Convolutional Neural Networks (CNN) Architectures II

UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

CNN Architectures II

Published as a conference paper at ICLR 2015

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.

1 INTRODUCTION

Convolutional networks (ConvNets) have recently enjoyed a great success in large-scale image and video recognition (Krizhevsky et al., 2012; Zeiler & Fergus, 2013; Sermanet et al., 2014; Simonyan & Zisserman, 2014) which has become possible due to the large public image repositories, such as ImageNet (Deng et al., 2009), and high-performance computing systems, such as GPUs or large-scale distributed clusters (Dean et al., 2012). In particular, an important role in the advance of deep visual recognition architectures has been played by the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) (Russakovsky et al., 2014), which has served as a testbed for a few generations of large-scale image classification systems, from high-dimensional shallow feature encodings (Perronnin et al., 2010) (the winner of ILSVRC-2011) to deep ConvNets (Krizhevsky et al., 2012) (the winner of ILSVRC-2012).

With ConvNets becoming more of a commodity in the computer vision field, a number of attempts have been made to improve the original architecture of Krizhevsky et al. (2012) in a bid to achieve better accuracy. For instance, the best-performing submissions to the ILSVRC-2013 (Zeiler & Fergus, 2013; Sermanet et al., 2014) utilised smaller receptive window size and smaller stride of the first convolutional layer. Another line of improvements dealt with training and testing the networks densely over the whole image and over multiple scales (Sermanet et al., 2014; Howard, 2014). In this paper, we address another important aspect of ConvNet architecture design – its depth. To this end, we 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.

As a result, we come up with significantly more accurate ConvNet architectures, which not only achieve the state-of-the-art accuracy on ILSVRC classification and localisation tasks, but are also applicable to other image recognition datasets, where they achieve excellent performance even when used as a part of a relatively simple pipelines (e.g. deep features classified by a linear SVM without fine-tuning). We have released our two best-performing models[1] to facilitate further research.

VGG

#16 #19

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

VGG #16 or
VGG #19

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

- Filter size is constant
- Number of filters increase along the architecture
- Pre-training

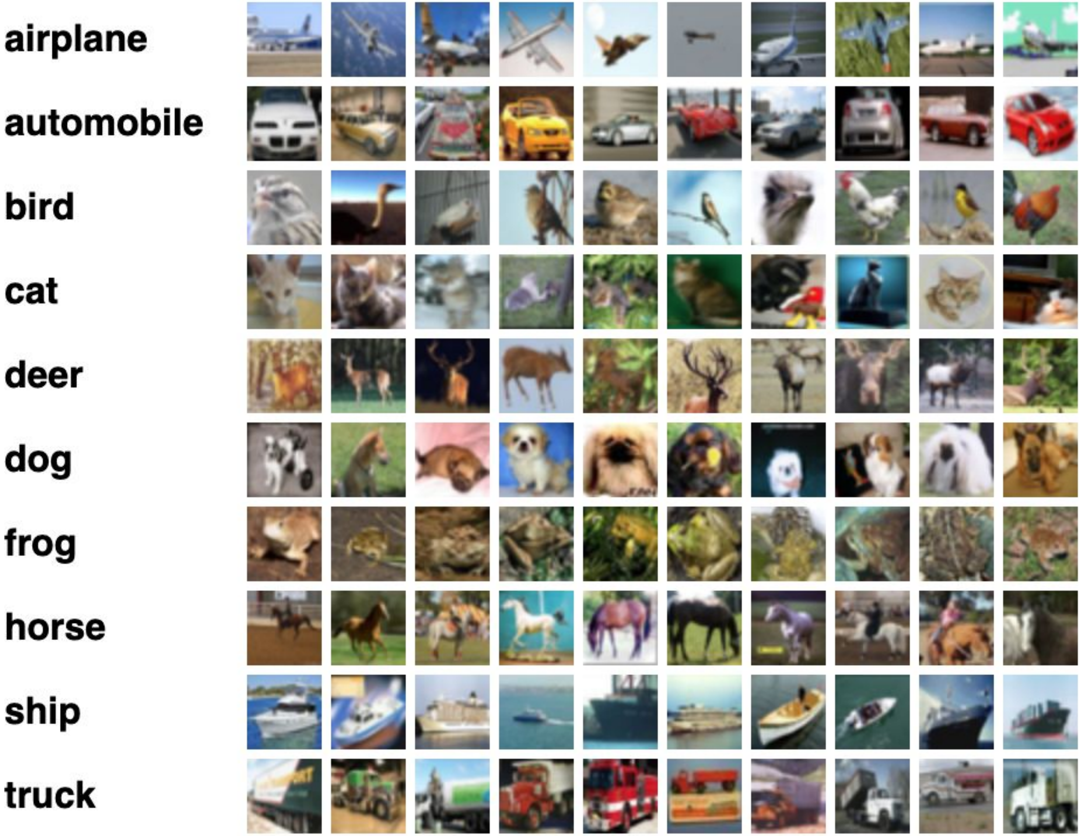
```python
# import the necessary packages
from tensorflow.keras.applications import VGG16

model = VGG16(weights="imagenet")
```

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
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

Layer Type	Output Size	Filter Size / Stride
INPUT IMAGE	$32 \times 32 \times 3$	
CONV	$32 \times 32 \times 32$	$3 \times 3, K = 32$
ACT	$32 \times 32 \times 32$	
BN	$32 \times 32 \times 32$	
CONV	$32 \times 32 \times 32$	$3 \times 3, K = 32$
ACT	$32 \times 32 \times 32$	
BN	$32 \times 32 \times 32$	
POOL	$16 \times 16 \times 32$	2×2
DROPOUT	$16 \times 16 \times 32$	
CONV	$16 \times 16 \times 64$	$3 \times 3, K = 64$
ACT	$16 \times 16 \times 64$	
BN	$16 \times 16 \times 64$	
CONV	$16 \times 16 \times 64$	$3 \times 3, K = 64$
ACT	$16 \times 16 \times 64$	
BN	$16 \times 16 \times 64$	
POOL	$8 \times 8 \times 64$	2×2
DROPOUT	$8 \times 8 \times 64$	
FC	512	
ACT	512	
BN	512	
DROPOUT	512	
FC	10	
SOFTMAX	10	

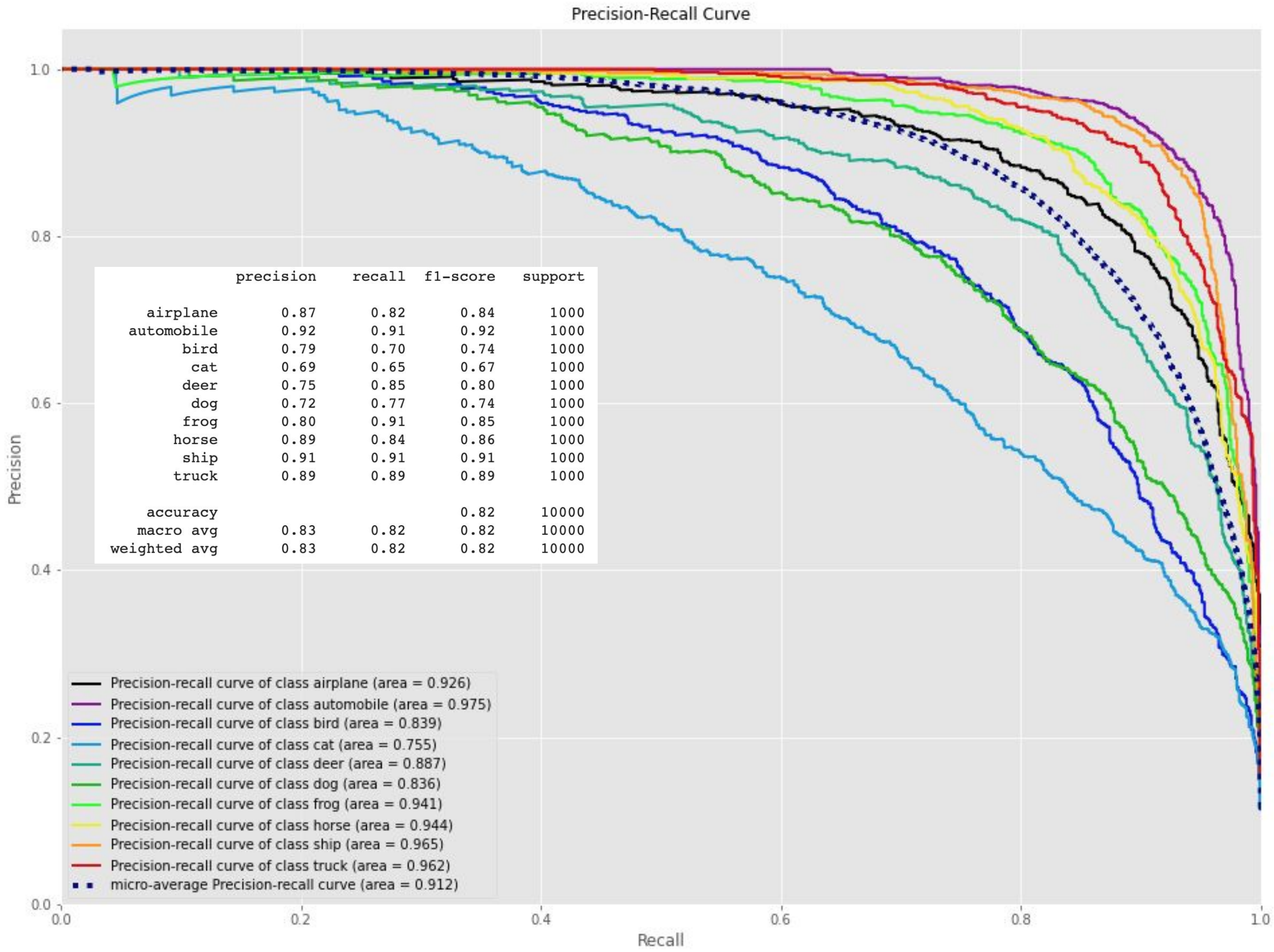
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 32)        896
_________________________________________________________________
activation (Activation)      (None, 32, 32, 32)        0
_________________________________________________________________
batch_normalization (BatchNo (None, 32, 32, 32)        128
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 32)        9248
_________________________________________________________________
activation_1 (Activation)    (None, 32, 32, 32)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 32, 32, 32)        128
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 64)        18496
_________________________________________________________________
activation_2 (Activation)    (None, 16, 16, 64)        0
_________________________________________________________________
batch_normalization_2 (Batch (None, 16, 16, 64)        256
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 16, 16, 64)        36928
_________________________________________________________________
activation_3 (Activation)    (None, 16, 16, 64)        0
_________________________________________________________________
batch_normalization_3 (Batch (None, 16, 16, 64)        256
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 8, 8, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 4096)              0
_________________________________________________________________
dense (Dense)                (None, 512)               2097664
_________________________________________________________________
activation_4 (Activation)    (None, 512)               0
_________________________________________________________________
batch_normalization_4 (Batch (None, 512)               2048
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                5130
_________________________________________________________________
activation_5 (Activation)    (None, 10)                0
=================================================================
Total params: 2,171,178
Trainable params: 2,169,770
Non-trainable params: 1,408
_________________________________________________________________
```

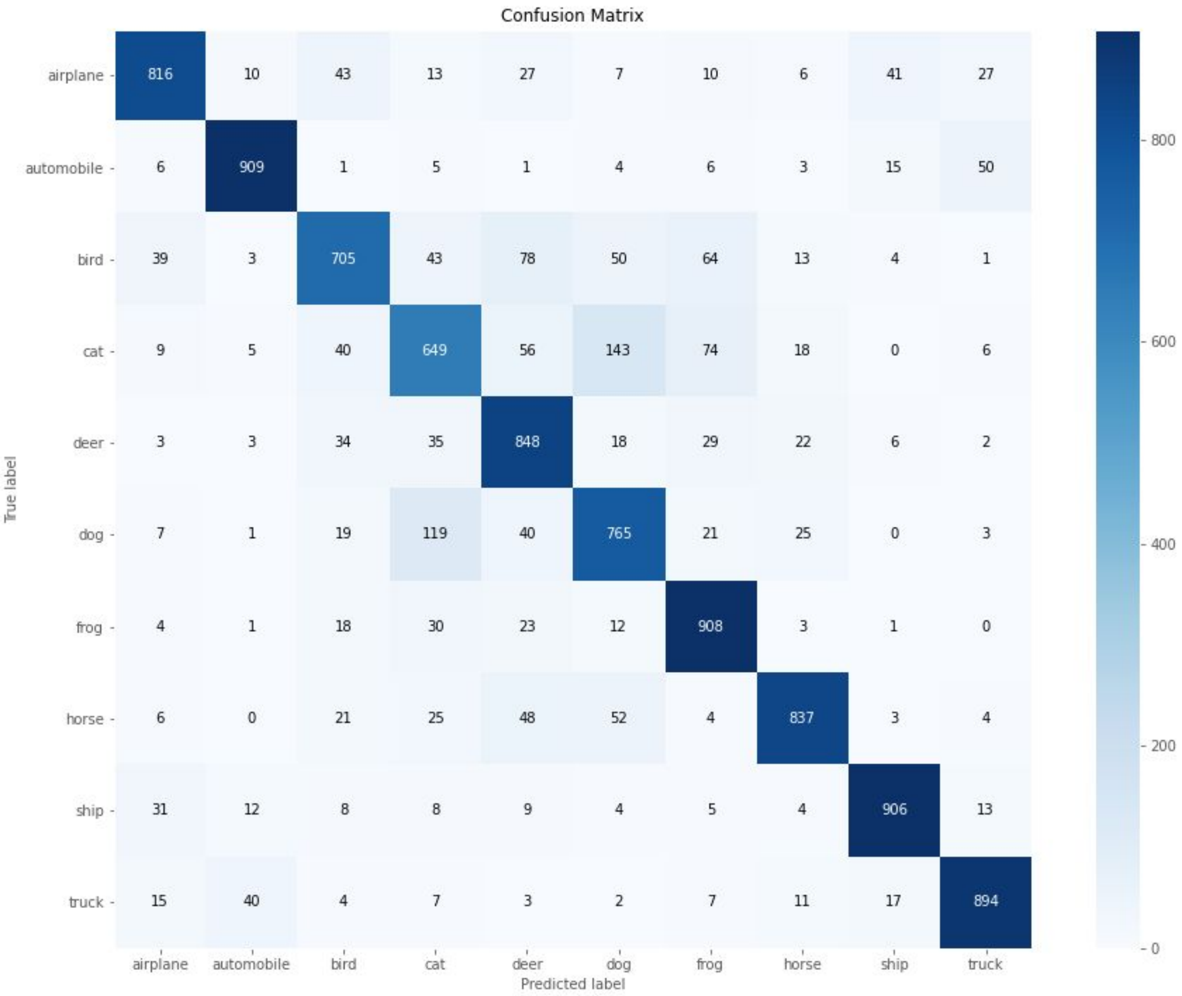


https://www.cs.toronto.edu/~kriz/cifar.html

The CIFAR-10 dataset

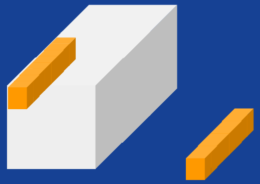
- 60,000 32x32x3 images
- 10 classes
- 6000 images per class
- 50,000 training images
- 10,000 test images


Precision-Recall Curve
1.0
0.8
0.6
0.4
0.2
0.0
Precision
0.0
0.2
0.4
0.6
0.8
1.0
Recall
precision recall f1-score support
airplane 0.87 0.82 0.84 1000
automobile 0.92 0.91 0.92 1000
bird 0.79 0.70 0.74 1000
cat 0.69 0.65 0.67 1000
deer 0.75 0.85 0.80 1000
dog 0.72 0.77 0.74 1000
frog 0.80 0.91 0.85 1000
horse 0.89 0.84 0.86 1000
ship 0.91 0.91 0.91 1000
truck 0.89 0.89 0.89 1000
accuracy 0.82 10000
macro avg 0.83 0.82 0.82 10000
weighted avg 0.83 0.82 0.82 10000
Precision-recall curve of class airplane (area = 0.926)
Precision-recall curve of class automobile (area = 0.975)
Precision-recall curve of class bird (area = 0.839)
Precision-recall curve of class cat (area = 0.755)
Precision-recall curve of class deer (area = 0.887)
Precision-recall curve of class dog (area = 0.836)
Precision-recall curve of class frog (area = 0.941)
Precision-recall curve of class horse (area = 0.944)
Precision-recall curve of class ship (area = 0.965)
Precision-recall curve of class truck (area = 0.962)
micro-average Precision-recall curve (area = 0.912)


Confusion Matrix
True label
Predicted label
airplane 816 10 43 13 27 7 10 6 41 27
automobile 6 909 1 5 1 4 6 3 15 50
bird 39 3 705 43 78 50 64 13 4 1
cat 9 5 40 649 56 143 74 18 0 6
deer 3 3 34 35 848 18 29 22 6 2
dog 7 1 19 119 40 765 21 25 0 3
frog 4 1 18 30 23 12 908 3 1 0
horse 6 0 21 25 48 52 4 837 3 4
ship 31 12 8 8 9 4 5 4 906 13
truck 15 40 4 7 3 2 7 11 17 894
airplane automobile bird cat deer dog frog horse ship truck
800
600
400
200
0

Predictions

How to Use 1x1 Convolutions to Manage Model Complexity



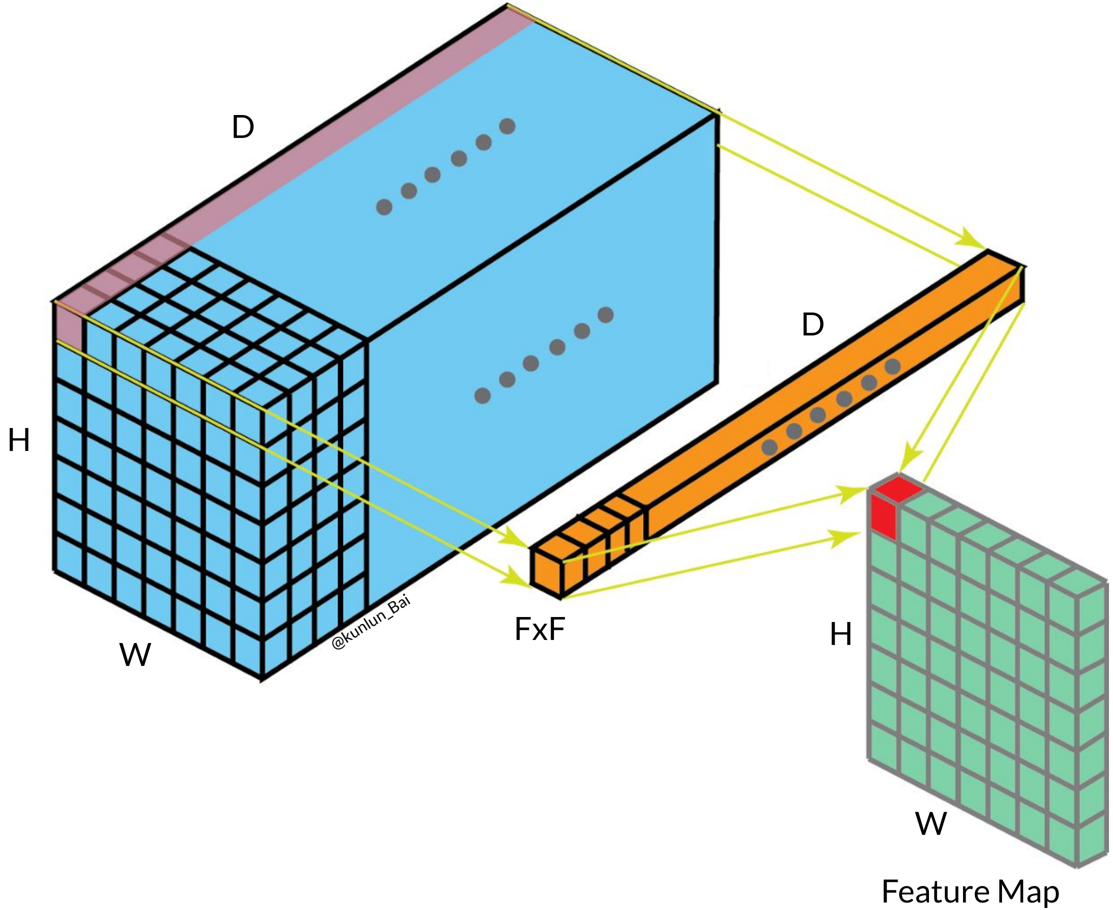
Network In Network

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

Abstract

We propose a novel deep network structure called "**N**etwork **I**n **N**etwork"(NIN) to enhance model discriminability for local patches within the receptive field. The conventional convolutional layer uses linear filters followed by a nonlinear activation function to scan the input. Instead, we build micro neural networks with more complex structures to abstract the data within the receptive field. We instantiate the micro neural network with a multilayer perceptron, which is a potent function approximator. The feature maps are obtained by sliding the micro networks over the input in a similar manner as CNN; they are then fed into the next layer. Deep NIN can be implemented by stacking mutiple of the above described structure. With enhanced local modeling via the micro network, we are able to utilize global average pooling over feature maps in the classification layer, which is easier to interpret and less prone to overfitting than traditional fully connected layers. We demonstrated the state-of-the-art classification performances with NIN on CIFAR-10 and CIFAR-100, and reasonable performances on SVHN and MNIST datasets.

1 Introduction



1x1 convolutional layer

The depth of the output of one convolutional layer is only defined by the number of parallel filters applied to the input.

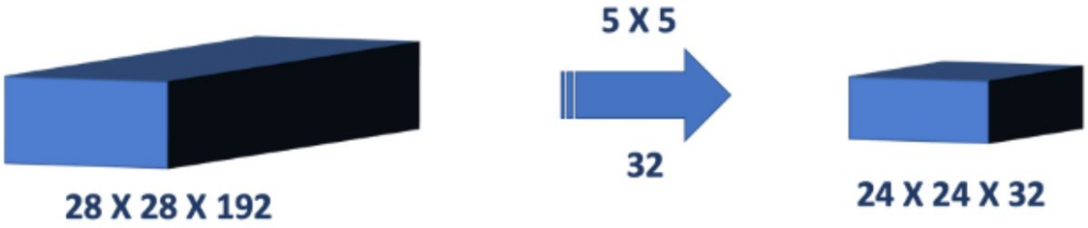
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Problem of Too Many Feature Maps

The depth of the input or number of filters used in convolutional layers often increases with the depth of the network, resulting in an increase in the number of resulting feature maps.

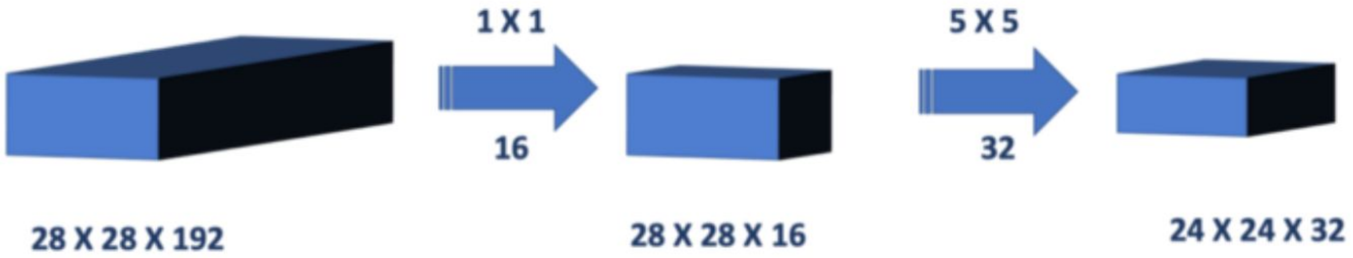
A large number of feature maps in a convolutional neural network can cause a problem of computational demand.

Pooling layers are designed to downscale feature maps. Nevertheless, pooling layers do not change the number of filters in the model, the depth, or number of channels.




```python
# example of a 1x1 filter for dimensionality reduction
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D
# create model
model = Sequential()
model.add(Conv2D(32, (5,5), padding="valid", activation="relu", input_shape=(28, 28, 192)))
# summarize model
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_17 (Conv2D)           (None, 24, 24, 32)        153632
=================================================================
Total params: 153,632
Trainable params: 153,632
Non-trainable params: 0
```




```python
# create model
model = Sequential()
model.add(Conv2D(16, (1,1), activation="relu", input_shape=(28, 28, 192)))
model.add(Conv2D(32, (5,5), padding="valid", activation="relu"))
# summarize model
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_14 (Conv2D)           (None, 28, 28, 16)        3088
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 24, 24, 32)        12832
=================================================================
Total params: 15,920
Trainable params: 15,920
Non-trainable params: 0
```

GoogLeNet

Going deeper with convolutions

Christian Szegedy
Google Inc.

Wei Liu
University of North Carolina, Chapel Hill

Yangqing Jia
Google Inc.

Pierre Sermanet
Google Inc.

Scott Reed
University of Michigan

Dragomir Anguelov
Google Inc.

Dumitru Erhan
Google Inc.

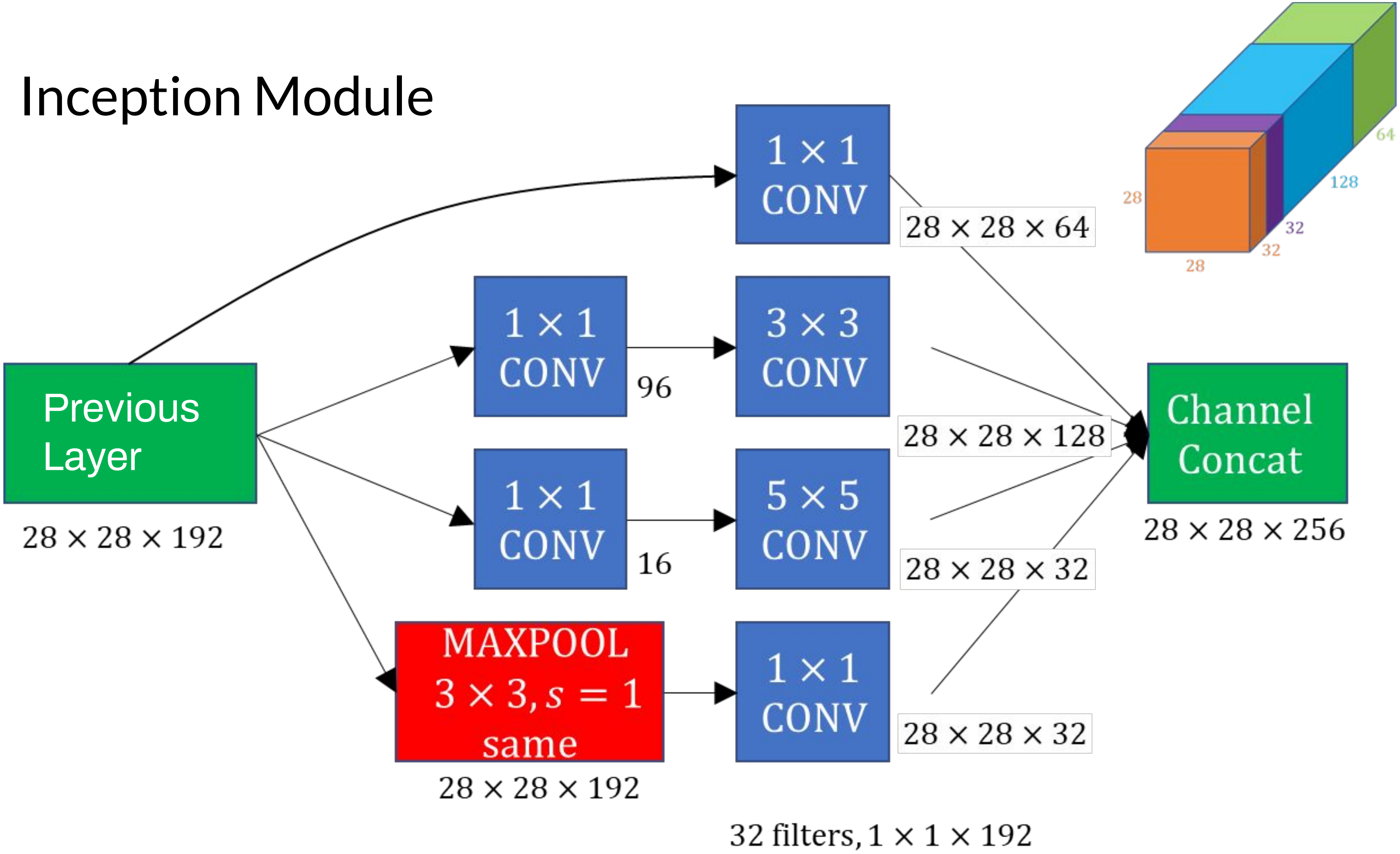
Vincent Vanhoucke
Google Inc.

Andrew Rabinovich
Google Inc.

Abstract

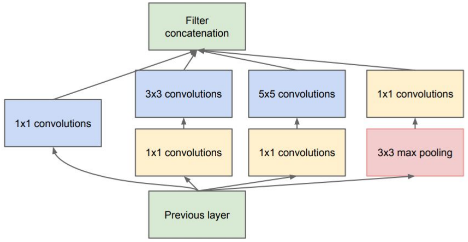
We propose a deep convolutional neural network architecture codenamed Inception, which was responsible for setting the new state of the art for classification and detection in the ImageNet Large-Scale Visual Recognition Challenge 2014 (ILSVRC14). The main hallmark of this architecture is the improved utilization of the computing resources inside the network. This was achieved by a carefully crafted design that allows for increasing the depth and width of the network while keeping the computational budget constant. To optimize quality, the architectural decisions were based on the Hebbian principle and the intuition of multi-scale processing. One particular incarnation used in our submission for ILSVRC14 is called GoogLeNet, a 22 layers deep network, the quality of which is assessed in the context of classification and detection.

Inception Module



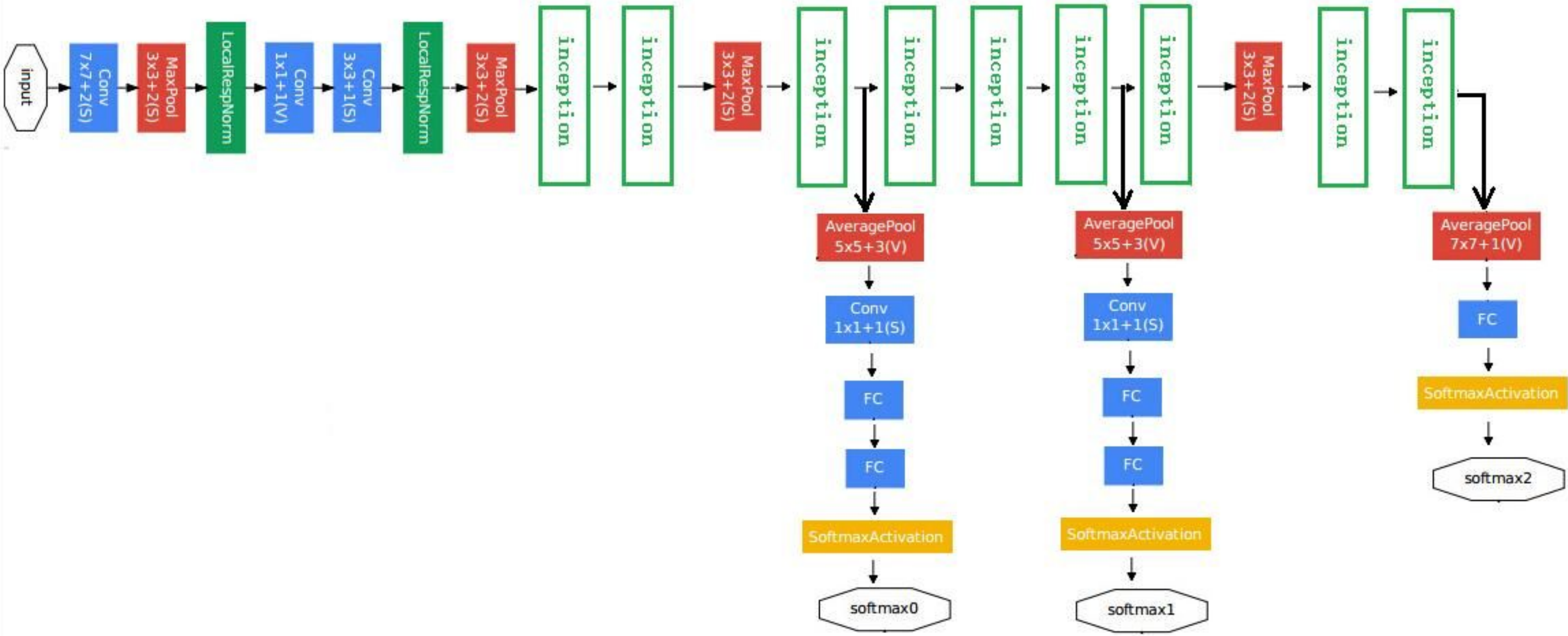
GoogLeNet incarnation of the Inception architecture

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								



+- 6.8M

```
# The total loss used by the inception net during training.
total_loss = real_loss + 0.3 * aux_loss_0 + 0.3 * aux_loss_1
```



MiniGoogLeNet

UNDERSTANDING DEEP LEARNING REQUIRES RE-THINKING GENERALIZATION

Chiyuan Zhang[*]
Massachusetts Institute of Technology
chiyuan@mit.edu

Samy Bengio
Google Brain
bengio@google.com

Moritz Hardt
Google Brain
mrtz@google.com

Benjamin Recht[†]
University of California, Berkeley
brecht@berkeley.edu

Oriol Vinyals
Google DeepMind
vinyals@google.com

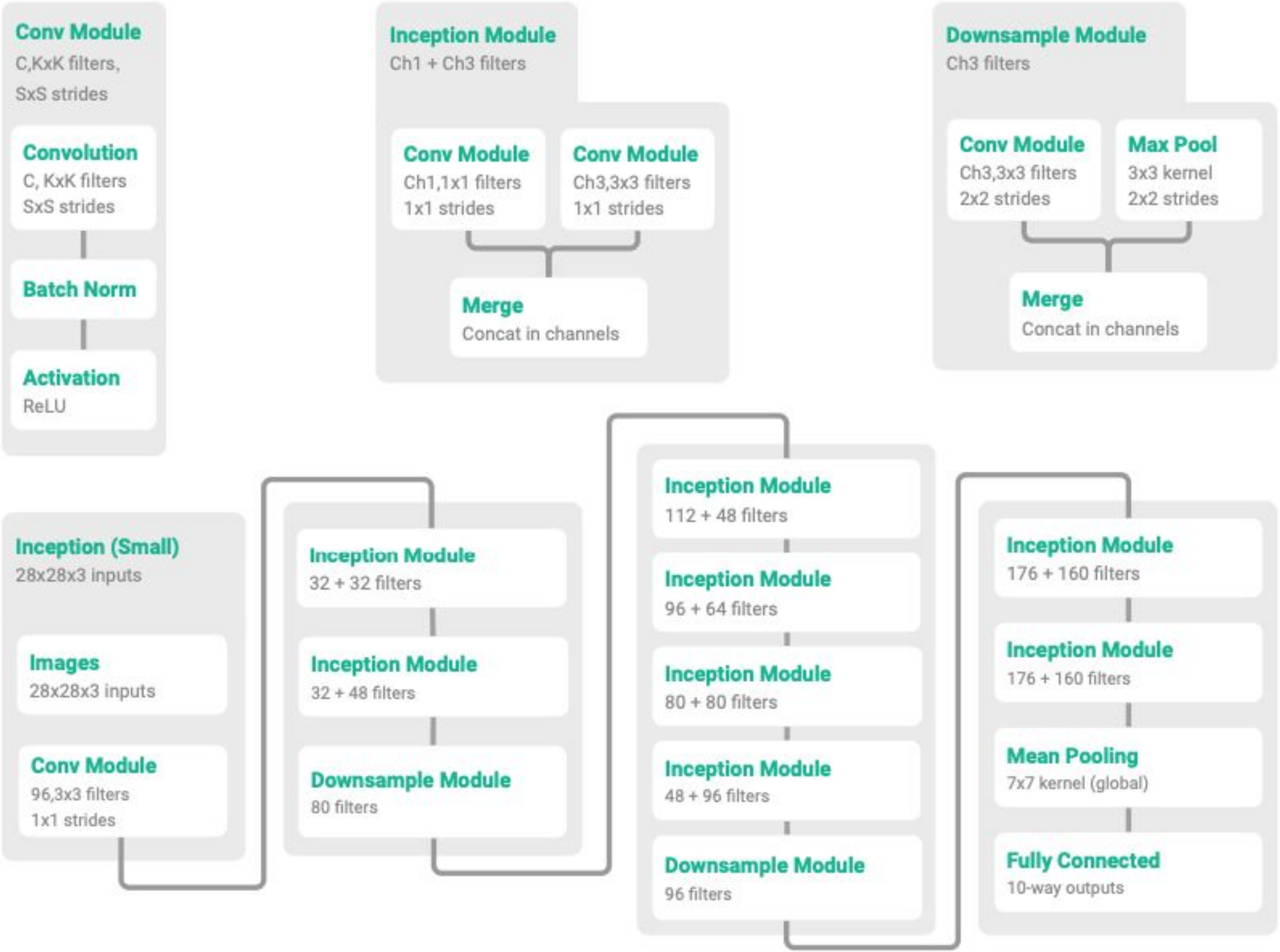
ABSTRACT

Despite their massive size, successful deep artificial neural networks can exhibit a remarkably small difference between training and test performance. Conventional wisdom attributes small generalization error either to properties of the model family, or to the regularization techniques used during training.

Through extensive systematic experiments, we show how these traditional approaches fail to explain why large neural networks generalize well in practice. Specifically, our experiments establish that state-of-the-art convolutional networks for image classification trained with stochastic gradient methods easily fit a random labeling of the training data. This phenomenon is qualitatively unaffected by explicit regularization, and occurs even if we replace the true images by completely unstructured random noise. We corroborate these experimental findings with a theoretical construction showing that simple depth two neural networks already have perfect finite sample expressivity as soon as the number of parameters exceeds the number of data points as it usually does in practice.

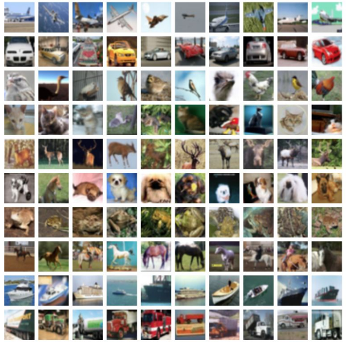
We interpret our experimental findings by comparison with traditional models.

1 INTRODUCTION



1.6M Params.

CIFAR-10



MiniGoogLeNet

	precision	recall	f1-score	support
airplane	0.91	0.92	0.92	1000
automobile	0.95	0.97	0.96	1000
bird	0.89	0.85	0.87	1000
cat	0.81	0.80	0.81	1000
deer	0.89	0.90	0.90	1000
dog	0.86	0.85	0.85	1000
frog	0.89	0.95	0.92	1000
horse	0.94	0.92	0.93	1000
ship	0.95	0.94	0.94	1000
truck	0.94	0.94	0.94	1000
accuracy			0.90	10000
macro avg	0.90	0.90	0.90	10000
weighted avg	0.90	0.90	0.90	10000

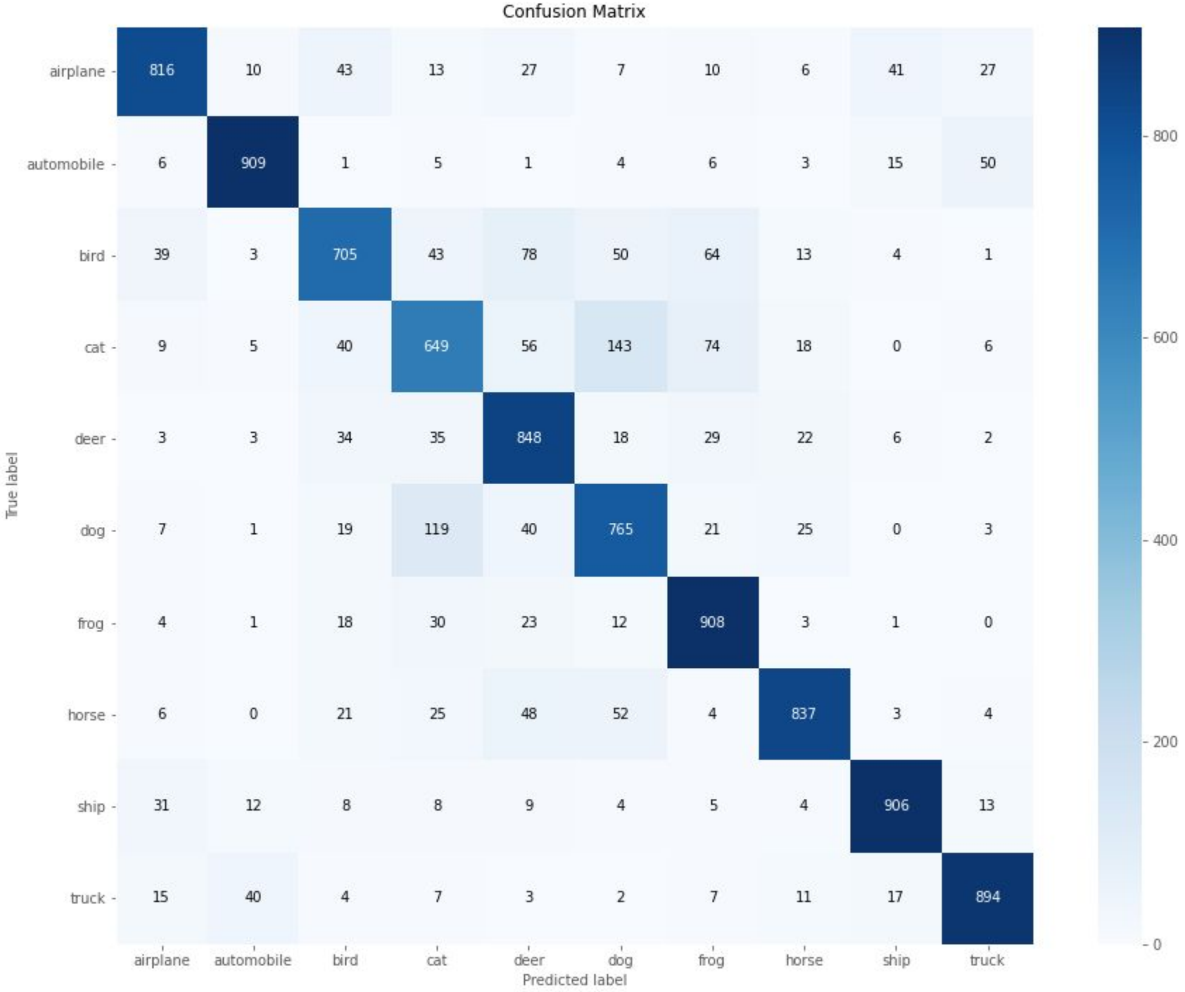
1.6M Params.

MiniVGG

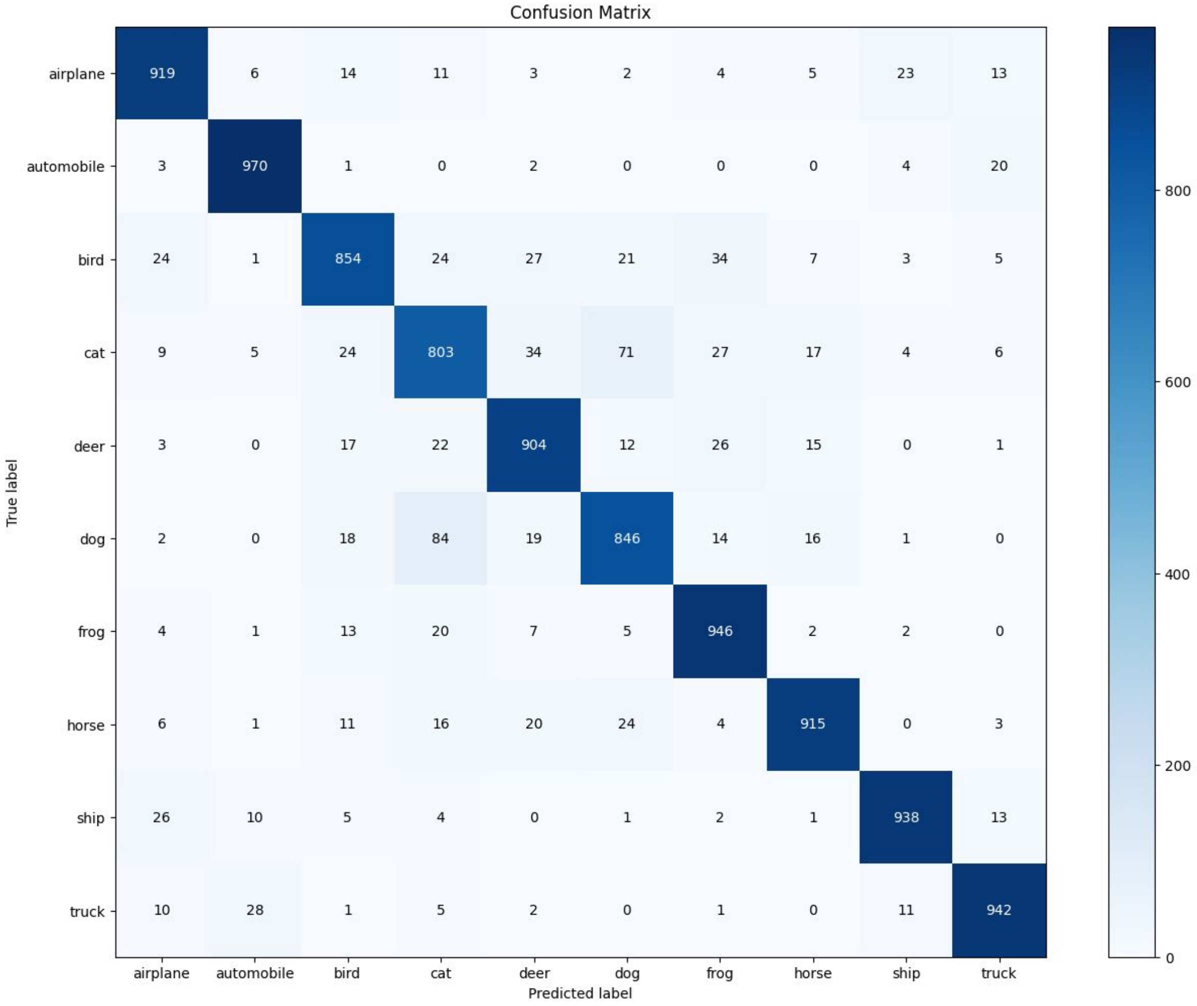
	precision	recall	f1-score	support
airplane	0.87	0.82	0.84	1000
automobile	0.92	0.91	0.92	1000
bird	0.79	0.70	0.74	1000
cat	0.69	0.65	0.67	1000
deer	0.75	0.85	0.80	1000
dog	0.72	0.77	0.74	1000
frog	0.80	0.91	0.85	1000
horse	0.89	0.84	0.86	1000
ship	0.91	0.91	0.91	1000
truck	0.89	0.89	0.89	1000
accuracy			0.82	10000
macro avg	0.83	0.82	0.82	10000
weighted avg	0.83	0.82	0.82	10000

2.1M Params.

MiniVGG


Confusion Matrix
airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck
816 10 43 13 27 7 10 6 41 27
6 909 1 5 1 4 6 3 15 50
39 3 705 43 78 50 64 13 4 1
9 5 40 649 56 143 74 18 0 6
3 3 34 35 848 18 29 22 6 2
7 1 19 119 40 765 21 25 0 3
4 1 18 30 23 12 908 3 1 0
6 0 21 25 48 52 4 837 3 4
31 12 8 8 9 4 5 4 906 13
15 40 4 7 3 2 7 11 17 894
True label
Predicted label
800
600
400
200
0

MiniGoogLeNet



The Tiny ImageNet Challenge

64 x 64 x 3

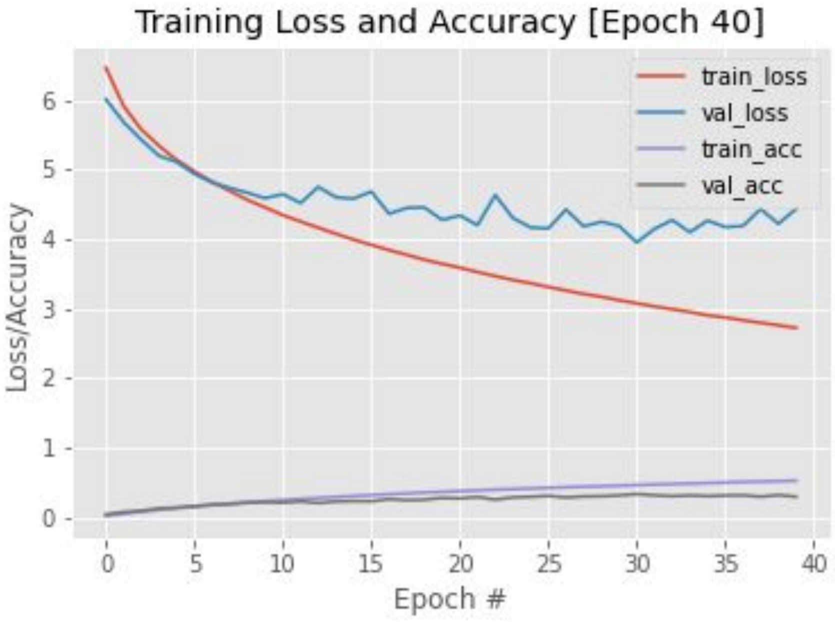
cs231n Stanford course on Convolutional Neural Networks

200 classes, each class includes 450 training images, 50 validation images, and 50 testing images
90k training images
10k validations
10k test

type	patch size/stride	output size	depth	#1x1	#3x3 reduce	#3x3	#5x5 reduce	#5x5	pool proj
convolution	5x5/1	64, 64, 64	1						
max pool	3x3/2	32, 32, 64	0						
convolution	3x3/1	32, 32, 192	2		64	192			
max pool	3x3/2	16, 16, 192	0						
inception (3a)		16, 16, 256	2	64	96	128	16	32	32
inception (3b)		16, 16, 480	2	128	128	192	32	96	64
max pool	3x3/2	8, 8, 480	0						
inception (4a)		8, 8, 512	2	192	96	208	16	48	64
inception (4b)		8, 8, 512	2	160	112	224	24	64	64
inception (4c)		8, 8, 512	2	128	128	256	24	64	64
inception (4d)		8, 8, 528	2	112	144	288	32	64	64
inception (4e)		8, 8, 832	2	256	160	320	32	128	128
max pool	3x3/2	4, 4, 832	0						
avg pool	4x4/1	1, 1, 832	0						
dropout (40%)		1, 1, 832	0						
linear		1, 1, 200	1						
softmax		1, 1, 200	0						

5a
5b

3.6M Params



- epoch_10.hdf5
- epoch_15.hdf5
- epoch_20.hdf5
- epoch_25.hdf5
- epoch_30.hdf5
- epoch_35.hdf5
- epoch_40.hdf5
- epoch_5.hdf5

```
opt = SGD(1e-3,momentum=0.9)
```

```
[INFO] predicting on test data...
[INFO] rank-1: 30.23%
[INFO] rank-5: 54.34%
```

DeepGoogLeNet

[Extensions]

1. Change the **conv_module** to use CONV => RELU => BN instead of the original CONV => BN => RELU ordering.
2. Attempt using **ELUs** instead of **ReLUs**.
3. Use Wandb.ai

Experiment #01

Epoch	Learning Rate
1 - 25	$1e-02$
26 - 35	$1e-03$
36 - 65	$1e-04$

Inception modules
4a-4e is removed
SGD

Experiment #02

Epoch	Learning Rate
1 - 20	1e-3
21 - 30	1e-4
31 - 40	1e-5

Inception modules
4a-4e is removed
Adam

Experiment #03

Epoch	Learning Rate
1 - 40	1e-3
41 - 60	1e-4
61 - 70	1e-5

Inception modules
4a-4e is enabled
Adam