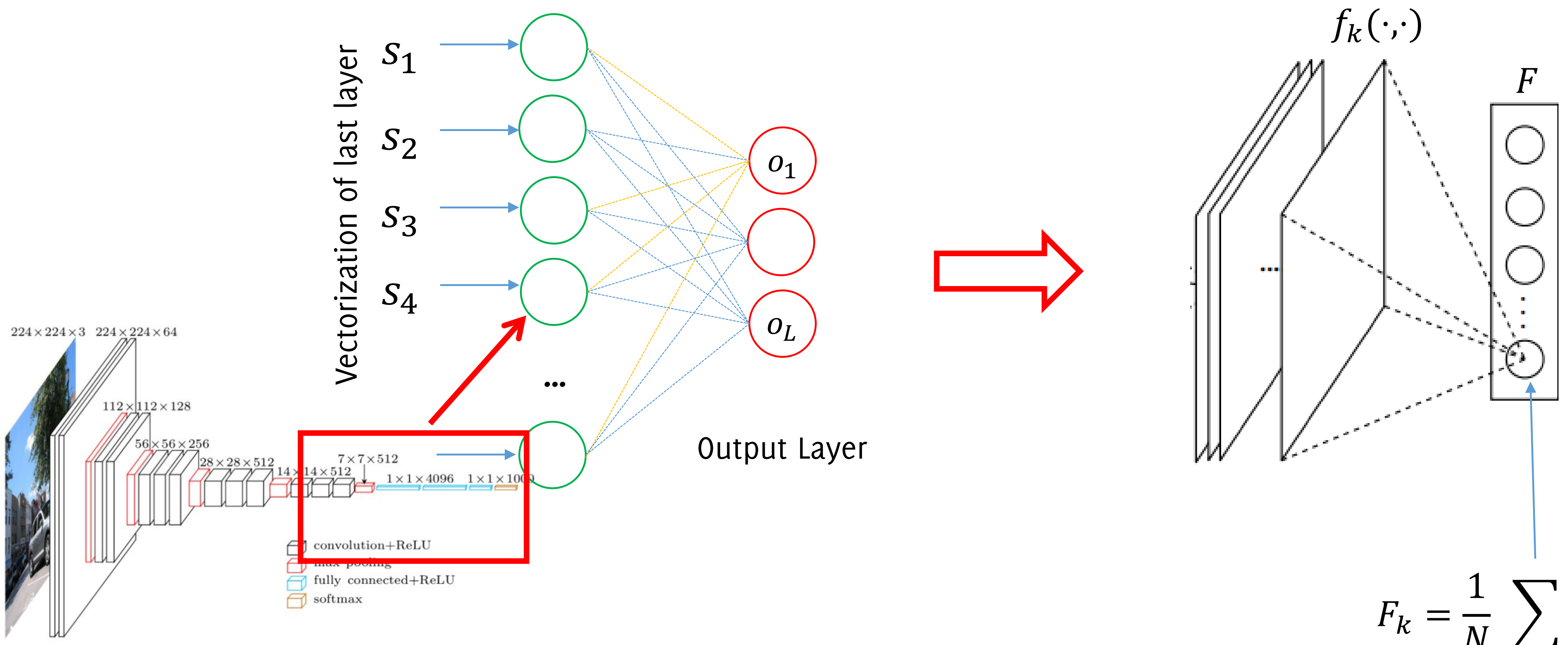
Lin, Min, Qiang Chen, and Shuicheng Yan. "Network in network." *ICLR 2014*

Network in Network

They also introduce Global Averaging Pooling Layers

Fully Connected Layer

Global Averaging Pooling Layer



$$F_k = \frac{1}{N} \sum_{(x,y)} f_k(x,y)$$

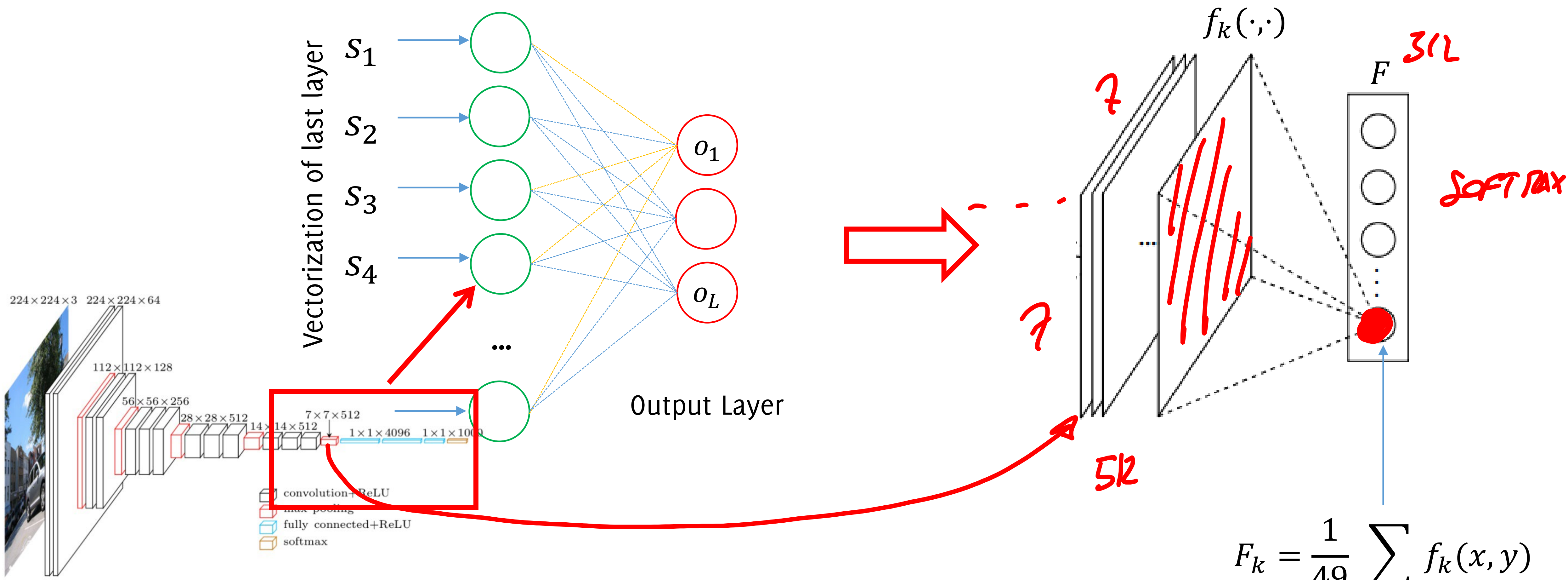
Lin, Min, Qiang Chen, and Shuicheng Yan. "Network in network." *ICLR 2014*

Network in Network

They also introduce Global Averaging Pooling Layers

Fully Connected Layer

GAP: Global Averaging Pooling Layer



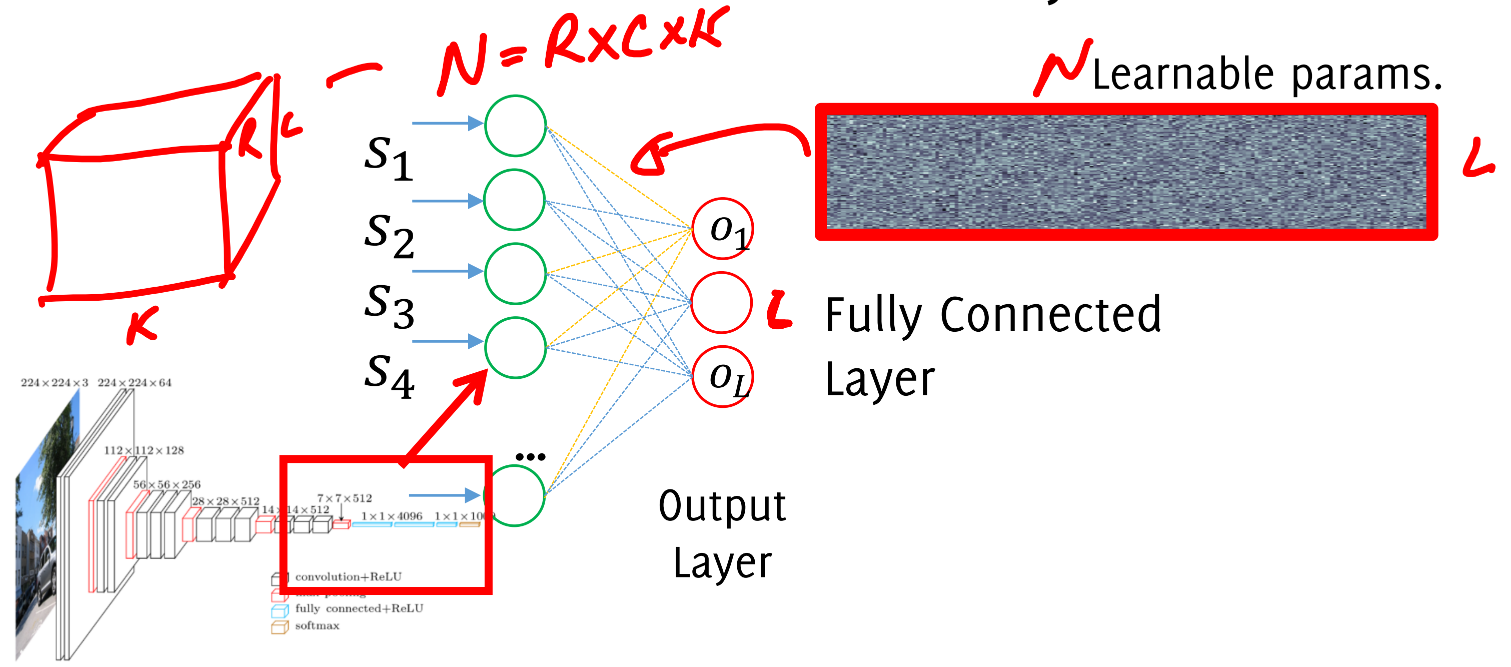
$$F_k = \frac{1}{49} \sum_{(x,y)} f_k(x,y)$$

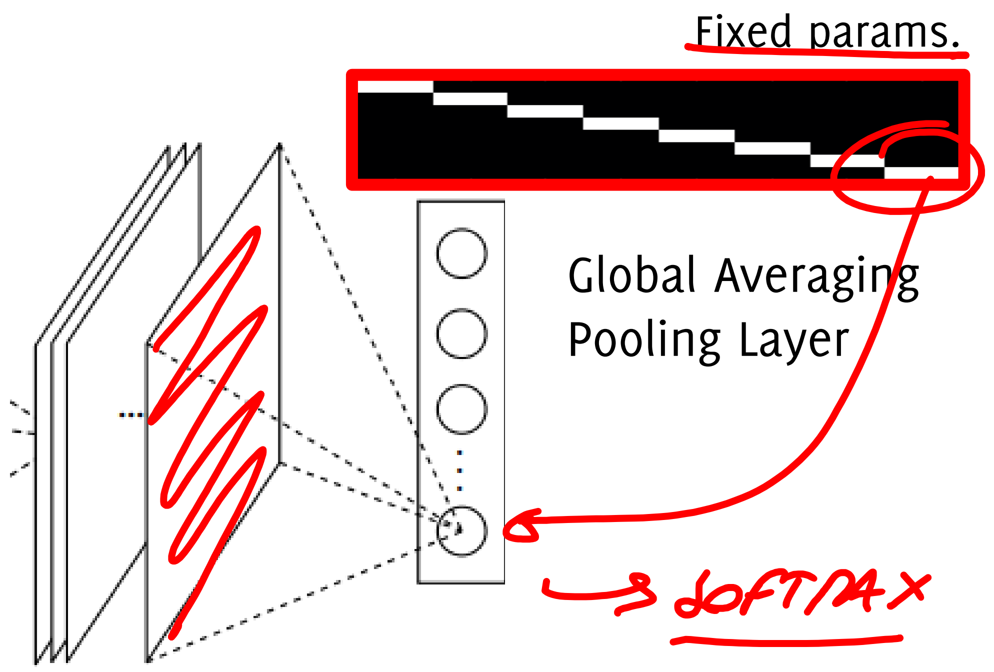
Lin, Min, Qiang Chen, and Shuicheng Yan. "Network in network." *ICLR 2014*

Network in Network: GAP

Global Averaging Pooling Layers: instead of a FC layer at the end of the network, **compute the average of each feature map.**

- The transformation corresponding **to GAP is a block diagonal, constant matrix** (consider the input unrolled layer-wise in a vector)
- The transformation of each layer in **MLP corresponds to a dense matrix.**





Rationale behind GAP

Fully connected layers are prone to overfitting

- They have many parameters
- Dropout was proposed as a regularized that randomly sets to zero a percentage of activations in the FC layers during training

The GAP was here used as follows:

1. **Remove the fully connected layer** at the end of the network!
2. **Introduce a GAP layer.**
3. **Predict by a simple soft-max after the GAP**.

Watch out: the number of feature maps has to correspond to the number of output classes! **In general, GAP can** be used with more/fewer classes than channels **provided an hidden layer to adjust feature dimension**

The Advantages of GAP Layers:

- No parameters to optimize, lighter networks less prone to overfitting
- Classification is performed by a softMax layer at the end of the GAP
- More interpretability, creates a direct connection between layers and classes output (we'll see in localization)
- This makes GAP a structural regularizer
- **Increases robustness to spatial transformation** of the input images
- **The network can be used to classify images of different sizes**

Lin, Min, Qiang Chen, and Shuicheng Yan. "Network in network." ICLR 2014

Network in Network

The whole NiN stacks

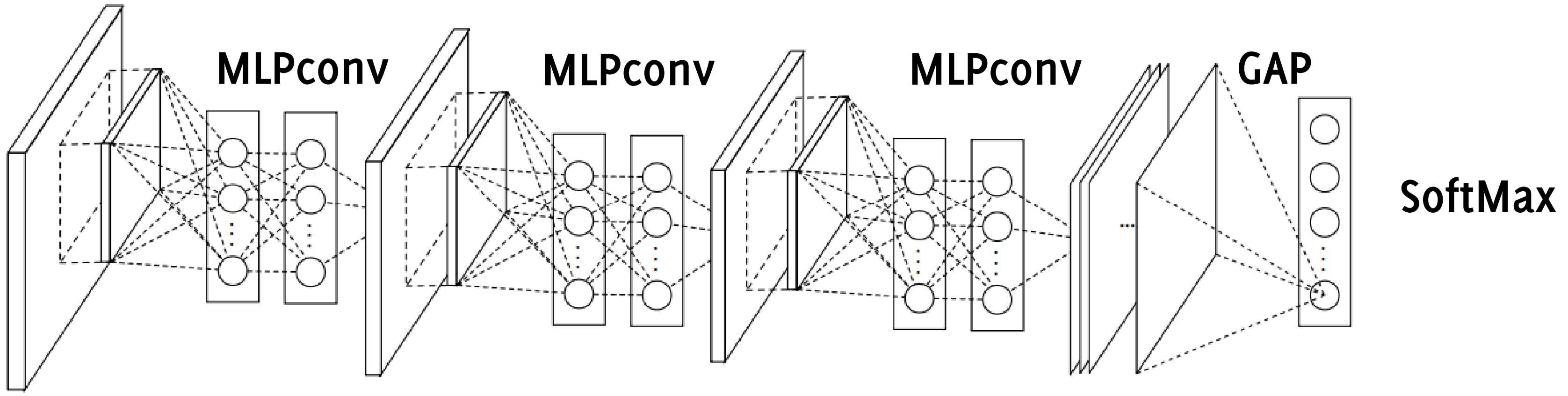
- mlpconv layers (RELU) + dropout
- Maxpooling

$\}$ A few layers of these

- Global Averaging Pooling (GAP) layer
- Softmax

$\}$ At the end of the network

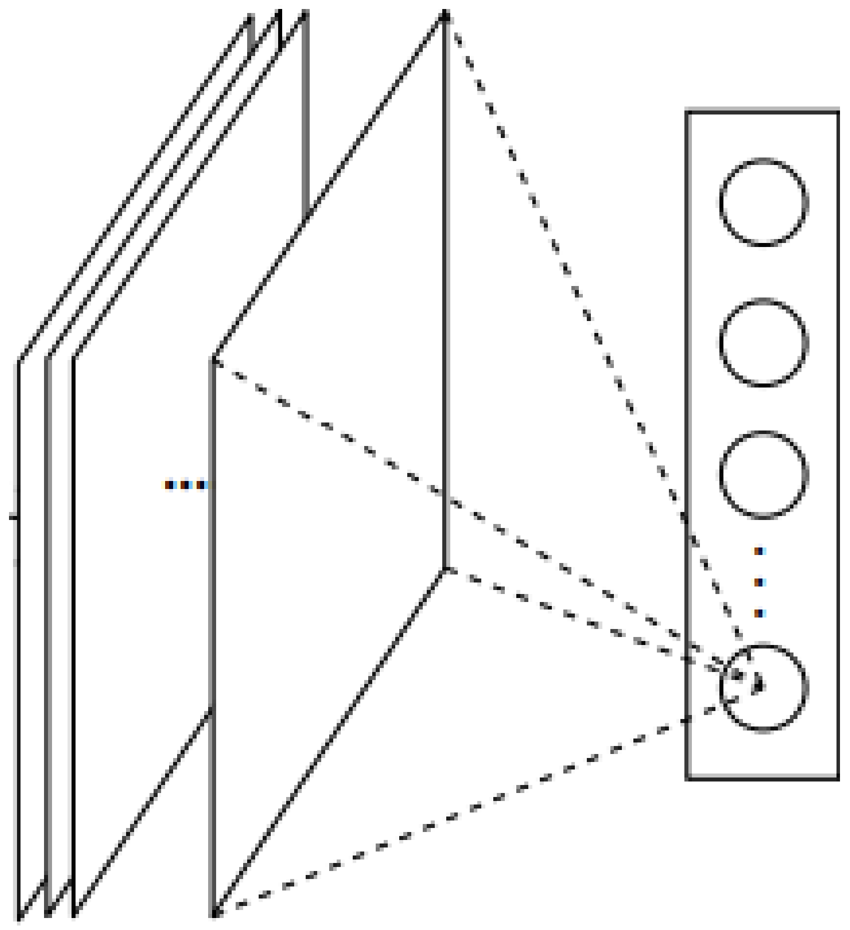
simple NiNs achieve state-of-the-art performance on «small» datasets (CIFAR10, CIFAR100, SVHN, MNIST) and that **GAP effectively reduces overfitting** w.r.t. FC



The Global Averaging Pooling (GAP) Layer

We indeed see that GAP is acting as a (structural) regularizer

Method	Testing Error
mlpconv + Fully Connected	11.59%
mlpconv + Fully Connected + Dropout	10.88%
mlpconv + Global Average Pooling	10.41%



GAP in Keras

```
gap = tfkl.GlobalAveragePooling2D(
            name='gap'
      )(x)
```

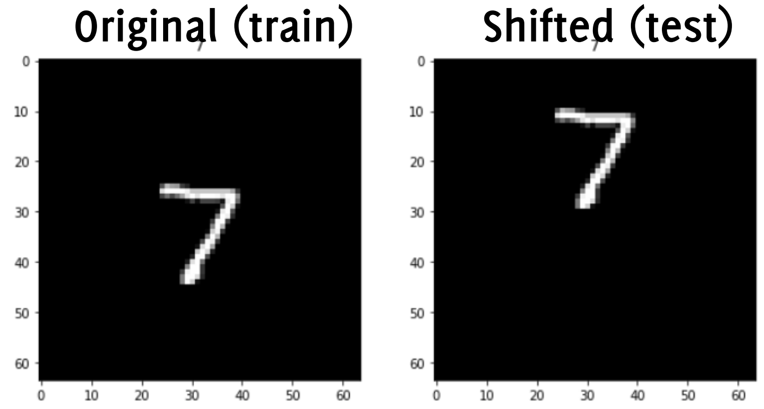
There are a couple of optional parameters but this are not relevant..

The output size of gap is (batch_size, channels)

GAP Increases Invariance to Shifts

Features extracted by the convolutional part of the network are invariant to shift of the input image

The MLP after the flattening is not invariant to shifts (different input neurons are connected by different weights)



Therefore, a CNN trained on centered images might not be able to correctly classify shifted ones

The GAP solves this problem, since there is no GAP and the two images lead to the same or very similar features

GAP Increases Invariance to Shifts

Example:

Dataset: a 64x64 (zero padded) MNIST

CNN-flattening: a traditional CNN with flattening, trained

CNN-GAP: the same architecture CNN but with GAP instead of MLP

Train both CNNs over the same training set without shift

Test both CNNs over both

- Original test set
- Sifted test set

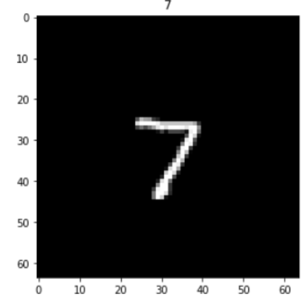
CNN-flattening Architecture

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 Input (InputLayer)          [(None, 64, 64)]          0

 reshape_1 (Reshape)         (None, 64, 64, 1)         0
 conv1 (Conv2D)              (None, 62, 62, 32)        320
 pool1 (MaxPooling2D)        (None, 31, 31, 32)        0
 conv2 (Conv2D)              (None, 29, 29, 64)        18496
 pool2 (MaxPooling2D)        (None, 14, 14, 64)        0
 conv3 (Conv2D)              (None, 12, 12, 128)       73856
 pool3 (MaxPooling2D)        (None, 6, 6, 128)         0
 flatten (Flatten)           (None, 4608)              0
 dropout1 (Dropout)          (None, 4608)              0
 classifier (Dense)          (None, 64)                294976
 dropout2 (Dropout)          (None, 64)                0
 Output (Dense)              (None, 10)                650

=================================================================
Total params: 388,298
Trainable params: 388,298
Non-trainable params: 0
```

CNN-GAP Architecture

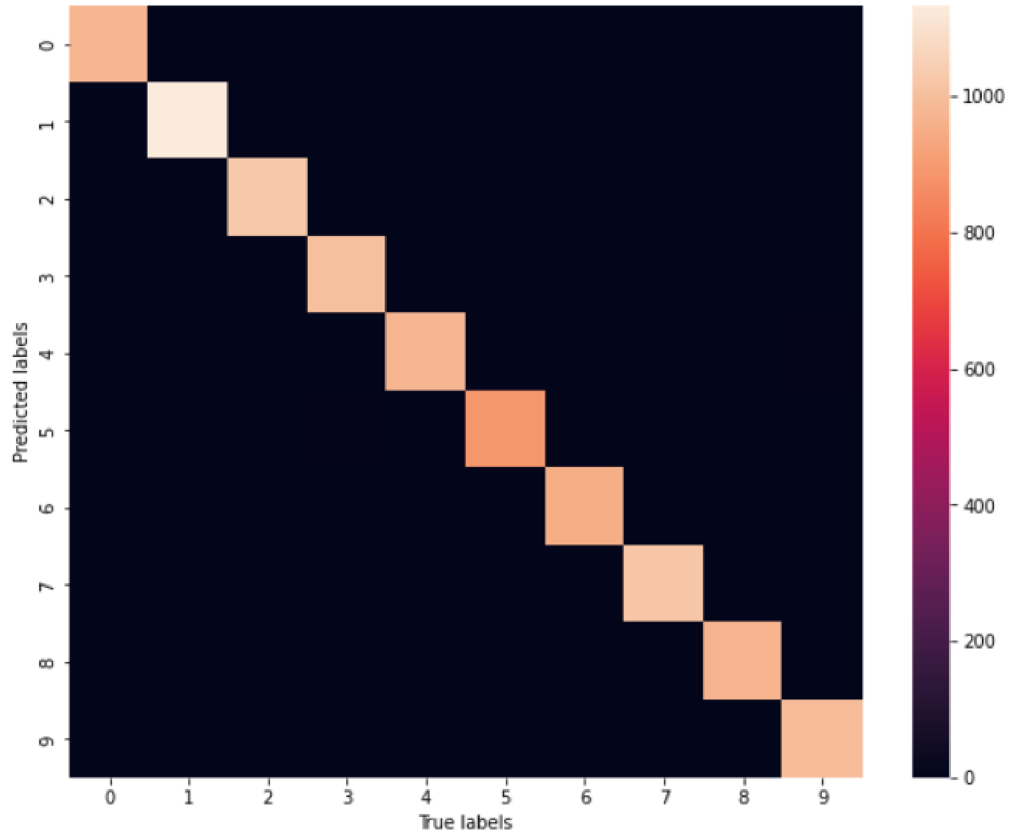
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Input (InputLayer)           [(None, 64, 64)]          0
reshape_3 (Reshape)          (None, 64, 64, 1)         0
conv1 (Conv2D)               (None, 62, 62, 32)        320
pool1 (MaxPooling2D)         (None, 31, 31, 32)        0
conv2 (Conv2D)               (None, 29, 29, 64)        18496
pool2 (MaxPooling2D)         (None, 14, 14, 64)        0
conv3 (Conv2D)               (None, 12, 12, 128)       73856
gpooling (GlobalAveragePooli (None, 128)               0
dropout1 (Dropout)           (None, 128)               0
classifier (Dense)           (None, 64)                8256
dropout2 (Dropout)           (None, 64)                0
Output (Dense)               (None, 10)                650

=================================================================
Total params: 101,578
Trainable params: 101,578
Non-trainable params: 0

_________________________________________________________________
```

Accuracy over Original Test Set



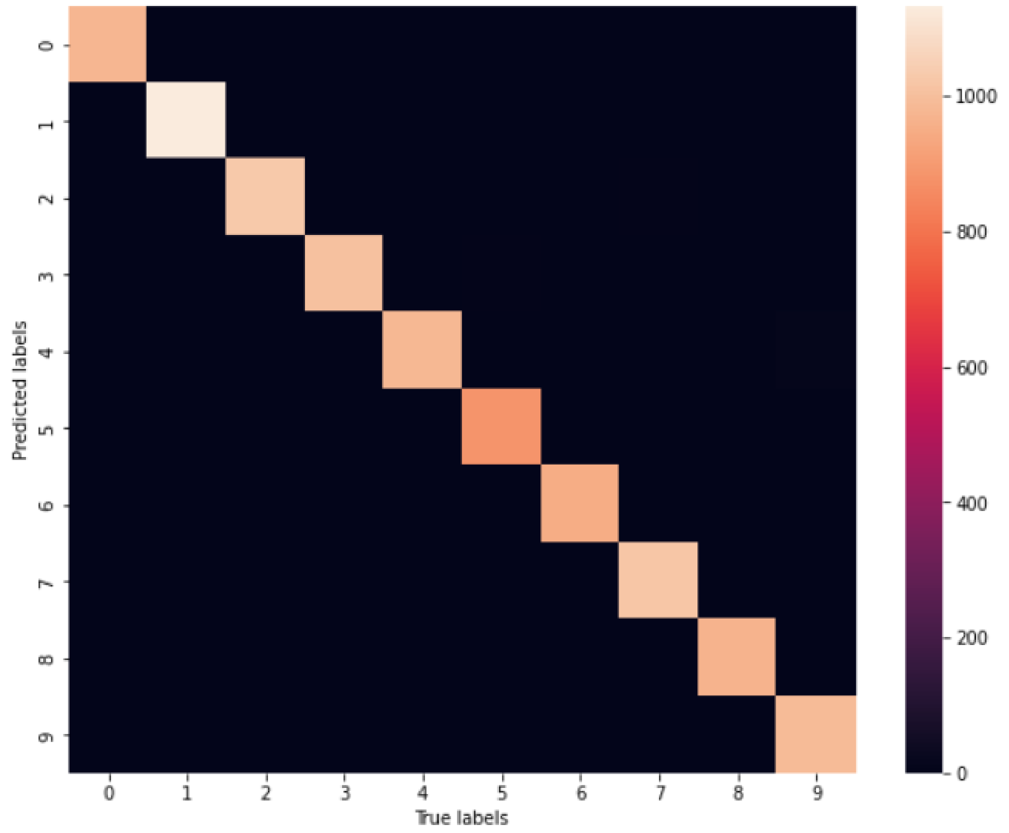
CNN-flattening:

```
Accuracy: 0.9936
Precision: 0.9935
Recall: 0.9936
F1: 0.9935
```

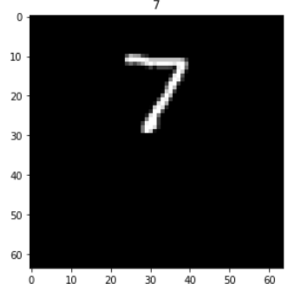


CNN-GAP

```
Accuracy: 0.9934
Precision: 0.9934
Recall: 0.9933
F1: 0.9933
```

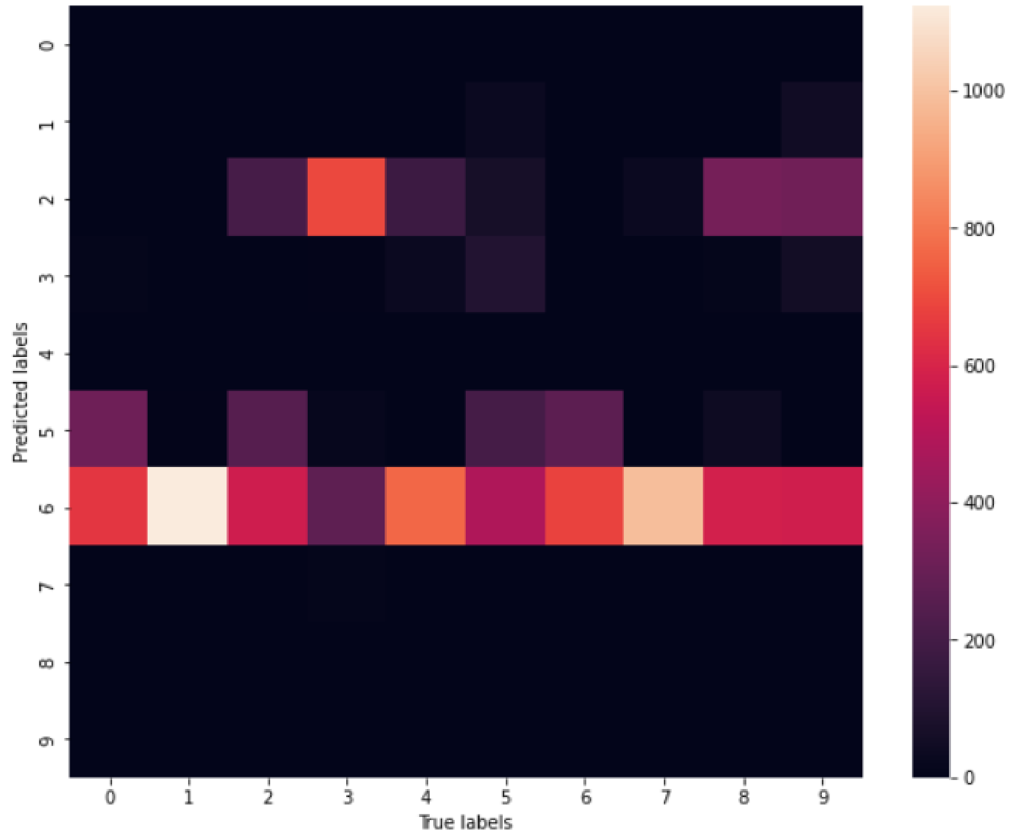


Accuracy over Shifted Test Set



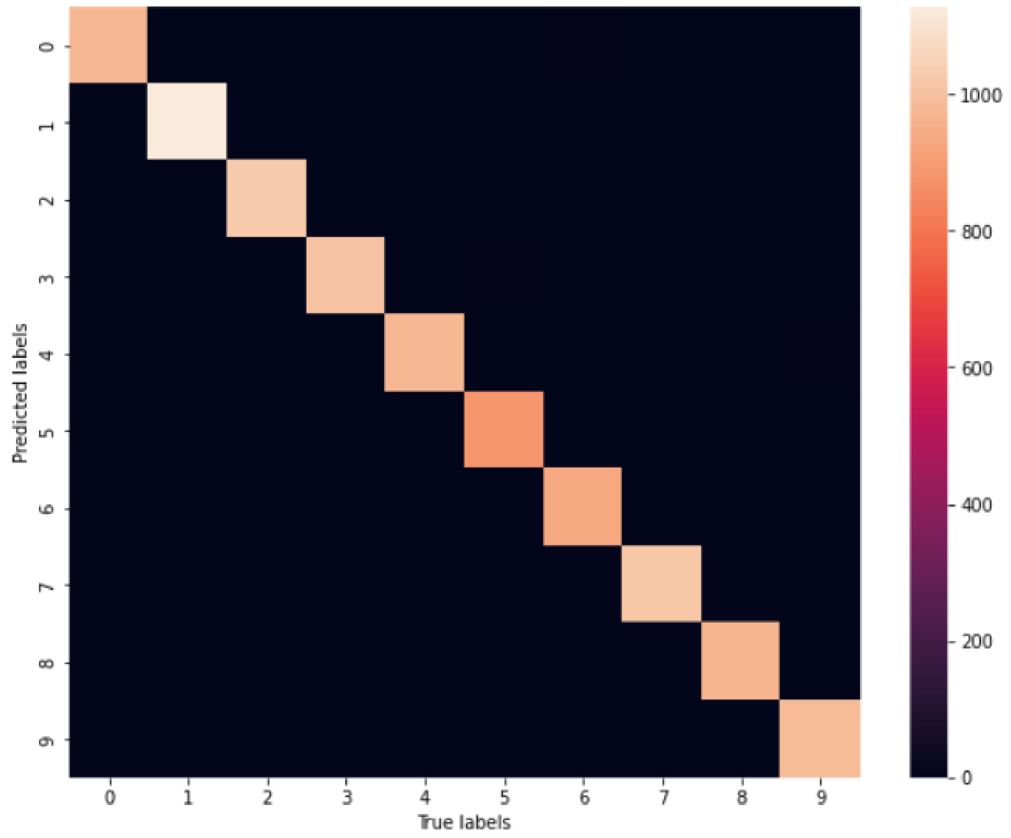
CNN-flattening:

```
Accuracy: 0.1103
Precision: 0.0435
Recall: 0.1151
F1: 0.0537
```



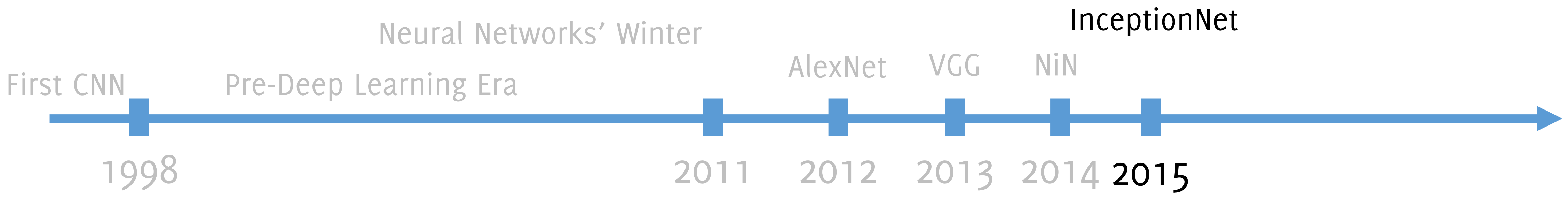
CNN-GAP

```
Accuracy: 0.9906
Precision: 0.9906
Recall: 0.9905
F1: 0.9905
```



https://colab.research.google.com/drive/1s108-0yIignuFBNTscfj9ZgDymi5sUOX?usp=sharing

InceptionNet: Multiple Branches



Going Deeper with Convolutions

Christian Szegedy[1], Wei Liu[2], Yangqing Jia[1], Pierre Sermanet[1], Scott Reed[3], Dragomir Anguelov[1], Dumitru Erhan[1], Vincent Vanhoucke[1], Andrew Rabinovich[4]

[1]Google Inc. [2]University of North Carolina, Chapel Hill

[3]University of Michigan, Ann Arbor [4]Magic Leap Inc.

[1]{szegedy,jiayq,sermanet,dragomir,dumitru,vanhoucke}@google.com

[2]wliu@cs.unc.edu, [3]reedscott@umich.edu, [4]arabinovich@magicleap.com

Inception Module

The most straightforward way of improving the performance of deep neural networks is by increasing their size (either in depth or width)

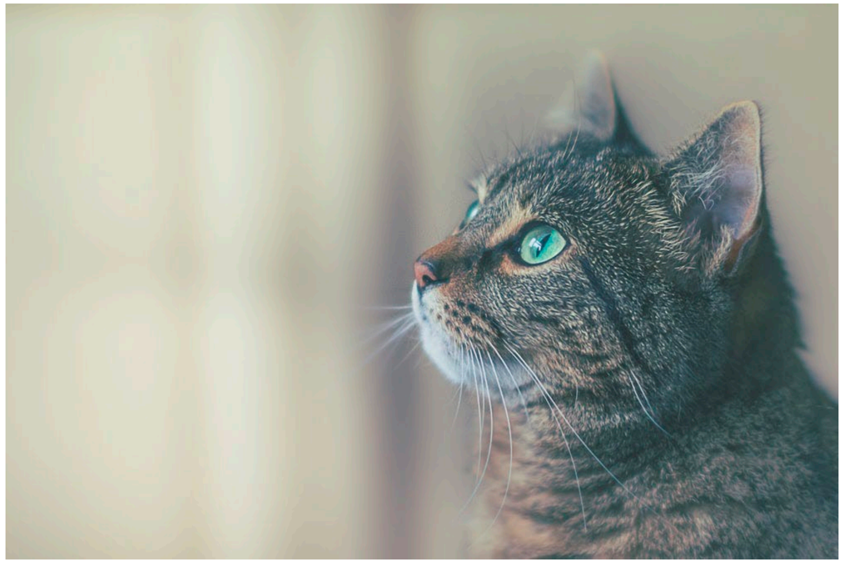
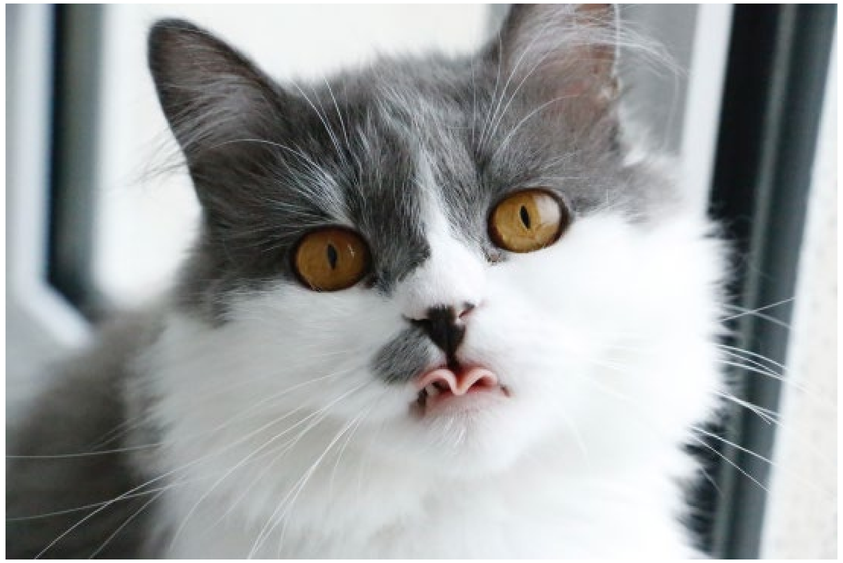
Bigger size typically means

- a larger number of parameters, which makes the enlarged network more prone to overfitting.
- dramatic increase in computational resources used.

Moreover image features might appear at different scale, and it is difficult to define the right filter size

Features might appear at different scales

Difficult to set the right kernel size!

Photos from Unsplash

GoogLeNet and Inception v1 (2014)

Deep network, with **high computational efficiency**

Only **5 million parameters**, **22 layers** of Inception modules

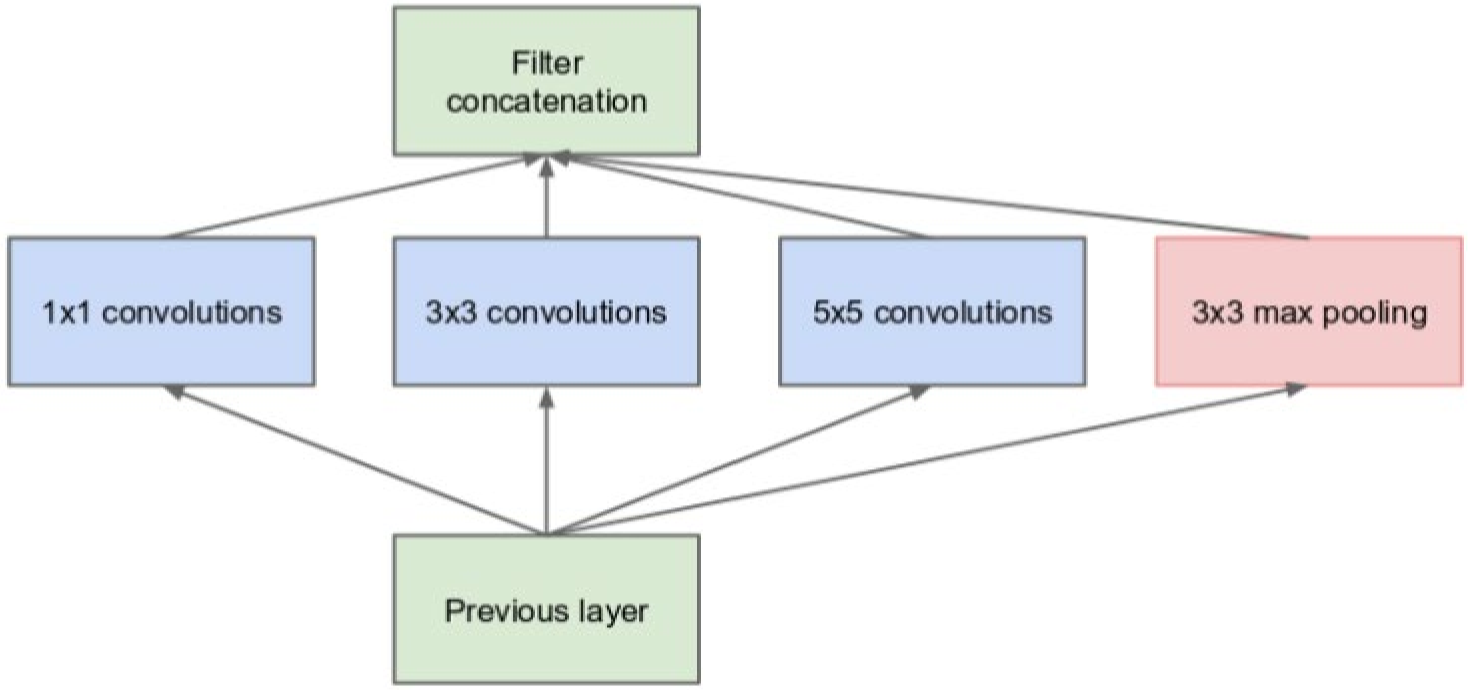
Won 2014 ILSVR-classification challenge (6,7% top 5 classification error)

Szegedy et al. "Going deeper with convolutions." ***CVPR 2015***

GoogLeNet and Inception v1 (2014)

It is based on inception modules, which are sort of «networks inside the network» or «local modules»

Concatenation preserves spatial resolution



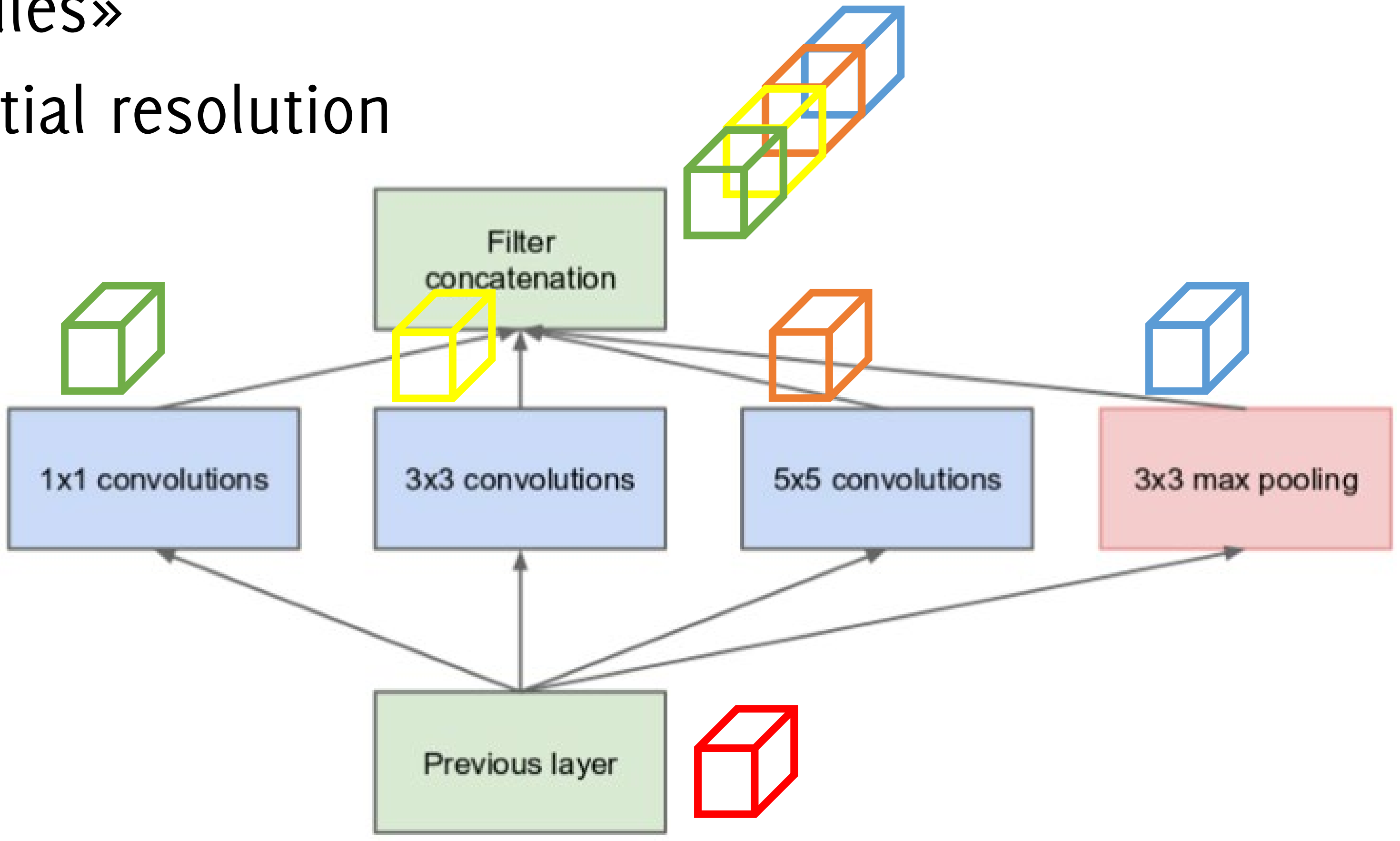
(a) Inception module, naïve version

Szegedy et al. "Going deeper with convolutions." *CVPR 2015*

GoogLeNet and Inception v1 (2014)

It is based on inception modules, which are sort of «networks inside the network» or «local modules»

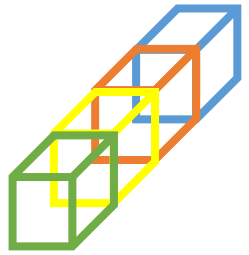
Concatenation preserves spatial resolution



(a) Inception module, naïve version

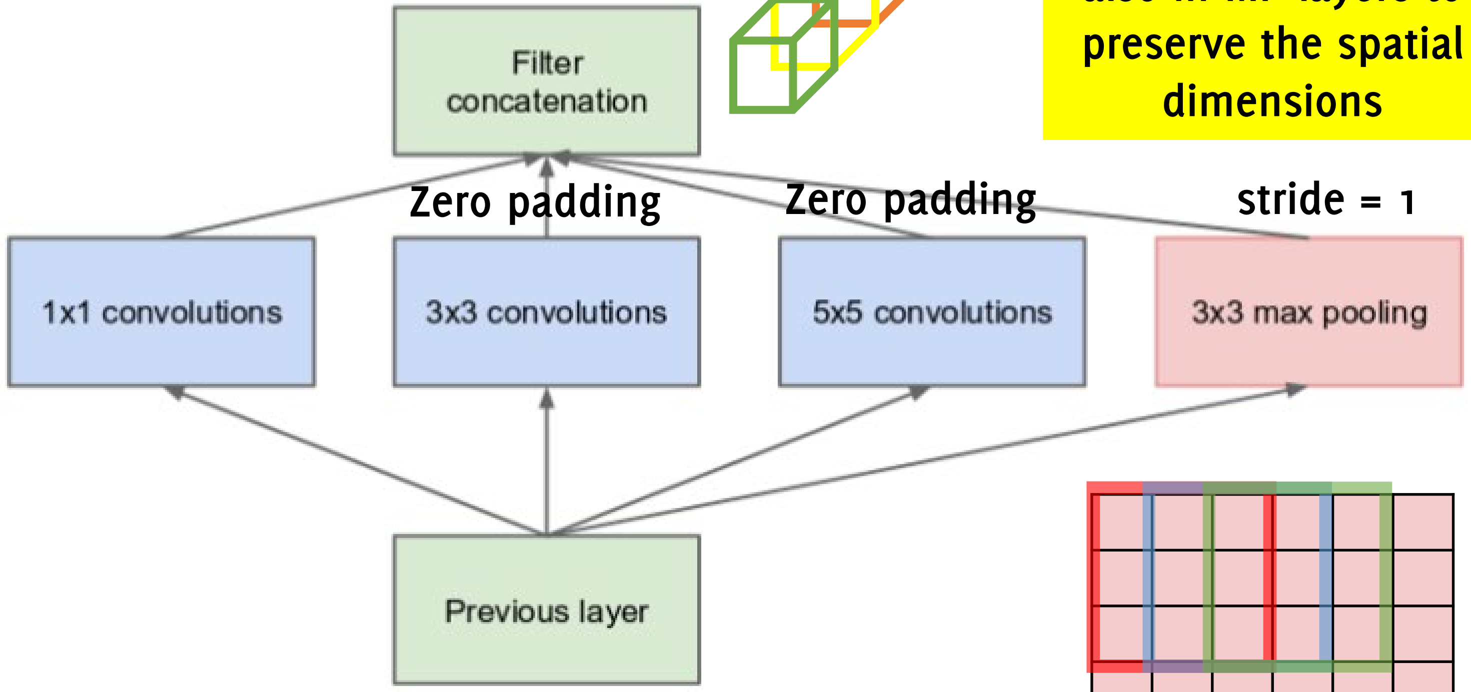
Inception Module (2014)

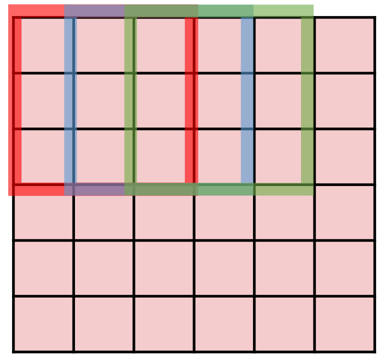
The solution is to **exploit multiple filter size** at the same level (1x1, 3x3, 5x5) and then **merge by concatenation** the output activation maps together



Stride=1 can be use also in MP layers to preserve the spatial dimensions

All the blocks preserve the spatial dimension by zero padding (convolutional filters) or by fractional stride (for Maxpooling)



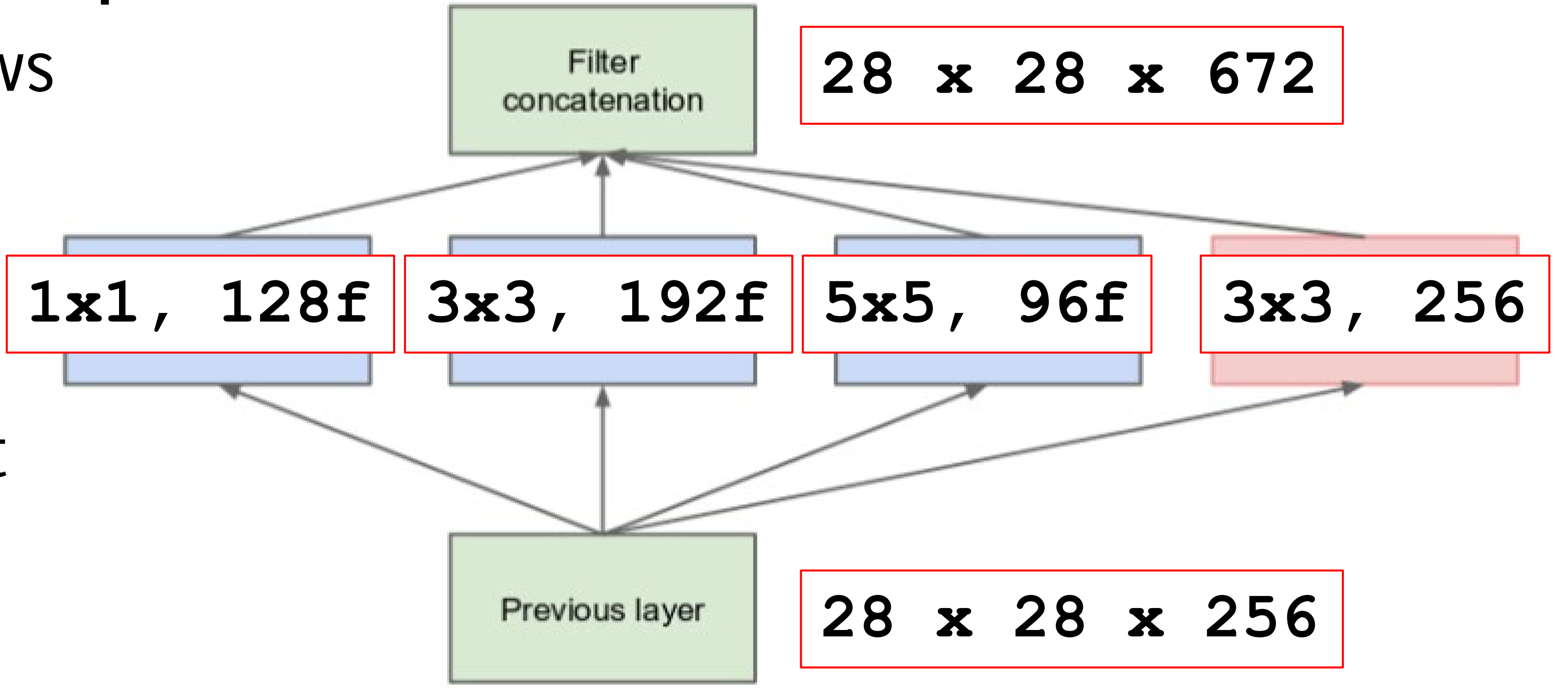


Thus, outputs can be concatenated depth-wise

Inception Module (2014)

The solution is to exploit multiple filter size at the same level (1x1, 3x3, 5x5) and then **merge by concatenation** the output activation maps together

- **Zero padding to preserve spatial size**
- The activation map grows much in depth
- A large number of operations to be performed due to the large depth of the input of each convolutional block: **854M operations** in this example



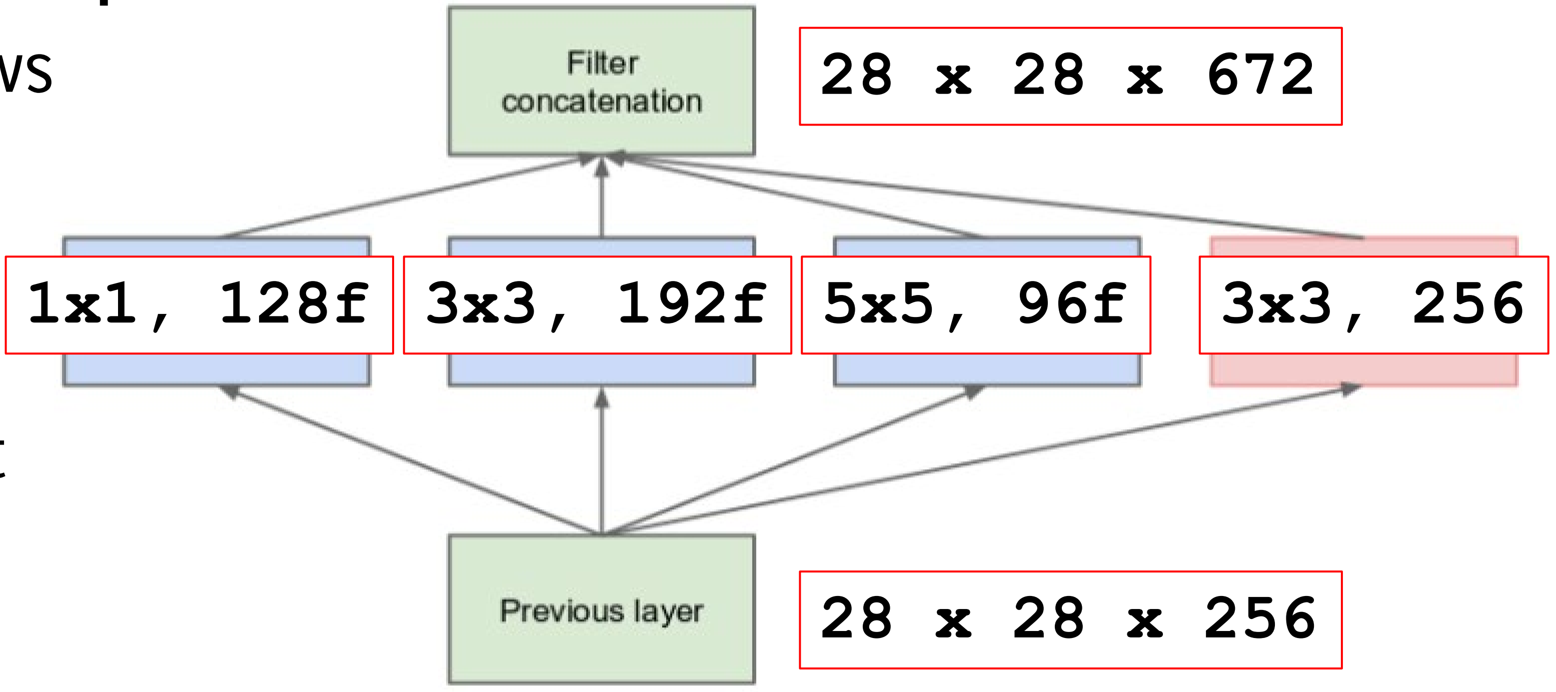
(a) Inception module, naïve version

Inception Module (2014)

The sol 3x3, 5x5) an togethe

The spatial extent is preserved, but the depth of the activation map is much expanded.
This is very expensive to compute

- **Zero**
- The activation map grows much in depth
- A large number of operations to be performed due to the large depth of the input of each convolutional block: **854M operations** in this example



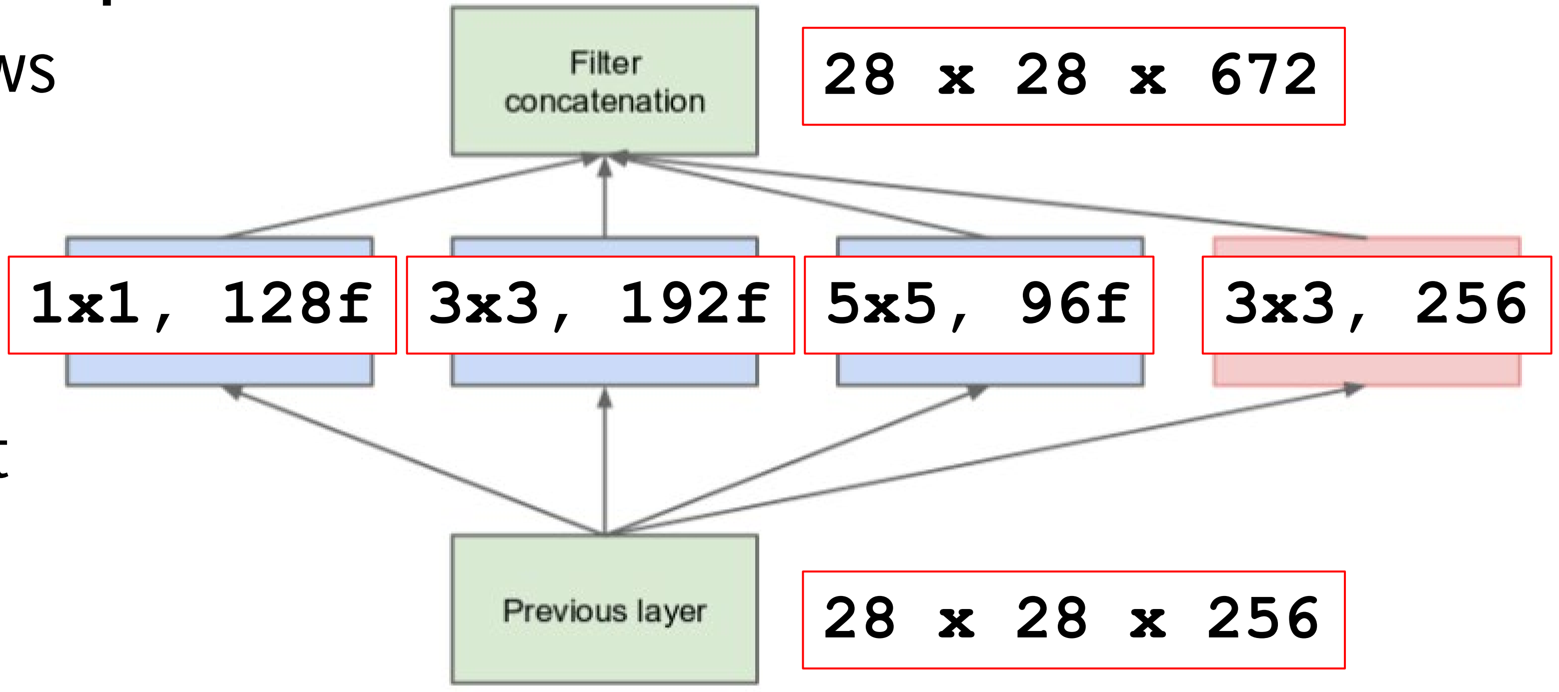
(a) Inception module, naïve version

Inception Module (2014)

The sol 3x3, 5x5) an togethe

Computational problems will get significantly worst when stacking multiple layers…

- **Zero padding to preserve spatial size**
- The activation map grows much in depth
- A large number of operations to be performed due to the large depth of the input of each convolutional block: **854M operations** in this example



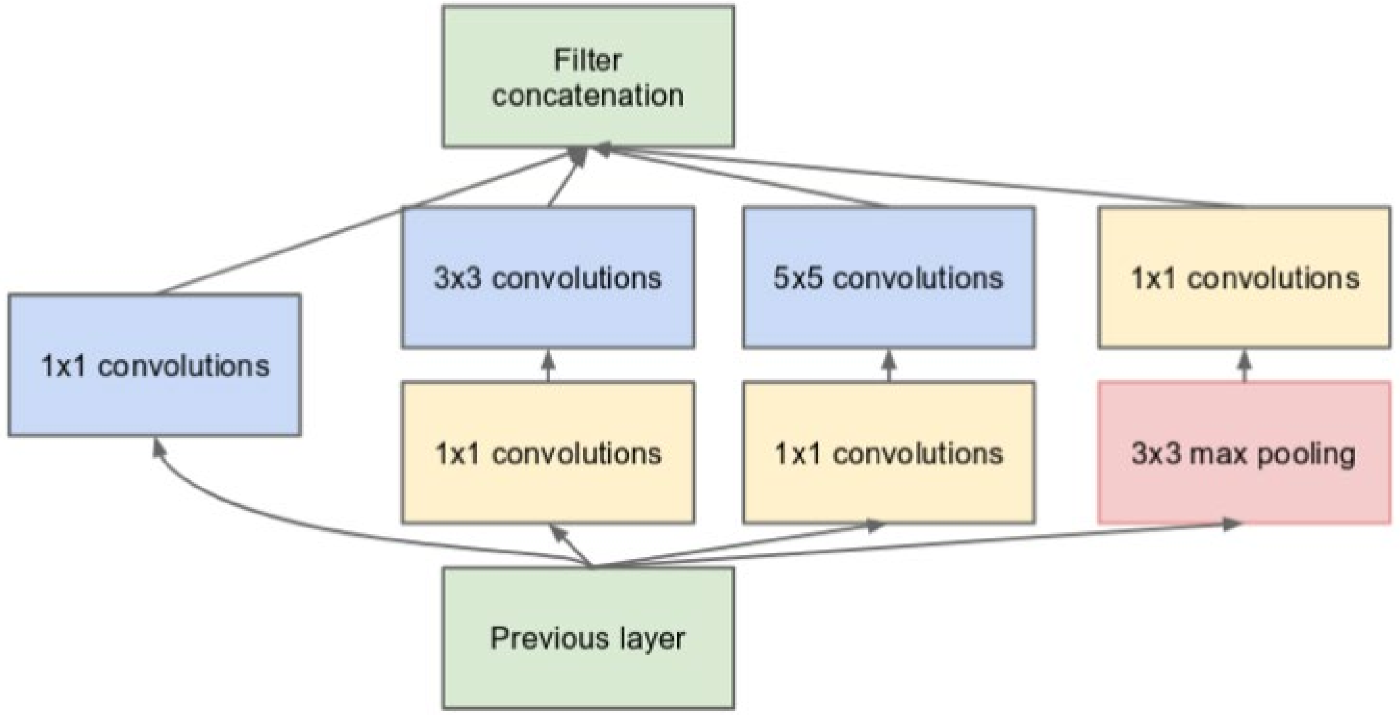
(a) Inception module, naïve version

Szegedy et al. "Going deeper with convolutions." *CVPR 2015*

Inception Module (2014)

Idea: To reduce the computational load of the network, the number of **input channels** of each conv layer is reduced thanks to **1x1 convolution** layers before the 3x3 and 5x5 convolutions

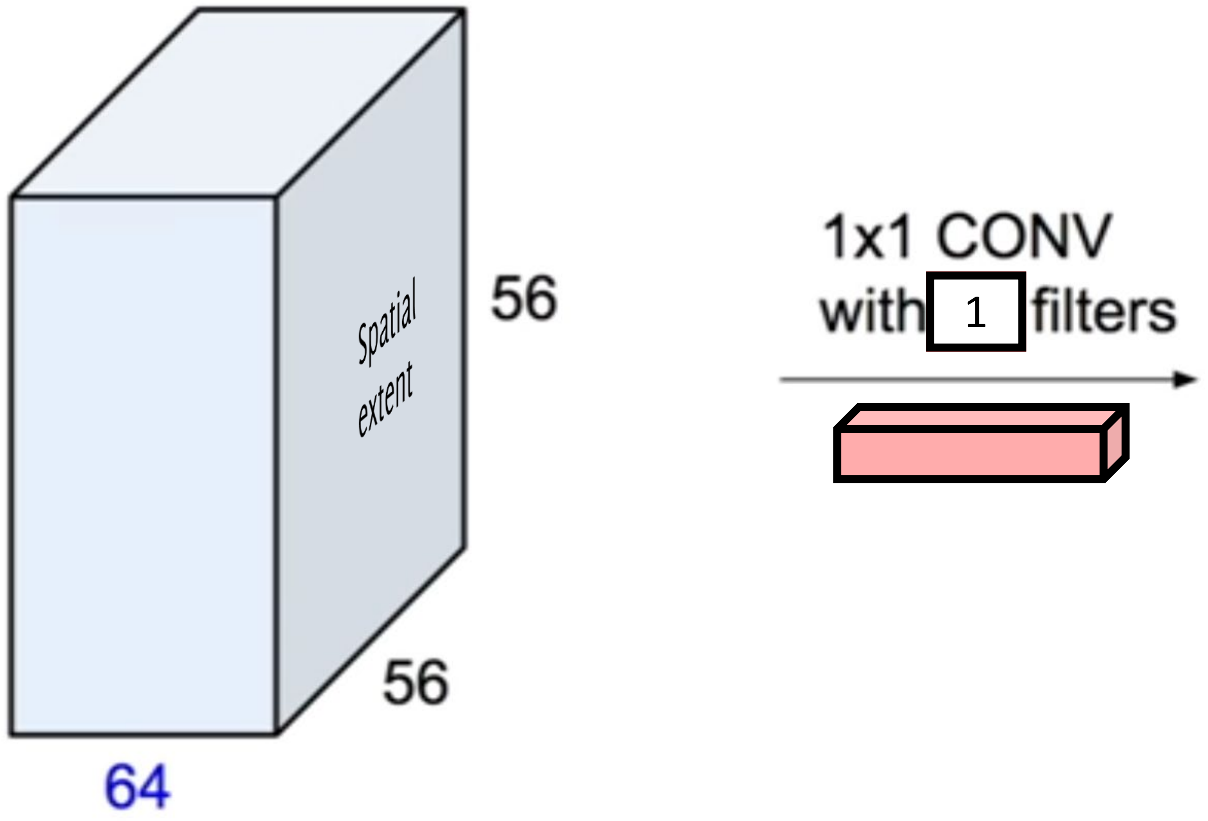
Using these 1x1 conv is referred to as **“bottleneck” layer**



(b) Inception module with dimension reductions

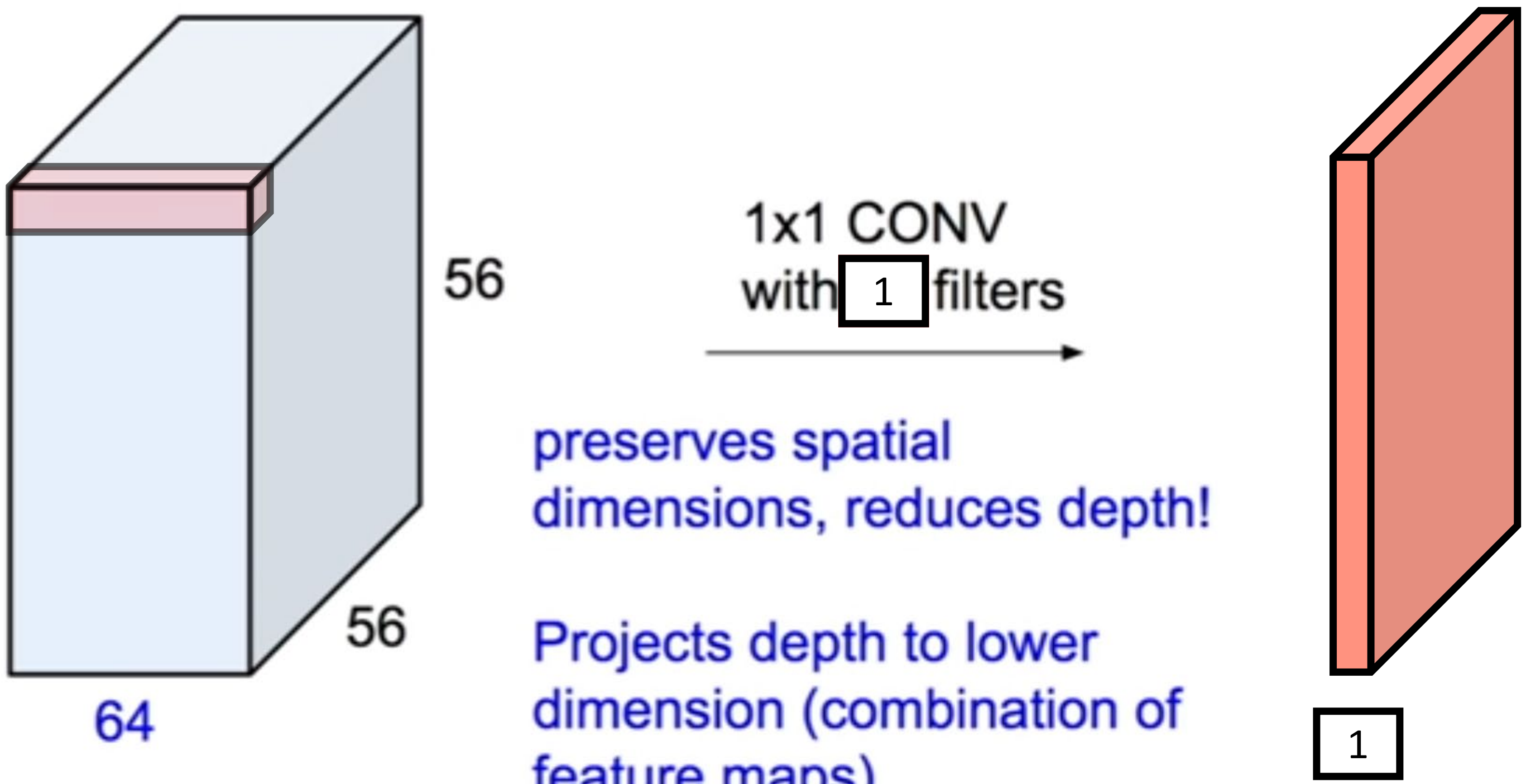
Szegedy et al. "Going deeper with convolutions." *CVPR 2015*

1x1 convolution layers as bottleneck



depth

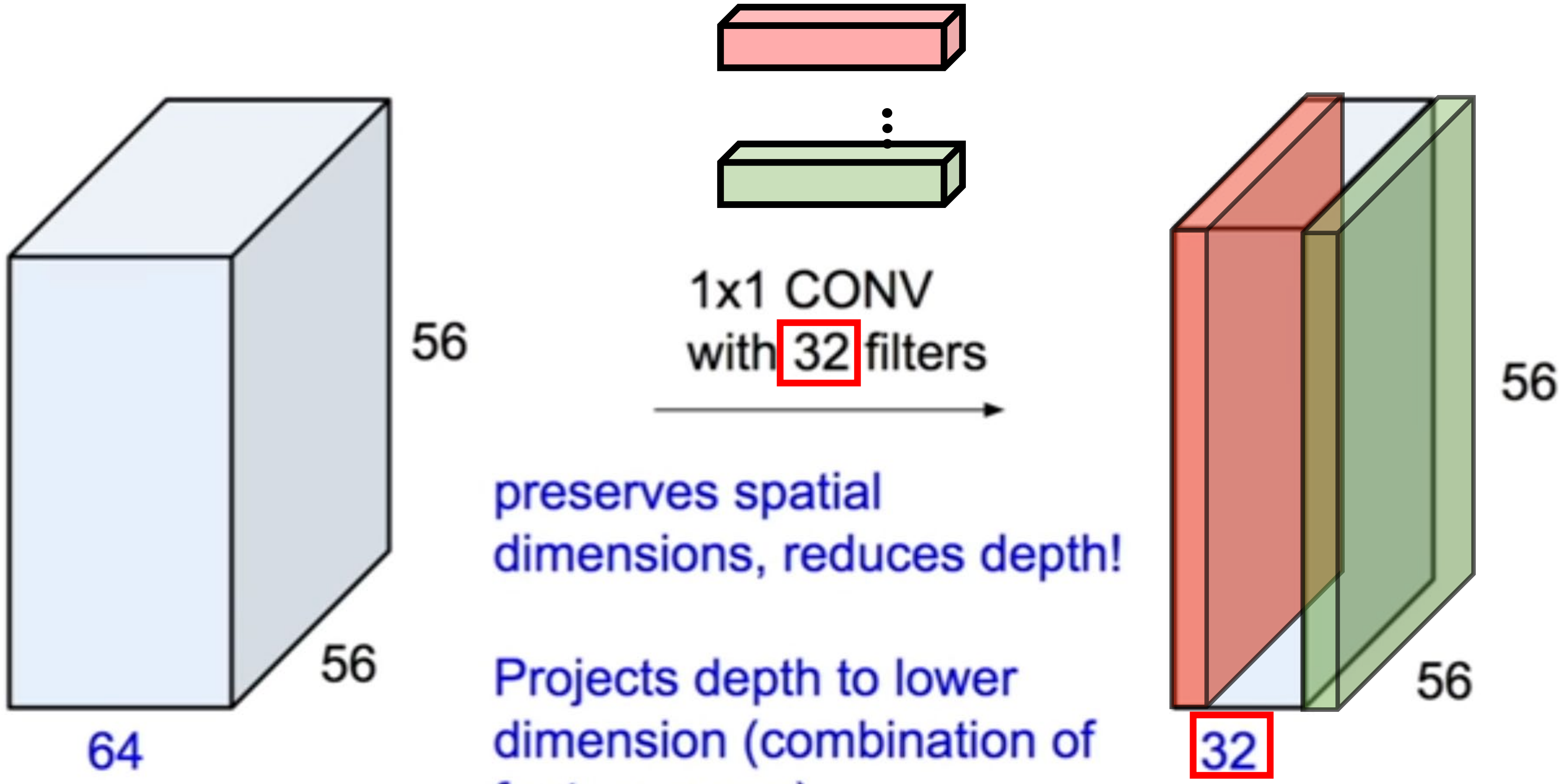
1x1 convolution layers as bottleneck



preserves spatial dimensions, reduces depth!

Projects depth to lower dimension (combination of feature maps)

1x1 convolution layers as bottleneck

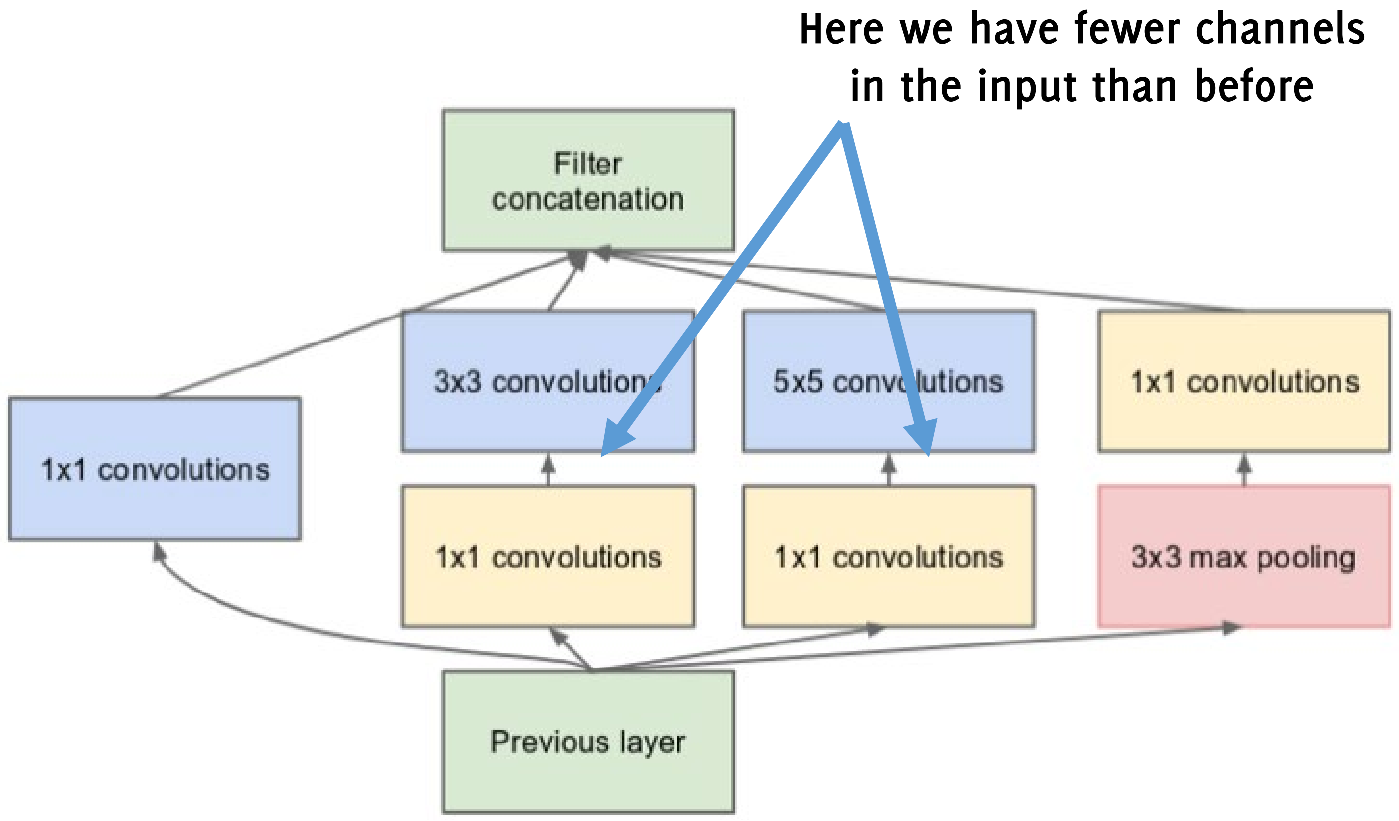

56
56
64
1x1 CONV
with 32 filters
preserves spatial dimensions, reduces depth!
Projects depth to lower dimension (combination of feature maps)
56
56
32

Inception Module (2014)

To reduce the computational load of the network, the number of **input channels** is reduced by adding an **1x1 convolution** layers before the 3x3 and 5x5 convolutions

The output volume has similar size, but the number of operation required is significantly reduced due to the 1x1 conv:
358M operations now

Adding 1x1 convolution layers increases the number of nonlinearities



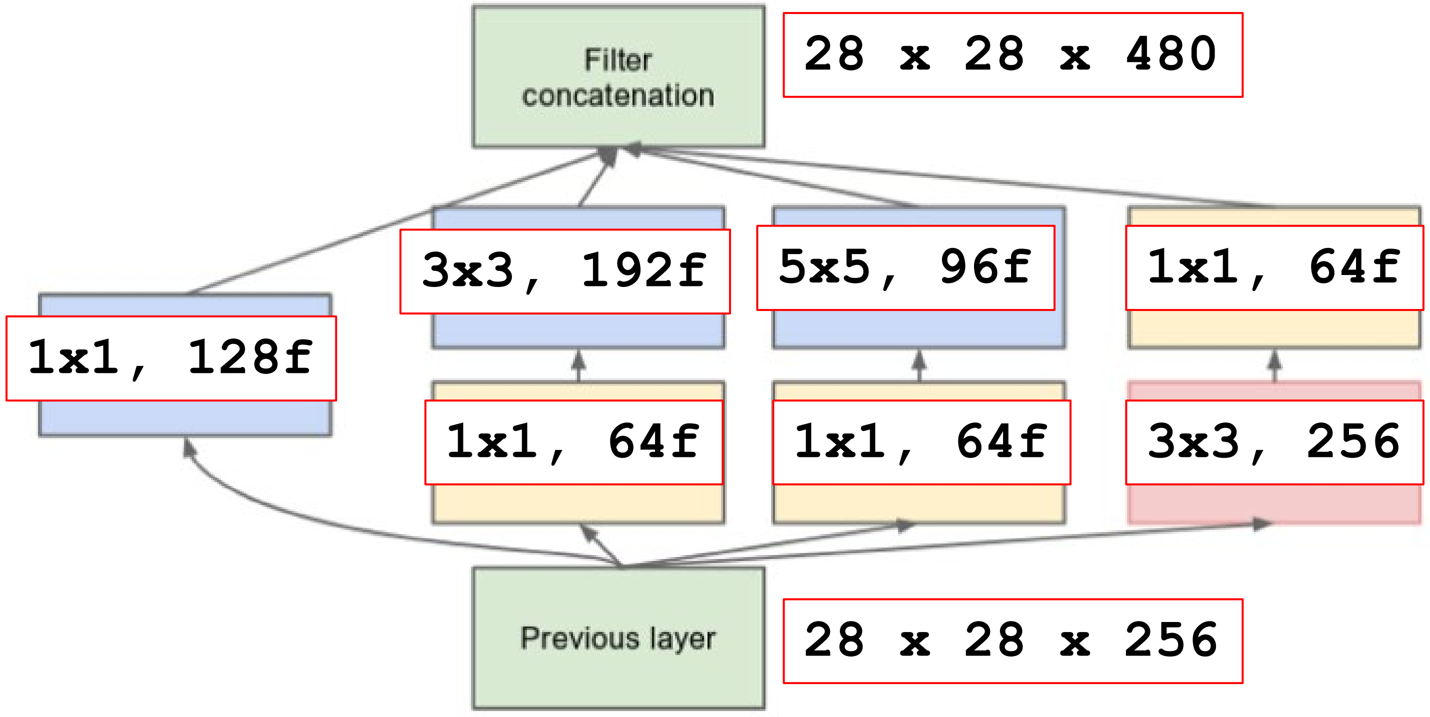
(b) Inception module with dimension reductions

Szegedy et al. "Going deeper with convolutions." *CVPR 2015*

Inception Module (2014)

To reduce the computational load of the network, the number of **input channels** is reduced by adding an **1x1 convolution** layers before the 3x3 and 5x5 convolutions

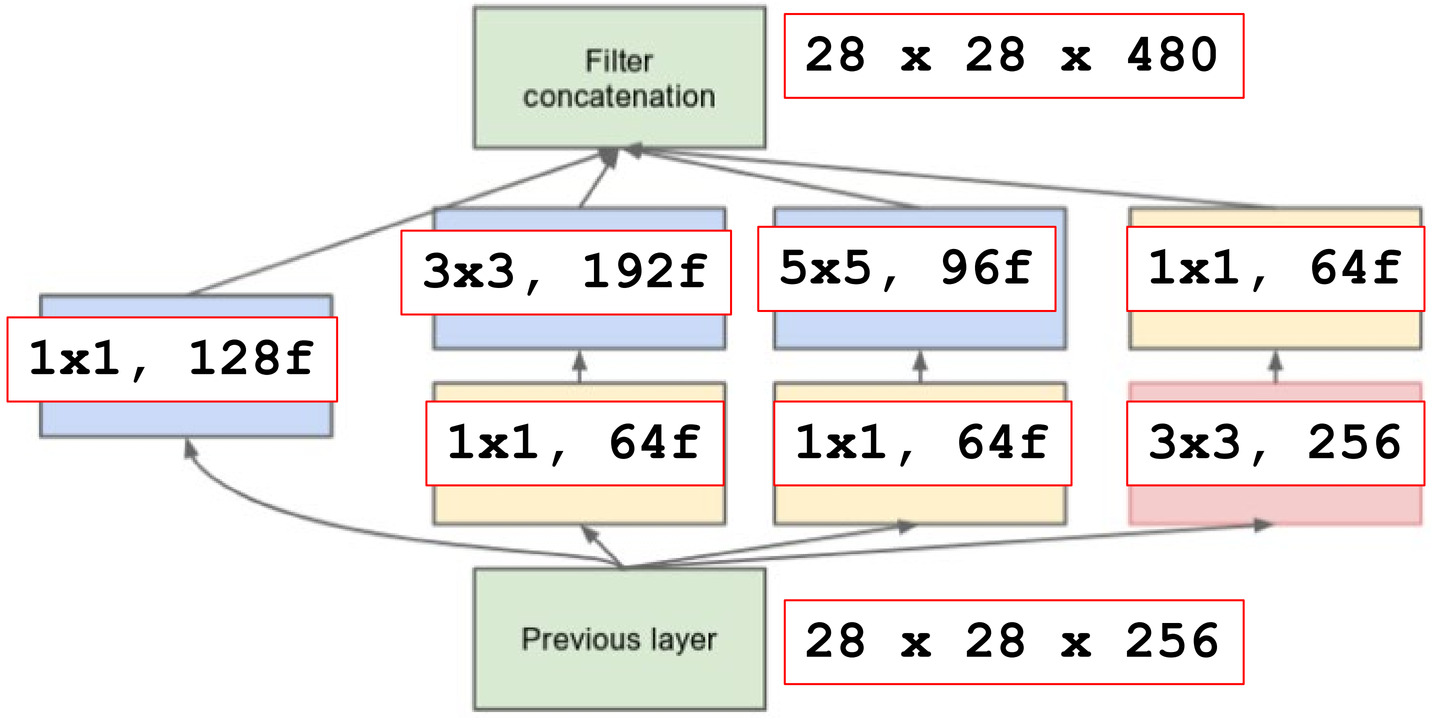
The output volume has similar size, but the number of operation required is significantly reduced due to the 1x1 conv:
358M operations now



(b) Inception module with dimension reductions

Inception Module (2014)

Network are no longer sequential.
There are parallel processing



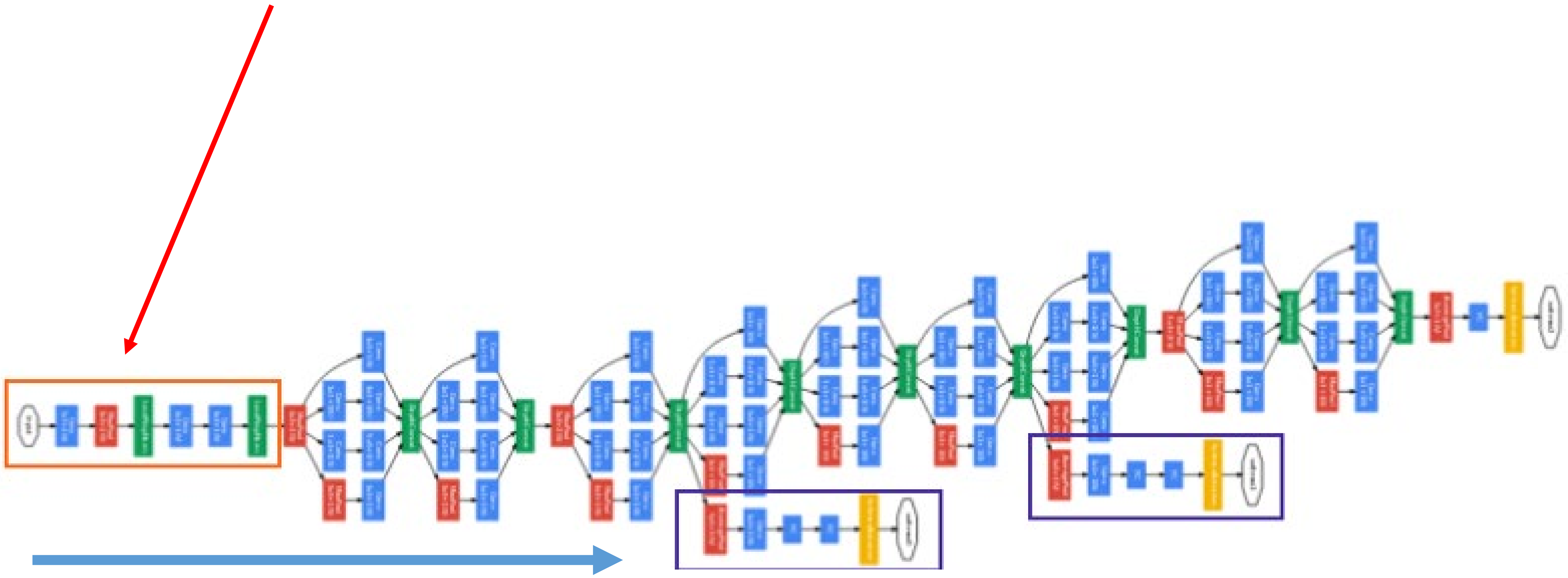
(b) Inception module with dimension reductions

Szegedy et al. "Going deeper with convolutions." *CVPR 2015*

GoogLeNet (2014)

GoogleNet stacks 27 layers considering pooling ones.

At the beginning there are two blocks of conv + pool layers

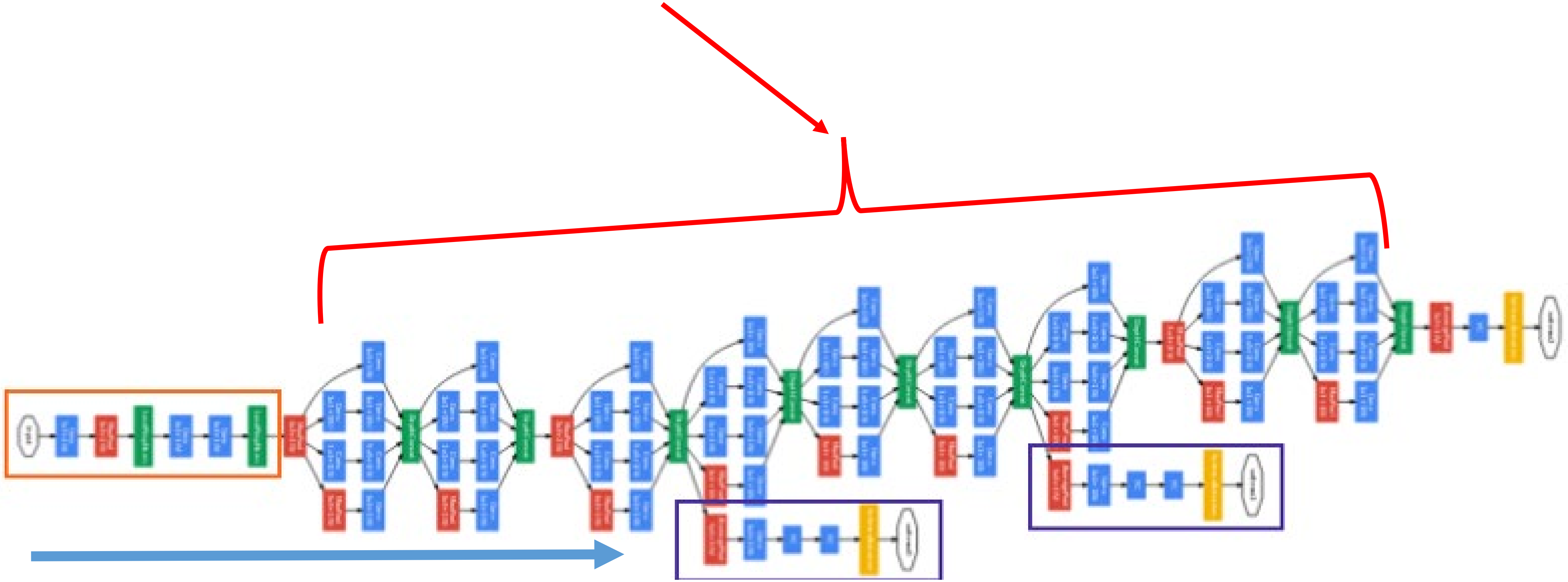


Szegedy et al. "Going deeper with convolutions." *CVPR 2015*

GoogLeNet (2014)

GoogleNet stacks 27 layers considering pooling ones.

Then, there are a stack of 9 of inception modules



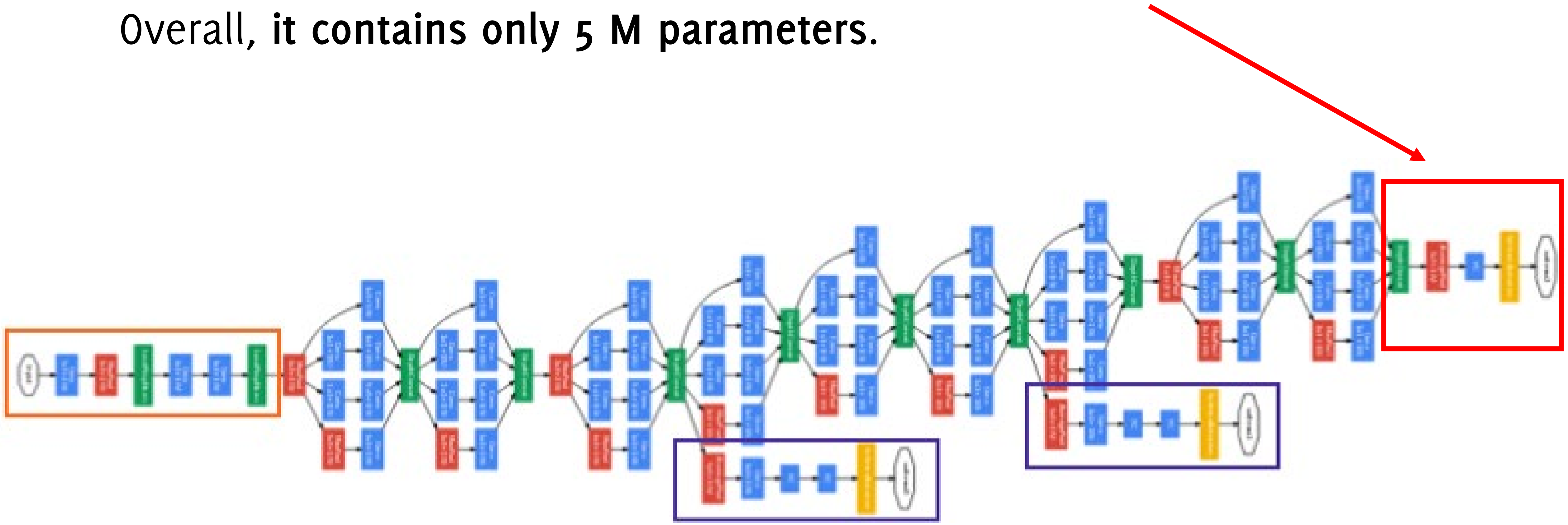
Szegedy et al. "Going deeper with convolutions." *CVPR 2015*

GoogLeNet (2014)

GoogleNet stacks 27 layers considering pooling ones.

No Fully connected layer at the end, **simple global averaging pooling (GAP) +** linear classifier + softmax.

Overall, **it contains only 5 M parameters.**

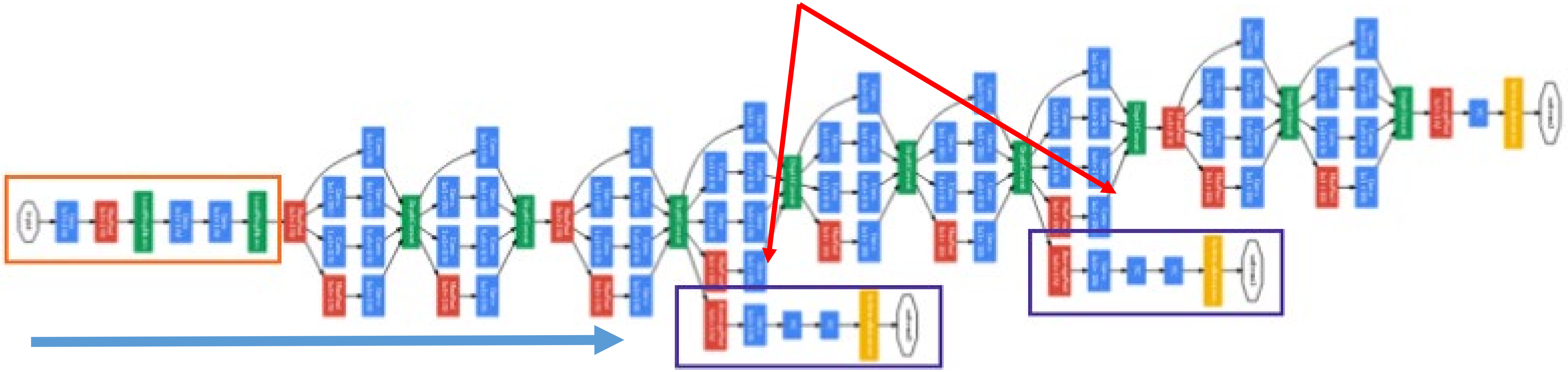


Szegedy et al. "Going deeper with convolutions." *CVPR 2015*

GoogLeNet (2014)

It also suffers of the **dying neuron problem**, therefore the authors **add two extra auxiliary classifiers** on the intermediate representation **to compute an intermediate loss that is used during training.**

You expect intermediate layers to provide meaningful features for classification as well.

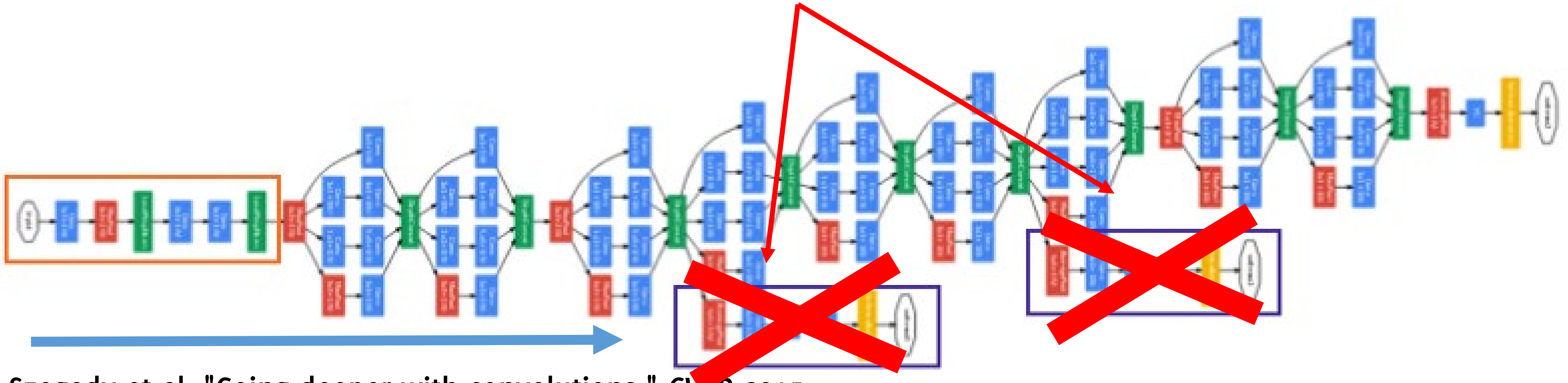


GoogLeNet (2014)

It also suffers of the **dying neuron problem**, therefore the authors **add two extra auxiliary classifiers** on the intermediate representation **to compute an intermediate loss that is used during training.**

You expect intermediate layers to provide meaningful features for classification as well.

Classification heads are then ignored / removed at inference time



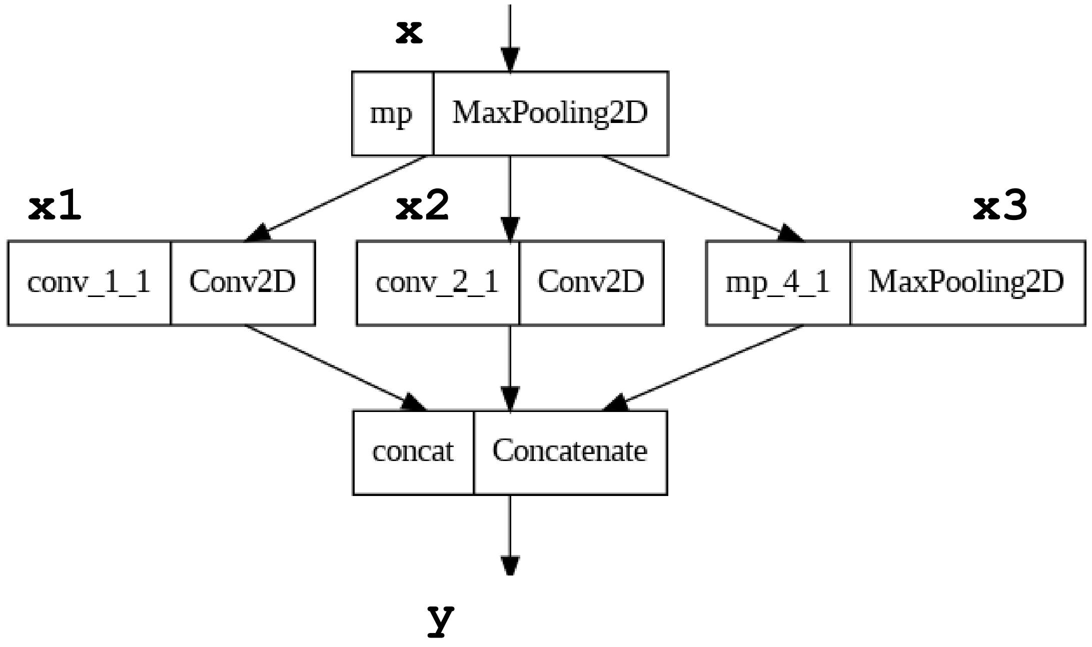
Szegedy et al. "Going deeper with convolutions." *CVPR 2015*

3 Take home messages

1. 1x1 convolutions: enable bottlenecks that reduce the number of operations and parameters of the network.
2. Blocks made of multiple connections instead of having a single tread.
3. Additional losses: you might want to train your network on additional tasks just for improving training convergence.

Inception Block in Kears

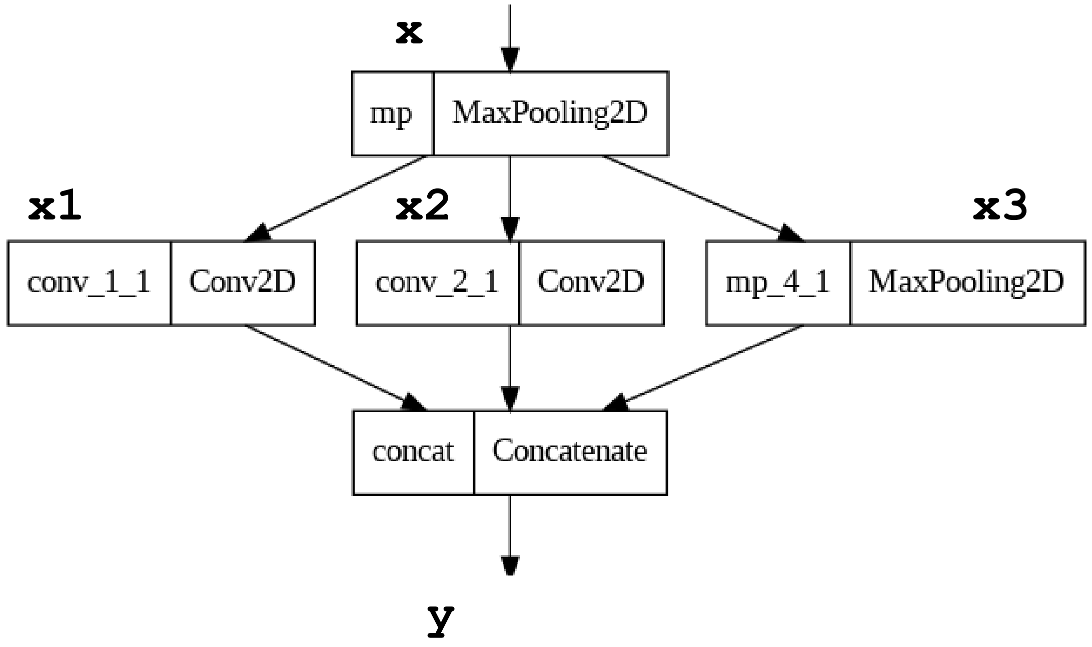
```
# input x
x = tfkl.MaxPooling2D(name='mp')(x)
x1 = tfkl.Conv2D(32,
   kernel_size=1,
   padding='same',
   activation='relu',
   name='conv_1_1')(x)
x2 = tfkl.Conv2D(64,
    kernel_size=1,
    padding='same',
    activation='relu',
   name='conv_2_1')(x)
x4 = tfkl.MaxPooling2D((3,3),
   strides=(1,1),
   padding='same',
   name='mp_4_1',)(x)
y = tfkl.Concatenate(
   axis=-1,
   name='concat')([x1, x2, x4])
```



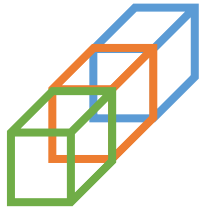
G. Boracchi

Inception Block in Kears

```
# input x
x = tfkl.MaxPooling2D(name='mp')(x)
x1 = tfkl.Conv2D(32,
   kernel_size=1,
   padding='same',
   activation='relu',
   name='conv_1_1')(x)
x2 = tfkl.Conv2D(64,
    kernel_size=1,
    padding='same',
    activation='relu',
   name='conv_2_1')(x)
x4 = tfkl.MaxPooling2D((3,3),
   strides=(1,1),
   padding='same',
   name='mp_4_1',)(x)
y = tfkl.Concatenate(
   axis=-1,
   name='concat')([x1, x2, x4])
```

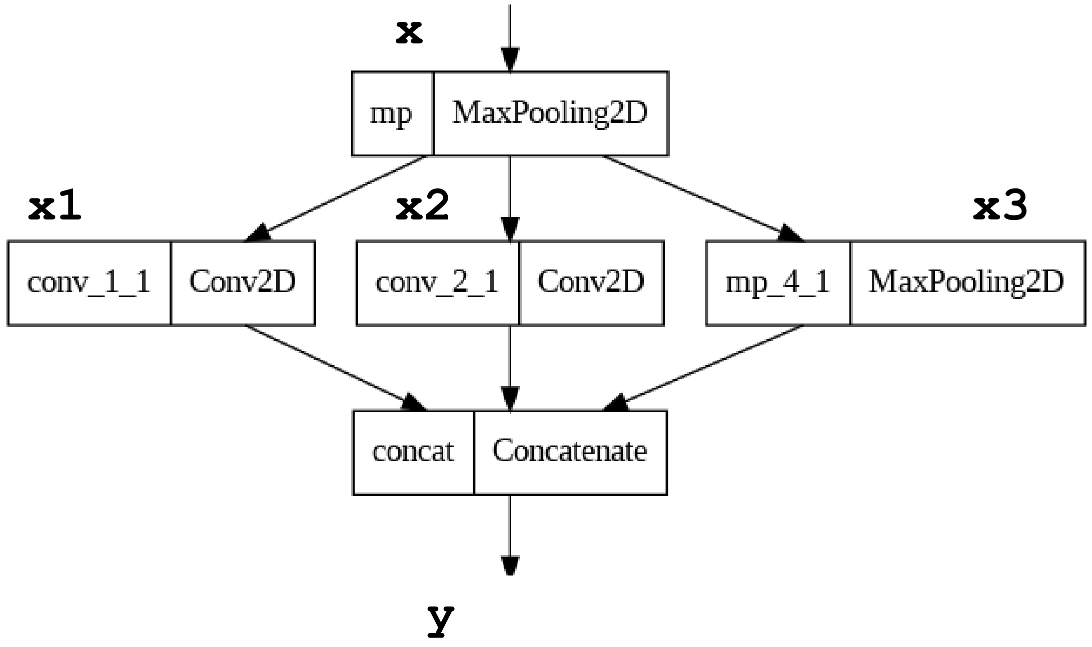


Concatenate layer to stack multiple activations along the last axis (**`axis=-1`**)



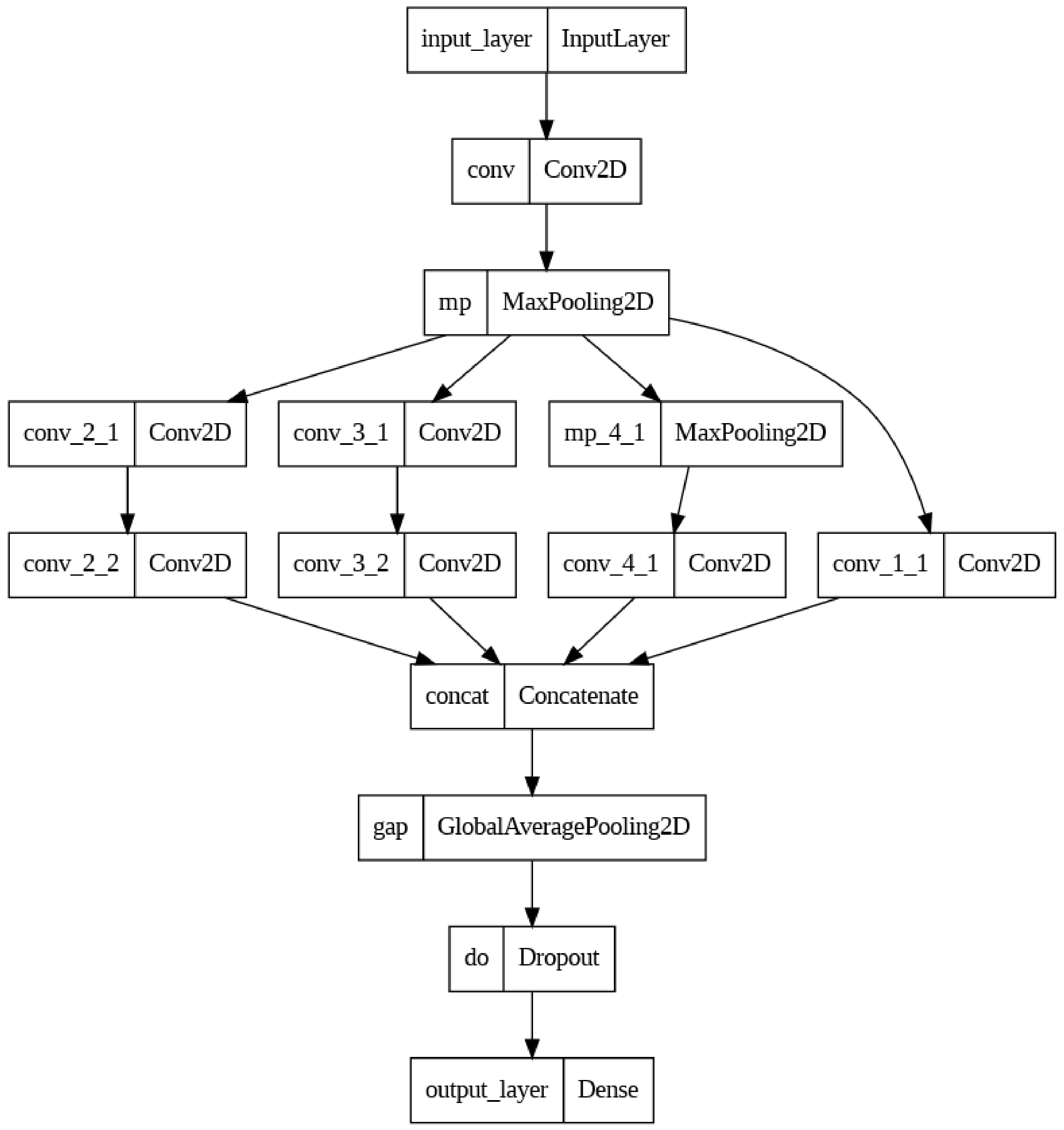
Inception Block in Kears

```
# input x
x = tfkl.MaxPooling2D(name='mp')(x)
x1 = tfkl.Conv2D(32,
   kernel_size=1,
   padding='same',
   activation='relu',
   name='conv_1_1')(x)
x2 = tfkl.Conv2D(64,
   kernel_size=1,
   padding='same',
   activation='relu',
   name='conv_2_1')(x)
x4 = tfkl.MaxPooling2D((3,3),
   strides=(1,1),
   padding='same',
   name='mp_4_1',)(x)
y = tfkl.Concatenate(
   axis=-1,
   name='concat')([x1, x2, x4])
```

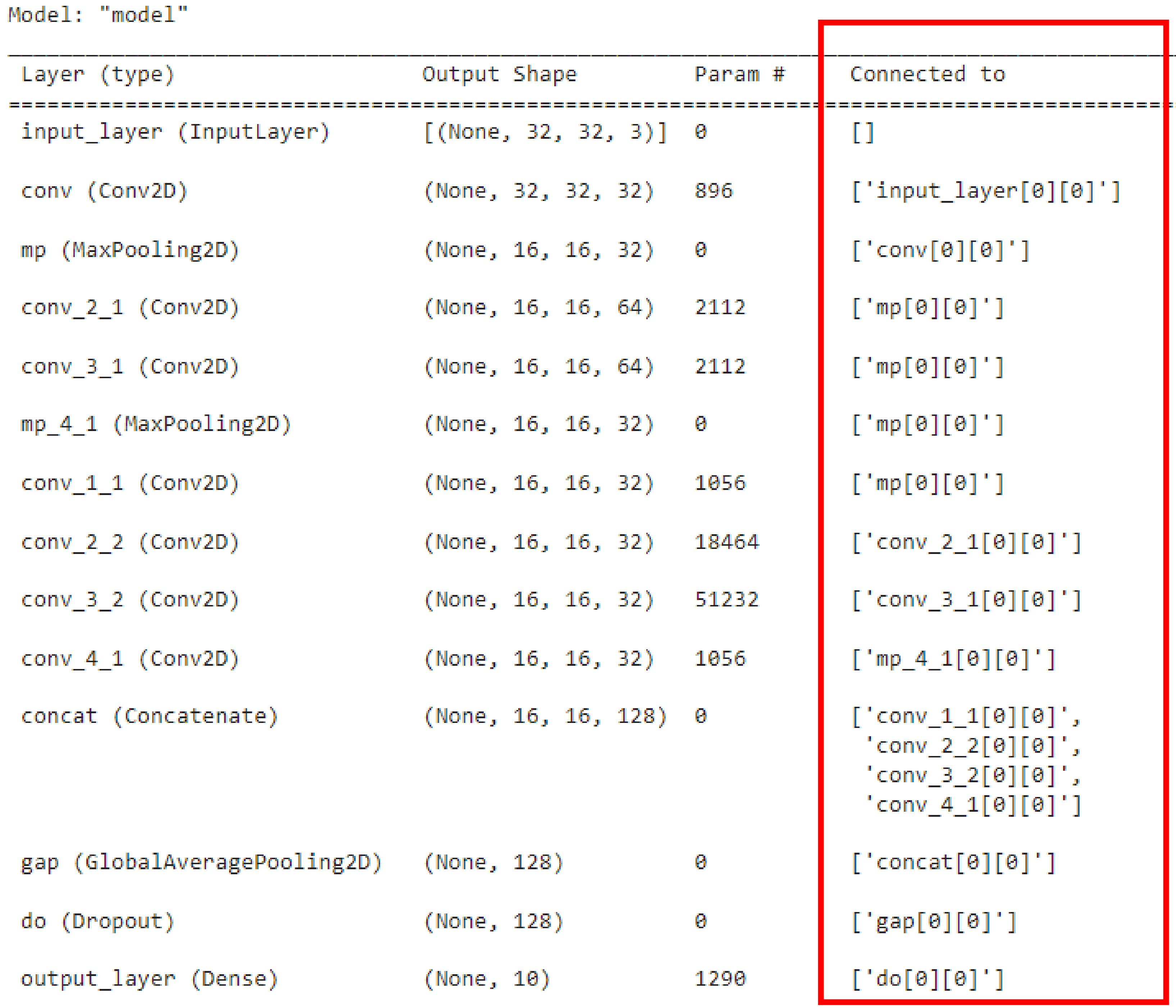


Spatial dimension should be preserved both by padding and stride in maxpooling

… and more

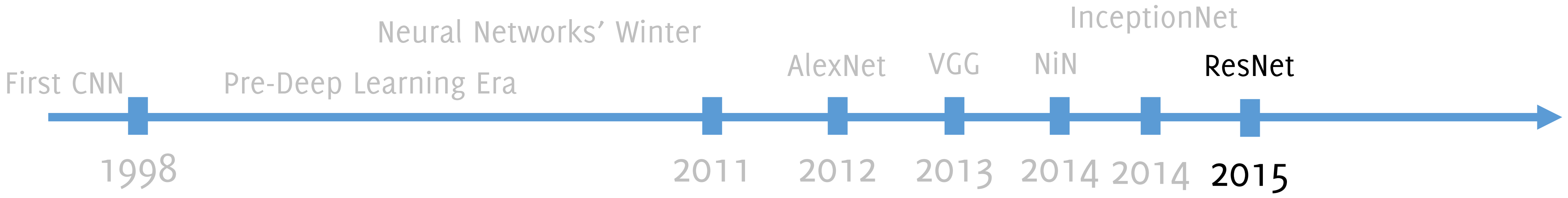

input_layer
InputLayer
conv
Conv2D
mp
MaxPooling2D
conv_2_1
Conv2D
conv_3_1
Conv2D
mp_4_1
MaxPooling2D
conv_2_2
Conv2D
conv_3_2
Conv2D
conv_4_1
Conv2D
conv_1_1
Conv2D
concat
Concatenate
gap
GlobalAveragePooling2D
do
Dropout
output_layer
Dense

model.summary()



```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_layer (InputLayer)       [(None, 32, 32, 3)]  0           []

 conv (Conv2D)                  (None, 32, 32, 32)   896         ['input_layer[0][0]']

 mp (MaxPooling2D)              (None, 16, 16, 32)   0           ['conv[0][0]']

 conv_2_1 (Conv2D)              (None, 16, 16, 64)   2112        ['mp[0][0]']

 conv_3_1 (Conv2D)              (None, 16, 16, 64)   2112        ['mp[0][0]']

 mp_4_1 (MaxPooling2D)          (None, 16, 16, 32)   0           ['mp[0][0]']

 conv_1_1 (Conv2D)              (None, 16, 16, 32)   1056        ['mp[0][0]']

 conv_2_2 (Conv2D)              (None, 16, 16, 32)   18464       ['conv_2_1[0][0]']

 conv_3_2 (Conv2D)              (None, 16, 16, 32)   51232       ['conv_3_1[0][0]']

 conv_4_1 (Conv2D)              (None, 16, 16, 32)   1056        ['mp_4_1[0][0]']

 concat (Concatenate)           (None, 16, 16, 128)  0           ['conv_1_1[0][0]',
                                                                  'conv_2_2[0][0]',
                                                                  'conv_3_2[0][0]',
                                                                  'conv_4_1[0][0]']


 gap (GlobalAveragePooling2D)   (None, 128)          0           ['concat[0][0]']

 do (Dropout)                   (None, 128)          0           ['gap[0][0]']

 output_layer (Dense)           (None, 10)           1290        ['do[0][0]']
```

ResNet: Residual Learning



G. Boracchi

Deep Residual Learning for Image Recognition

Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun
Microsoft Research
{kahe, v-xiangz, v-shren, jiansun}@microsoft.com

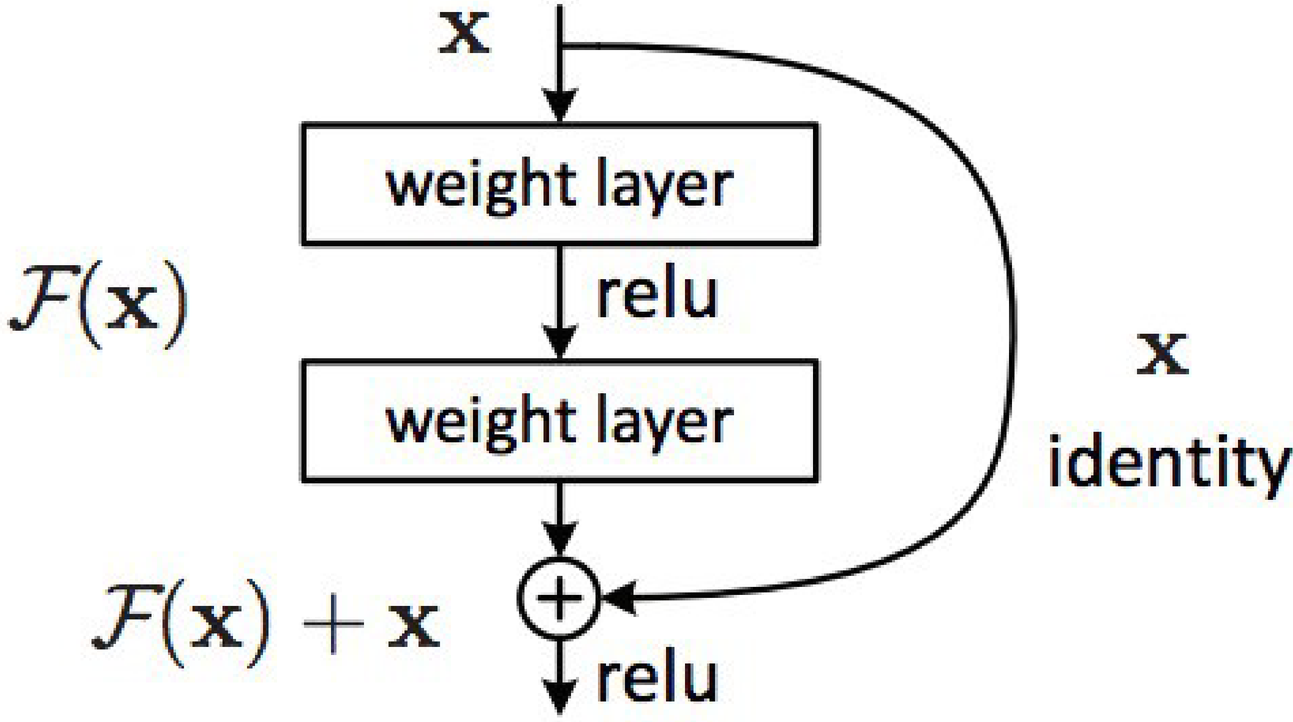
ResNet (2015)

Very Deep network: 152 layers for a deep network trained on Imagenet!

1202 layers on CIFAR!

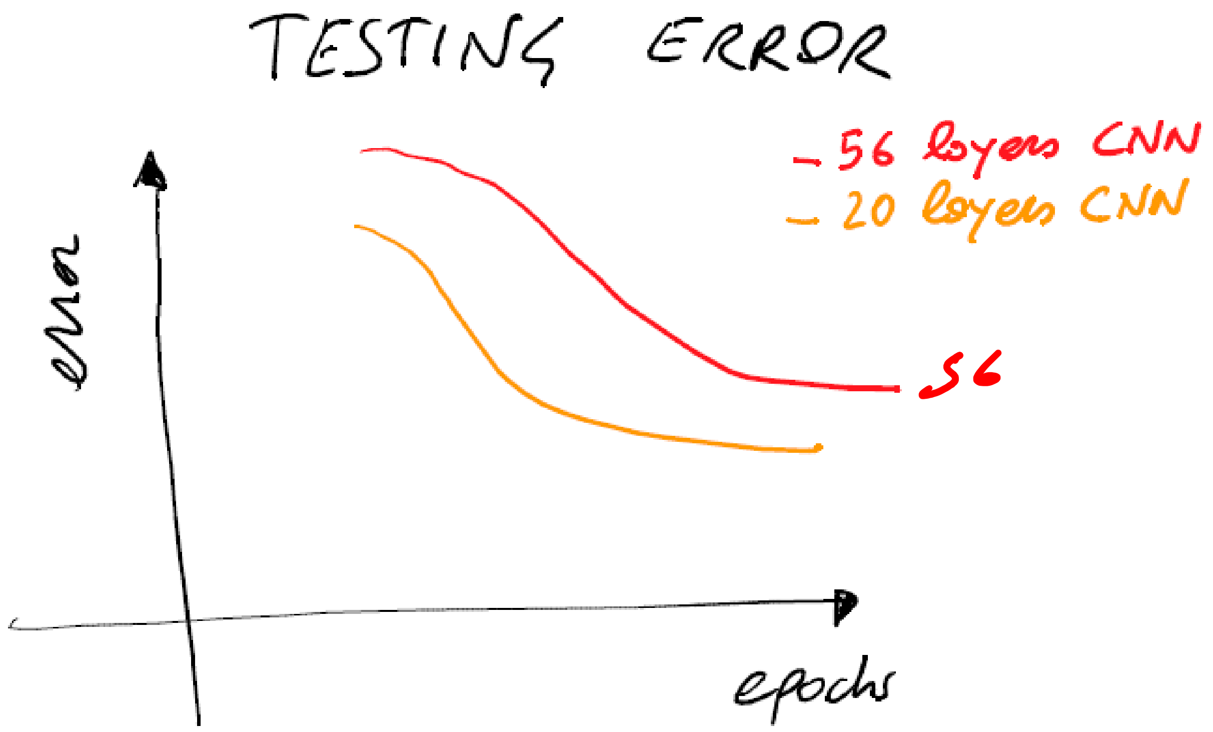
2015 ILSVR winner both localization and classification (3.57% top 5 classification error). **Better than *human performance***

The main investigation was: is it possible to continuosly improve accuracy by stacking more and more layers



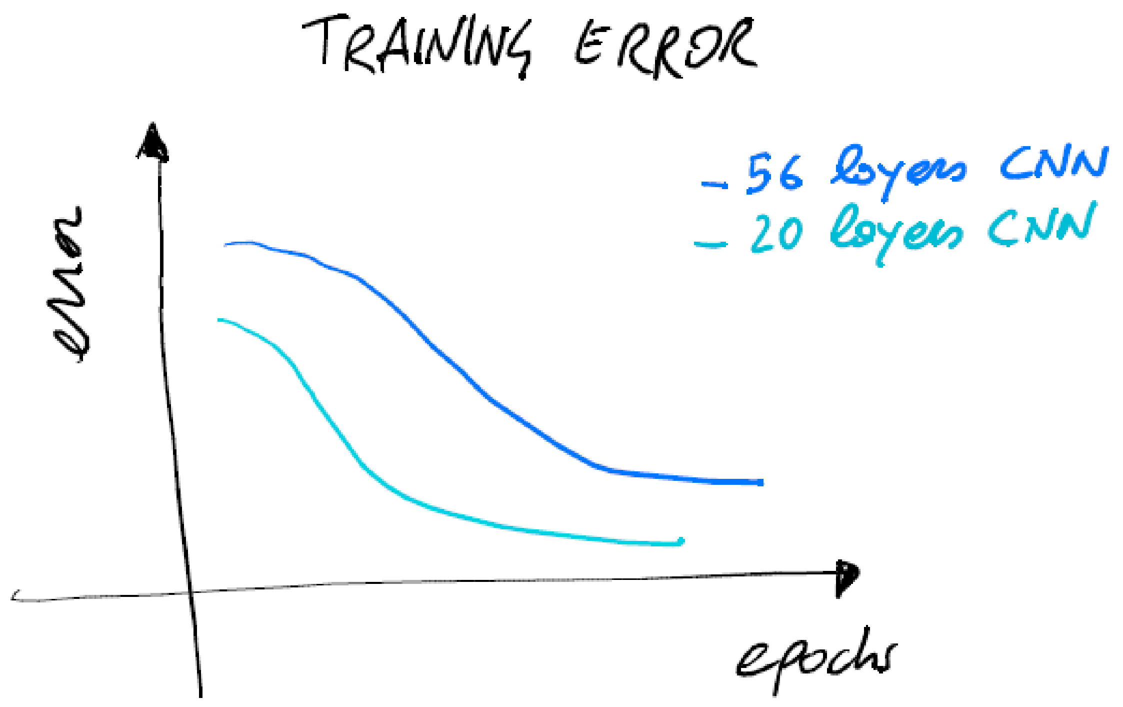
ResNet (2015): The rationale

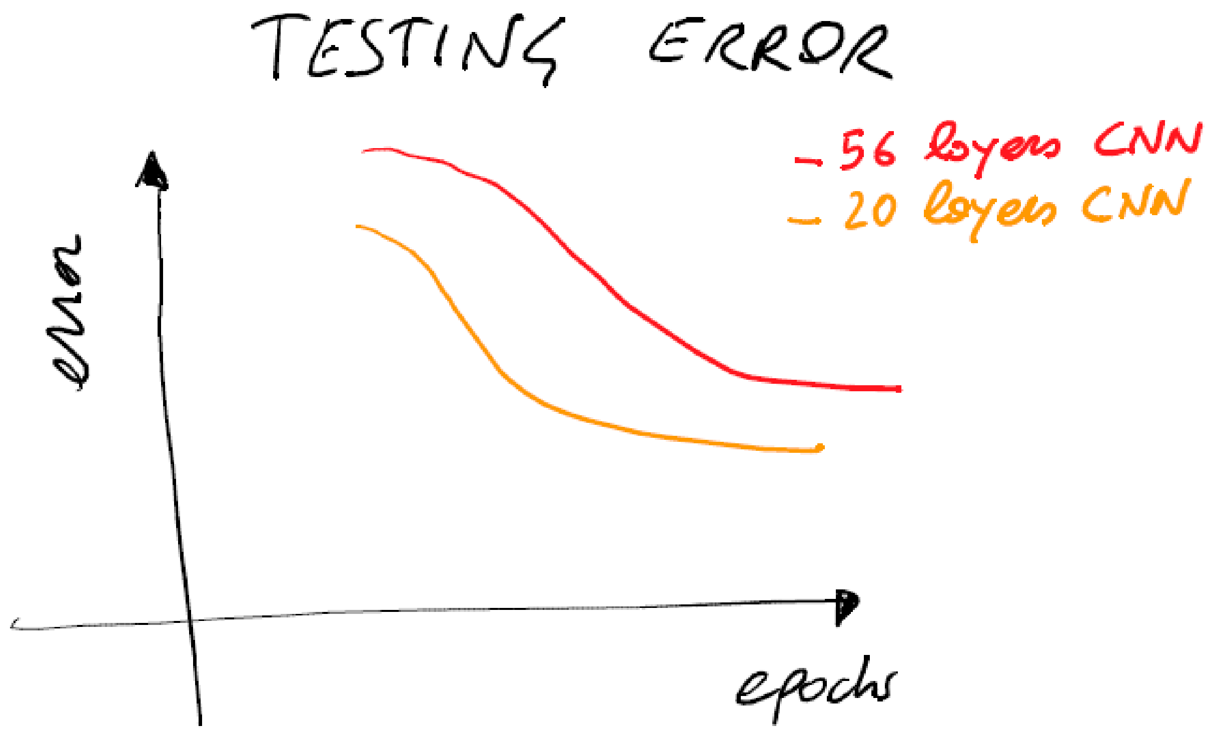
Empirical observation: **Increasing the network depth**, by stacking an increasingly number of layers, **does not always improve performance**



ResNet (2015): The rationale

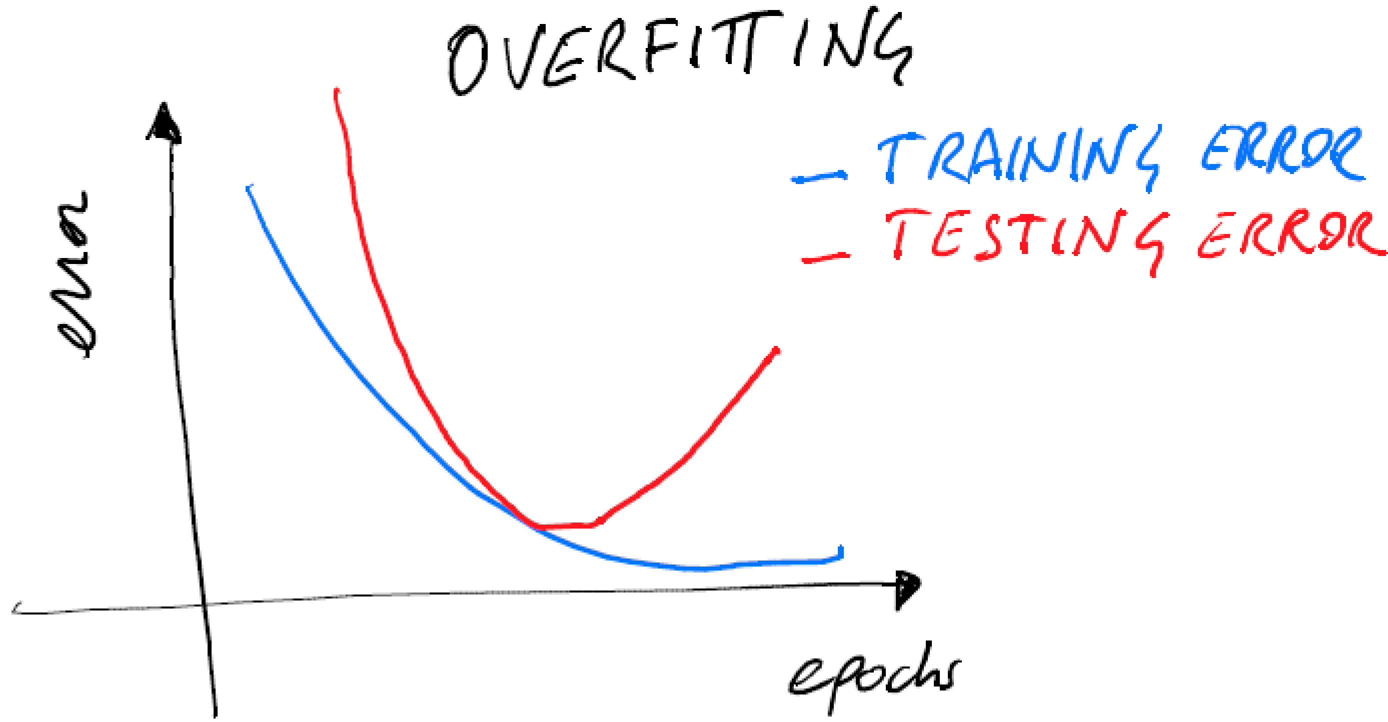
But **this is not due to overfitting**, since the same trend is shown in the training error





ResNet (2015): The rationale

But **this is not due to overfitting**, since the same trend is shown in the training error, while for overfitting we have that training and test error diverge

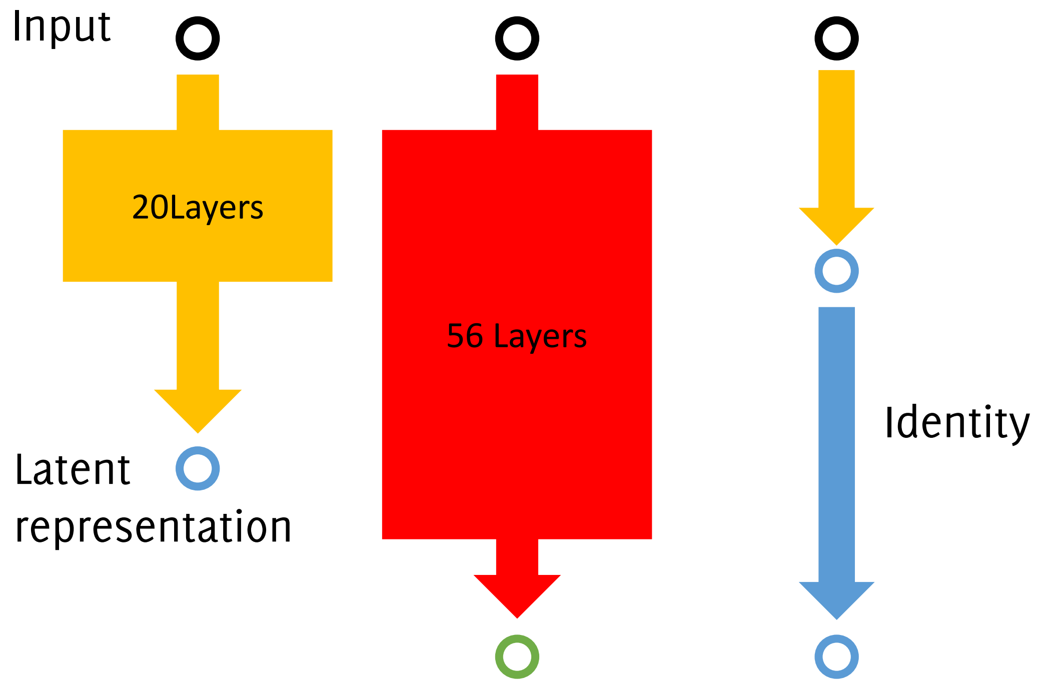


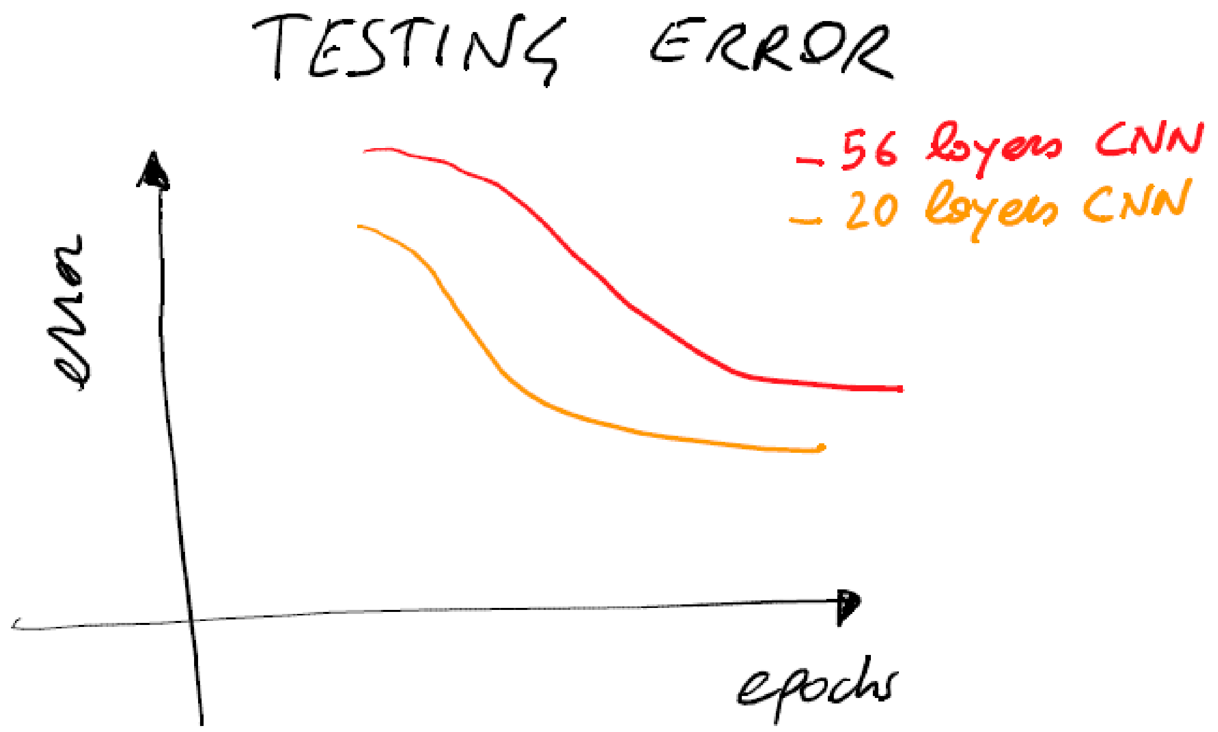
G. Boracchi

ResNet (2015): the intuition

Deeper model are harder to optimize than shallower models.

However, we might in principle **copy the parameters of the shallow** network **in the deeper one** and then **in the remaining part**, set the weights to **yield an identity mapping.**





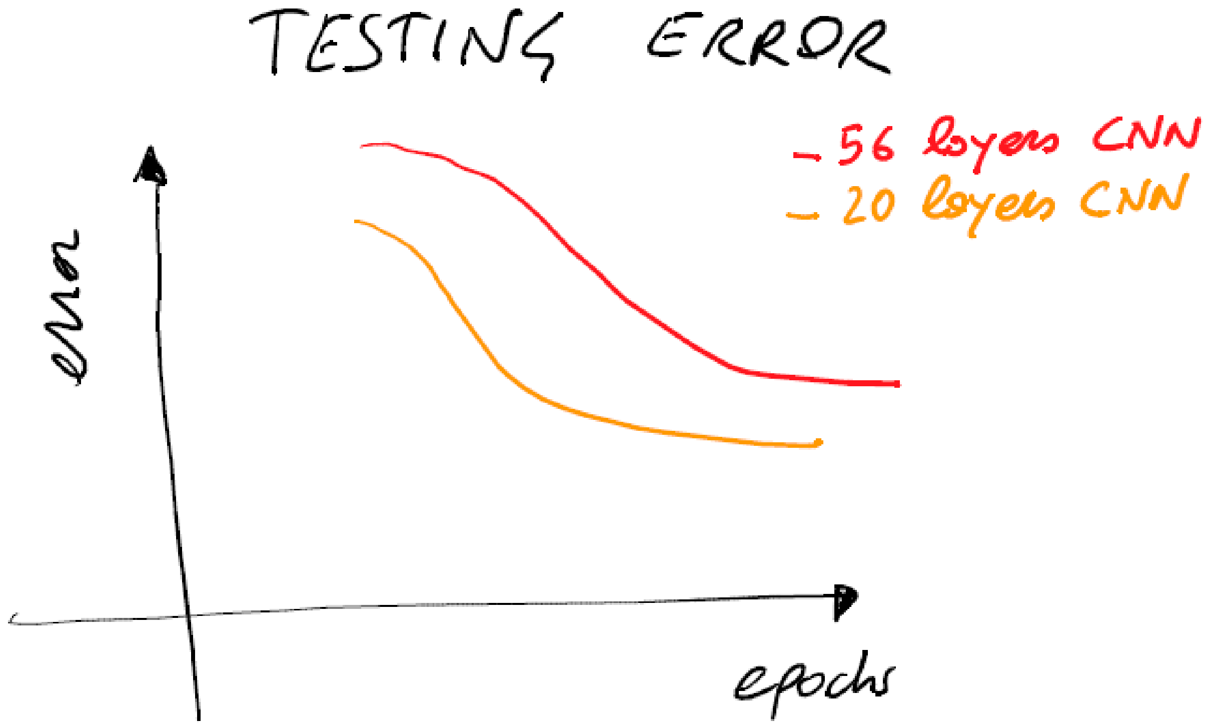
ResNet (2015): the intuition

Deeper model are harder to optimize than shallower models.

However, we might in principle **copy the parameters of the shallow** network **in the deeper one** and then in the remaining part, set the weights to **yield an identity mapping**.

Therefore, deeper networks should be in principle as good as the shallow ones

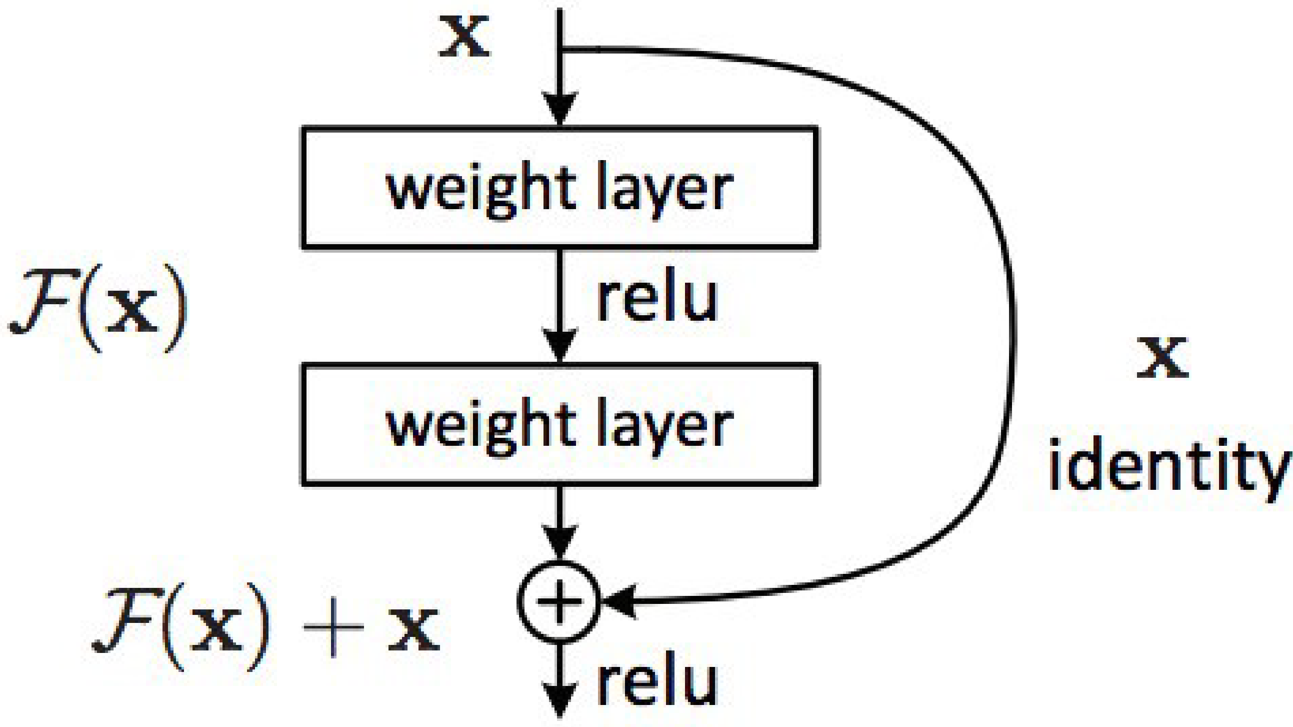
Since the experimental evidence, is different **the identity function is not easy to learn!**



ResNet: Very deep by residual connections

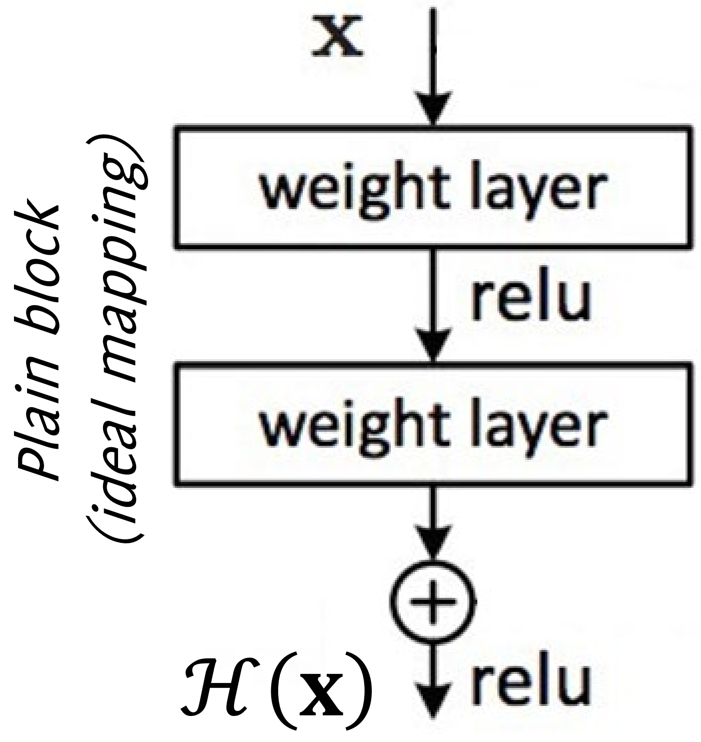
Adding an "identity shortcut connection" :

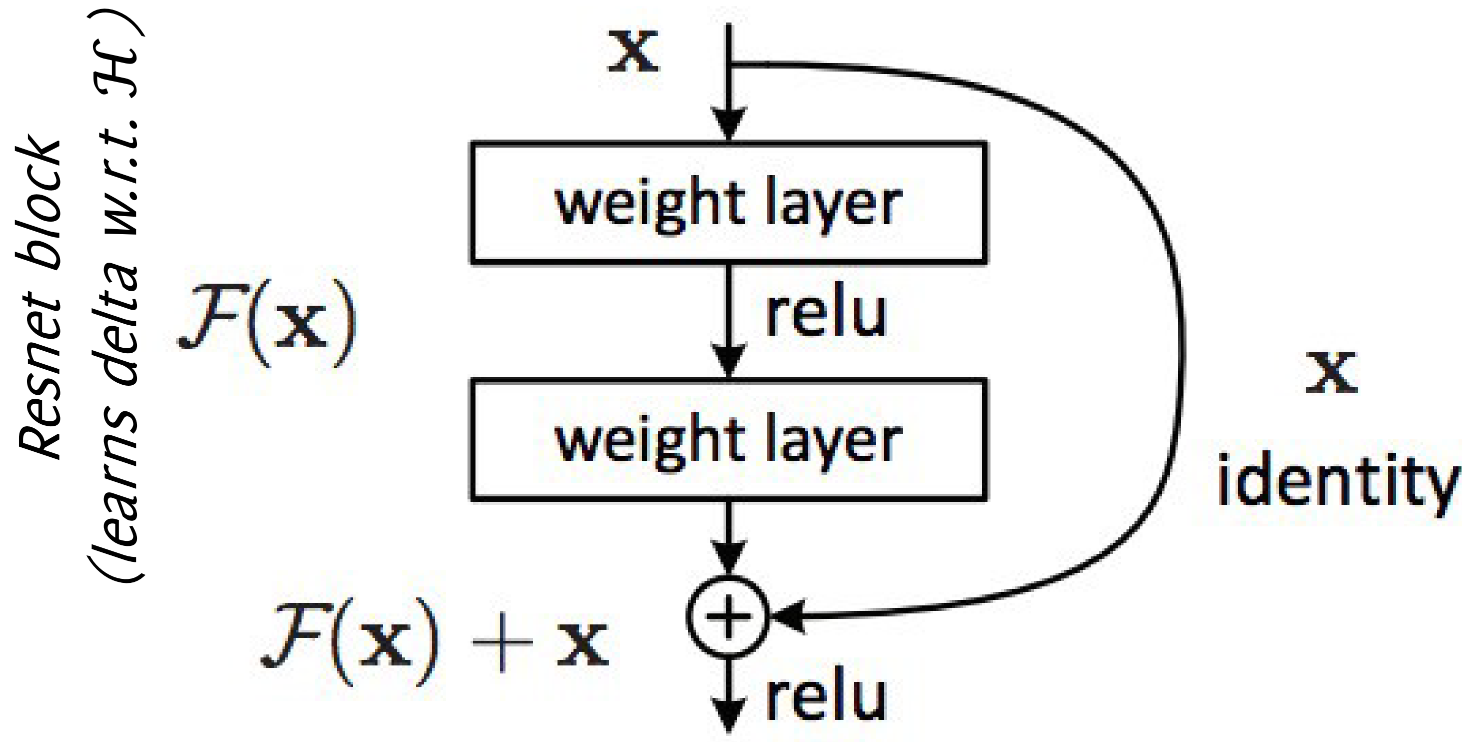
- helps in **mitigating the vanishing gradient problem** and enables deeper architectures
- Does not add parameters
- In case the **previous network was optimal**, the **weights to be learned goes to zero** and information is propagated by the identity
- The network can still be trained through back-propagation



ResNet: Very deep by residual connections

Intuition: force the network to learn a different task in each block. If $\mathcal{H}(\mathbf{x})$ is the **ideal mapping** to be learned from a plain network, by skip connections **we force the network to learn $\mathcal{F}(\mathbf{x}) = \mathcal{H}(\mathbf{x}) - \mathbf{x}$,** here the term *residual.*



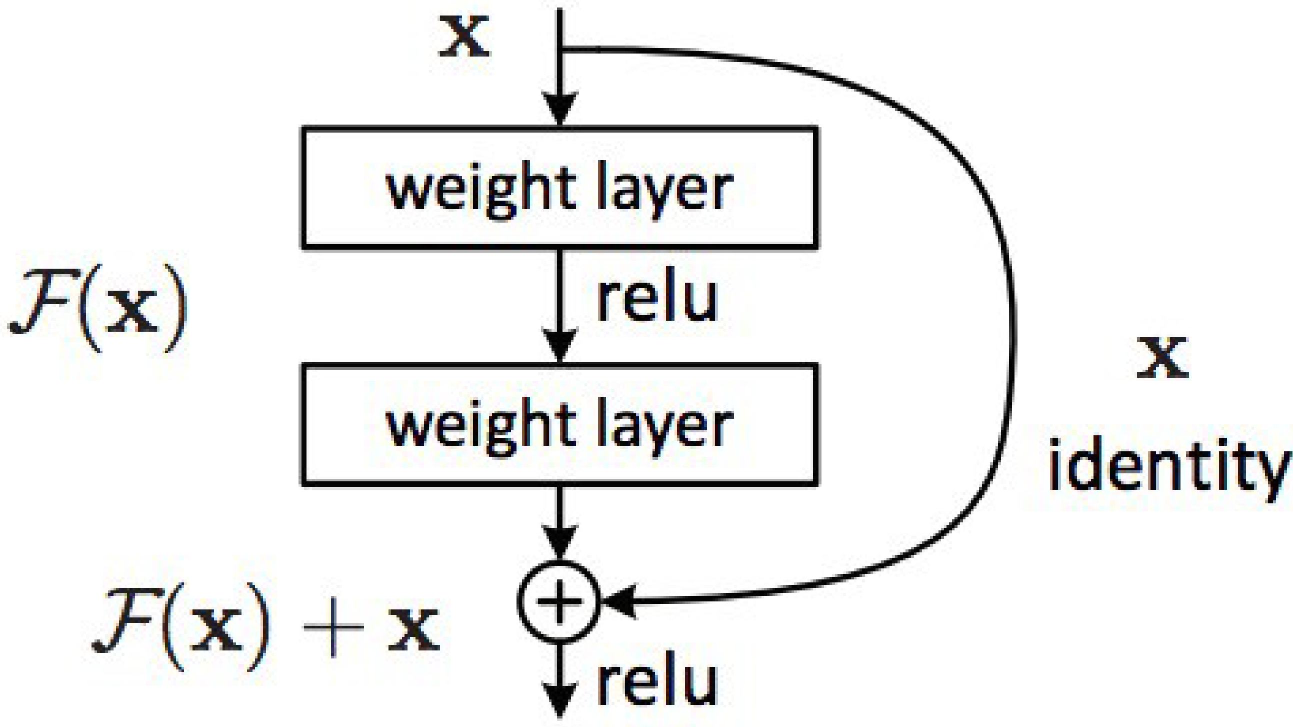


Resnet block
(learns delta w.r.t. $\mathcal{H}$)
x
$\mathcal{F}(\mathbf{x})$
weight layer
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Kaiming He, et al. "Deep Residual Learning for Image Recognition“ CVPR 2016

ResNet: Very deep by residual connections

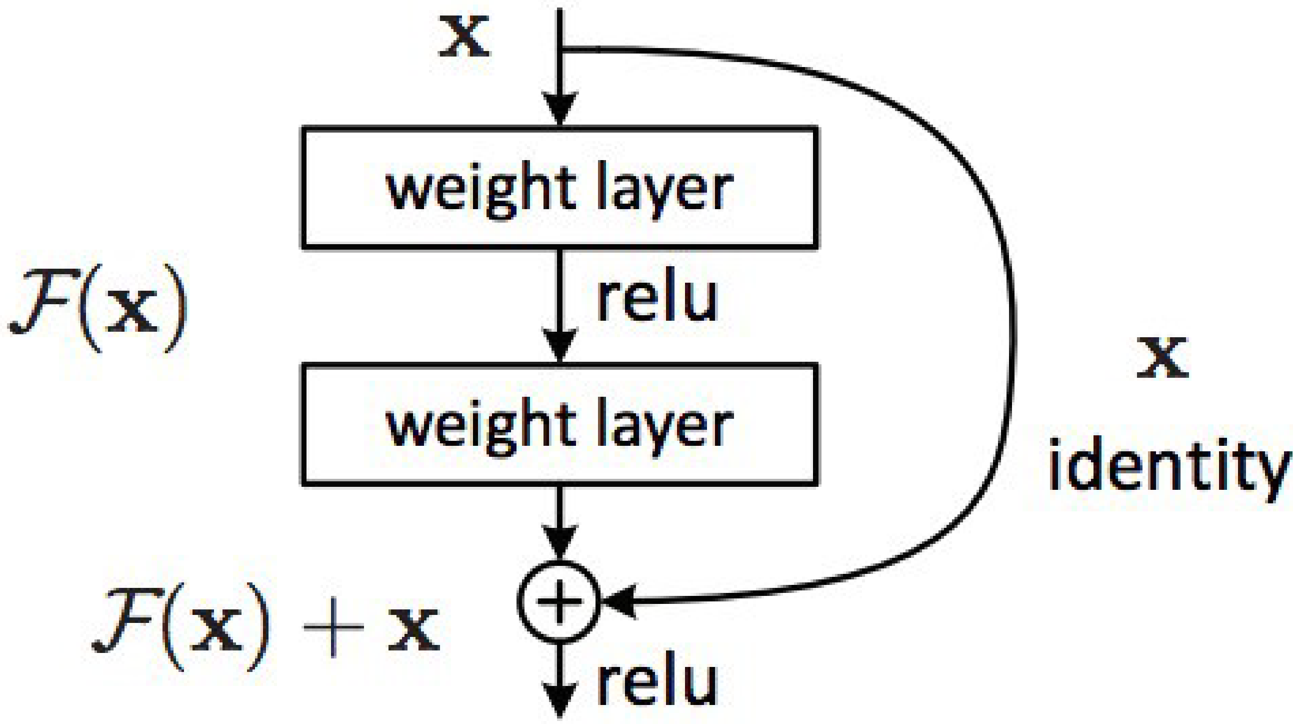
- $\mathcal{F}(\mathbf{x})$ is called the **residual** (something to add on top of identity), which turns to be easier to train in deep networks.
- **Weights in between the skip connection can be used to learn a «delta», a residual i.e., $\mathcal{F}(\mathbf{x})$ to improve over the solution that can be achieved by a shallow network.**
- Since $\mathbf{x}$ and $\mathcal{F}(\mathbf{x})$ must have the same size. Thus the weights (convolutional layers) are such that to **preserve dimension depth-wise or are re-arranged by 1x1 convolutions.**



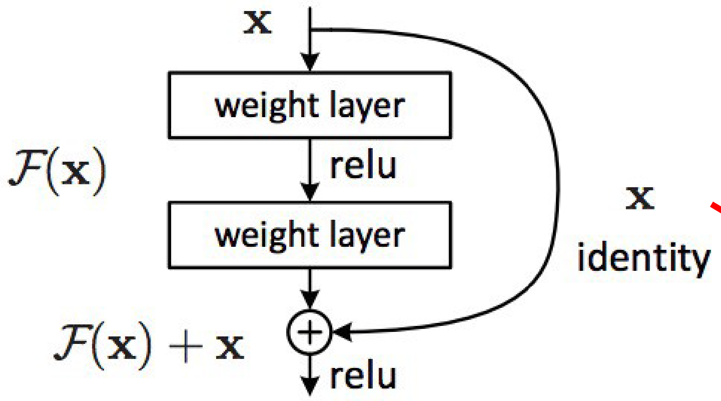
ResNet (2015)

The rationale behind adding this identity mapping is that:

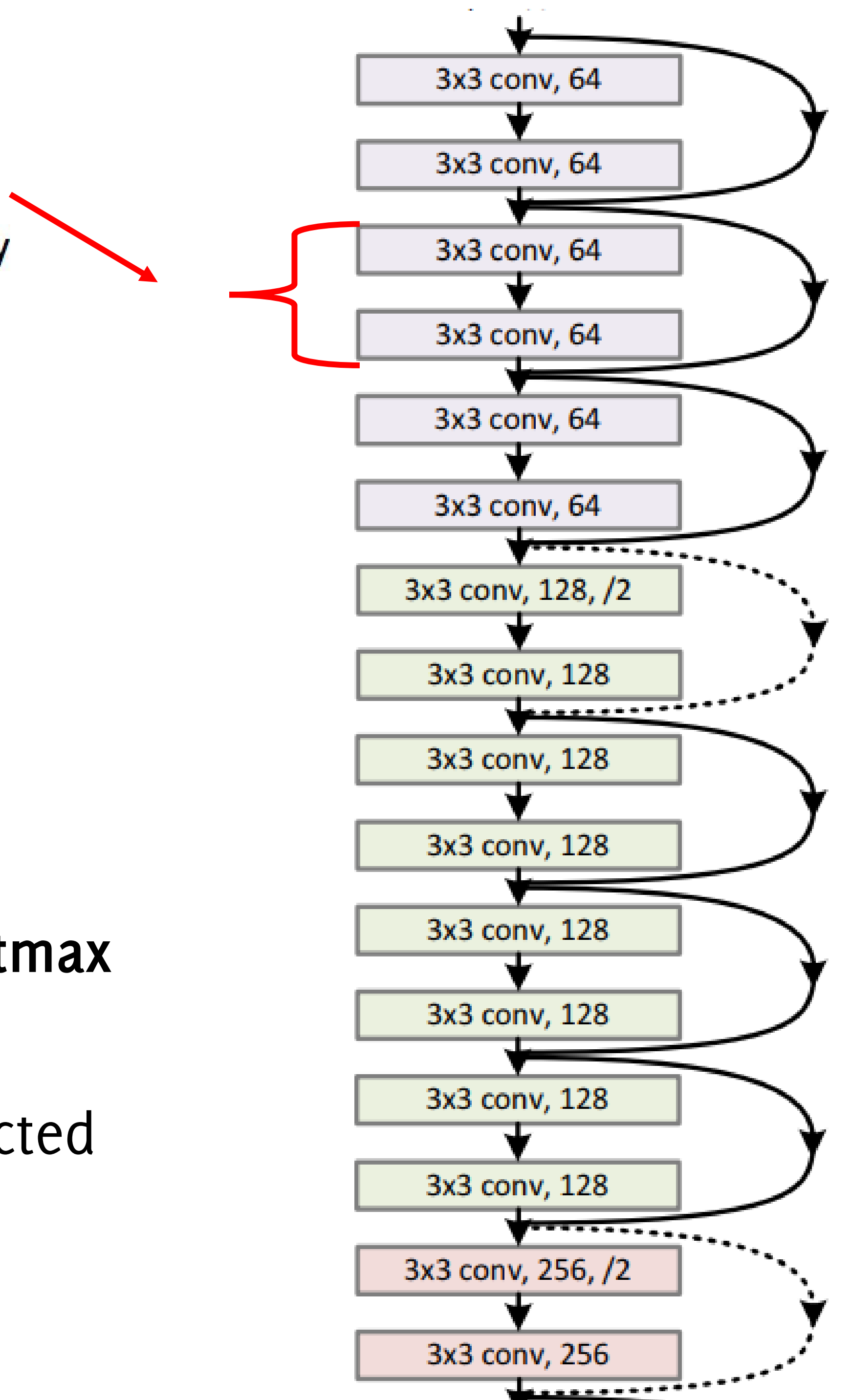
- It is **easier** for the following layers **to learn features on top of the input value**
- In practice the layers between an identity mapping would otherwise fail at learning the identity function to transfer the input to the output
- The performance achieved by resNet suggests that **probably most of the deep layers have to be close to the identity**!



ResNet (2015)



x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu



The ResNet is a stack of 152 layers of this module

The network alternates

- some spatial **pooling by convolution with stride 2**
- **doubling the number of filters**

At the beginning there is a **convolutional layer**

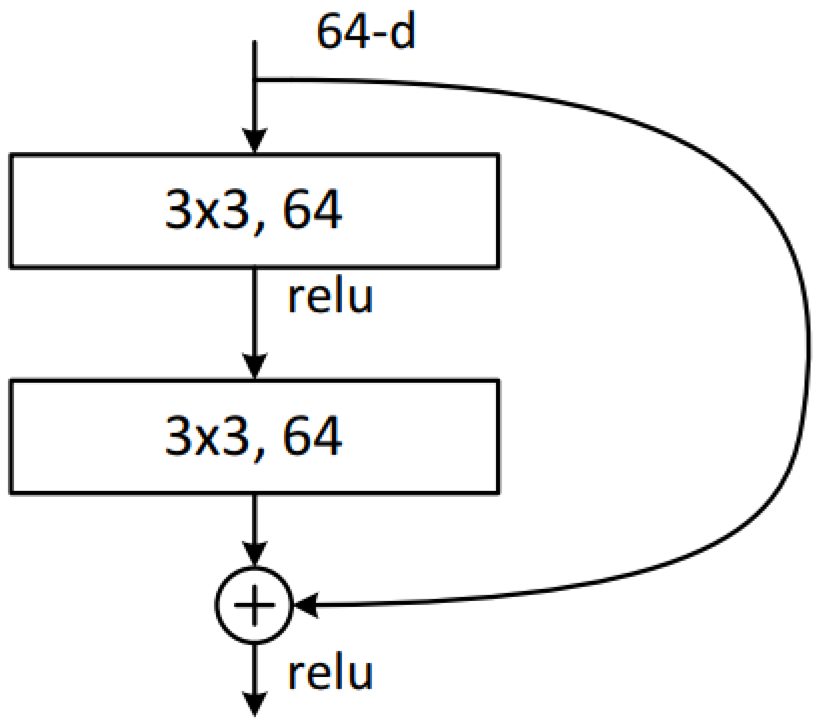
At the end: no FC but just a **GAP to be fed in the final softmax**

Deeper networks are able to achieve lower errors as expected

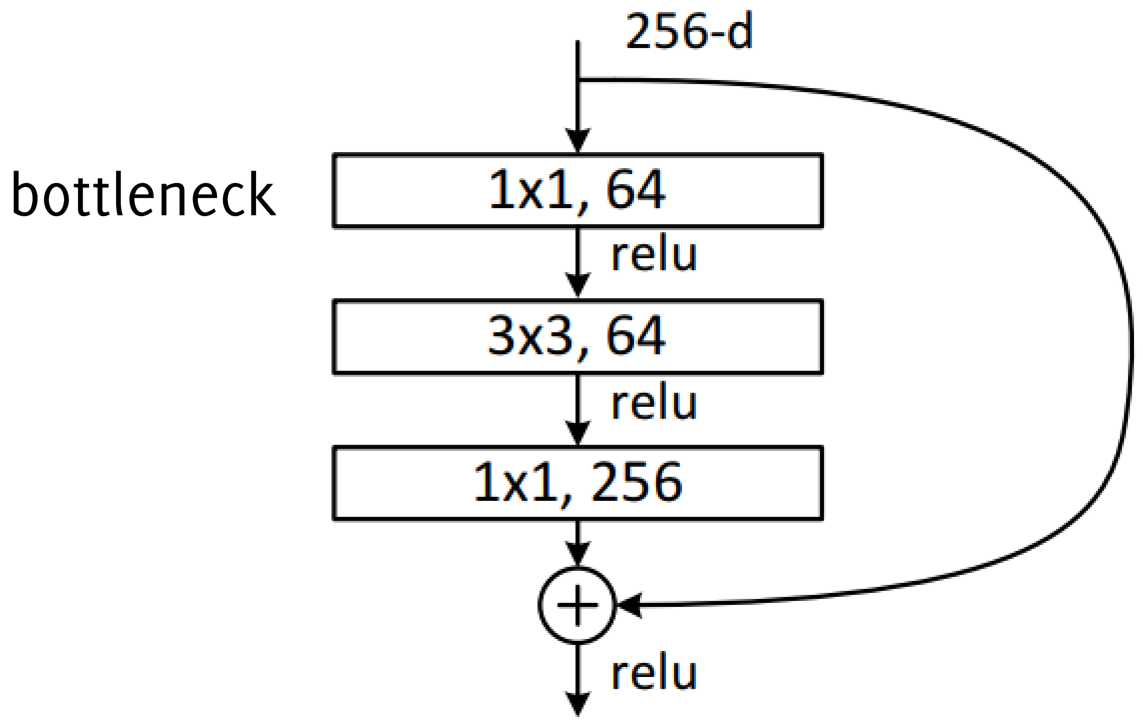
ResNet (2015)

Very deep architecture (say more than 50 layers) **adopt a bottleneck layer to reduce the depth within each block, thus the computational complexity** of the network (as in the inception module)

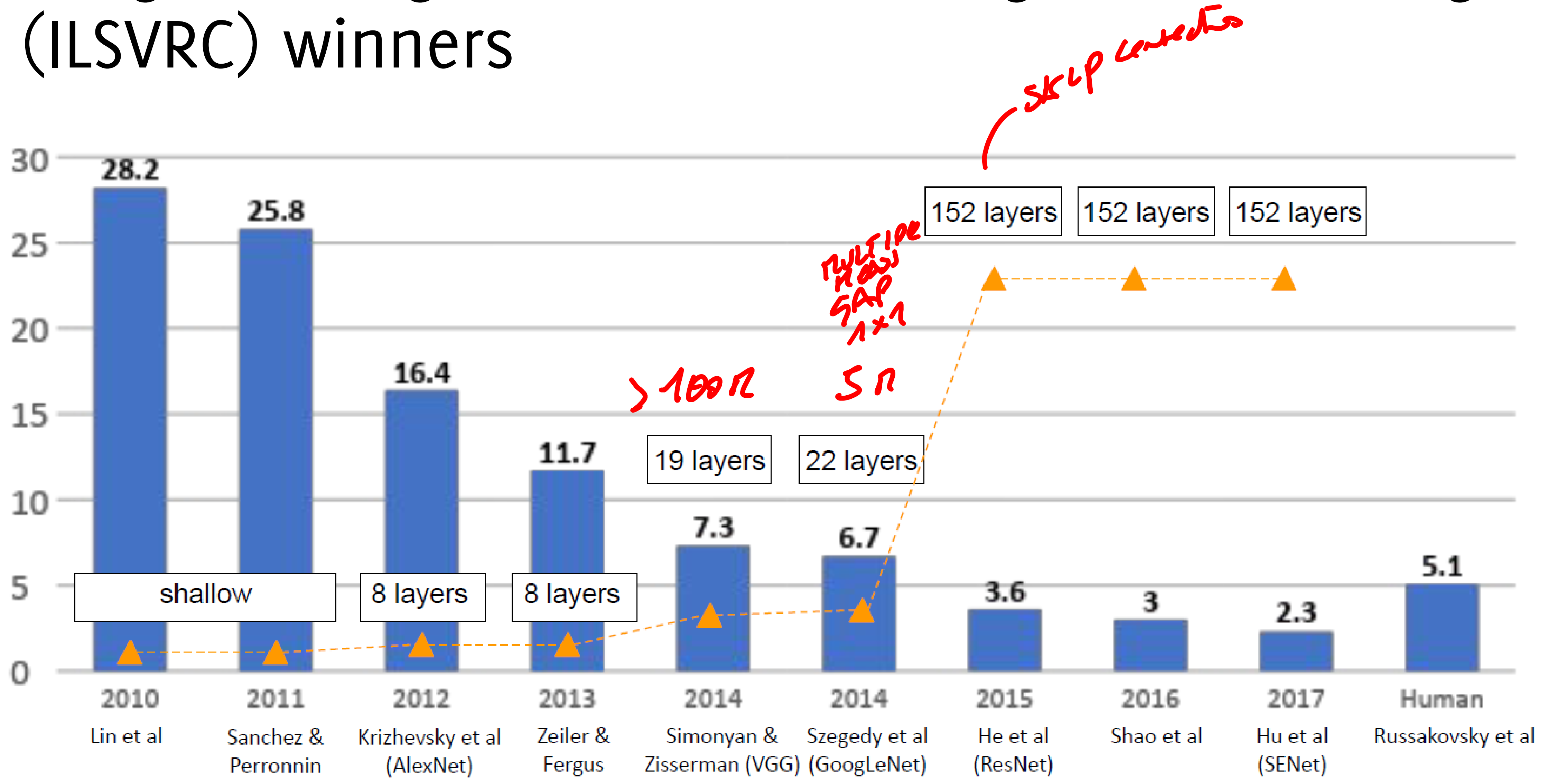
Few channels



Many channels

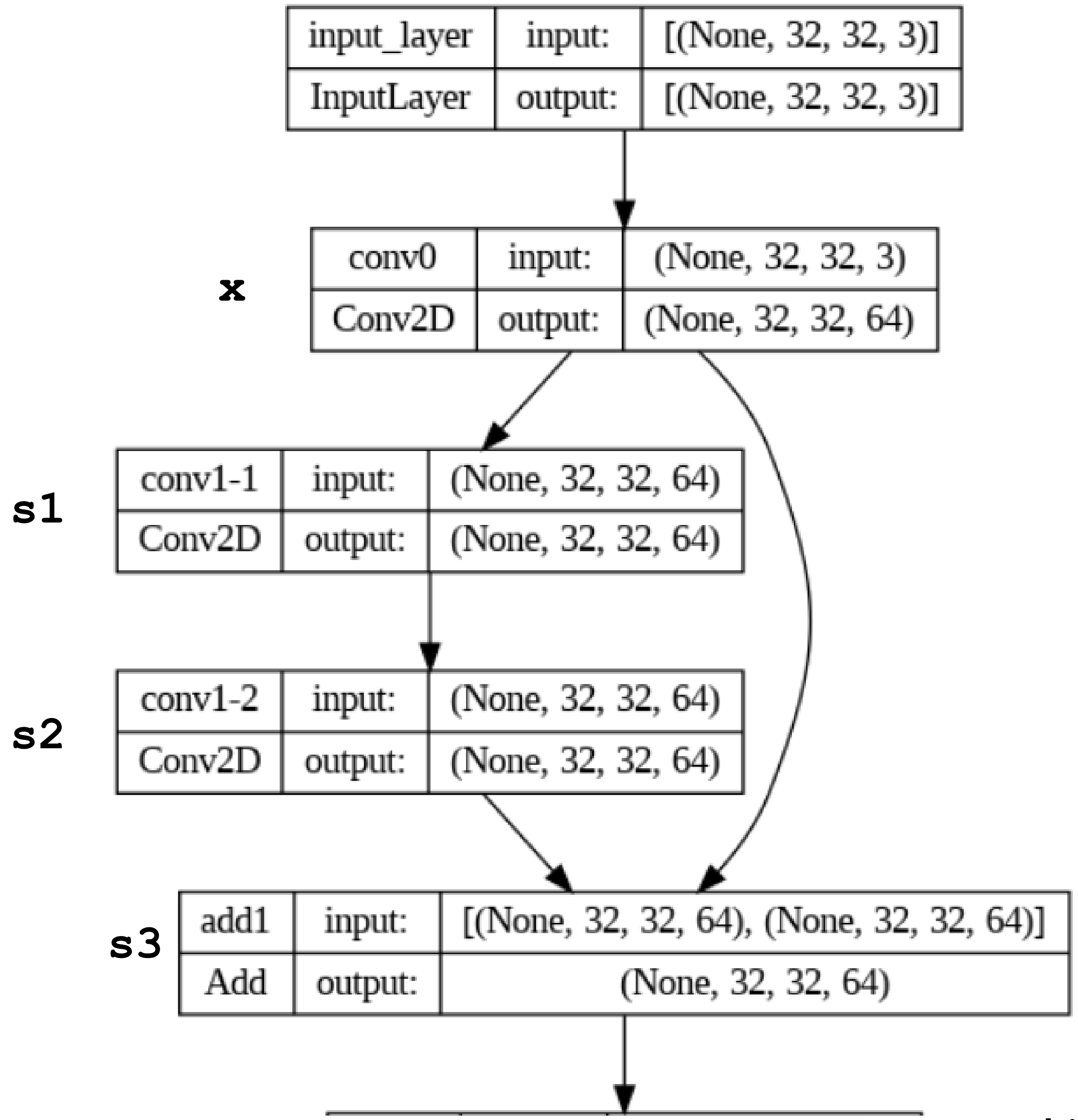


ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



Resnet Block in Kears

```
# input x
s1 = tfkl.Conv2D(
        filters=filters,
        kernel_size=3,
        padding='same',
        activation='relu',
        name='conv'+name+'-'+str(1)
    )(x)
 s2 = tfkl.Conv2D(
        filters=filters,
        kernel_size=3,
        padding='same',
        activation='relu',
        name='conv'+name+'-'+str(c+2)
        )(s1)
s3 = tfkl.Add(name='add'+name)([x,s2])
s4 = tfkl.ReLU(name='relu'+name)(s3)
s5 = tfkl.MaxPooling2D(name='pooling'+name)(s4)
```



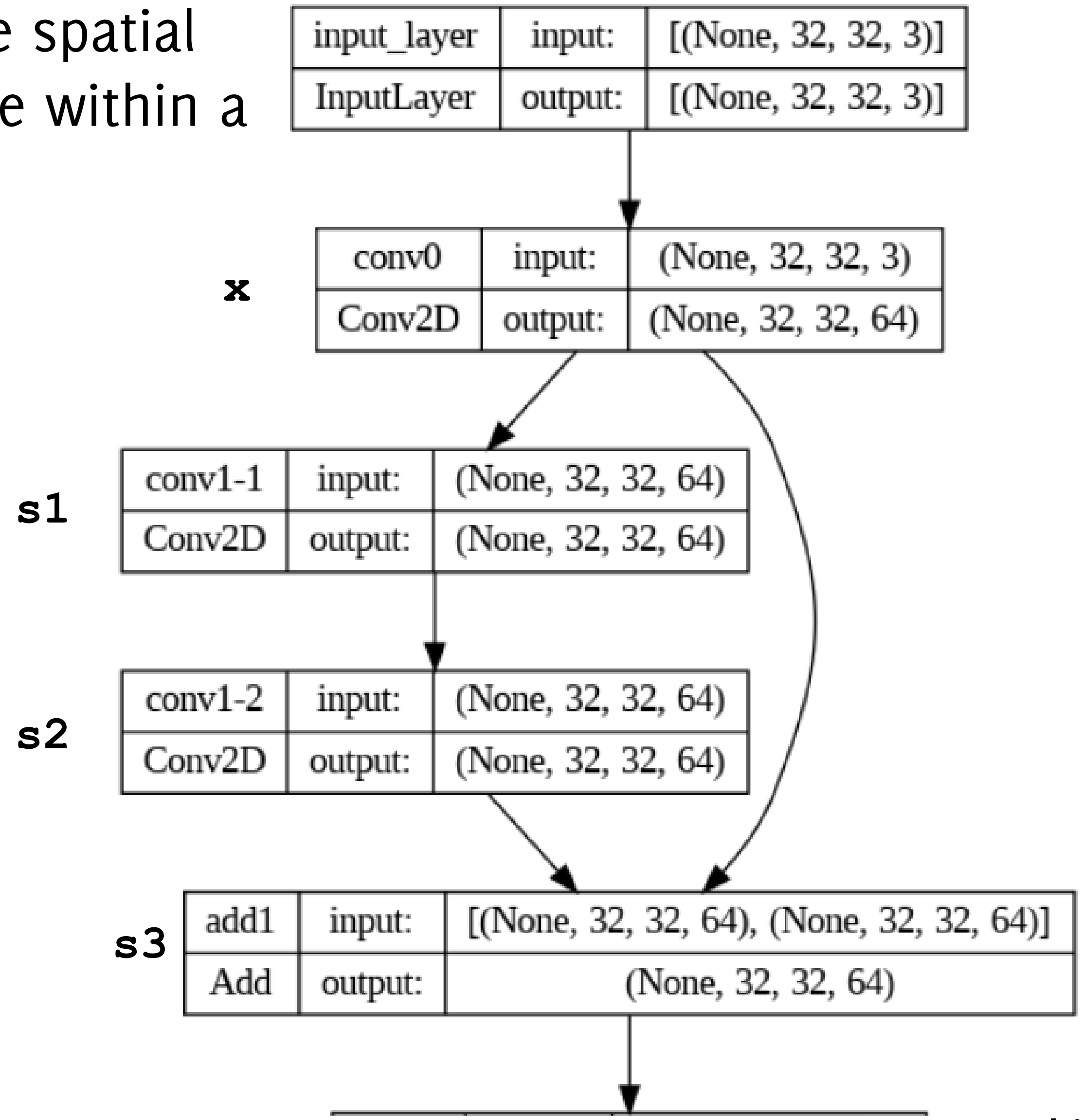
input_layer | input: | [(None, 32, 32, 3)]
InputLayer | output: | [(None, 32, 32, 3)]
x
conv0 | input: | (None, 32, 32, 3)
Conv2D | output: | (None, 32, 32, 64)
s1
conv1-1 | input: | (None, 32, 32, 64)
Conv2D | output: | (None, 32, 32, 64)
s2
conv1-2 | input: | (None, 32, 32, 64)
Conv2D | output: | (None, 32, 32, 64)
s3
add1 | input: | [(None, 32, 32, 64), (None, 32, 32, 64)]
Add | output: | (None, 32, 32, 64)

G. Boracchi

Resnet Block in Kears

```
# input x
s1 = tfkl.Conv2D(
        filters=filters,
        kernel_size=3,
        padding='same',
        activation='relu',
        name='conv'+name+'-'+str(1)
    )(x)
 s2 = tfkl.Conv2D(
        filters=filters,
        kernel_size=3,
        padding='same',
        activation='relu',
        name='conv'+name+'-'+str(c+2)
        )(s1)
s3 = tfkl.Add(name='add'+name)([x,s2])
s4 = tfkl.ReLU(name='relu'+name)(s3)
s5 = tfkl.MaxPooling2D(name='pooling'+name)(s4)
```

It is important that the spatial extent does not change within a resnet block



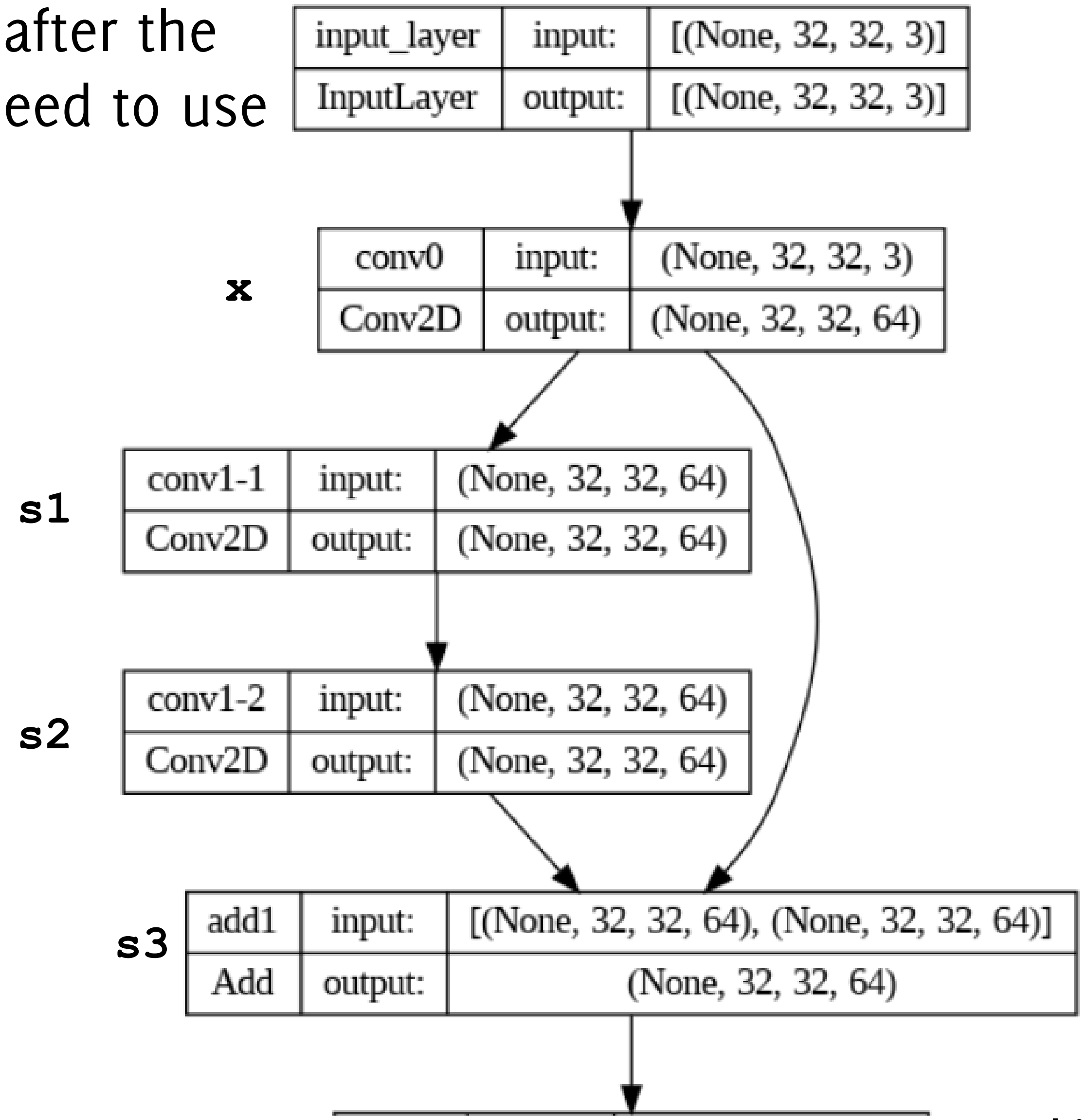
input_layer | input: | [(None, 32, 32, 3)]
InputLayer | output: | [(None, 32, 32, 3)]
x
conv0 | input: | (None, 32, 32, 3)
Conv2D | output: | (None, 32, 32, 64)
s1
conv1-1 | input: | (None, 32, 32, 64)
Conv2D | output: | (None, 32, 32, 64)
s2
conv1-2 | input: | (None, 32, 32, 64)
Conv2D | output: | (None, 32, 32, 64)
s3
add1 | input: | [(None, 32, 32, 64), (None, 32, 32, 64)]
Add | output: | (None, 32, 32, 64)

G. Boracchi

Resnet Block in Kears

```
# input x
s1 = tfkl.Conv2D(
        filters=filters,
        kernel_size=3,
        padding='same',
        activation='relu',
        name='conv'+name+'-'+str(1)
    )(x)
 s2 = tfkl.Conv2D(
        filters=filters,
        kernel_size=3,
        padding='same',
        activation='relu',
        name='conv'+name+'-'+str(c+2)
        )(s1)
s3 = tfkl.Add(name='add'+name)([x,s2])
s4 = tfkl.ReLU(name='relu'+name)(s3)
s5 = tfkl.MaxPooling2D(name='pooling'+name)(s4)
```

Include a nonlinearity after the add layer. You might need to use a sepecific layer



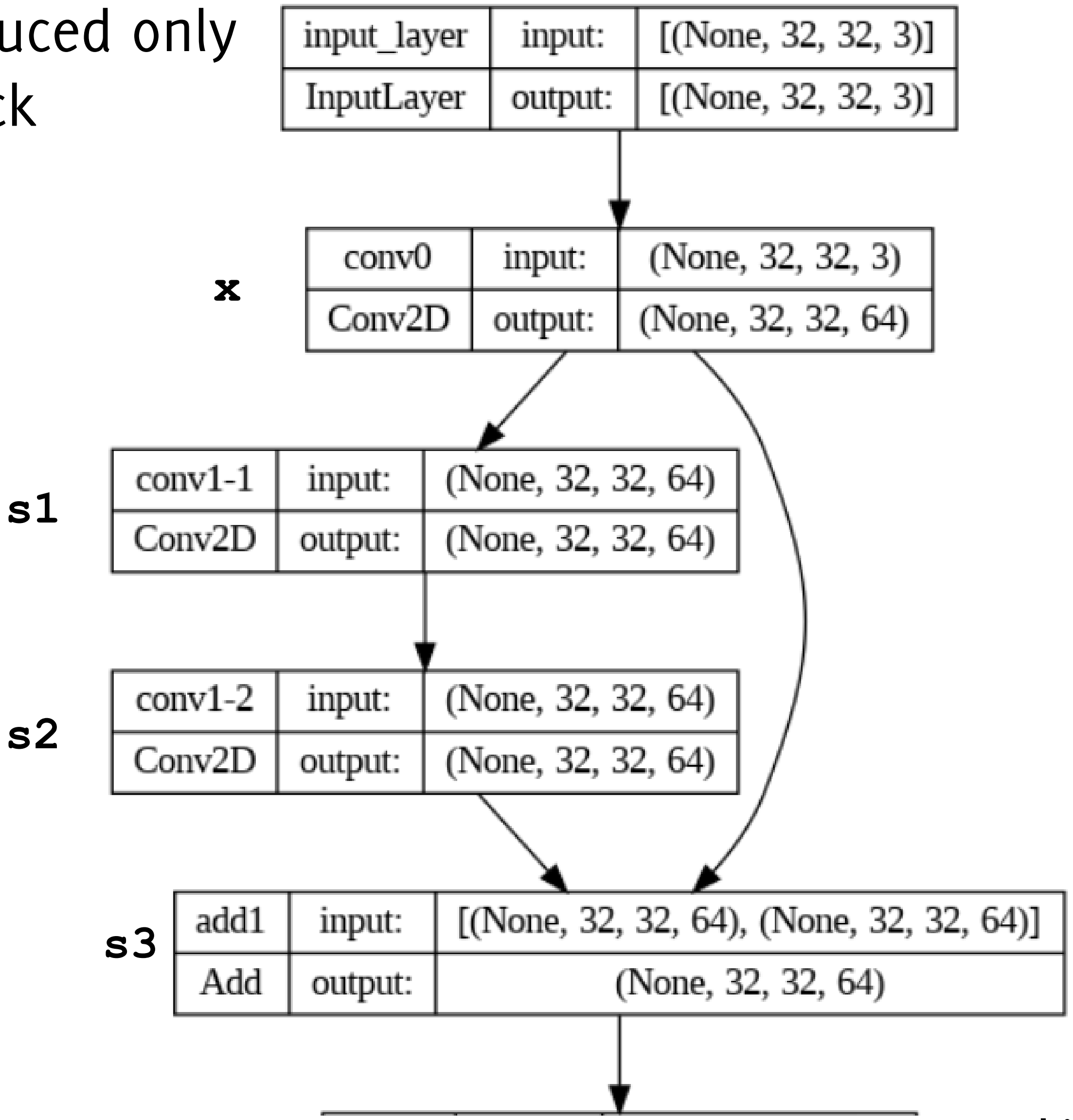
input_layer | input: | [(None, 32, 32, 3)]
InputLayer | output: | [(None, 32, 32, 3)]
x
conv0 | input: | (None, 32, 32, 3)
Conv2D | output: | (None, 32, 32, 64)
s1
conv1-1 | input: | (None, 32, 32, 64)
Conv2D | output: | (None, 32, 32, 64)
s2
conv1-2 | input: | (None, 32, 32, 64)
Conv2D | output: | (None, 32, 32, 64)
s3
add1 | input: | [(None, 32, 32, 64), (None, 32, 32, 64)]
Add | output: | (None, 32, 32, 64)

G. Boracchi

Resnet Block in Kears

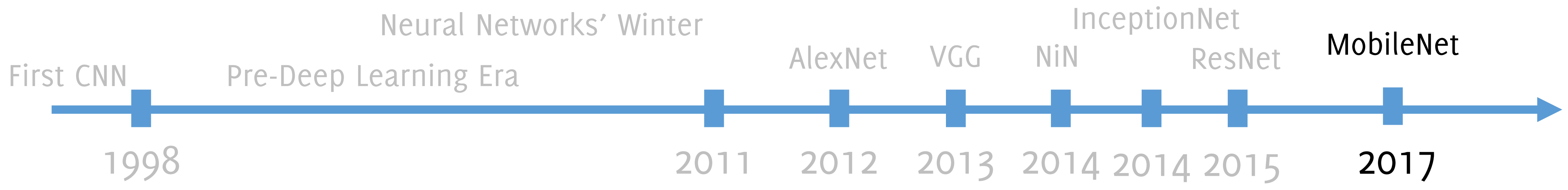
```
# input x
s1 = tfkl.Conv2D(
        filters=filters,
        kernel_size=3,
        padding='same',
        activation='relu',
        name='conv'+name+'-'+str(1)
    )(x)
 s2 = tfkl.Conv2D(
        filters=filters,
        kernel_size=3,
        padding='same',
        activation='relu',
        name='conv'+name+'-'+str(c+2)
        )(s1)
s3 = tfkl.Add(name='add'+name)([x,s2])
s4 = tfkl.ReLU(name='relu'+name)(s3)
s5 = tfkl.MaxPooling2D(name='pooling'+name)(s4)
```

Spatial size can be reduced only outside the resnet block



G. Boracchi

MobileNet: Reducing Computational Costs



MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications

Andrew G. Howard Menglong Zhu Bo Chen Dmitry Kalenichenko
Weijun Wang Tobias Weyand Marco Andreetto Hartwig Adam

Google Inc.

{howarda,menglong,bochen,dkalenichenko,weijunw,weyand,anm,hadam}@google.com

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... & Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861.

Mobilenets

Designed to reduce the number of parameters and of operations, to embed networks in mobile application.

Issues preventing use in mobile devices:

- conv2D layers have quite a few parameters
- conv2D layers are quite computationally demending

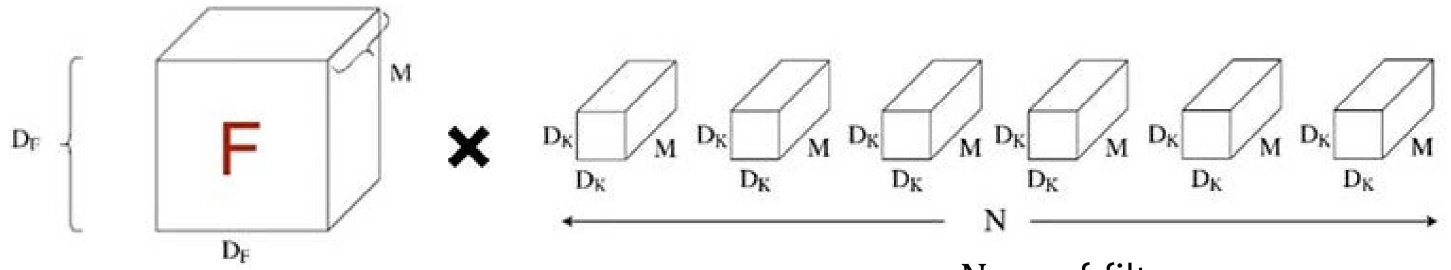
Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... & Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861.

Mobilenets

Nr of operations: $(D_K \times D_K \times M \times D_F \times D_F) \times N$

Traditional Conv2D

Each filter, mixes all the input channels

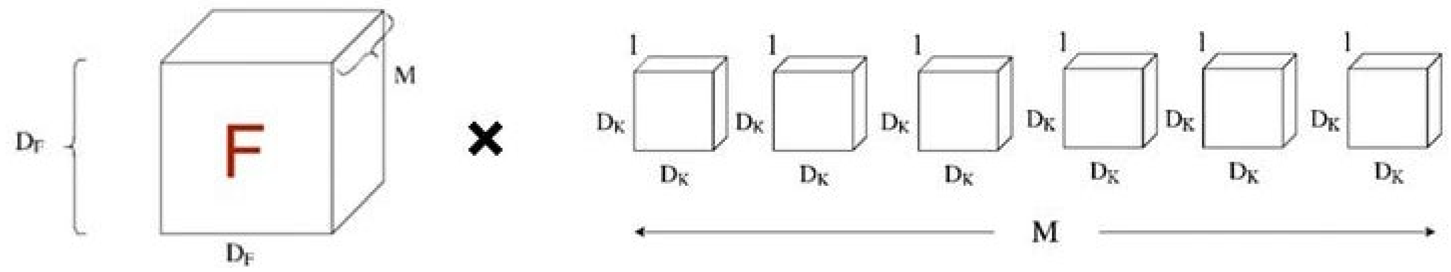


N=nr of filters

Separable Convolution, made of two steps

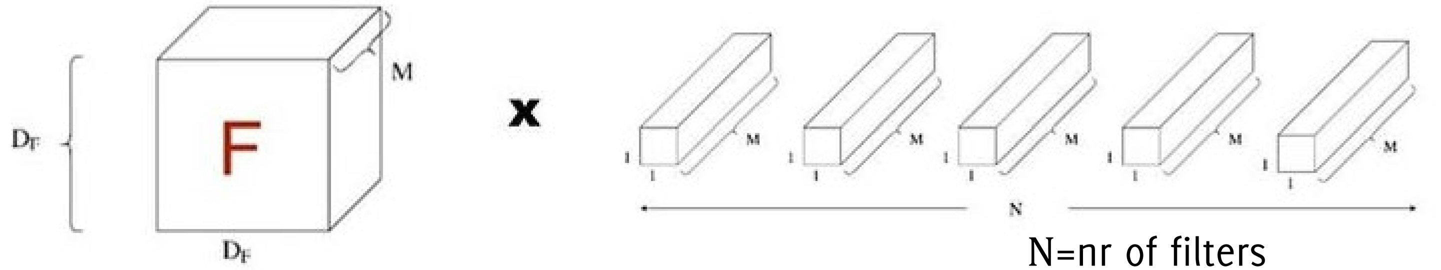
Nr of operations: $(D_K \times D_K \times M \times D_F \times D_F)$

1) Depth-wise convolution
this does not mix channels, it is like 2D convolution on each channel of input activation F.



2) Point-wise convolution:
Combines the **output of dept-wise convolution** by N filters that are 1×1.
It does not perform spatial convolution anymore

Nr of operations: $(M \times D_F \times D_F \times N)$



N=nr of filters

Image Credits: https://medium.com/@godeep48/an-overview-on-mobilenet-an-efficient-mobile-vision-cnn-f301141db94d

Depth-wise Separable Convolutions

All in all, a layer of dept-wise separable convolution using N filters costs

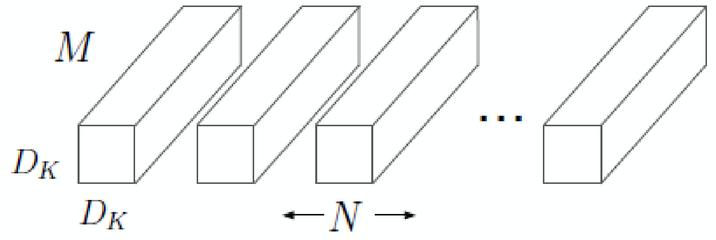
$$(D_K^2 \times M \times D_F^2) + M \times D_F^2 \times N$$

Which compared to conv2D layers

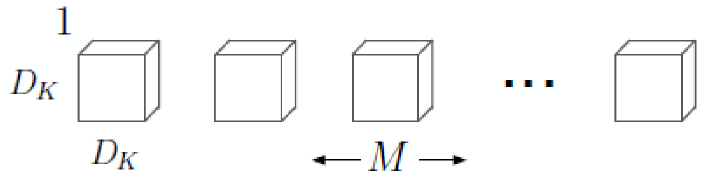
$$\frac{(D_K^2 \times M \times D_F^2) + M \times D_F^2 \times N}{D_K^2 \times M \times D_F^2 \times N} =$$

$$\frac{1}{N} + \frac{1}{D_K^2}$$

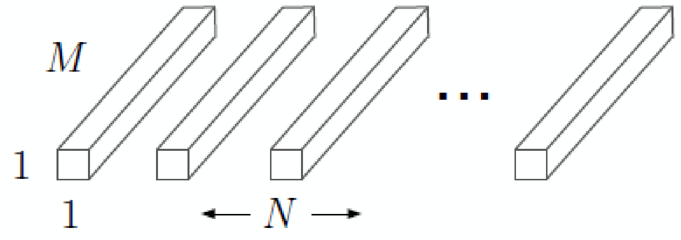
Which denotes a substantial savings when N and D_K are large



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



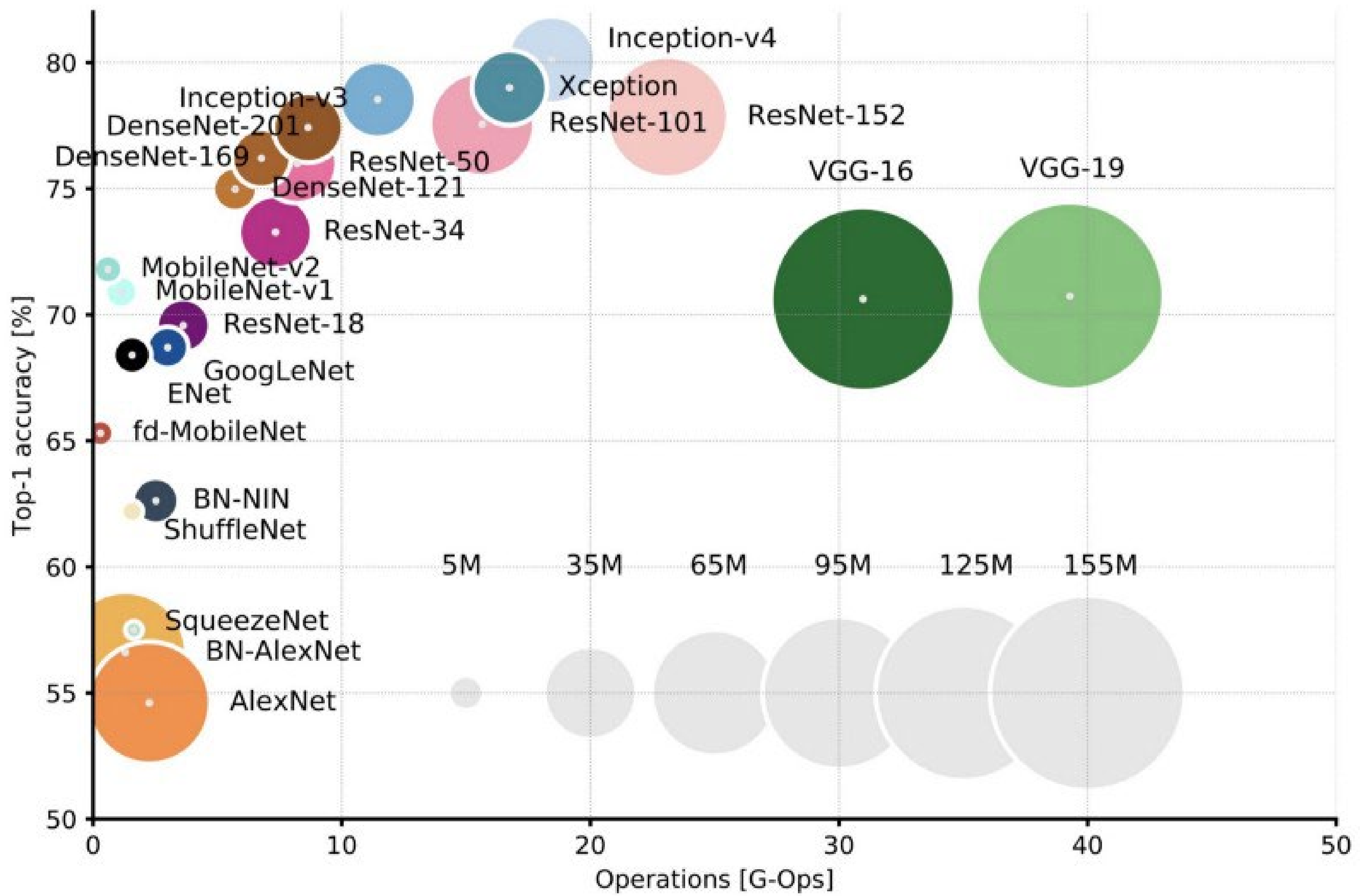
(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

Figure 2. The standard convolutional filters in (a) are replaced by two layers: depthwise convolution in (b) and pointwise convolution in (c) to build a depthwise separable filter.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... & Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861.

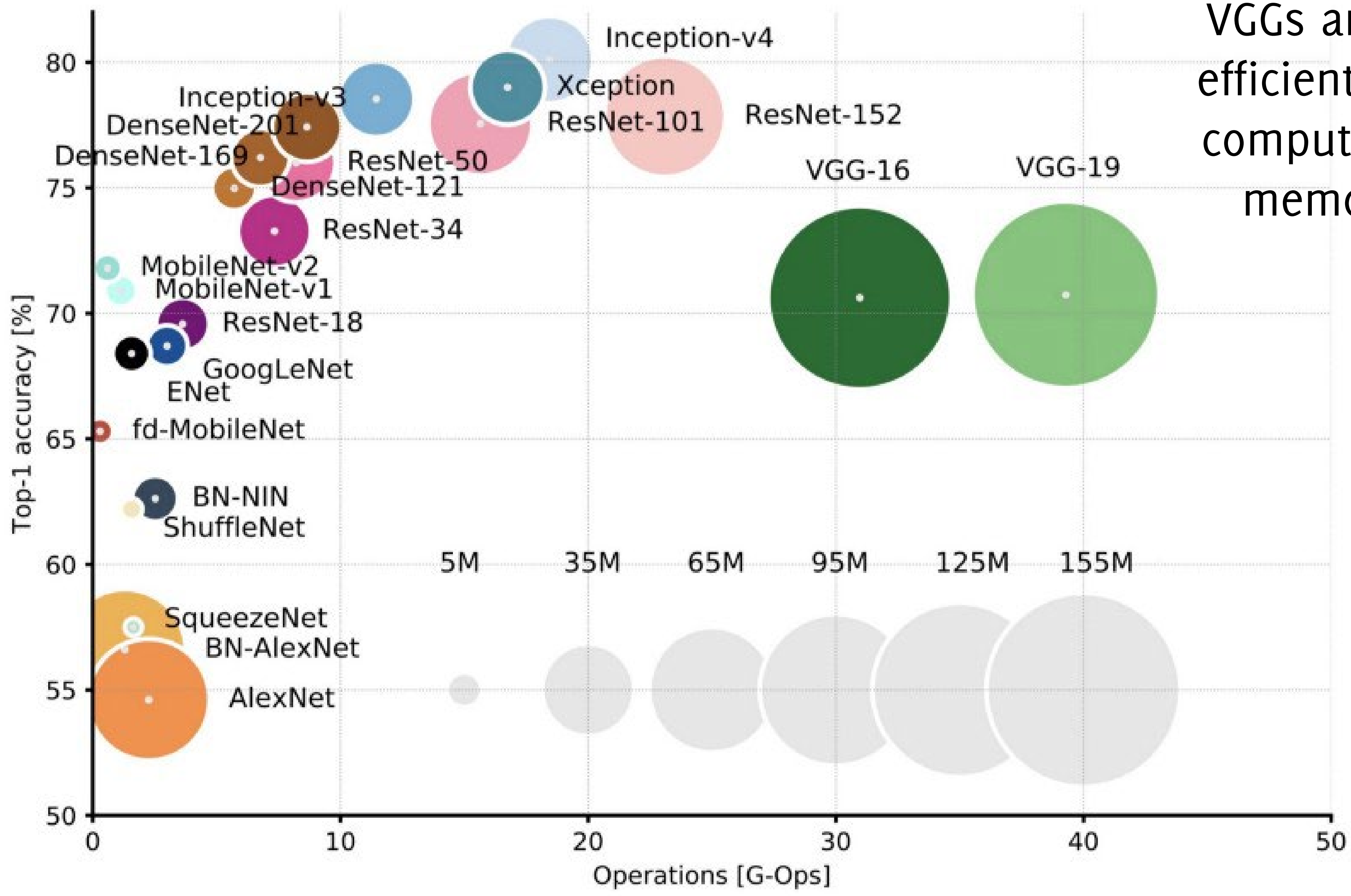
A Comparison

Comparison



Canziani, A, Paszke A., Culurciello E.. "An analysis of deep neural network models for practical applications." *arXiv preprint* (2016).

Comparison

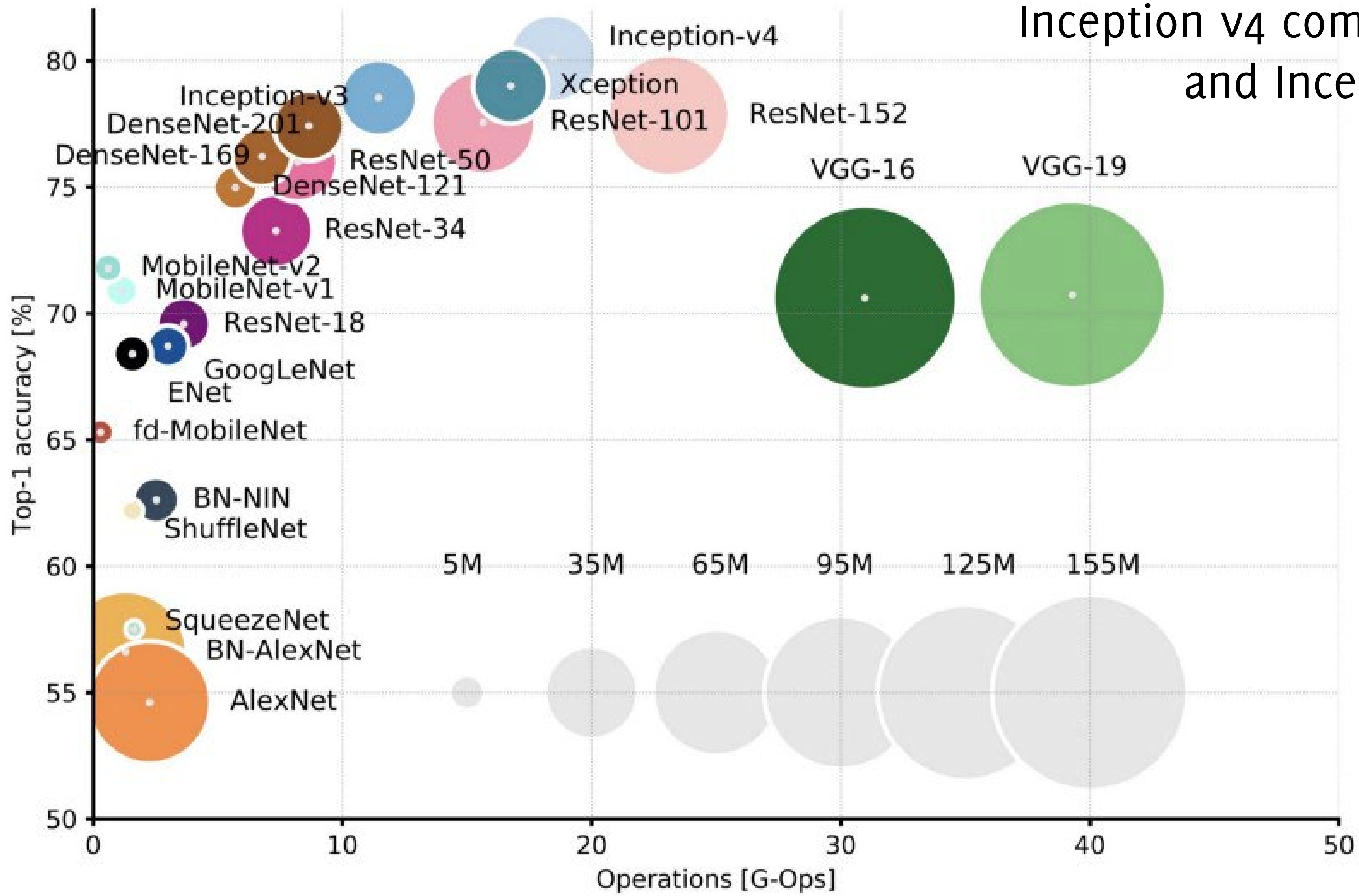


VGGs are the least efficient: very large computational and memory usage

Canziani, A, Paszke A., Culurciello E.. "An analysis of deep neural network models for practical applications." *arXiv preprint* (2016).

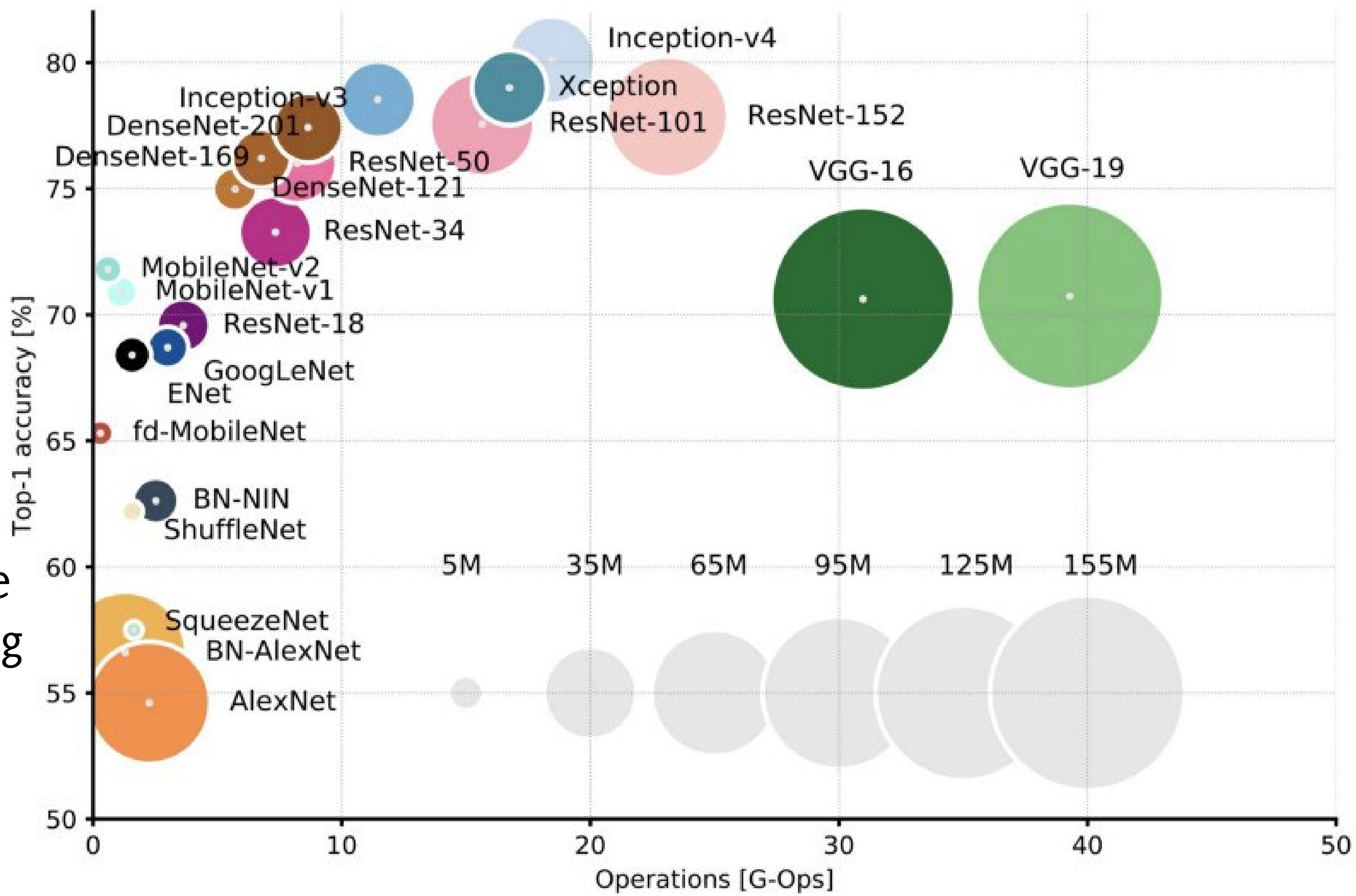
Comparison

Inception models are the most efficient and best performing
Inception v4 combines ResNet and Inception



Canziani, A, Paszke A., Culurciello E.. "An analysis of deep neural network models for practical applications." *arXiv preprint* (2016).

Comparison

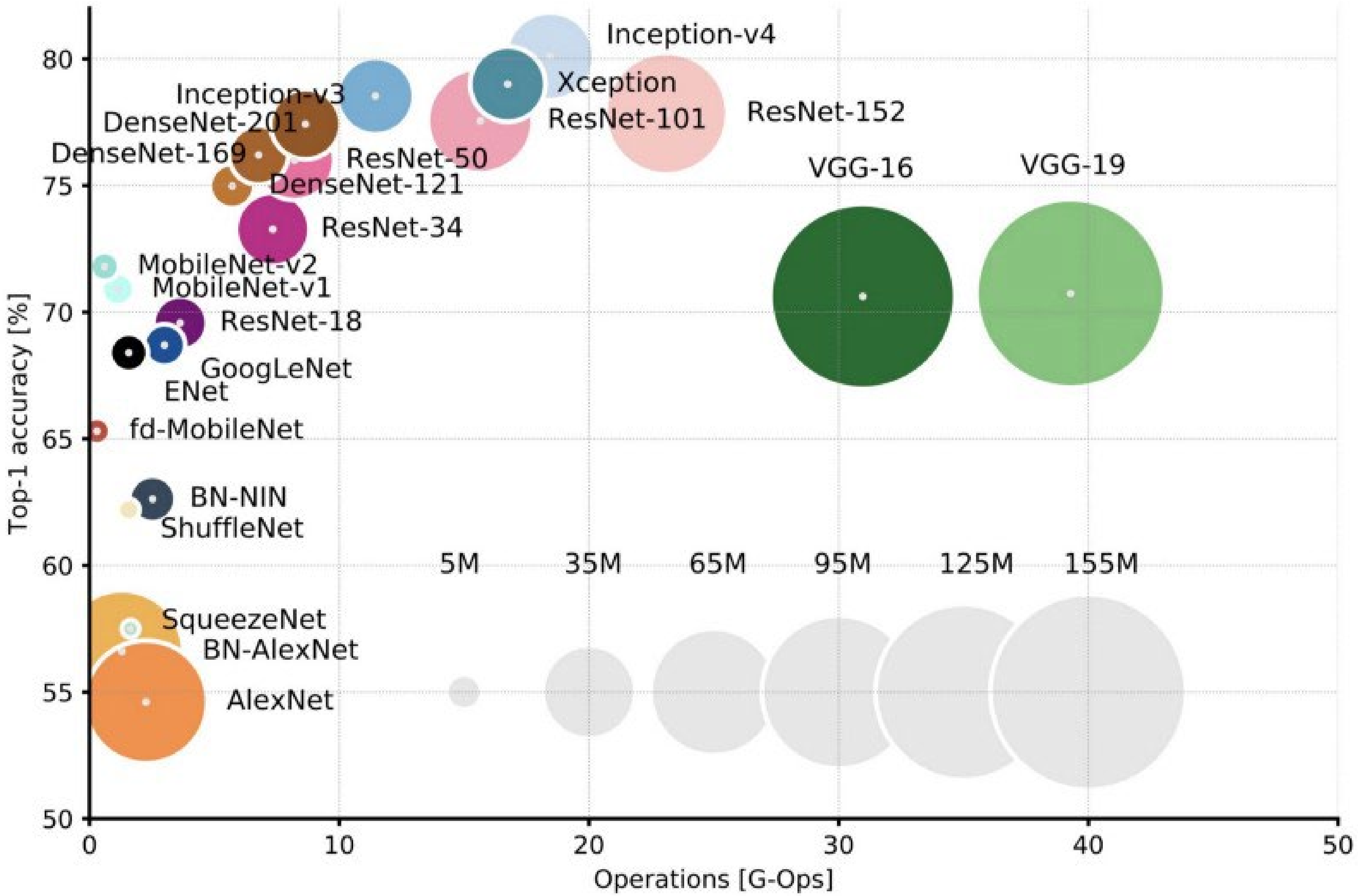


AlexNet are the least performing and not particularly efficient

Canziani, A, Paszke A., Culurciello E.. "An analysis of deep neural network models for practical applications." *arXiv preprint* (2016).

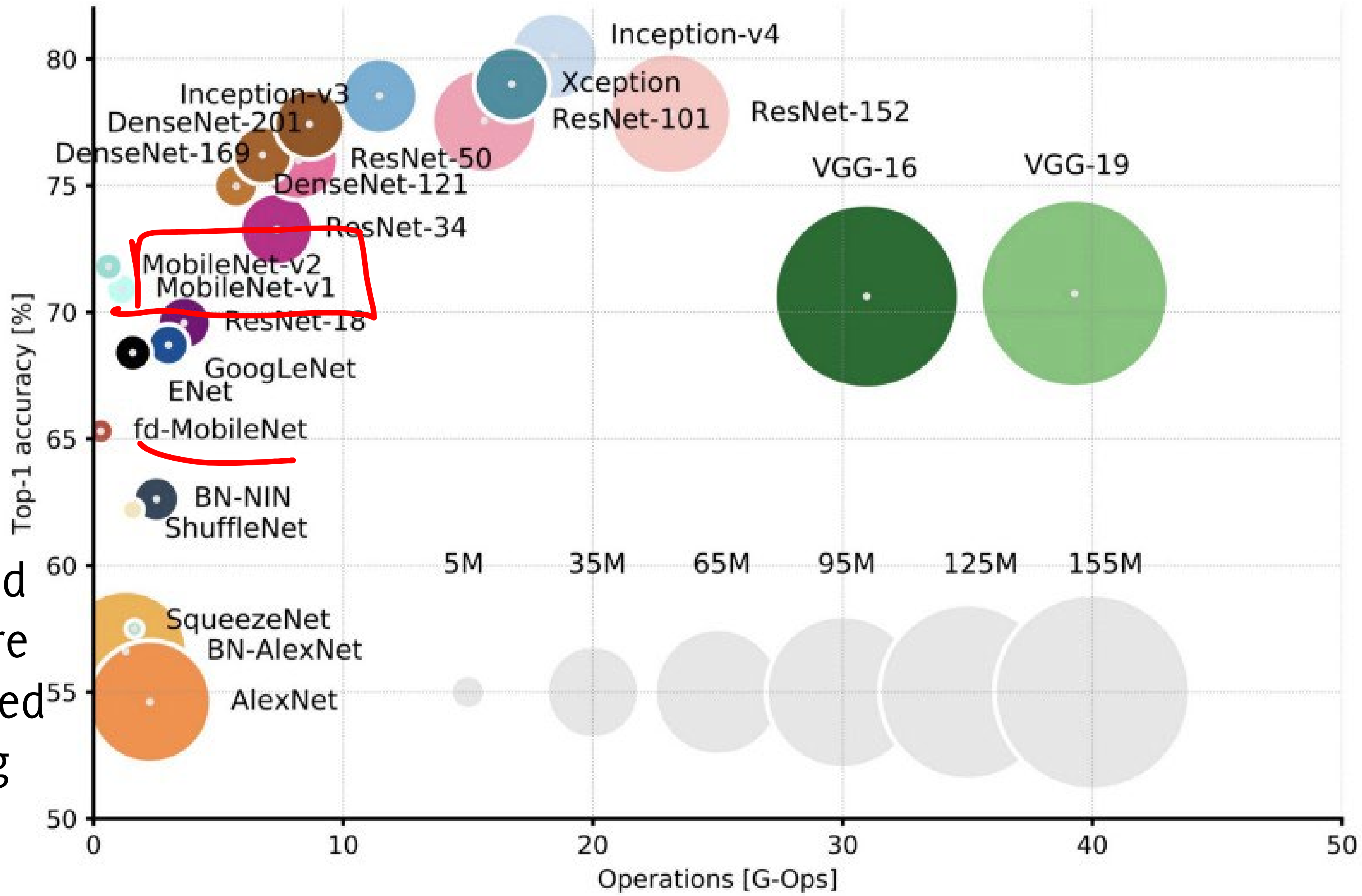
Comparison

Inception-v4:
Resnet + Inception



Canziani, A, Paszke A., Culurciello E.. "An analysis of deep neural network models for practical applications." *arXiv preprint* (2016).

Comparison



Mobile net and squeezenet are instead designed for improving efficiency

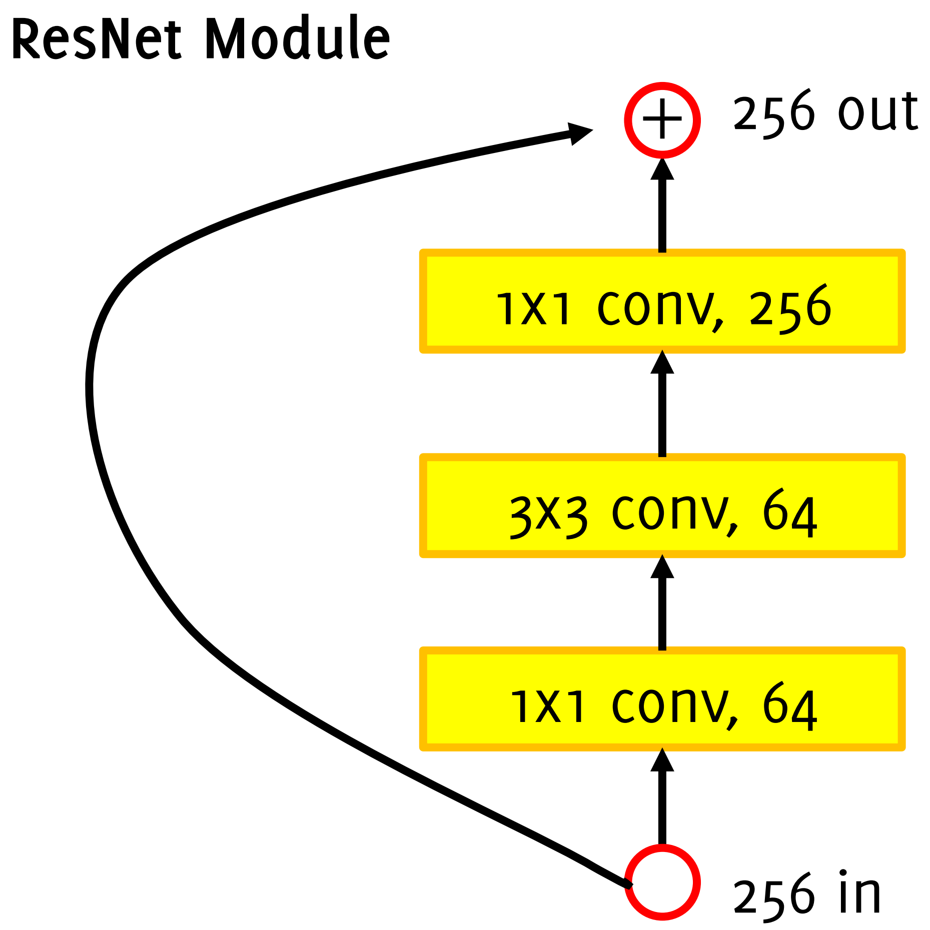
Canziani, A, Paszke A., Culurciello E.. "An analysis of deep neural network models for practical applications." *arXiv preprint* (2016).

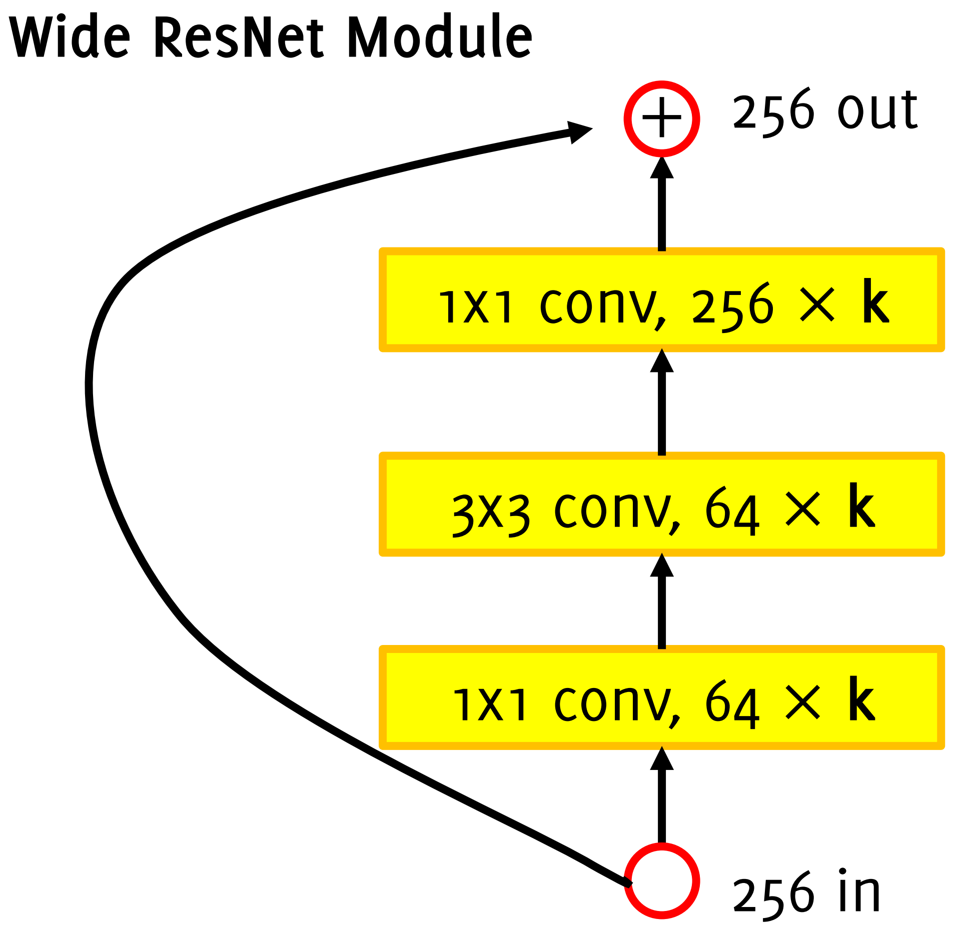
Latest Developments in Image Classification

Aggregated Residual Transformations for Deep Neural Networks

Saining Xie[1] Ross Girshick[2] Piotr Dollár[2] Zhuowen Tu[1] Kaiming He[2]
[1]UC San Diego [2]Facebook AI Research
{s9xie,ztu}@ucsd.edu {rbg,pdollar,kaiminghe}@fb.com

Wide Resnet

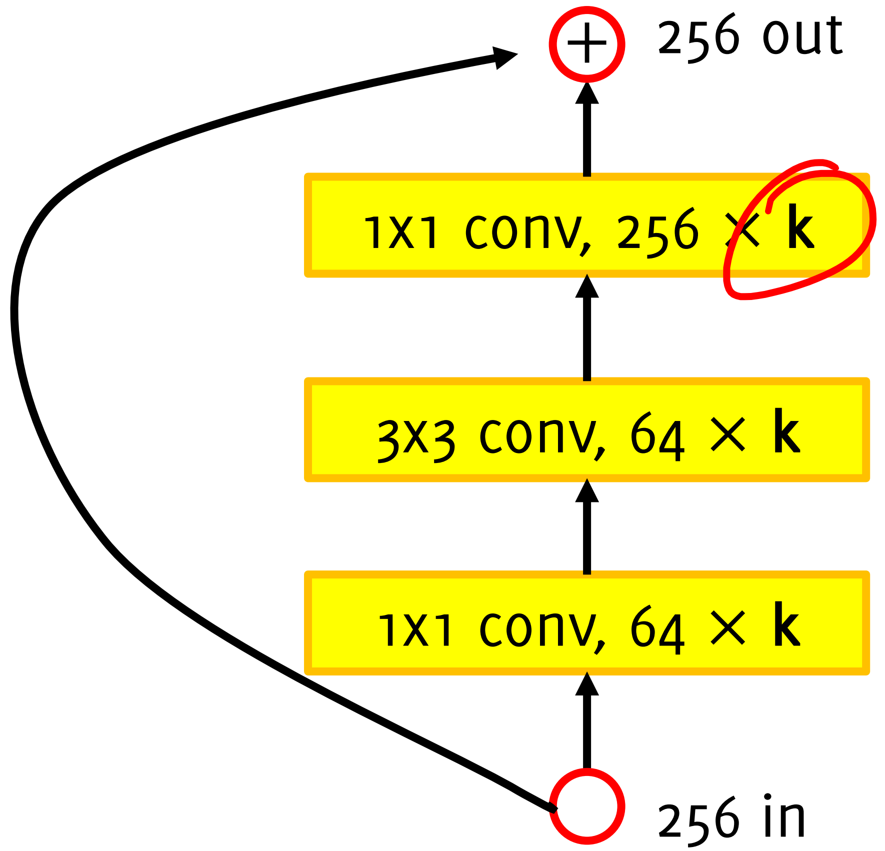

ResNet Module
256 out
1x1 conv, 256
3x3 conv, 64
1x1 conv, 64
256 in


Wide ResNet Module
256 out
1x1 conv, 256 × k
3x3 conv, 64 × k
1x1 conv, 64 × k
256 in

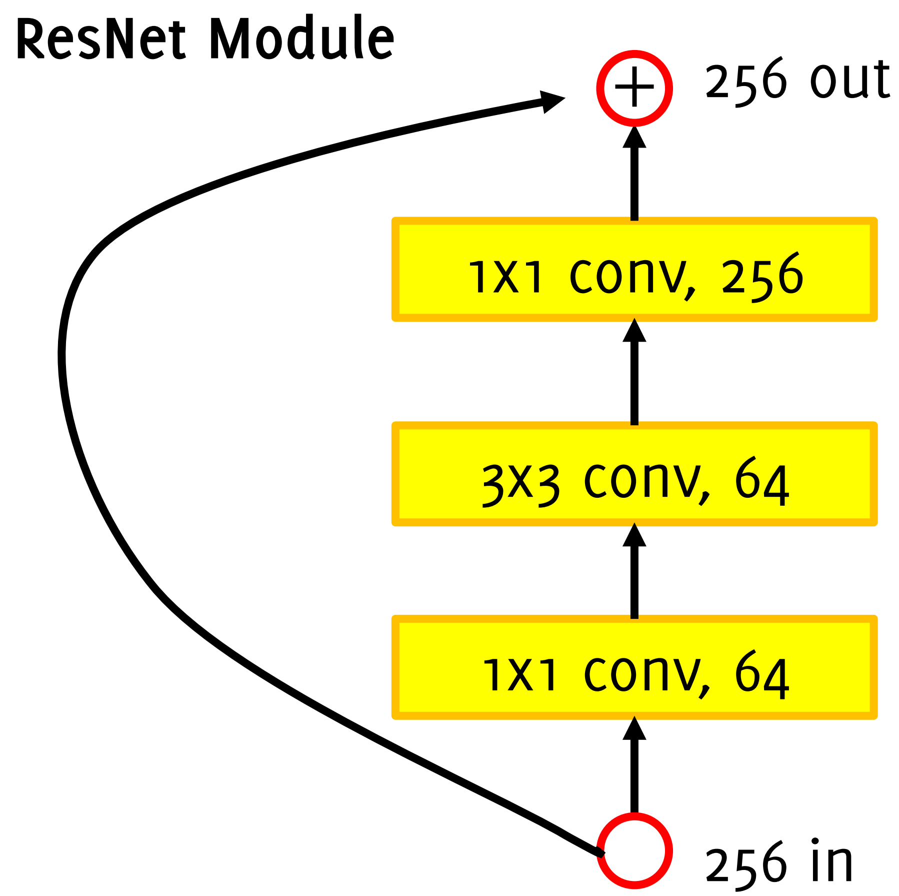
Wide Resnet

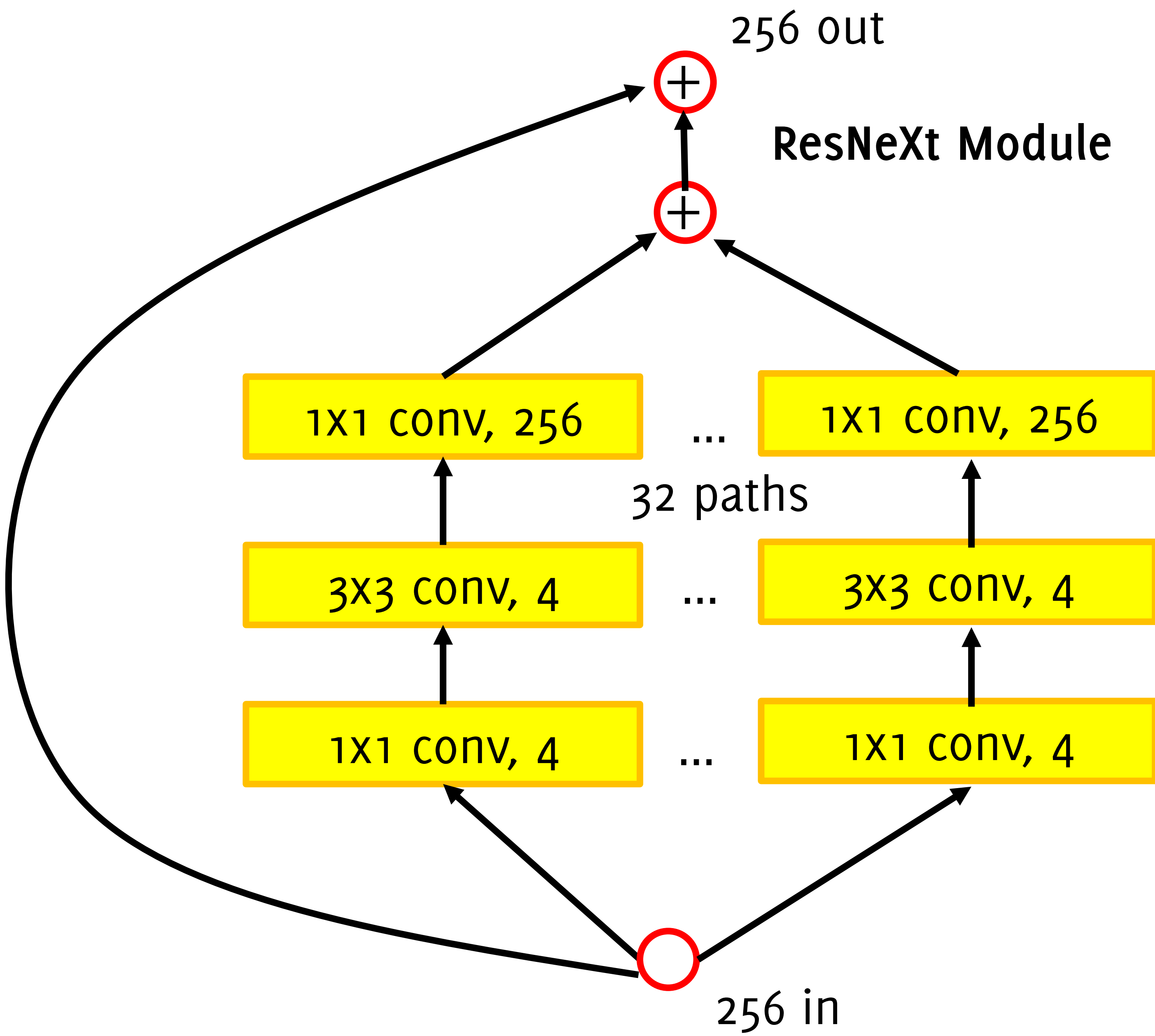
- Use wider residual blocks (F x k filters instead of F filters in each layer)
- **50-layer wide ResNet outperforms 152-layer original ResNet**
- Increasing width instead of depth more computationally efficient (parallelizable)

Wide ResNet Module



ResNeXt


ResNet Module
256 out
1x1 conv, 256
3x3 conv, 64
1x1 conv, 64
256 in

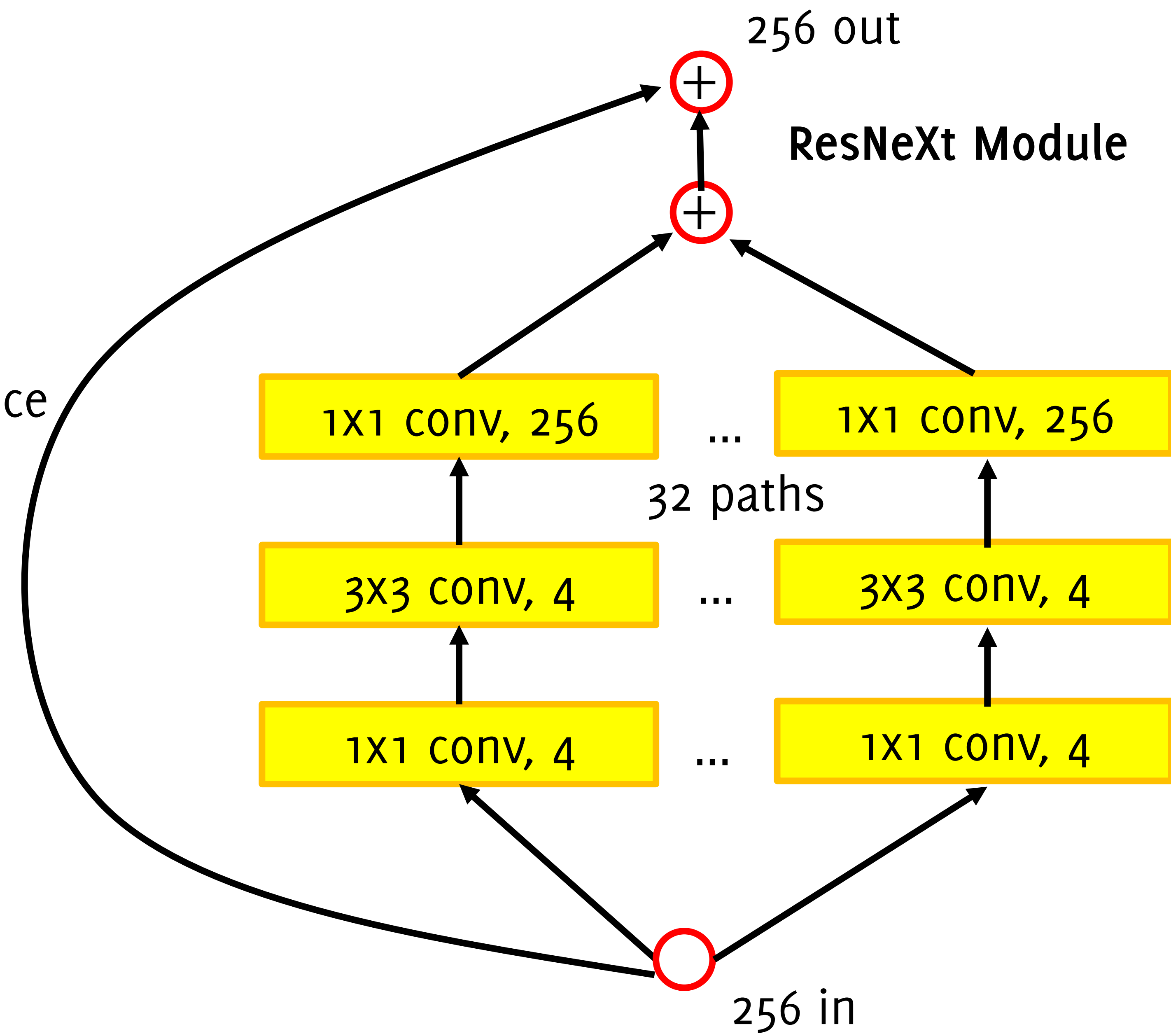

256 out
ResNeXt Module
1x1 conv, 256
...
1x1 conv, 256
32 paths
3x3 conv, 4
...
3x3 conv, 4
1x1 conv, 4
...
1x1 conv, 4
256 in

ResNeXt

Widen the ResNet module by adding multiple pathways in parallel (previous wide Resnet was just increasing the number of filters and showing it achieves similar performance with fewer blocks)

Similar to inception module where the **activation maps are being processed in parallel**

Different from inception module, all the paths share the same topology



Densely Connected Convolutional Networks

Gao Huang*
Cornell University
gh349@cornell.edu

Zhuang Liu*
Tsinghua University
liuzhuang13@mails.tsinghua.edu.cn

Laurens van der Maaten
Facebook AI Research
lvdmaaten@fb.com

Kilian Q. Weinberger
Cornell University
kqw4@cornell.edu

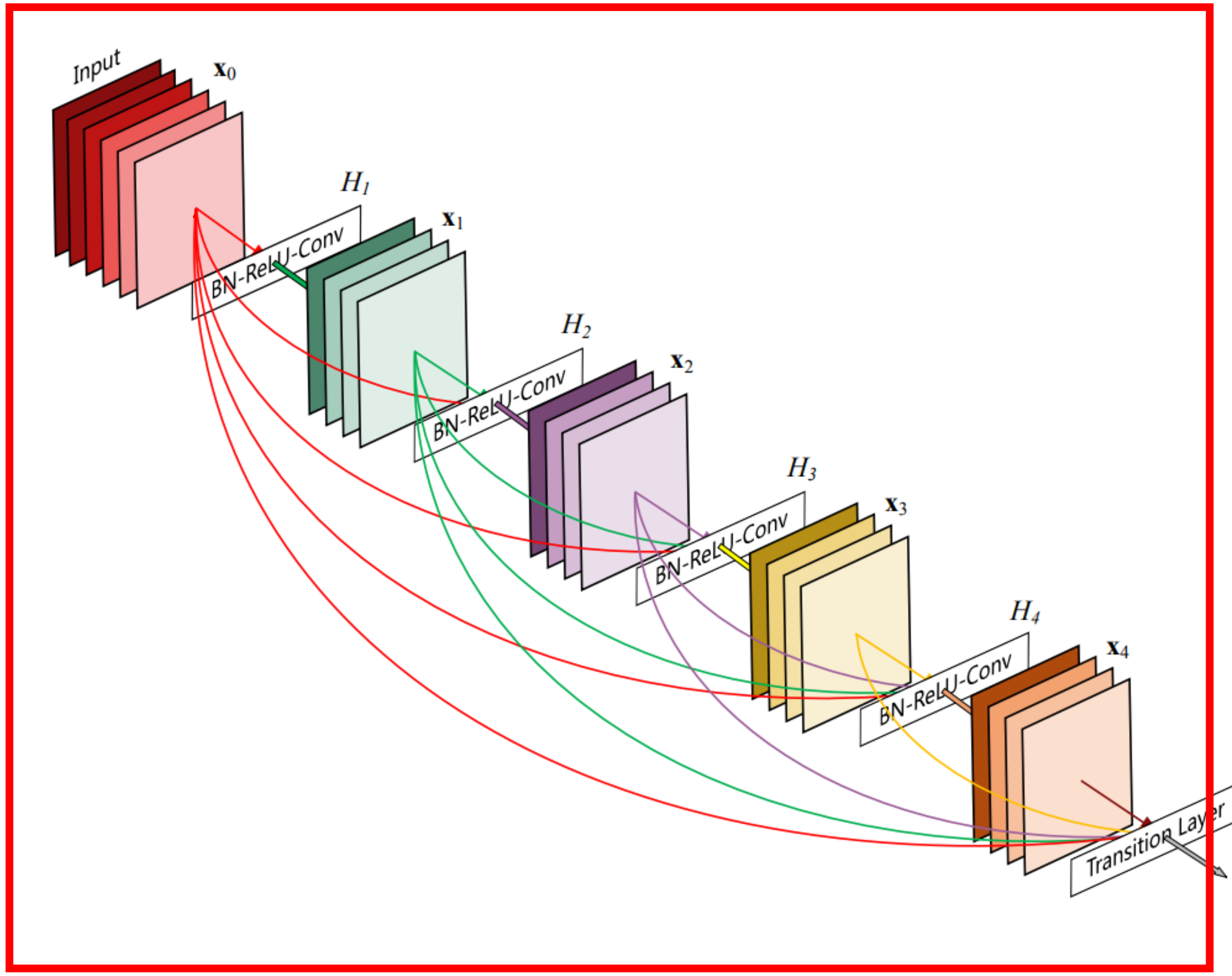
DenseNet

In **each block** of a DenseNet, **each convolutional layer takes as input the output of the previous layers**

Short connections between convolutional layers of the network

Each layer is connected to every other layer in a feed-forward fashion

Dense block



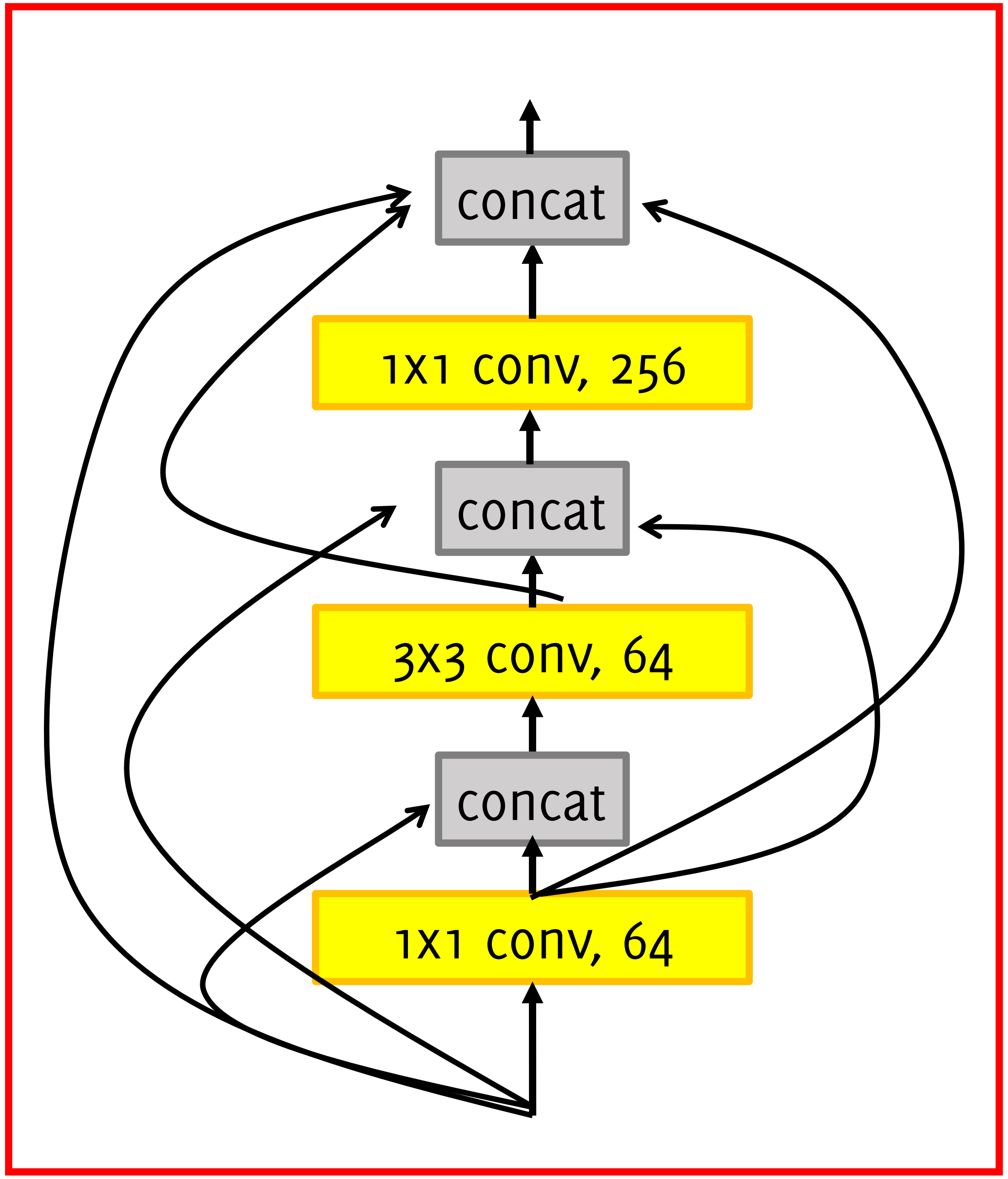
DenseNet

In each block a DenseNet, each **convolutional layer takes as input the output of the previous layers**

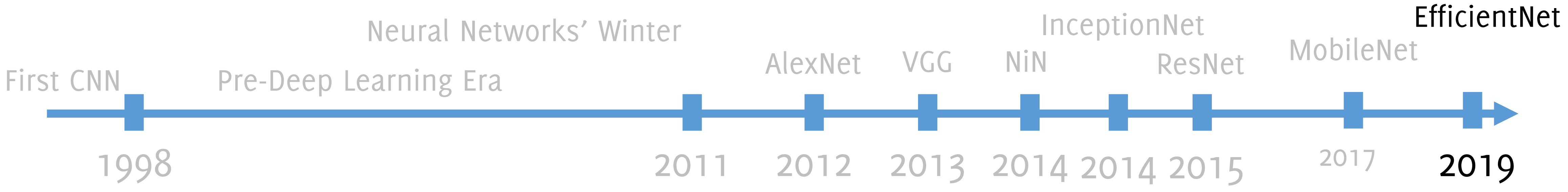
Each layer is connected to every other layer in a feed-forward fashion

This alleviates vanishing gradient problem, promotes feature re-use since each feature is spread through the network

Dense block



EfficientNet: a family of networks



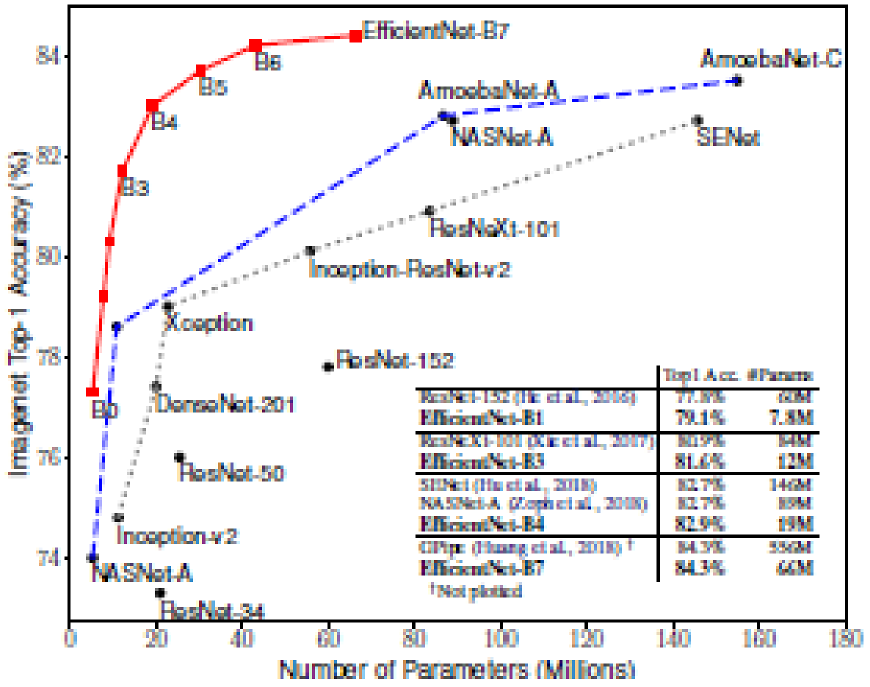
G. Boracchi

EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks

Mingxing Tan [1] Quoc V. Le [1]

Abstract

Convolutional Neural Networks (ConvNets) are commonly developed at a fixed resource budget, and then scaled up for better accuracy if more resources are available. In this paper, we systematically study model scaling and identify that carefully balancing network depth, width, and resolution can lead to better performance. Based on this observation, we propose a new scaling method that uniformly scales all dimensions of depth/width/resolution using a simple yet highly effective *compound coefficient*. We demonstrate the effectiveness of this method on scaling up MobileNets and ResNet.

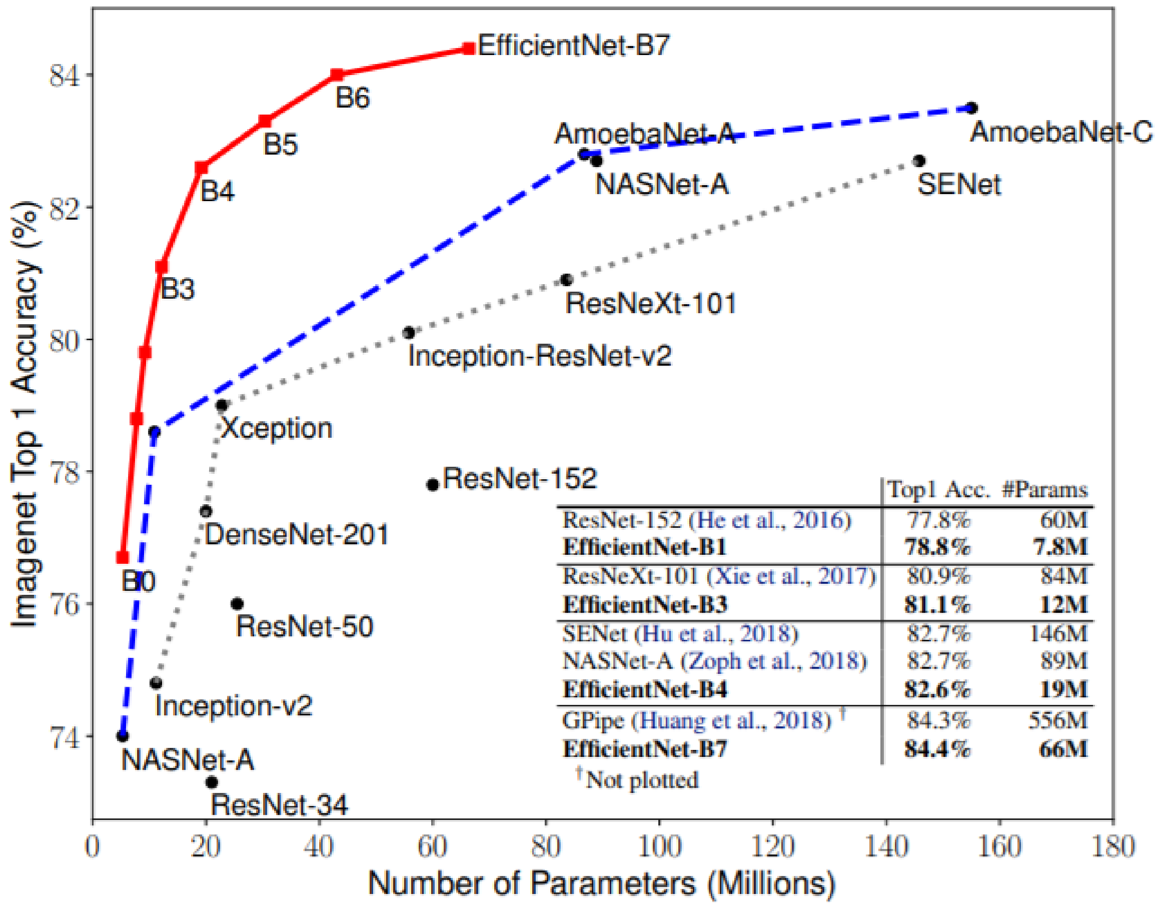


	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**79.1%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.6%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.9%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.3%**	**66M**

† Not plotted

EfficientNet:

We propose a new scaling method that uniformly scales all dimensions of depth/width/resolution using a simple yet highly effective compound coefficient



	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**78.8%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.1%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.6%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.4%**	**66M**

† Not plotted

Figure 1. **Model Size vs. ImageNet Accuracy.** All numbers are for single-crop, single-model. Our EfficientNets significantly outperform other ConvNets. In particular, EfficientNet-B7 achieves new state-of-the-art 84.4% top-1 accuracy but being 8.4x smaller and 6.1x faster than GPipe. EfficientNet-B1 is 7.6x smaller and 5.7x faster than ResNet-152. Details are in Table 2 and 4.