**یادگیری عمیق**

**جلسه ۱۴، ۱۵ و ۱۶**

# یادگیری عمیق برای بینایی کامپیوتری

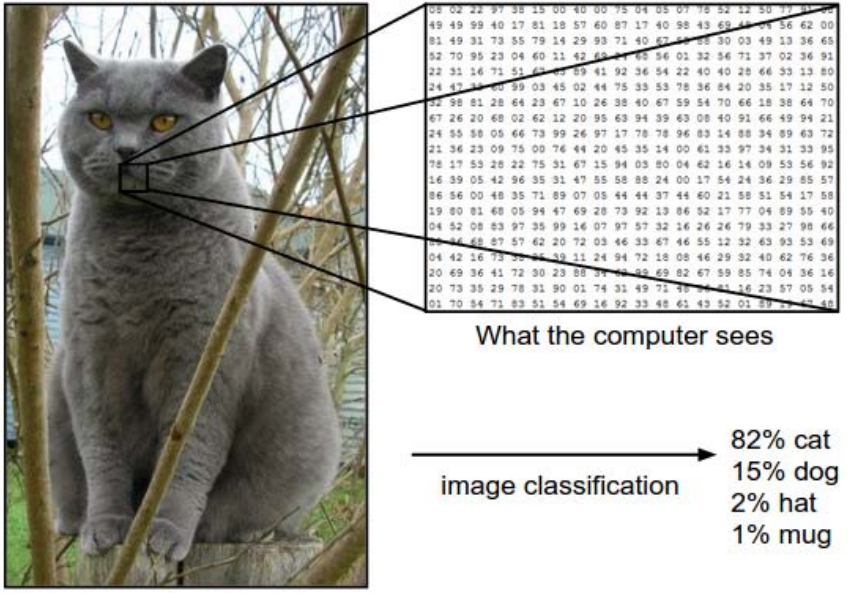
## Deep Learning for Computer Vision


کاظم فولادی قلعه

دانشکده مهندسی، پردیس فارابی

دانشگاه تهران

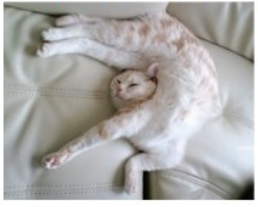

http://courses.fouladi.ir/deep


What the computer sees
image classification
82% cat
15% dog
2% hat
1% mug

Viewpoint variation

Scale variation

Deformation



Occlusion

Illumination conditions

Background clutter

Intra-class variation

یادگیری عمیق

یادگیری عمیق برای بینایی کامپیوتری

# ۱

# معرفی شبکه‌های عصبی کانوولوشنال (convnets)

# شبکه‌های عصبی کانوولوشنال

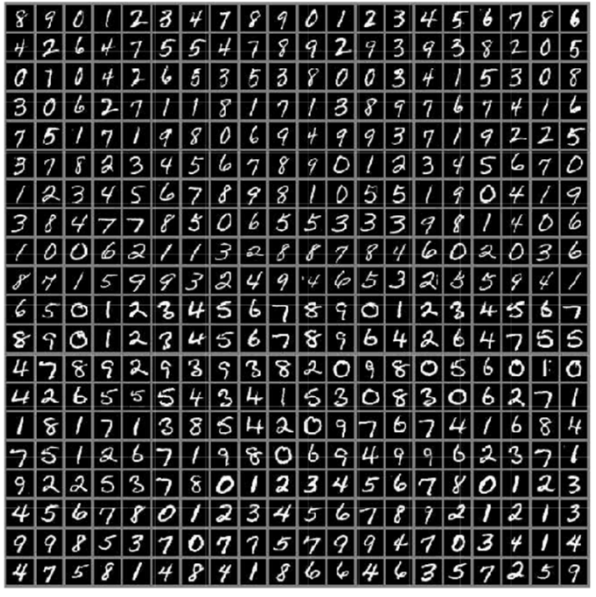
## CONVOLUTIONAL NEURAL NETWORKS (CONVNETS)

شبکه‌های عصبی کانوولوشنال،
مدلی از یادگیری عمیق هستند
که تقریباً در عموم کاربردهای بینایی کامپیوتری مورد استفاده قرار می‌گیرند.

# شبکه‌های عصبی کانوولوشنال

## یک مثال کاربردی: بازشناسی ارقام دست‌نویس

The problem we're trying to solve here is to classify grayscale images of handwritten digits (28 × 28 pixels) into their 10 categories (0 through 9).



# MNIST Dataset

A set of 60,000 training images,
plus 10,000 test images,
assembled by
the National Institute of Standards and Technology
(the NIST in MNIST)
in the 1980s.

# شبکه‌های عصبی کانوولوشنال

یک مثال کاربردی: بازشناسی ارقام دست‌نویس

## Instantiating a small convnet

```
from keras import layers
from keras import models
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

**نکته‌ی مهم:**

یک convnet تانسورهایی با شکل
`(image_height, image_width, image_channels)`
را به‌عنوان ورودی دریافت می‌کند.

# شبکه‌های عصبی کانوولوشنال

## یک مثال کاربردی: بازشناسی ارقام دست‌نویس: معماری convnet تا اینجا

```
>>> model.summary()
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
maxpooling2d_1 (MaxPooling2D)  (None, 13, 13, 32)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
maxpooling2d_2 (MaxPooling2D)  (None, 5, 5, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 3, 3, 64)          36928
=================================================================
Total params: 55,744
Trainable params: 55,744
Non-trainable params: 0
```

خروجی هر لایه‌ی Conv2D و MaxPooling2D یک تانسور سه‌بعدی با شکل
`(height, width, channels)`
است.

هرچه در عمیق شبکه پیش می‌رویم، بعدهای عرض و ارتفاع کوچک‌تر می‌شوند.
تعداد کانال‌ها با اولین آرگومان لایه‌های Conv2D مشخص می‌شود.

# شبکه‌های عصبی کانوولوشنال

یک مثال کاربردی: بازشناسی ارقام دست‌نویس: اضافه کردن پشته‌ای از لایه‌های متراکم

## Adding a classifier on top of the convnet

```
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

آخرین تانسور با شکل
(3,3,64)
را به‌عنوان ورودی به یک شبکه‌ی طبقه‌بندی‌کننده‌ی مصل متراکم وارد می‌کنیم:
پشته‌ای از لایه‌های متراکم (Dense).

چون این طبقه‌بندی‌کننده بردارهای یک–بعدی را پردازش می‌کند،
ابتدا تانسور سه–بعدی را تخت (flat) می‌کنیم.

# شبکه‌های عصبی کانوولوشنال

## یک مثال کاربردی: بازشناسی ارقام دست‌نویس: معماری convnet

```
>>> model.summary()
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
maxpooling2d_1 (MaxPooling2D)  (None, 13, 13, 32)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
maxpooling2d_2 (MaxPooling2D)  (None, 5, 5, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 3, 3, 64)          36928
_________________________________________________________________
flatten_1 (Flatten)          (None, 576)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                36928
_________________________________________________________________
dense_2 (Dense)              (None, 10)                650
=================================================================
Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0
```

اکنون یک طبقه‌بندی‌کننده‌ی ۱۰–طبقه‌ای با استفاده از ۱۰ خروجی با تابع فعالیت softmax داریم.

# شبکه‌های عصبی کانوولوشنال

یک مثال کاربردی: بازشناسی ارقام دست‌نویس: آموزش شبکه

## Training the convnet on MNIST images

```
from keras.datasets import mnist
from keras.utils import to_categorical
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images = train_images.reshape((60000, 28, 28, 1))
train_images = train_images.astype('float32') / 255
test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype('float32') / 255
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
model.compile(optimizer='rmsprop',
loss='categorical_crossentropy',
metrics=['accuracy'])
model.fit(train_images, train_labels, epochs=5, batch_size=64)
```

# شبکه‌های عصبی کانوولوشنال

یک مثال کاربردی: بازشناسی ارقام دست‌نویس: آزمایش شبکه

```
>>> test_loss, test_acc = model.evaluate(test_images, test_labels)
>>> test_acc
0.99080000000000001
```

Whereas the densely connected network from chapter 2 had a test accuracy of **97.8%**, the basic convnet has a test accuracy of **99.3%**:

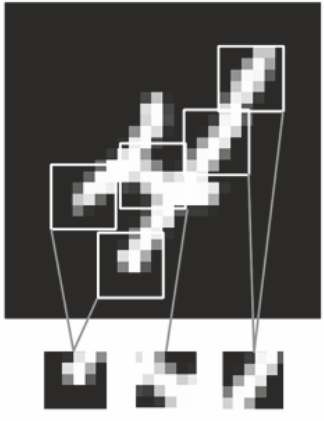
we decreased the error rate by 68% (relative).

# عمل کانوولوشن

مقایسه‌ی لایه‌ی متصل متراکم با لایه‌ی کانوولوشنال

THE CONVOLUTION OPERATION

| **لایه‌ی متصل متراکم**<br>*Densely Connected Layer* | **لایه‌ی کانوولوشنال**<br>*Convolution Layer* |
|---|---|
| در فضای ویژگی ورودی خود الگوهای **سراسری** را یاد می‌گیرند.<br>(مثلاً در MNIST الگوها حاوی همه‌ی پیکسل‌ها هستند) | در فضای ویژگی ورودی خود الگوهای **محلی** را یاد می‌گیرند.<br>(در تصاویر، الگوها در پنجره‌های کوچک دوبعدی از ورودی یافت می‌شوند؛ مثلاً پنجره‌های 3*3) |



تصویرها می‌توانند به الگوهای محلی مانند لبه‌ها، بافت‌ها و ... شکسته شوند.

# عمل کانوولوشن

دو ویژگی جالب ناشی از خصوصیت کلیدی convnet (یادگیری الگوهای محلی)

## THE CONVOLUTION OPERATION

**۱**

**یادگیری الگوهای تغییرناپذیر در جابه‌جایی**
*The patterns they learn are translation invariant*

جهان بصری اساساً نسبت به تغییر مکان، نامتغیر است.

پس از یادگیری یک الگو در قسمتی از تصویر، convnet می‌تواند آن الگو را در هر مکانی بازشناسی کند.
در حالی‌که شبکه‌ی dense باید الگو در مکان جدید را دوباره یاد بگیرد.

**۲**

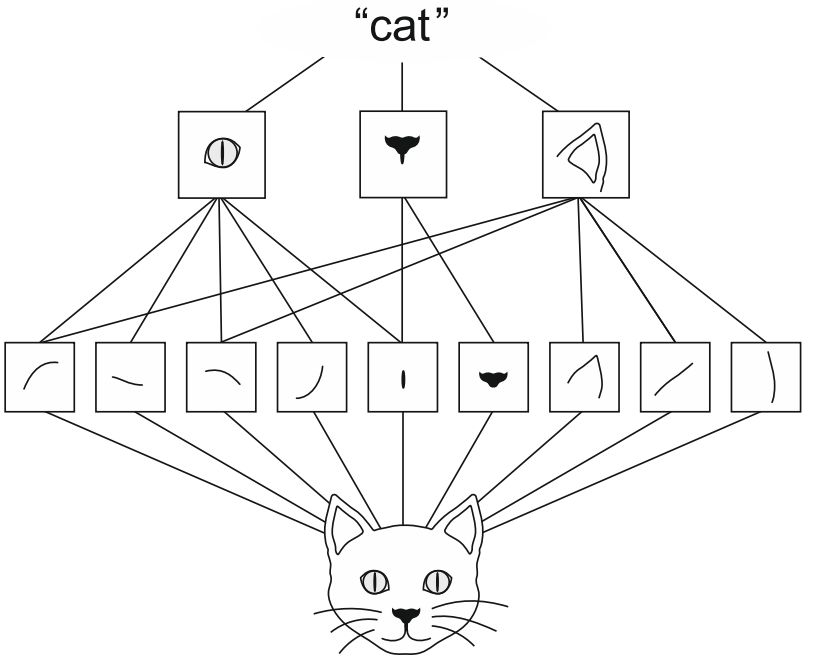
**یادگیری سلسله‌مراتب فضایی الگوها**
*They can learn spatial hierarchies of patterns*

جهان بصری اساساً سلسله‌مراتبی فضایی است.

اولین لایه‌ی کانوولوشنال الگوهای محلی کوچک مانند **لبه‌ها** را یاد می‌گیرد.
دومین لایه‌ی کانوولوشنال الگوهای بزرگ‌تر ساخته شده از ویژگی‌های اولین لایه را یاد خواهد گرفت و ...
این به convnet اجازه می‌دهد که مفاهیم بصری انتزاعی و پیچیده را به‌صورت افزایشی یاد بگیرد.

# عمل کانوولوشن

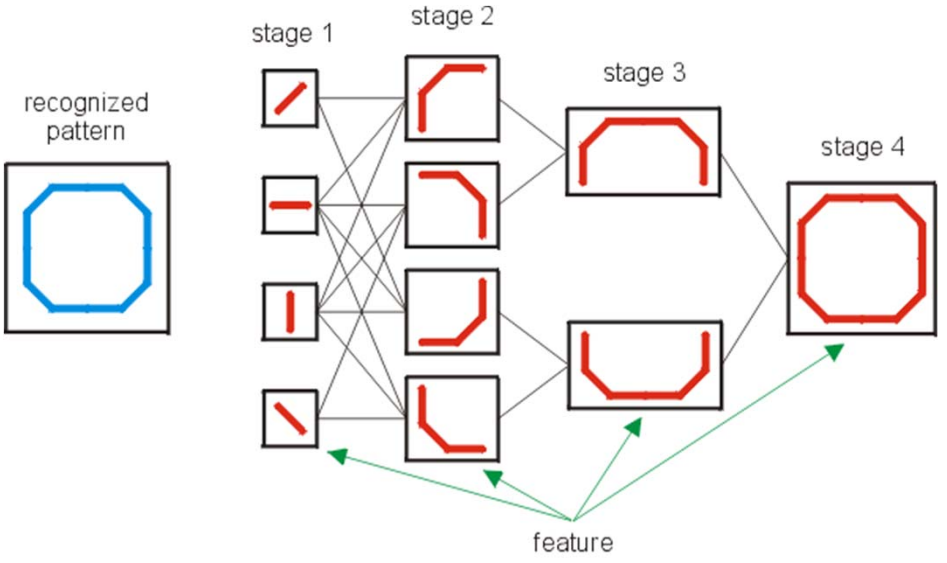
## یادگیری سلسله‌مراتب فضایی الگوها: مثال

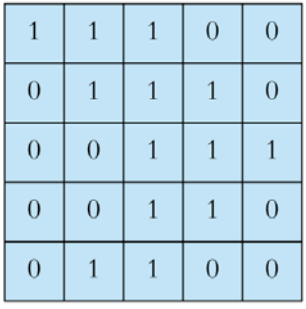



یک واژه‌ی دیداری (visual word)، یک سلسله‌مراتب فضایی از ماژول‌های دیداری را شکل می‌دهد:
لبه‌های فرامحلی در اشیای محلی (مانند چشم‌ها یا گوش‌ها) ترکیب می‌شوند،
که خود آنها در مفاهیم سطح بالاتر مانند «گربه» ترکیب می‌شوند.

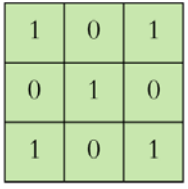
# عمل کانوولوشن

یادگیری سلسله‌مراتب فضایی الگوها: مثال: کارکرد نوعی لایه‌ها

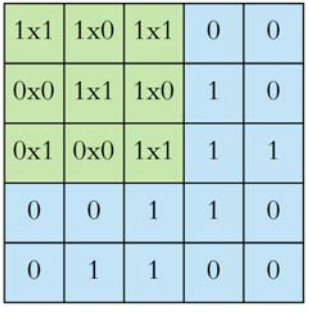




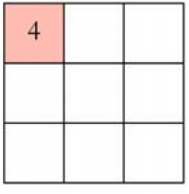

Input



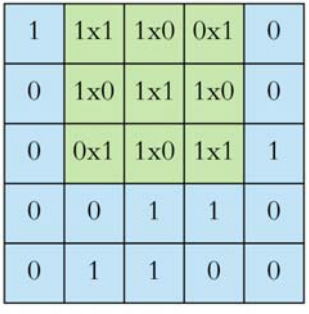

Filter / Kernel



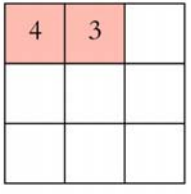

Input x Filter




Feature Map

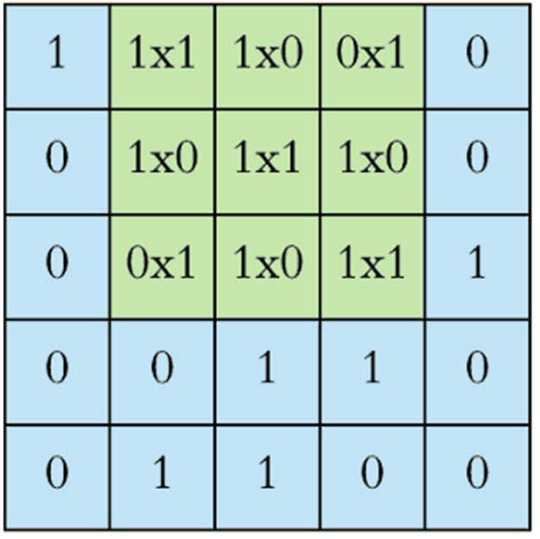


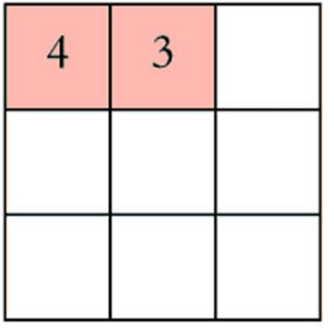

Input x Filter




Feature Map

https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2




https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

# عمل کانوولوشن

## نقشه‌های ویژگی

FEATURE MAPS

کانوولوشن‌ها روی تانسورهای سه–بعدی (نقشه‌های ویژگی)
با دو محور فضایی [ارتفاع و عرض] و یک محور عمق [کانال] انجام می‌شود.

عملیات کانوولوشن، پچ‌هایی را از نقشه‌ی ویژگی ورودی خود استخراج می‌کند
و تبدیل یکسانی را روی همه‌ی این پچ‌ها اعمال می‌کند
تا یک نقشه‌ی ویژگی خروجی ایجاد شود.

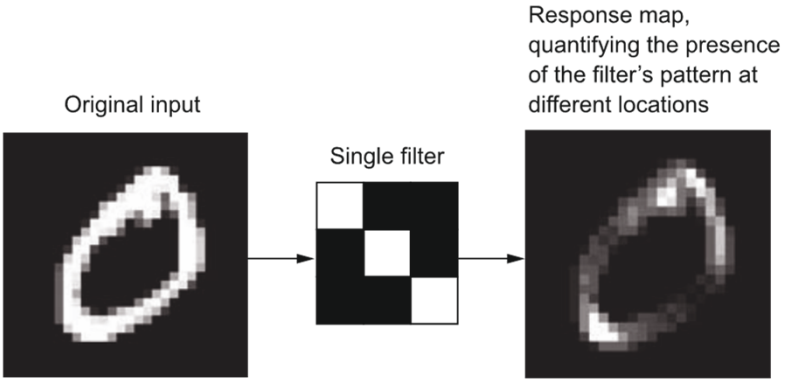

عمق نقشه‌ی ویژگی خروجی می‌تواند داخواه باشد،
زیرا عمق خروجی یک پارامتر لایه است.

کانال‌های مخلف در خروجی بیان‌گر رنگ نیستند، بلکه متناظر با **فیلترها** هستند.
فیلترها جنبه‌های مختلف داده‌های ورودی را کدگذاری می‌کنند:
در سطح بالا، یک فیلتر می‌تواند مثلاً مفهوم «حضور یک چهره در ورودی» را کدگذاری کند.

# عمل کانوولوشن

## نقشه‌های ویژگی: مثال

FEATURE MAPS




مفهوم نقشه‌ی پاسخ:
یک نقشه‌ی دو-بعدی از وجود
یک الگوی خاص
در قسمت‌های مختلف یک ورودی

در مثال MNIST، اولین لایه‌ی کانوولوشنی، یک نقشه‌ی ویژگی با اندازه‌ی (28,28,1) می‌گیرد
و یک نقشه‌ی ویژگی خروجی با اندازه‌ی (26,26,32) می‌دهد:
یعنی ۳۲ فیلتر را روی ورودی خودش محاسبه می‌کند.
هر یک از این ۳۲ کانال خروجی حاوی یک توری 26*26 از مقادیر است که نقشه‌ی پاسخ آن فیلتر روی ورودی است؛
و پاسخ آن فیلتر در مکان‌های مختلف ورودی را مشخص می‌کند.

# عمل کانوولوشن

پارامترهای کلیدی در تعریف عمل کانوولوشن

۱

**اندازه‌ی پچ‌ها**
*Size of the patches*

اندازه‌ی پچ‌های استخراج شده از ورودی‌ها

معمولاً ۳ × ۳ یا ۵ × ۵ است
انتخاب ۳ × ۳ معمول است.

۲

**عمق خروجی**
*Depth of the output*

عمق نقشه‌ی ویژگی خروجی

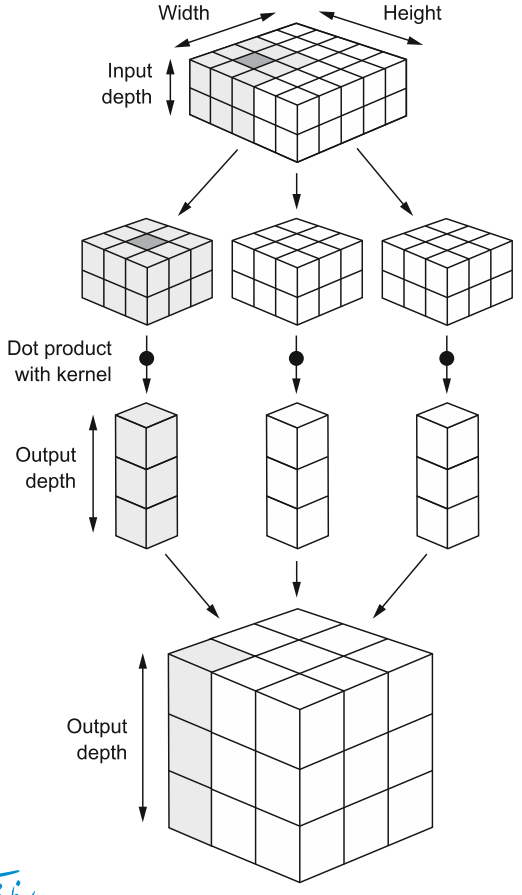
تعداد فیلترهایی که با عمل کانوولوشن محاسبه می‌شود.
در مثال: با عمق ۳۲ شروع شد و با عمق ۶۴ پایان یافت.

```
Conv2D(output_depth, (window_height, window_width))
```

در Keras، این پارامترها اولین آرگومان‌هایی هستند که به لایه‌ی Conv2D گذر داده می‌شوند.

# عمل کانوولوشن

عملکرد




در کانوولوشن:

› پنجره‌های کوچک (مثلاً ۳×۳) بر روی نقشه‌ی ویژگی ورودی سه–بعدی حرکت داده می‌شود، در هر مکان ممکن توقف می‌کند،

و یک پچ سه–بعدی از ویژگی‌های پیرامون خود به شکل زیر استخراج می‌کند:

```
(window_height, window_width, input_depth)
```

› سپس هر پچ سه–بعدی تحت تبدیل قرار می‌گیرد (از طریق ضرب یک تانسوری با ماتریس وزن یادگیری شده که به آن هسته‌ی کانوولوشن / convolution kernel گفته می‌شود.) تا به یک بردار یک–بعدی به شکل `output_depth` تبدیل شود.

› سپس همه‌ی این بردارها به‌صورت فضایی در کنار هم قرار می‌گیرند تا یک نقشه‌ی خروجی سه–بعدی به شکل زیر ایجاد شود:

```
(height, width, output_depth)
```

هر مکان در نقشه‌ی ویژگی خروجی، متناظر با همان مکان در نقشه‌ی ویژگی ورودی است.

# عمل کانوولوشن

## تفاوت عرض و ارتفاع در نقشه‌های ویژگی ورودی و خروجی

عرض و ارتفاع در نقشه‌ی ویژگی ورودی ممکن است با
عرض و ارتفاع در نقشه‌ی ویژگی خروجی متفاوت باشد.

به دو دلیل:

۱ **اثرات کناره**
*Border effects*

می‌تواند از طریق پدگذاری روی نقشه‌ی ویژگی ورودی مقابله شود.

۲ **اندازه‌ی قدم**
*Strides*

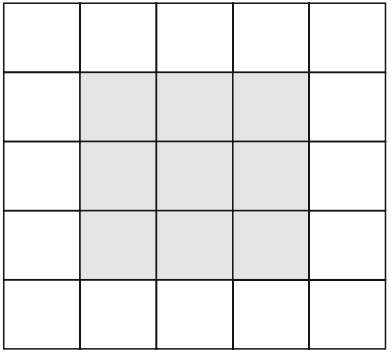
اندازه‌ی فاصله بین قرارگیری دو پنجره‌ی متوالی

# عمل کانوولوشن

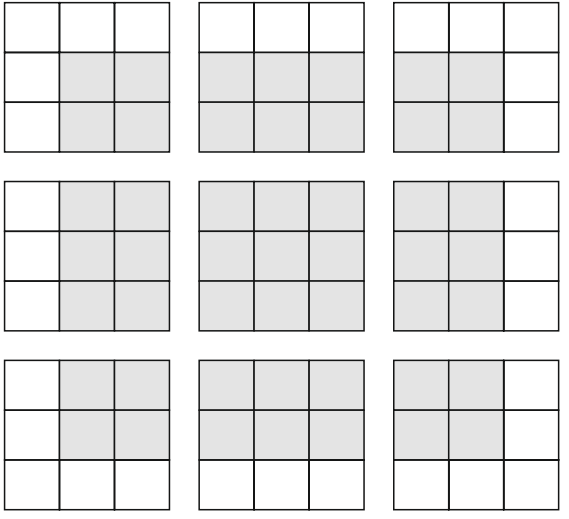
## اثرات کناره

## BORDER EFFECTS

اگر یک نقشه‌ی ویژگی ۵×۵ داشته باشیم،
تنها ۹ خانه وجود دارد که می‌توان یک پنجره‌ی ۳×۳
را در آن جای داد.



پس یک توری ۳×۳ تشکیل می‌شود ⇐
نقشه‌ی ویژگی خروجی ۳×۳ خواهد شد.



خروجی در هر بعد، ۲ واحد کوچک‌تر از ورودی می‌شود.

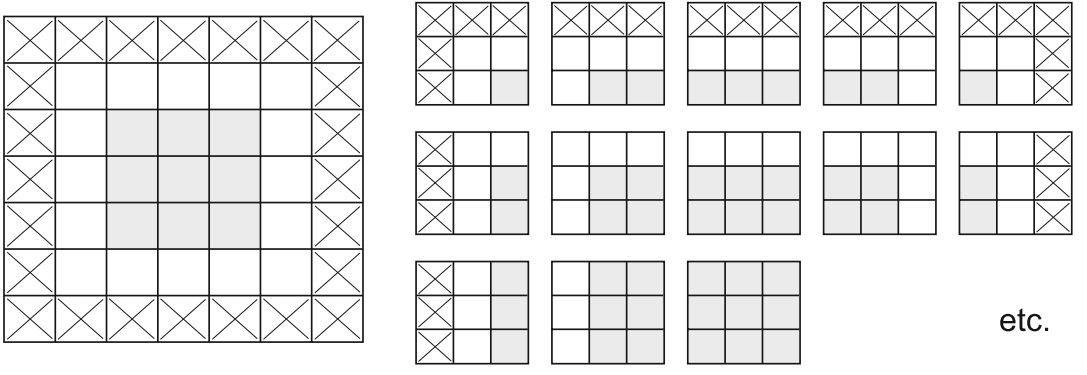
# عمل کانوولوشن

## پدگذاری

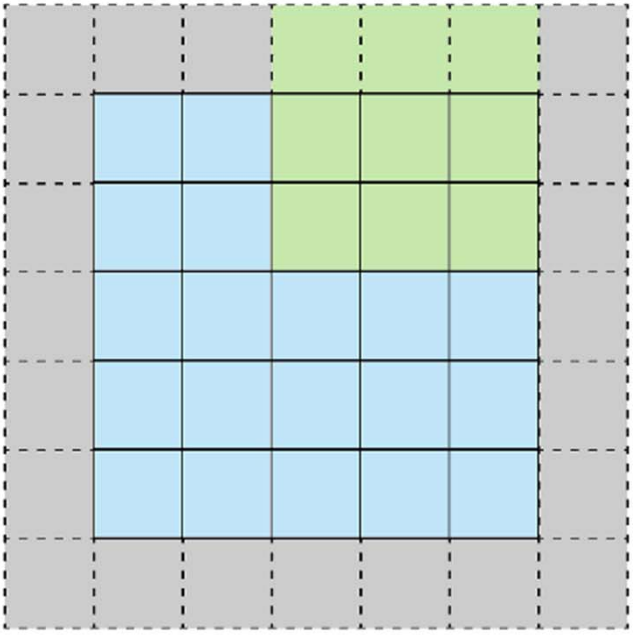
## PADDING

اگر بخواهیم نقشه‌ی ویژگی خروجی دارای ابعاد فضایی یکسان با نقشه‌ی ویژگی ورودی باشد، می‌توانیم از **پدگذاری** استفاده کنیم.

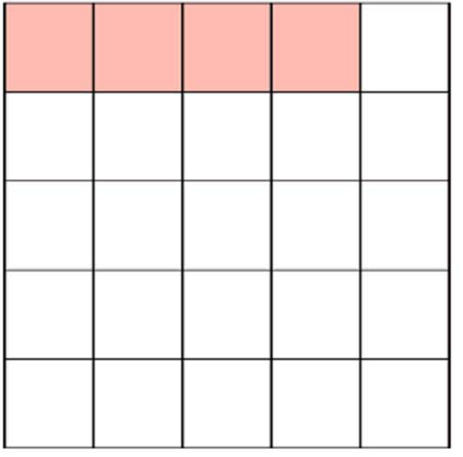
| **پدگذاری** *Padding* | افزودن تعدادی سطر/ستون به هر یک از کناره‌های نقشه‌ی ویژگی به‌طوری‌که جای‌دهی پنجره‌های کانوولوشن بر روی هر خانه‌ی ورودی ممکن شود. |
|---|---|




برای یک پنجره‌ی ۳×۳، یک سطر به بالا و پایین و یک ستون به چپ و راست اضافه می‌کنیم.
برای یک پنجره‌ی ۵×۵، دو سطر به بالا و پایین و دو ستون به چپ و راست اضافه می‌کنیم.



Stride 1 with Padding



Feature Map

https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

# عمل کانوولوشن

## پدگذاری

### PADDING

در لایه‌های Conv2D در Keras پدگذاری از طریق آرگومان padding پیکربندی می‌شود.

| padding = “same” | padding = “valid” |
| --- | --- |
| پدگذاری به‌گونه‌ای که<br>عرض و ارتفاع خروجی مشابه<br>عرض و ارتفاع ورودی باشد. | بدون پدگذاری<br>(فقط استفاده از مکان‌های مجاز)<br><br>(مقدار پیش‌فرض) |

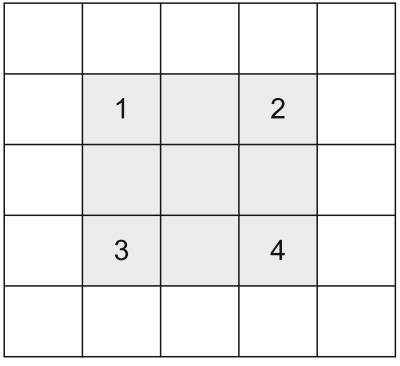
# عمل کانوولوشن

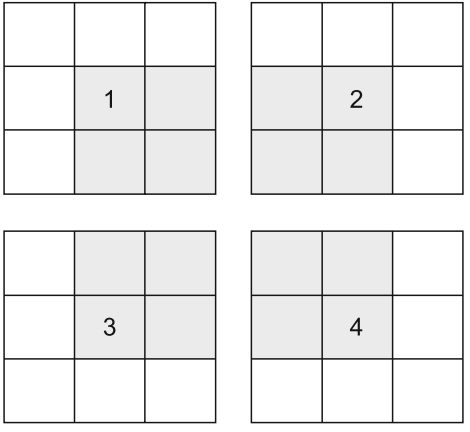
## اندازه قدم‌های کانوولوشن

## CONVOLUTION STRIDES

خانه‌های مرکزی پنجره‌های کانوولوشن می‌توانند مجاور نباشند
و اندازه‌ی قدم به‌جای 1 می‌تواند بزرگ‌تر باشد.

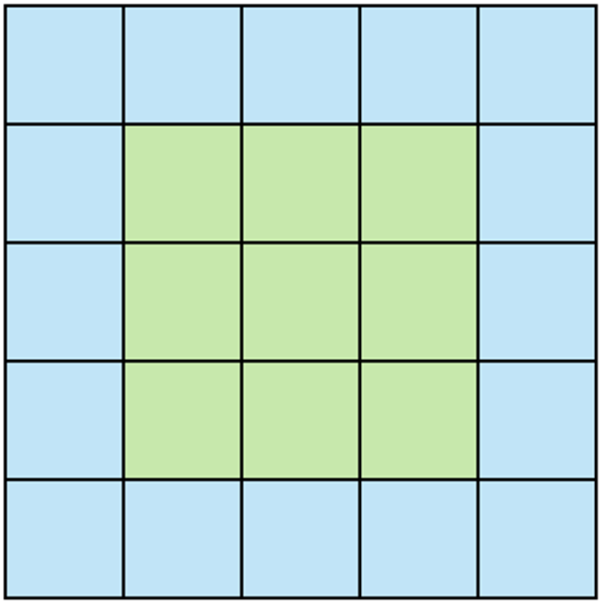
پچ‌های
کانوولوشنی
۳×۳
با اندازه قدم‌های
۲×۲



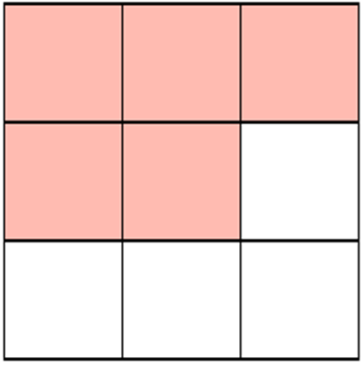




استفاده از اندازه قدم ۲ به معنی این است که عرض و ارتفاع نقشه‌ی ویژگی با فاکتور ۲ زیرنمونه‌برداری می‌شود.

در عمل به‌ندرت از کانوولوشن‌های strided استفاده می‌شود.
برای زیرنمونه‌برداری نقشه‌های ویژگی معمولاً از عملیات pooling استفاده می‌شود.

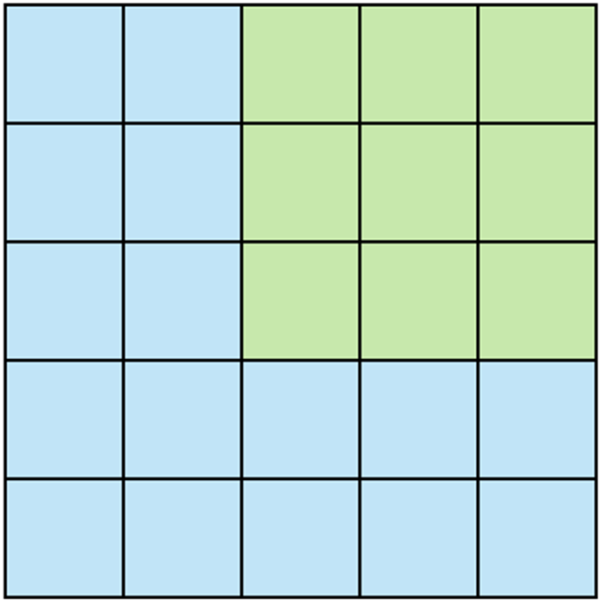


Stride 1

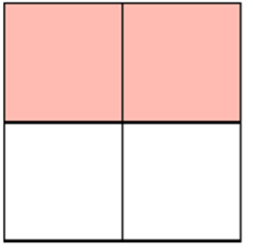


Feature Map

https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2



Stride 2



Feature Map

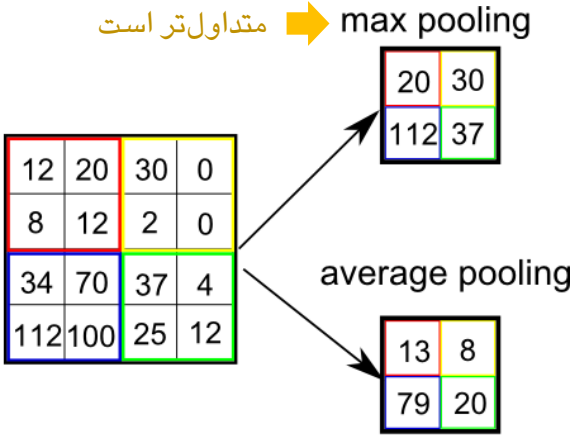
https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

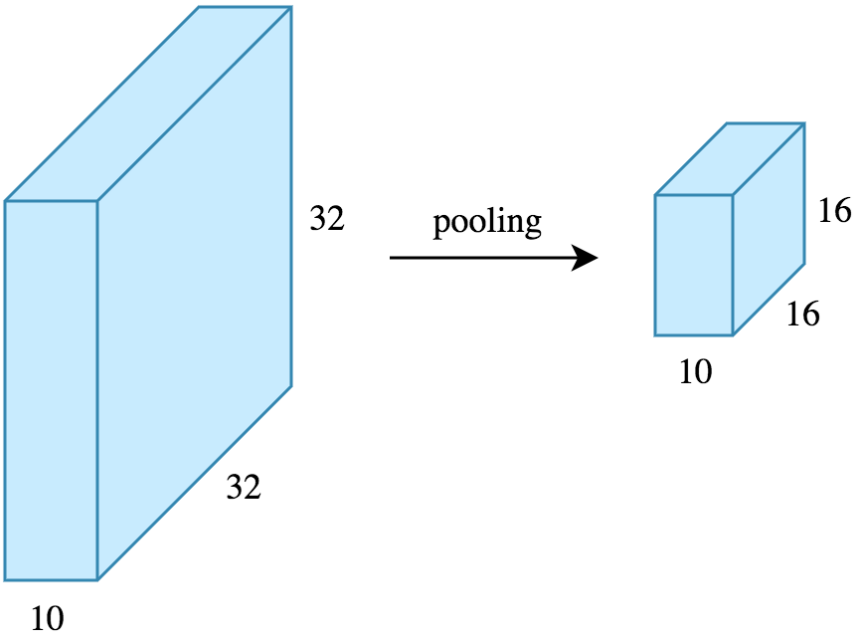
# عمل تلفیق

## POOLING OPERATION

عمل تلفیق از نظر مفهومی شبیه کانوولوشن است،
با این تفاوت که به‌جای یک تبدیل خطی یادگرفته شده بر روی پچ‌های محلی (هسته‌ی کانوولوشن)،
تبدیل از طریق یک عملیات تانسوری ثابت (مثل ماکزیمم یا میانگین یا ...) انجام می‌شود.

همچنین تلفیق معمولاً با پنجره‌های ۲×۲ با اندازه قدم‌های ۲ انجام می‌شود
⇐ موجب زیرنمونه‌برداری نقشه‌ی ویژگی با فاکتور ۲ می‌شود.






https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

# عمل تلفیق

مثالی از یک شبکه‌ی کانوولوشنال بدون استفاده از تلفیق

## POOLING OPERATION

اگر از عمل pooling در مدل استفاده نکنیم:

```
model_no_max_pool = models.Sequential()
model_no_max_pool.add(layers.Conv2D(32, (3, 3), activation='relu',
                                    input_shape=(28, 28, 1)))
model_no_max_pool.add(layers.Conv2D(64, (3, 3), activation='relu'))
model_no_max_pool.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

```
>>> model_no_max_pool.summary()

Layer (type)          Output Shape          Param #
=================================================================
conv2d_4 (Conv2D)    (None, 26, 26, 32)     320
_________________________________________________________________
conv2d_5 (Conv2D)    (None, 24, 24, 64)     18496
_________________________________________________________________
conv2d_6 (Conv2D)    (None, 22, 22, 64)     36928
=================================================================
Total params: 55,744
Trainable params: 55,744
Non-trainable params: 0
```

# عمل تلفیق

## POOLING OPERATION

**اگر از تلفیق استفاده نکنیم**

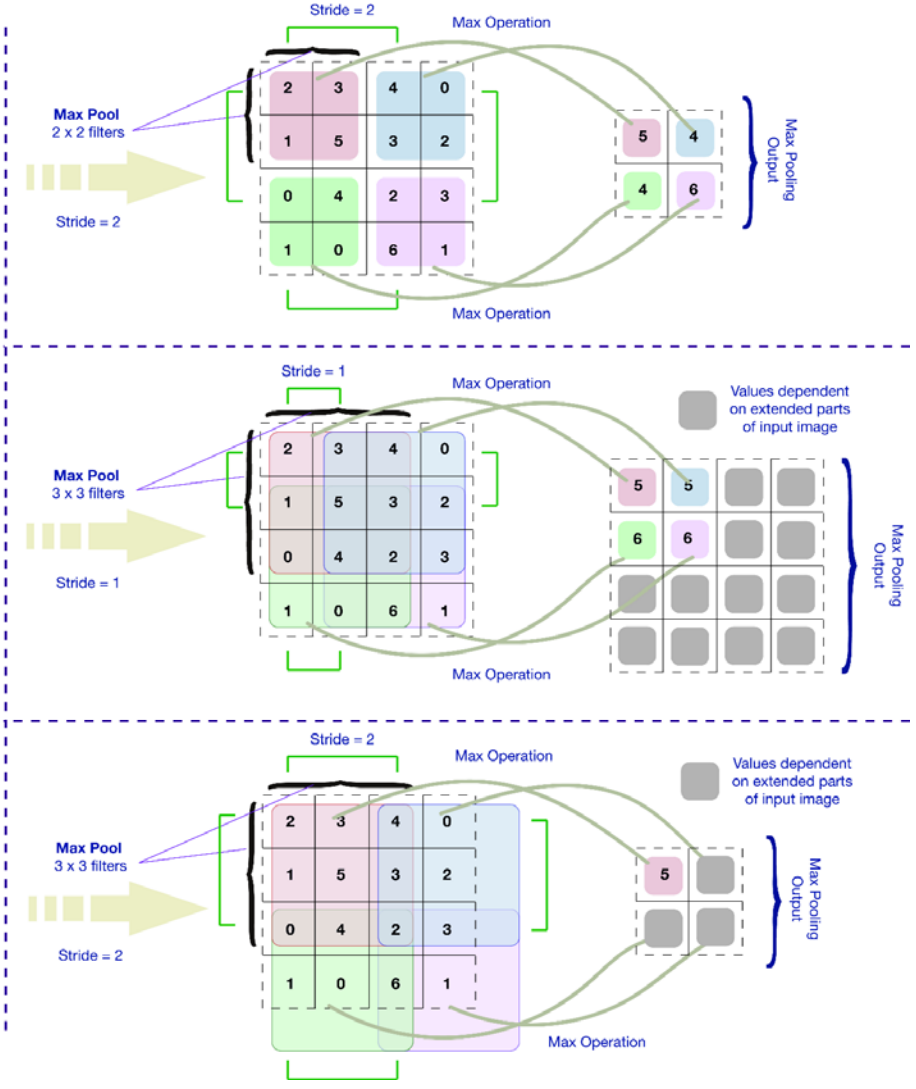
- **فرآیند به‌سمت یادگیری سلسله‌مراتب فضایی ویژگی‌ها هدایت نمی‌شود.**
  ⇐ پنجره‌های لایه‌های بعدی، حاوی همان اطلاعات پنجره‌های لایه‌های قبل خواهند بود.
- **نقشه‌ی ویژگی نهایی تعداد پارامترهای بسیار زیادی برای هر نمونه خواهد داشت.**
  ⇐ مشکل بیش‌برازش

هدف از زیرنمونه‌برداری (downsampling)، کاهش تعداد ضرایب نقشه‌ی ویژگی برای پردازش است، و همچنین، القای سلسله‌مراتب‌های فیلتر فضایی با ایجاد لایه‌های کانوولوشن متوالی.

عملکرد max-pooling از average-pooling در عمل بهتر است:
ویژگی‌ها متمایل هستند حضور فضایی برخی الگوها یا مفاهیم را بر روی کاشی‌های مختلف نقشه‌ی ویژگی کدگذاری کنند، و نگاه کردن به حضور ماکزیمم ویژگی‌های مختلف نسبت به حضور میانگین آنها حاوی اطلاعات بیشتری است.

An example Image Portion
for Max Pooling
Numbers represent
the pixel values

| | | | |
|---|---|---|---|
| 2 | 3 | 4 | 0 |
| 1 | 5 | 3 | 2 |
| 0 | 4 | 2 | 3 |
| 1 | 0 | 6 | 1 |



یادگیری عمیق برای بینایی کامپیوتری

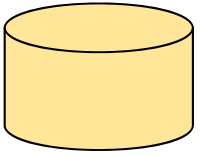
# ۲

# آموزش یک convnet از صفر بر روی یک مجموعه‌داده کوچک

# دورنما

استفاده از شبکه‌های عصبی کانوولوشنال برای مسئله‌ی طبقه‌بندی تصاویر

## مسئله‌ی طبقه‌بندی تصویر سگ/گربه



مجموعه‌داده حاوی ۴۰۰۰ تصویر (۲۰۰۰ گربه، ۲۰۰۰ سگ)

| مجموعه‌ی آموزشی *Training Set* | مجموعه‌ی اعتبارسنجی *Validation Set* | مجموعه‌ی آزمایشی *Test Set* |
|---|---|---|
| ۲۰۰۰ تصویر | ۱۰۰۰ تصویر | ۱۰۰۰ تصویر |

| | | |
|---|---|---|
| **آموزش مدل از صفر** (بدون رگولاریزاسیون) *Training a Model from Scratch* | دقت طبقه‌بندی accuracy = 71% | مشکل اصلی: **بیش‌برازش** |
| **استفاده از داده افزایی** *Data Augmentation* | دقت طبقه‌بندی accuracy = 82% | |
| **استخراج ویژگی با شبکه‌ی پیش‌آموزش دیده** *Feature Extraction with a Pretrained Network* | دقت طبقه‌بندی accuracy = 90 ~ 96% | |
| **تنظیم دقیق یک شبکه‌ی پیش‌آموزش دیده** *Fine-Tuning a Pretrained Network* | دقت طبقه‌بندی accuracy = 97% | |

# آموزش یک convnet از صفر بر روی یک مجموعه‌داده کوچک

TRAINING A CONVNET FROM SCRATCH ON A SMALL DATASET

**مجموعه داده‌ی کوچک** می‌تواند حاوی چند صد تا ده‌ها هزار نمونه باشد.

یادگیری عمیق، معمولاً تنها زمانی کار می‌کند که مقدار زیادی داده وجود داشته باشد.
چون یافتن ویژگی‌های مفید در داده‌های آموزشی بدون مهندسی ویژگی دستی زمانی محقق می‌شود که
حجم زیادی نمونه‌ی آموزشی وجود داشته باشد.
این موضوع صحیح است به‌خصوص زمانی که نمونه‌های ورودی دارای ابعاد بسیار بالا باشند (مانند تصویر).

تعداد زیاد نمونه‌ها **نسبی** است
و وابسته به اندازه به اندازه و عمق شبکه‌ای است که می‌خواهیم آن را آموزش بدهیم.

از آنجا که شبکه‌های عصبی کانوولوشنال
ویژگی‌های **محلی تغییرناپذیر تحت جابه‌جایی** را یاد می‌گیرند،
برای مسائل ادراکی بسیار کارآمد هستند.

آموزش یک شبکه‌ی عصبی کانوولوشنال از صفر بر روی یک مجموعه داده‌ی بسیار کوچک،
بدون نیاز به هرگونه مهندسی ویژگی سفارشی،
با وجود کم بودن نسبی داده‌ها، هنوز می‌تواند نتایج معقولی تولید کند.

# قابلیت تغییر منظور در یادگیری عمیق

REPURPOSABILILTY

مدل‌های یادگیری عمیق، به‌طور ذاتی بسیار **قابلیت تغییر منظور** دارند.

مثلاً می‌توان یک مدل طبقه‌بندی تصویر یا تبدیل گفتار به متن
که بر روی یک مجموعه‌داده‌ی بزرگ–مقیاس آموزش داده شده است را گرفت
و تنها با ایجاد کمی تغییر در آن،
از آن در مسئله‌ی بسیار متفاوتی استفاده‌ی مجدد (reuse) کرد.

# مسئله‌ی طبقه‌بندی تصویر سگ/گربه

## DOGS VS. CAT IMAGE CLASSIFICATION

# مسئله‌ی طبقه‌بندی تصویر سگ/گربه

## مجموعه‌داده

DOGS VS. CAT IMAGE CLASSIFICATION

نمونه‌هایی از مجموعه داده:
نمونه‌ها از نظر اندازه، ظاهر و ... ناهمگن هستند.

تصاویر با رزولوشن متوسط با فرمت JPEG هستند.

You can download the original dataset from www.kaggle.com/c/dogs-vs-cats/data

مجموعه‌داده حاوی ۲۵۰۰۰ تصویر (۱۲۵۰۰ گربه، ۱۲۵۰۰ سگ)

ایجاد یک مجموعه‌داده‌ی جدید از این مجموعه با سه زیرمجموعه:

| مجموعه‌ی آموزشی *Training Set* | مجموعه‌ی اعتبارسنجی *Validation Set* | مجموعه‌ی آزمایشی *Test Set* |
|---|---|---|
| ۲۰۰۰ تصویر (۱۰۰۰ تا از هر کلاس) | ۱۰۰۰ تصویر (۵۰۰ تا از هر کلاس) | ۱۰۰۰ تصویر (۵۰۰ تا از هر کلاس) |

# مسئله‌ی طبقه‌بندی تصویر سگ/گربه

ایجاد کپی از تصاویر برای پوشه‌های آموزش، اعتبارسنجی و آزمایش (۱ از ۴)

## Copying images to training, validation, and test directories

```
import os, shutil


original_dataset_dir = '/Users/fchollet/Downloads/kaggle_original_data'


base_dir = '/Users/fchollet/Downloads/cats_and_dogs_small'
os.mkdir(base_dir)


train_dir = os.path.join(base_dir, 'train')
os.mkdir(train_dir)
validation_dir = os.path.join(base_dir, 'validation')
os.mkdir(validation_dir)
test_dir = os.path.join(base_dir, 'test')
```

**Path to the directory where the original dataset was uncompressed**

**Directory where you'll store your smaller dataset**

**Directories for the training, validation, and test splits**

# مسئله‌ی طبقه‌بندی تصویر سگ/گربه

## ایجاد کپی از تصاویر برای پوشه‌های آموزش، اعتبارسنجی و آزمایش (۲ از ۴)

```
train_cats_dir = os.path.join(train_dir, 'cats')
os.mkdir(train_cats_dir)
```
Directory with training cat pictures

```
train_dogs_dir = os.path.join(train_dir, 'dogs')
os.mkdir(train_dogs_dir)
```
Directory with training dog pictures

```
validation_cats_dir = os.path.join(validation_dir, 'cats')
os.mkdir(validation_cats_dir)
```
Directory with validation cat pics

```
validation_dogs_dir = os.path.join(validation_dir, 'dogs')
os.mkdir(validation_dogs_dir)
```
Directory with validation dog pics

```
test_cats_dir = os.path.join(test_dir, 'cats')
os.mkdir(test_cats_dir)
```
Directory with test cat pictures

```
test_dogs_dir = os.path.join(test_dir, 'dogs')
os.mkdir(test_dogs_dir)
```
Directory with test dog pictures

# مسئله‌ی طبقه‌بندی تصویر سگ/گربه

## ایجاد کپی از تصاویر برای پوشه‌های آموزش، اعتبارسنجی و آزمایش (۳ از ۴)

```
fnames = ['cat.{}.jpg'.format(i) for i in range(1000)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(train_cats_dir, fname)
    shutil.copyfile(src, dst)
```

**Copies the first 1,000 cat images to train_cats_dir**

```
fnames = ['cat.{}.jpg'.format(i) for i in range(1000, 1500)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(validation_cats_dir, fname)
    shutil.copyfile(src, dst)
```

**Copies the next 500 cat images to validation_cats_dir**

```
fnames = ['cat.{}.jpg'.format(i) for i in range(1500, 2000)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(test_cats_dir, fname)
    shutil.copyfile(src, dst)
```

**Copies the next 500 cat images to test_cats_dir**

# مسئله‌ی طبقه‌بندی تصویر سگ/گربه

## ایجاد کپی از تصاویر برای پوشه‌های آموزش، اعتبارسنجی و آزمایش (۴ از ۴)

```
fnames = ['dog.{}.jpg'.format(i) for i in range(1000)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(train_dogs_dir, fname)
    shutil.copyfile(src, dst)
```

**Copies the first 1,000 dog images to train_dogs_dir**

```
fnames = ['dog.{}.jpg'.format(i) for i in range(1000, 1500)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(validation_dogs_dir, fname)
    shutil.copyfile(src, dst)
```

**Copies the next 500 dog images to validation_dogs_dir**

```
fnames = ['dog.{}.jpg'.format(i) for i in range(1500, 2000)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(test_dogs_dir, fname)
    shutil.copyfile(src, dst)
```

**Copies the next 500 dog images to test_dogs_dir**

## مسئله‌ی طبقه‌بندی تصویر سگ/گربه

بررسی تعداد تصاویر هر کلاس در هر یک از مجموعه‌های آموزش، اعتبارسنجی و آزمایش

```
>>> print('total training cat images:', len(os.listdir(train_cats_dir)))
total training cat images: 1000
>>> print('total training dog images:', len(os.listdir(train_dogs_dir)))
total training dog images: 1000
>>> print('total validation cat images:', len(os.listdir(validation_cats_dir)))
total validation cat images: 500
>>> print('total validation dog images:', len(os.listdir(validation_dogs_dir)))
total validation dog images: 500
>>> print('total test cat images:', len(os.listdir(test_cats_dir)))
total test cat images: 500
>>> print('total test dog images:', len(os.listdir(test_dogs_dir)))
total test dog images: 500
```

چون تعداد نمونه‌ها از هر کلاس مساوی است،
یک مسئله‌ی **طبقه‌بندی دودویی متوازن** (balanced binary classification) داریم
و معیار موفقیت مناسب، دقت (accuracy) است.

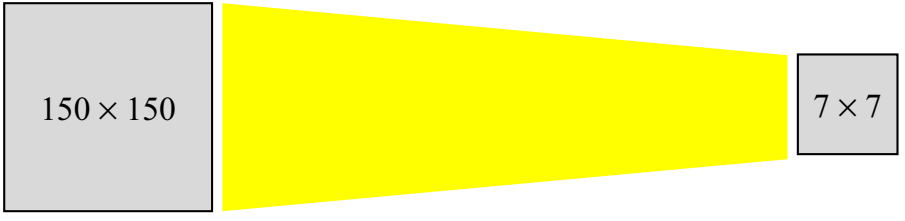
# مسئله‌ی طبقه‌بندی تصویر سگ/گربه

## ساخت شبکه

## BUILDING THE NETWORK

یک convnet را به‌صورت پشته‌ای لایه‌های متناوب Conv2D (با تابع فعالیت relu) و MaxPooling2D می‌سازیم.

چون با تصاویر بزرگ‌تری نسبت به MNIST سروکار داریم و مسئله پیچیده‌تر است،
باید شبکه‌ی بزرگ‌تری ایجاد کنیم: تعداد مراحل conv + pool بیشتر؛
این کار موجب (۱) افزایش ظرفیت شبکه
و (۲) کاهش کافی اندازه‌ی نقشه‌های ویژگی تا رسیدن به لایه‌ی تخت‌سازی (Flatten) می‌شود.




چون با یک مسئله‌ی طبقه‌بندی دودویی سروکار داریم،
در نهایت یک لایه‌ی Dense با یک واحد با تابع فعالیت sigmoid خواهیم داشت.

# شبکه‌های عصبی کانوولوشنال

الگوی افزایش تدریجی عمق نقشه‌های ویژگی همزمان با کاهش اندازه‌ی آنها

در اکثر شبکه‌های عصبی کانوولوشنال
عمق نقشه‌های ویژگی در شبکه به‌تدریج افزایش می‌یابد،
در حالی‌که اندازه‌ی نقشه‌های ویژگی به‌تدریج کاهش می‌یابد.

# مسئله‌ی طبقه‌بندی تصویر سگ/گربه

ساخت شبکه

## Instantiating a small convnet for dogs vs. cats classification

```
from keras import layers
from keras import models

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu’,
                                    input_shape=(150, 150, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

چون با یک مسئله‌ی طبقه‌بندی دودویی سروکار داریم،
در نهایت یک لایه‌ی Dense با یک واحد با تابع فعالیت sigmoid خواهیم داشت.
خروجی احتمال تعلق آنچه شبکه می‌بیند به یکی از کلاس‌ها را کدگذاری می‌کند.

# مسئله‌ی طبقه‌بندی تصویر سگ/گربه

## مشاهده‌ی تغییر ابعاد نقشه‌های ویژگی طی لایه‌های متوالی

```
>>> model.summary()
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 148, 148, 32)      896
_________________________________________________________________
maxpooling2d_1 (MaxPooling2D)(None, 74, 74, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 72, 72, 64)        18496
_________________________________________________________________
maxpooling2d_2 (MaxPooling2D)(None, 36, 36, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 34, 34, 128)       73856
_________________________________________________________________
maxpooling2d_3 (MaxPooling2D)(None, 17, 17, 128)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 15, 15, 128)       147584
_________________________________________________________________
maxpooling2d_4 (MaxPooling2D)(None, 7, 7, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 6272)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               3211776
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 513
=================================================================
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0
```

# مسئله‌ی طبقه‌بندی تصویر سگ/گربه

پیکربندی مدل برای آموزش

## Configuring the model for training

```
from keras import optimizers

model.compile(loss='binary_crossentropy’,
              optimizer=optimizers.RMSprop(lr=1e-4),
              metrics=['acc'])
```

چون از تابع فعالیت sigmoid در لایه‌ی آخر استفاده کرده‌ایم،
از تابع اتلاف binary cross entropy استفاده می‌کنیم.

# مسئله‌ی طبقه‌بندی تصویر سگ/گربه

## پیش‌پردازش داده‌ها

## DATA PREPROCESSING

داده‌ها باید در قالب تانسورهای ممیزشناور پیش‌پردازش شده‌ی مناسب قرار بگیرند.

مراحل کار:

1) Read the picture files.
2) Decode the JPEG content to RGB grids of pixels.
3) Convert these into floating-point tensors.
4) Rescale the pixel values (between 0 and 255) to the [0, 1] interval (as you know, neural networks prefer to deal with small input values).

در Keras ماژولی برای پردازش تصویر وجود دارد که کارهای فوق را انجام می‌دهد:
keras.preprocessing.image

# مسئله‌ی طبقه‌بندی تصویر سگ/گربه

پیش‌پردازش داده‌ها: استفاده از کتابخانه‌ی موجود در کراس برای خواندن تصاویر

## Using `ImageDataGenerator` to read images from directories

```
from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        train_dir,              # Target directory
        target_size=(150, 150) # Resizes all images to 150 × 150
        batch_size=20,
        class_mode='binary')    # Because you use binary_crossentropy loss,
                                # you need binary labels

validation_generator = test_datagen.flow_from_directory(
        validation_dir,
        target_size=(150, 150),
        batch_size=20,
        class_mode='binary')
```

**Rescales all images by 1/255**

## Understanding Python generators

A *Python generator* is an object that acts as an iterator: it's an object you can use with the `for ... in` operator. Generators are built using the `yield` operator.

Here is an example of a generator that yields integers:

```
def generator():
    i = 0
    while True:
        i += 1
        yield i

for item in generator():
    print(item)
    if item > 4:
        break
```

It prints this:

```
1
2
3
4
5
```

# مسئله‌ی طبقه‌بندی تصویر سگ/گربه

## پیش‌پردازش داده‌ها: عملکرد مولد تصاویر

عملکرد هر یک از مولدها:

ایجاد دسته‌هایی حاوی ۲۰ نمونه از تصاویر RGB با ابعاد ۱۵۰×۱۵۰ (شکل: (20,150,150,3))
و برچسب‌های دودویی (شکل: (20,))

```
>>> for data_batch, labels_batch in train_generator:
>>> print('data batch shape:', data_batch.shape)
>>> print('labels batch shape:', labels_batch.shape)
>>> break
data batch shape: (20, 150, 150, 3)
labels batch shape: (20,)
```

چون مولد دسته‌ها (batches) را به‌صورت نامتناهی تولید می‌کند،
به‌صورت بی‌پایان روی پوشه‌ی مقصد، حلقه می‌زند.
برای همین باید حلقه‌ی تکرار را در نقطه‌ای شکست (break).

# مسئله‌ی طبقه‌بندی تصویر سگ/گربه

## برازش مدل با استفاده از یک مولد دسته‌ای

حال نوبت برازش مدل با استفاده از داده‌های تولید شده توسط مدل است.
برای این کار از متد `fit_generator` استفاده می‌کنیم (معادل `fit` برای مولدهای داده).
آرگومان اول آن، یک مولد Python است (که یک دسته از ورودی‌ها و تارگت‌ها را به‌صورت نامحدود تولید می‌کند).
آرگومان دوم آن، تعداد نمونه‌هایی که باید از مولد استخراج شود را مشخص می‌کند.

In this case, batches are 20 samples, so it will take 100 batches until you see your target of 2,000 samples.

### Fitting the model using a batch generator

```
history = model.fit_generator(
      train_generator,
      steps_per_epoch=100,
      epochs=30,
      validation_data=validation_generator,
      validation_steps=50)
```

### Saving the model

```
model.save('cats_and_dogs_small_1.h5')
```

# مسئله‌ی طبقه‌بندی تصویر سگ/گربه

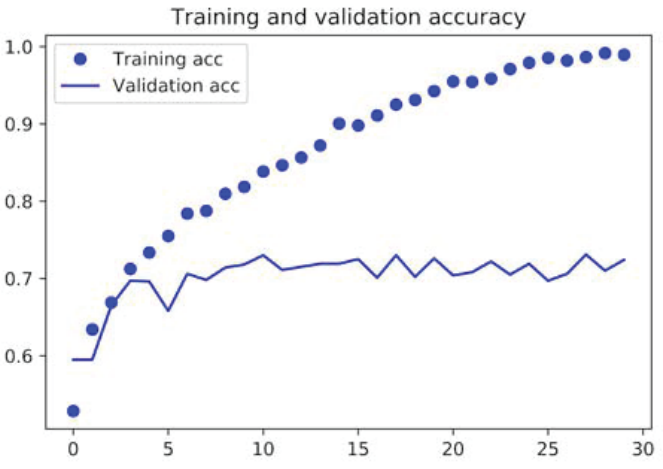
نمایش منحنی‌های اتلاف و دقت در حین آموزش

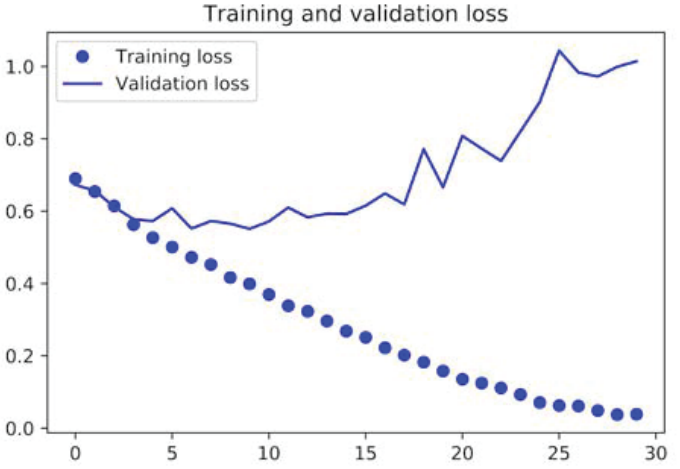
## Displaying curves of loss and accuracy during training

```
import matplotlib.pyplot as plt

acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss’]

epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```

# مسئله‌ی طبقه‌بندی تصویر سگ/گربه

## نمودارهای اتلاف و دقت روی داده‌های آموزشی و اعتبارسنجی




هر دو نمودار، مشخصه‌ی بیش‌برازش هستند.

دقت روی داده‌های آموزشی به‌صورت خطی در طول زمان افزایش می‌یابد تا به نزدیک ۱۰۰% می‌رسد،
اما دقت روی داده‌های اعتبارسنجی در حد ۷۰ تا ۷۲% می‌ماند.

اتلاف روی داده‌های اعتبارسنجی پس از ۵ اپک به می‌نیمم خود می‌رسد و ثابت می‌ماند،
اما اتلاف روی داده‌های آموزشی به‌صورت خطی کاهش می‌یابد تا به نزدیک صفر می‌رسد.

با توجه به کم بودن نسبی نمونه‌های آموزشی، **بیش‌برازش** مهم‌ترین نگرانی ماست.

# استفاده از داده‌افزایی

## USING DATA AUGMENTATION

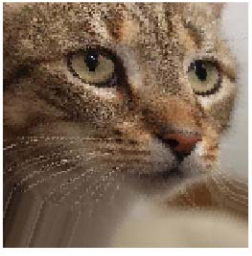
بیش‌برازش، در اثر بسیار کم بودن نمونه‌های یادگیری اتفاق می‌افتد؛
و باعث می‌شود نتوانیم مدلی را آموزش بدهیم که بتواند به داده‌های جدید تعمیم پیدا کند.

اگر بی‌نهایت داده داشته باشیم،
مدل ما در معرض همه‌ی جنبه‌های توزیع داده‌های موجود قرار می‌گیرد و هرگز بیش‌برازش نمی‌کند.

| | |
|---|---|
| **داده‌افزایی**<br>*Data Augmentation* | تولید داده‌های بیشتر از روی داده‌های آموزشی موجود،<br>با افزودن نمونه‌ها از طریق چند تبدیل تصادفی که به نمونه‌های باورپذیر منجر شود. |

هدف این است که:
در زمان آموزش، مدل ما هرگز یک تصویر (دقیقاً ثابت) را دو مرتبه مشاهده نکند.
این کمک می‌کند که مدل ما در معرض جنبه‌های بیشتری از داده‌ها قرار بگیرد و بهتر تعمیم بدهد.

داده‌افزایی، یک روش خاص بینایی کامپیوتری برای مقابله با بیش‌برازش
در هنگام پردازش تصاویر با مدل‌های یادگیری عمیق است که تقریباً همیشه استفاده می‌شود.



https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

# استفاده از داده‌افزایی

در پردازش تصویر

## Setting up a data augmentation configuration via `ImageDataGenerator`

```
datagen = ImageDataGenerator(
      rotation_range=40,
      width_shift_range=0.2,
      height_shift_range=0.2,
      shear_range=0.2,
      zoom_range=0.2,
      horizontal_flip=True,
      fill_mode='nearest')
```

در Keras می‌توان تعدادی تبدیل تصادفی را برای اعمال روی تصاویر پیکربندی کرد:
با استفاده از instance `ImageDataGenerator`

- `rotation_range` is a value in degrees (0–180), a range within which to randomly rotate pictures.
- `width_shift` and `height_shift` are ranges (as a fraction of total width or height) within which to randomly translate pictures vertically or horizontally.
- `shear_range` is for randomly applying shearing transformations.
- `zoom_range` is for randomly zooming inside pictures.
- `horizontal_flip` is for randomly flipping half the images horizontally—relevant when there are no assumptions of horizontal asymmetry (for example, real-world pictures).
- `fill_mode` is the strategy used for filling in newly created pixels, which can appear after a rotation or a width/height shift.

# مسئله‌ی طبقه‌بندی تصویر سگ/گربه

استفاده از داده‌افزایی: نمایش چند تصویر آموزشی افزوده شده تصادفی

## Displaying some randomly augmented training images

```
from keras.preprocessing import image

fnames = [os.path.join(train_cats_dir, fname) for
     fname in os.listdir(train_cats_dir)]

img_path = fnames[3]

img = image.load_img(img_path, target_size=(150, 150))

x = image.img_to_array(img)

x = x.reshape((1,) + x.shape)

i = 0
for batch in datagen.flow(x, batch_size=1):
    plt.figure(i)
    imgplot = plt.imshow(image.array_to_img(batch[0]))
    i += 1
    if i % 4 == 0:
      break
plt.show()
```

**Module with image preprocessing utilities**

**Chooses one image to augment**

**Reads the image and resizes it**

**Generates batches of randomly transformed images. Loops indefinitely, so you need to break the loop at some point!**

# مسئله‌ی طبقه‌بندی تصویر سگ/گربه

استفاده از داده‌افزایی: نمایش چند تصویر آموزشی افزوده شده تصادفی

# استفاده از دیگر تکنیک‌های رگولاریزاسیون در کنار داده‌افزایی

اگر یک شبکه‌ی جدید را با استفاده از داده‌افزایی آموزش بدهیم،
شبکه هرگز دو ورودی یکسان را نخواهد دید.
اما
ورودی‌هایی که شبکه می‌بیند، دارای همبستگی شدیدی میان یکدیگر هستند.
زیرا داده‌ها از تعداد اندکی از داده‌های اصلی آمده‌اند
(اطلاعات جدیدی تولید نشده است، بلکه تنها اطلاعات موجود مجدداً مخلوط شده‌اند)

⇓

برای رهایی از بیش‌برازش، ممکن است داده‌افزایی کافی نباشد.

پس برای مقابله‌ی بیشتر با بیش‌برازش، باید از دیگر روش‌های رگولاریزاسیون نیز استفاده کنیم.
(مثلاً یک لایه‌ی Dropout دقیقاً قبل از طبقه‌بندی‌کننده‌ی متصل متراکم قرار می‌دهیم.)

# مسئله‌ی طبقه‌بندی تصویر سگ/گربه

تعریف یک شبکه‌ی عصبی کانوولوشنال جدید که شامل dropout است

## Defining a new convnet that includes dropout

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                                    input_shape=(150, 150, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dropout(0.5))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid’))

model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-4),
              metrics=['acc'])
```

# مسئله‌ی طبقه‌بندی تصویر سگ/گربه

## آموزش یک شبکه‌ی عصبی کانوولوشنال با استفاده از مولدهای داده‌افزایی (۱ از ۲)

شبکه‌ای را آموزش می‌دهیم که از داده‌افزایی و dropout استفاده می‌کند:

### Training the convnet using data-augmentation generators

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,)

test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        train_dir,   # Target directory
        target_size=(150, 150),  # Resizes all images to 150 × 150
        batch_size=32,
        class_mode='binary')
```

**Note that the validation data shouldn't be augmented!**

**Because you use binary_crossentropy loss, you need binary labels.**

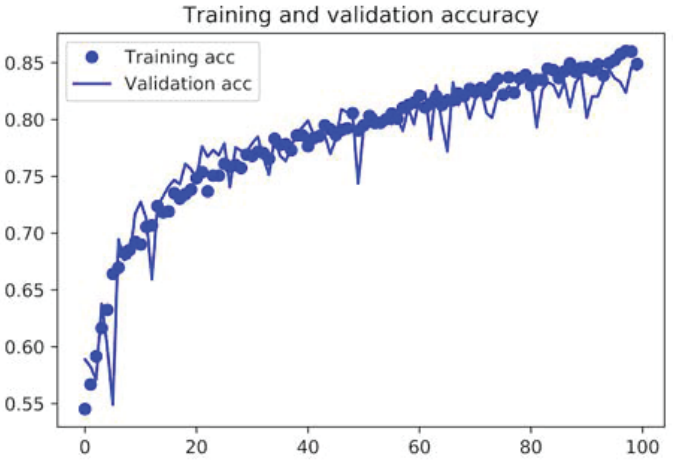
# مسئله‌ی طبقه‌بندی تصویر سگ/گربه

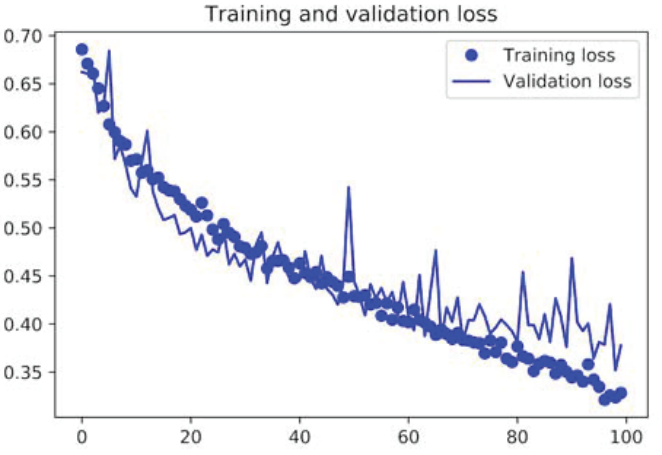
## آموزش یک شبکه‌ی عصبی کانوولوشنال با استفاده از مولدهای داده‌افزایی (۲ از ۲)

```
validation_generator = test_datagen.flow_from_directory(
        validation_dir,
        target_size=(150, 150),
        batch_size=32,
        class_mode='binary’)

history = model.fit_generator(
      train_generator,
      steps_per_epoch=100,
      epochs=100,
      validation_data=validation_generator,
      validation_steps=50)

model.save('cats_and_dogs_small_2.h5’)   # Saving the model
```

# مسئله‌ی طبقه‌بندی تصویر سگ/گربه

## نمودارهای اتلاف و دقت روی داده‌های آموزشی و اعتبارسنجی (با داده‌افزایی)




در اثر داده‌افزایی و dropout دیگر بیش‌برازش وجود ندارد.
منحنی‌های آموزش به‌طور نزدیکی منحنی‌های اعتبارسنجی را دنبال می‌کنند.

در اینجا به دقت ۸۲% رسیده‌ایم.
(حدود ۱۵% بهبود نسبت به مدل رگولاریزه نشده)

## مسئله‌ی طبقه‌بندی تصویر سگ/گربه

بهبود نتایج با استفاده از سایر تکنیک‌های رگولاریزاسیون

با استفاده‌ی بیشتر از تکنیک‌های رگولاریزاسیون،
و با تنظیم پارامترهای شبکه
(مانند تعداد فیلترها در هر لایه‌ی کانوولوشن، تعداد لایه‌ها در شبکه و ...)
ممکن است بتوانیم به دقت‌های بهتر در حد ۸۶% یا ۸۷% برسیم.

اما
از آنجا که داده‌ها بسیار کم هستند، با آموزش شبکه از صفر، دقت بالاتری حاصل نمی‌شود.

یادگیری عمیق

یادگیری عمیق برای بینایی کامپیوتری

# ۳

# استفاده از یک convnet پیش‌آموزش دیده

# استفاده از یک convnet پیش‌آموزش دیده

یادگیری انتقالی با CNN

USING A PRETRAINED CONVNET (TRANSFER LEARNING WITH CNN)

یک روی‌کرد معمول و بسیار مؤثر در یادگیری عمیق بر روی مجموعه‌داده‌های کوچک از تصاویر،
استفاده از یک شبکه‌ی پیش‌آموزش دیده است.

یک شبکه‌ی پیش‌آموزش دیده،
یک شبکه‌ی ذخیره شده است که قبلاً بر روی یک مجموعه‌داده‌ی بزرگ آموزش دیده است.
(معمولاً بر روی وظیفه‌ی طبقه‌بندی تصاویر بزرگ–مقیاس)

اگر مجموعه‌داده‌ی اصلی به‌اندازه‌ی کافی بزرگ (large) و به‌اندازه‌ی کافی عمومی (general) باشد،
آن‌گاه سلسله‌مراتب فضایی ویژگی‌های یادگرفته شده توسط آن شبکه‌ی پیش‌آموزش دیده،
می‌تواند به‌طور مؤثری به‌عنوان یک مدل عام (generic) از دنیای بصری عمل کند.
⇐ ویژگی‌های آن می‌تواند برای مسائل متعدد و متفاوت بینایی کامپیوتری مفید باشد،
حتی اگر کلاس‌های مسئله‌ی جدید کاملاً متفاوت با کلاس‌های وظیفه‌ی اصلی شبکه باشد.

برای مثال،
ممکن است یک شبکه را بر روی ImageNet آموزش دهیم (کلاس‌ها: عمدتاً حیوانات و اشیای روزمره)،
و سپس از این شبکه برای منظور متفاوتی (مانند شناسایی مبلمان در تصاویر) استفاده کنیم.

این ویژگی «قابلیت حمل ویژگی‌های یادگیری شده بین مسائل مختلف» یکی از مزایای کلیدی یادگیری عمیق،
در مقایسه با روی‌کردهای قدیمی‌تر کم‌عمق یادگیری است؛
و یادگیری عمیق را برای مسائل با داده‌های کوچک بسیار مؤثر می‌کند.

# مسئله‌ی طبقه‌بندی تصویر سگ/گربه

با استفاده از یک convnet پیش‌آموزش دیده

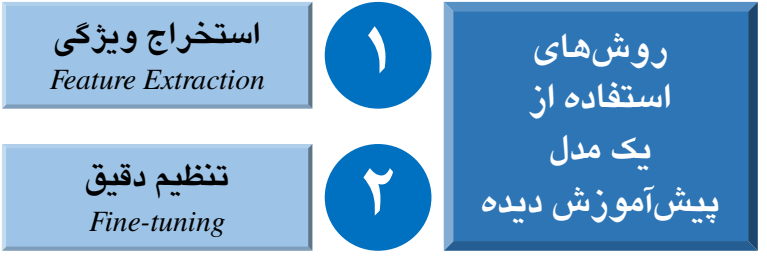
برای مسئله‌ی طبقه‌بندی تصویر سگ / گربه
یک convnet بزرگ را در نظر می‌گیریم که با مجموعه‌داده‌ی ImageNet آموزش یافته است:
(۱,۴ میلیون تصویر برچسب‌دار با ۱۰۰۰ کلاس مختلف)
حاوی کلاس‌های مختلف از جمله گونه‌های مختلف گربه‌ها و سگ‌ها ⇐ انتظار داریم در این مسئله هم به‌خوبی عمل کند.

معماری VGG16

Karen Simonyan and Andrew Zisserman,
"Very Deep Convolutional Networks for Large-Scale Image Recognition,"
arXiv (2014), https://arxiv.org/abs/1409.1556.

# شبکه‌های عصبی کانوولوشنال

## روش‌های استفاده از یک مدل پیش‌آموزش دیده



# شبکه‌های عصبی کانوولوشنال

## روش‌های استفاده از یک مدل پیش‌آموزش دیده: استخراج ویژگی

## FEATURE EXTRACTION

روش‌های استفاده از یک مدل پیش‌آموزش دیده

۱ **استخراج ویژگی** *Feature Extraction*

استفاده از بازنمایی‌های یادگرفته شده توسط یک شبکه‌ی قبلی، برای استخراج ویژگی‌ها از نمونه‌های جدید

۲ تنظیم دقیق *Fine-tuning*

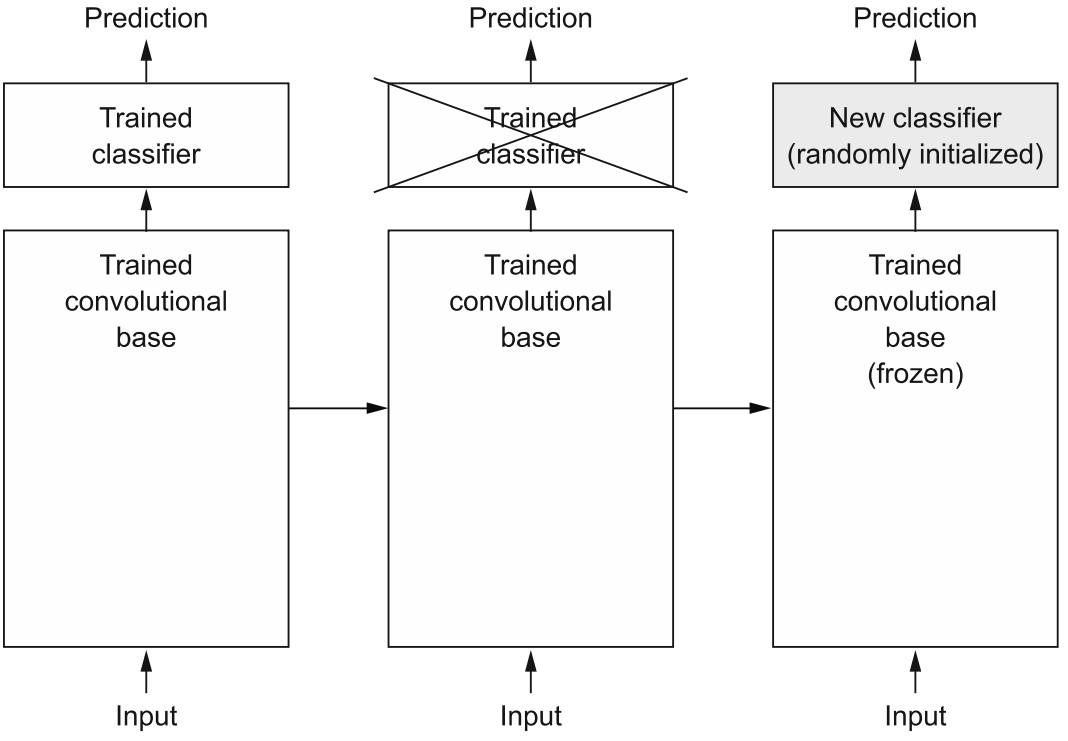
این ویژگی‌ها سپس به یک طبقه‌بندی‌کننده‌ی جدید که از صفر آموزش داده می‌شود، داده می‌شوند.

convnetهای مورد استفاده برای طبقه‌بندی تصاویر، از دو بخش تشکیل شده‌اند:
شروع با یک توالی از لایه‌های کانوولوشنال و پولینگ (**پایه‌ی کانوولوشنال** مدل: convolutional base)
و پایان با یک طبقه‌بندی‌کننده‌ی متصل متراکم

در استفاده از convnetها، استخراج ویژگی با گرفتن پایه‌ی کانوولوشنال از شبکه‌ی قبلاً آموزش دیده، وارد کردن داده‌های جدید به آن، و آموزش یک طبقه‌بندی‌کننده‌ی جدید بر بالای خروجی آن انجام می‌شود.

# شبکه‌های عصبی کانوولوشنال

روش‌های استفاده از یک مدل پیش‌آموزش دیده: استخراج ویژگی



# شبکه‌های عصبی کانوولوشنال

روش‌های استفاده از یک مدل پیش‌آموزش دیده: استخراج ویژگی: دلایل استفاده از پایه‌ی کانوولوشنال

چرا تنها پایه‌ی کانوولوشنال را مورد استفاده‌ی مجدد قرار می‌دهیم؟
آیا نمی‌توانیم طبقه‌بندی‌کننده‌ی متصل متراکم بالای آن را هم مورد استفاده‌ی مجدد قرار دهیم؟

در حالت کلی، از انجام چنین کاری اجتناب می‌کنیم.
زیرا:
بازنمایی‌های یادگیری شده توسط پایه‌ی کانوولوشنال احتمالاً **عام‌تر** هستند
و بنابراین بیشتر قابل استفاده‌ی مجدد هستند.
اما
بازنمایی‌های یادگیری شده توسط طبقه‌بندی‌کننده‌ی متصل متراکم لزوماً خاص مجموعه کلاس‌هایی هستند
که مدل بر روی آنها آموزش دیده است.
همچنین بازنمایی‌های یافت شده در لایه‌های متصل متراکم،
دیگر حاوی هیچ اطلاعاتی در مورد مکان قرارگیری اشیا در تصویر ورودی نیستند.

(لایه‌های تماماً متصل از مفهوم فضا رها می‌شوند، در حالی که مکان شیئ همچنان توسط نقشه‌های ویژگی کانوولوشنال توصیف می‌شوند.)

# شبکه‌های عصبی کانوولوشنال

روش‌های استفاده از یک مدل پیش‌آموزش دیده: استخراج ویژگی: عمومیت ویژگی‌ها در پایه‌ی کانوولوشنال

سطح عمومیت (و در نتیجه قابلیت استفاده‌ی مجدد) بازنمایی‌های استخراج شده
توسط لایه‌های خاص کانوولوشنال،
وابسته به عمق لایه در آن مدل است.

لایه‌های اولیه‌ی مدل: استخراج نقشه‌های ویژگی محلی و شدیداً عمومی (لبه‌های دیداری، رنگ‌ها، بافت‌ها و ...)
لایه‌های سطوح بالاتر: مفاهیم انتزاعی‌تر (چشم گربه، گوش سگ، ...)

⇓

اگر مجموعه‌داده‌ی جدید با مجموعه‌داده‌ی اصلی آموزش‌دیده تفاوت زیادی داشته باشد،
ممکن است بهتر باشد به‌جای استفاده از کل پایه‌ی کانوولوشنال،
تنها از چند لایه‌ی اول آن برای استخراج ویژگی استفاده کنیم.

# مسئله‌ی طبقه‌بندی تصویر سگ/گربه

## استفاده از ویژگی‌های استخراج‌شده روی ImageNet

از آنجایی که کلاس‌های موجود در ImageNet حاوی چند کلاس سگ و گربه است،
احتمالاً استفاده‌ی مجدد از اطلاعات موجود در لایه‌های متصل متراکم مدل اصلی سودمند خواهند بود،
اما ما از آن استفاده نمی‌کنیم
تا حالت کلی‌تری را پوشش دهیم که در آن کلاس‌های مسئله‌ی جدید با کلاس‌های مدل اصلی همپوشانی ندارد.

برای پیاده‌سازی عملی، از پایه‌ی کانوولوشنال شبکه‌ی VGG16 استفاده می‌کنیم
که روی ImageNet آموزش یافته است
تا ویژگی‌های جالب از تصاویر سگ و گربه را استخراج کند.
سپس یک طبقه‌بندی‌کننده‌ی Dogs vs. Cats را در بالای این ویژگی‌ها آموزش می‌دهیم.

نمونه مدل‌های پیش‌آموزش دیده روی ImageNet موجود در ماژول `keras.application`

| Xception | Inception V3 | ResNet50 | VGG16 | VGG19 | MobileNet |
|---|---|---|---|---|---|

# مسئله‌ی طبقه‌بندی تصویر سگ/گربه

## استفاده از ویژگی‌های استخراج‌شده روی ImageNet: ایجاد نمونه‌ای از پایه‌ی کانوولوشنال VGG16

### Instantiating the VGG16 convolutional base

```
from keras.applications import VGG16

conv_base = VGG16(weights='imagenet',
                  include_top=False,
                  input_shape=(150, 150, 3))
```

سه آرگومان این سازنده:

- `weights` specifies the weight checkpoint from which to initialize the model.
- `include_top` refers to including (/not) the densely connected classifier on top of the network.
  By default, this densely connected classifier corresponds to the 1,000 classes from ImageNet. Because you intend to use your own densely connected classifier (with only two classes: cat and dog), you don't need to include it.
- `input_shape` is the shape of the image tensors that you'll feed to the network.
  This argument is purely optional: if you don't pass it, the network will be able to process inputs of any size.

# مسئله‌ی طبقه‌بندی تصویر سگ/گربه

## استفاده از ویژگی‌های استخراج‌شده روی ImageNet: جزئیات معماری پایه‌ی کانوولوشنال VGG16

```
>>> conv_base.summary()
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 150, 150, 3)       0
_________________________________________________________________
block1_conv1 (Convolution2D) (None, 150, 150, 64)      1792
_________________________________________________________________
block1_conv2 (Convolution2D) (None, 150, 150, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 75, 75, 64)        0
_________________________________________________________________
block2_conv1 (Convolution2D) (None, 75, 75, 128)       73856
_________________________________________________________________
block2_conv2 (Convolution2D) (None, 75, 75, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 37, 37, 128)       0
_________________________________________________________________
block3_conv1 (Convolution2D) (None, 37, 37, 256)       295168
_________________________________________________________________
block3_conv2 (Convolution2D) (None, 37, 37, 256)       590080
_________________________________________________________________
block3_conv3 (Convolution2D) (None, 37, 37, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 18, 18, 256)       0
_________________________________________________________________
block4_conv1 (Convolution2D) (None, 18, 18, 512)       1180160
_________________________________________________________________
block4_conv2 (Convolution2D) (None, 18, 18, 512)       2359808
_________________________________________________________________
block4_conv3 (Convolution2D) (None, 18, 18, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 9, 9, 512)         0
_________________________________________________________________
block5_conv1 (Convolution2D) (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_conv2 (Convolution2D) (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_conv3 (Convolution2D) (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 4, 4, 512)         0
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
```

شکل نهایی نقشه‌ی ویژگی به‌صورت (4, 4, 512) است.
در بالای این ویژگی باید یک طبقه‌بندی‌کننده‌ی متصل متراکم قرار بگیرد.

# مسئله‌ی طبقه‌بندی تصویر سگ/گربه

استفاده از ویژگی‌های استخراج‌شده روی ImageNet: دو مسیر برای ادامه‌ی کار

**مسیر اول**

اجرای پایه‌ی کانوولوشنال روی مجموعه‌داده،
ضبط کردن خروجی آن به‌صورت یک آرایه‌ی NumPy بر روی دیسک، و
استفاده از این داده‌ها به‌عنوان ورودی به یک طبقه‌بندی‌کننده‌ی متصل متراکم

☺ اجرای این راه‌حل سریع و ارزان است،
زیرا تنها به یک بار اجرای پایه‌ی کانوولوشنال روی هر تصویر ورودی نیاز دارد
و پایه‌ی کانوولوشنال، پرهزینه‌ترین بخش این خط لوله است.
☹ این تکنیک امکان استفاده از داده‌افزایی را نمی‌دهد.

**مسیر دوم**

گسترش مدل موجود با اضافه کردن لایه‌های Dense به بالای آن، و
اجرای کل مدل به‌صورت انتها به انتها بر روی داده‌های ورودی

☺ این تکنیک امکان استفاده از داده‌افزایی را می‌دهد،
زیرا هر تصویر ورودی در هر زمان که توسط مدل دیده می‌شود، از پایه‌ی کانوولوشنال می‌گذرد.
☹ این تکنیک کندتر و پرهزینه‌تر است.

# مسئله‌ی طبقه‌بندی تصویر سگ/گربه

## روش سریع استخراج ویژگی بدون داده‌افزایی: استخراج ویژگی با استفاده از پایه‌ی کانوولوشنال پیش‌آموزش دیده (۱ از ۲)

FAST FEATURE EXTRACTION WITHOUT DATA AUGMENTATION

با اجرای نمونه‌ها از مولد `ImageDataGenerator` شروع می‌کنیم
تا تصاویر را به‌صورت آرایه‌های Numpy به‌همراه برچسب آنها استخراج کند.
سپس ویژگی‌ها را با فراخوانی متد `predict` از مدل `conv_base` از این تصاویر استخراج می‌کنیم.

### Extracting features using the pretrained convolutional base

```
import os
import numpy as np
from keras.preprocessing.image import ImageDataGenerator

base_dir = '/Users/fchollet/Downloads/cats_and_dogs_small'
train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')
test_dir = os.path.join(base_dir, 'test')

datagen = ImageDataGenerator(rescale=1./255)
batch_size = 20
```

# مسئله‌ی طبقه‌بندی تصویر سگ/گربه

## روش سریع استخراج ویژگی بدون داده‌افزایی: استخراج ویژگی با استفاده از پایه‌ی کانوولوشنال پیش‌آموزش دیده (۲ از ۲)

```
def extract_features(directory, sample_count):
    features = np.zeros(shape=(sample_count, 4, 4, 512))
    labels = np.zeros(shape=(sample_count))
    generator = datagen.flow_from_directory(
        directory,
        target_size=(150, 150),
        batch_size=batch_size,
        class_mode='binary')
   i = 0
   for inputs_batch, labels_batch in generator:
       features_batch = conv_base.predict(inputs_batch)
       features[i * batch_size : (i + 1) * batch_size] = features_batch
       labels[i * batch_size : (i + 1) * batch_size] = labels_batch
       i += 1
       if i * batch_size >= sample_count:
           break
   return features, labels

train_features, train_labels = extract_features(train_dir, 2000)
validation_features, validation_labels = extract_features(validation_dir, 1000)
test_features, test_labels = extract_features(test_dir, 1000)
# Flatten fatures
train_features = np.reshape(train_features, (2000, 4 * 4 * 512))
validation_features = np.reshape(validation_features, (1000, 4 * 4 * 512))
test_features = np.reshape(test_features, (1000, 4 * 4 * 512))
```

**Note that because generators yield data indefinitely in a loop, you must break after every image has been seen once.**

## مسئله‌ی طبقه‌بندی تصویر سگ/گربه

روش سریع استخراج ویژگی بدون داده‌افزایی: تعریف و آموزش طبقه‌بندی‌کننده‌ی متصل متراکم

حال طبقه‌بندی‌کننده‌ی متصل متراکم را تعریف می‌کنیم،
از dropout برای رگولاریزاسیون استفاده می‌کنیم و آن را با داده‌ها و برچسب‌های اخیراً ذخیره شده آموزش می‌دهیم.

### Defining and training the densely connected classifier

```
from keras import models
from keras import layers
from keras import optimizers

model = models.Sequential()
model.add(layers.Dense(256, activation='relu', input_dim=4 * 4 * 512))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer=optimizers.RMSprop(lr=2e-5),
              loss='binary_crossentropy',
              metrics=['acc'])

history = model.fit(train_features, train_labels,
                    epochs=30,
                    batch_size=20,
                    validation_data=(validation_features, validation_labels))
```

آموزش بسیار سریع است، زیرا تنها با دو لایه‌ی Dense سروکار داریم.

# مسئله‌ی طبقه‌بندی تصویر سگ/گربه

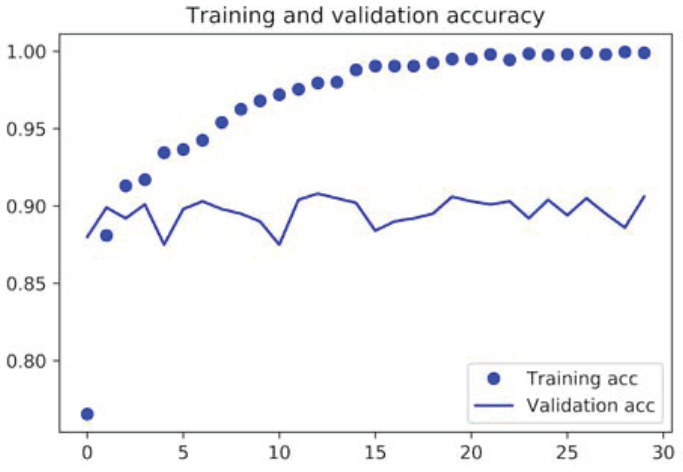
روش سریع استخراج ویژگی بدون داده‌افزایی: رسم نتایج

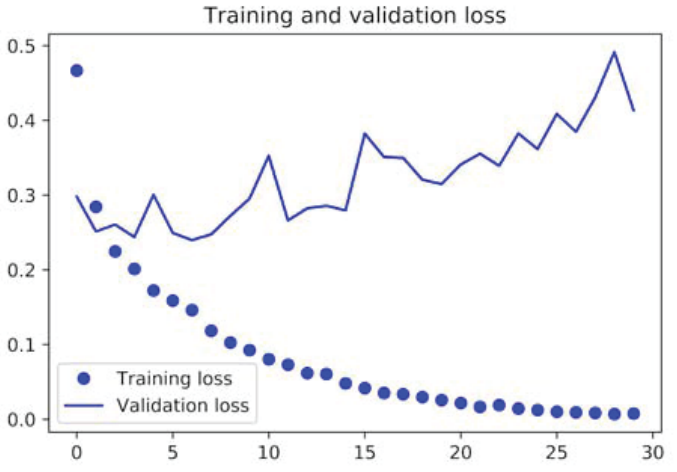
## Plotting the results

```
import matplotlib.pyplot as plt

acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss’]

epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```

# مسئله‌ی طبقه‌بندی تصویر سگ/گربه

روش سریع استخراج ویژگی بدون داده‌افزایی: نمودارهای اتلاف و دقت روی داده‌های آموزشی و اعتبارسنجی




در اینجا به دقتی حدود ۹۰% روی داده‌های اعتبارسنجی رسیده‌ایم
(بسیار بهتر از آنچه در مدل کوچک آموزش داده شده از صفر رسیدیم)
اما
تقریباً از نقطه‌ی شروع با بیش‌برازش مواجه شده‌ایم؛
با وجود اینکه از dropout با نرخ تقریباً بالایی استفاده کرده‌ایم.
زیرا: این تکنیک از داده‌افزایی استفاده نکرده است
و داده‌افزایی برای جلوگیری از بیش‌برازش در حالتی که با مجموعه‌داده‌های کوچک تصویری کار می‌کنیم ضروری است.

# مسئله‌ی طبقه‌بندی تصویر سگ/گربه

روش استخراج ویژگی با داده‌افزایی

FEATURE EXTRACTION WITH DATA AUGMENTATION

این روش کندتر و پرهزینه‌تر است،
اما اجازه می‌دهد در حین آموزش از داده‌افزایی استفاده کنیم:
مدل conv_base را گسترش می‌دهیم و آن را به‌صورت انتها به انتها روی ورودی‌ها اجرا می‌کنیم.

## Adding a densely connected classifier on top of the convolutional base

```
from keras import models
from keras import layers
model = models.Sequential()
model.add(conv_base)
model.add(layers.Flatten())
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

چون مدل‌ها دقیقاً مانند لایه‌ها رفتار می‌کنند،
می‌توان یک مدل را به مدل Sequential اضافه نمود، دقیقاً مانند اضافه کردن یک لایه.

# مسئله‌ی طبقه‌بندی تصویر سگ/گربه

## روش استخراج ویژگی با داده‌افزایی: جزئیات مدل

```
>>> model.summary()
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 4, 4, 512)         14714688
_________________________________________________________________
flatten_1 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               2097408
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 257
=================================================================
Total params: 16,812,353
Trainable params: 16,812,353
Non-trainable params: 0
```

مشاهده می‌شود که پایه‌ی کانوولوشنال حدود ۱۵ میلیون پارامتر دارد که بسیار زیاد است.
طبقه‌بندی‌کننده‌ای که در بالا اضافه شده است، حدود ۲ میلیون پارامتر دارد.

# مسئله‌ی طبقه‌بندی تصویر سگ/گربه

روش استخراج ویژگی با داده‌افزایی: انجماد پایه‌ی کانوولوشنال

## انجماد یک لایه
*Freezing a Layer*

جلوگیری از به‌هنگام‌سازی وزن‌های آن لایه در حین آموزش

پیش از کامپایل کردن و شروع آموزش مدل، بسیار مهم است که پایه‌ی کانوولوشنال را منجمد کنیم.
اگر این کار را نکنیم بازنمایی‌هایی که قبلاً توسط پایه‌ی کانوولوشنال یاد گرفته شده‌اند، در حین آموزش تغییر می‌یابند، زیرا لایه‌های Dense در بالا به‌صورت تصادفی مقداردهی اولیه شده‌اند و تغییر وزن‌های بسیار بزرگی را در شبکه منتشر می‌کنند که به‌طور مؤثر، بازنمایی‌های یادگرفته‌شده‌ی قبلی را تخریب می‌کنند.

```
>>> print('This is the number of trainable weights '
          'before freezing the conv base:', len(model.trainable_weights))
This is the number of trainable weights before freezing the conv base: 30
>>> conv_base.trainable = False
>>> print('This is the number of trainable weights '
          'after freezing the conv base:', len(model.trainable_weights))
This is the number of trainable weights after freezing the conv base: 4
```

با تنظیمات فوق، تنها وزن‌های دو لایه‌ی Dense که اضافه شده‌اند آموزش می‌یابند؛
در مجموع ۴ تانسور وزن خواهیم داشت (برای هر لایه ۲ تا: ماتریس وزن اصلی و بردار بایاس)

وقتی آموزش‌پذیری وزن را پس از کامپایل کردن مدل تغییر می‌دهیم،
باید آن مدل را مجدداً کامپایل کنیم. وگرنه این تغییرات نادیده گرفته می‌شوند.

# مسئله‌ی طبقه‌بندی تصویر سگ/گربه

روش استخراج ویژگی با داده‌افزایی: آموزش مدل انتها به انتها با یک پایه‌ی کانوولوشنال منجمد (۱ از ۲)

## Training the model end to end with a frozen convolutional base

```
from keras.preprocessing.image import ImageDataGenerator
from keras import optimizers

train_datagen = ImageDataGenerator(
      rescale=1./255,
      rotation_range=40,
      width_shift_range=0.2,
      height_shift_range=0.2,
      shear_range=0.2,
      zoom_range=0.2,
      horizontal_flip=True,
      fill_mode='nearest')

test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(150, 150),
        batch_size=20,
        class_mode='binary')
```

Note that the validation data shouldn't be augmented!

Because you use binary_crossentropy loss, you need binary labels.

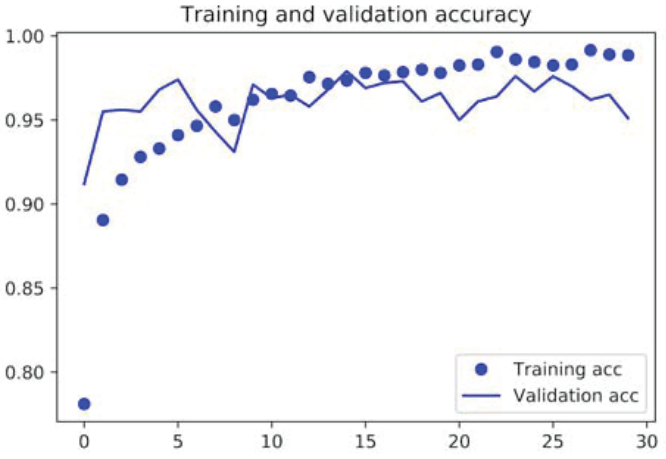
# مسئله‌ی طبقه‌بندی تصویر سگ/گربه

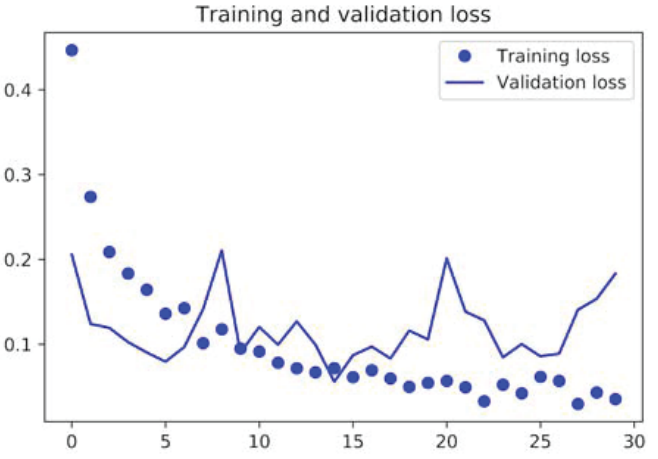
## روش استخراج ویژگی با داده‌افزایی: آموزش مدل انتها به انتها با یک پایه‌ی کانوولوشنال منجمد (۲ از ۲)

```
validation_generator = test_datagen.flow_from_directory(
        validation_dir,
        target_size=(150, 150),
        batch_size=20,
        class_mode='binary’)

model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=2e-5),
              metrics=['acc’])

history = model.fit_generator(
      train_generator,
      steps_per_epoch=100,
      epochs=30,
      validation_data=validation_generator,
      validation_steps=50)
```

# مسئله‌ی طبقه‌بندی تصویر سگ/گربه

روش استخراج ویژگی با داده‌افزایی: نمودارهای اتلاف و دقت روی داده‌های آموزشی و اعتبارسنجی




در اینجا به دقتی حدود ۹۶% روی داده‌های اعتبارسنجی رسیده‌ایم
(بسیار بهتر از آنچه در مدل کوچک آموزش داده شده از صفر رسیدیم)

# شبکه‌های عصبی کانوولوشنال

## روش‌های استفاده از یک مدل پیش‌آموزش دیده: تنظیم دقیق

**روش‌های استفاده از یک مدل پیش‌آموزش دیده**

۱ استخراج ویژگی
*Feature Extraction*

۲ **تنظیم دقیق**
*Fine-tuning*

خارج ساختن تعدادی از لایه‌های بالایی مدل از حالت انجماد و آموزش توأم آن همراه با لایه‌های اضافه شده به مدل

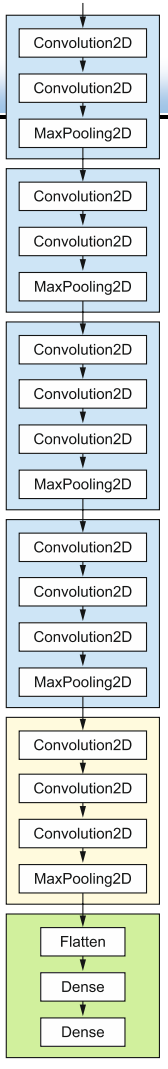
این روش مکملی برای روش استخراج ویژگی است.

تعداد کمی از لایه‌های بالایی مدل منجمد شده‌ای که برای استخراج ویژگی‌ها به‌کار رفت را از انجماد خارج می‌کنیم، و هم بخشی از مدل که اخیراً اضافه شده (در اینجا طبقه‌بندی‌کننده‌ی تماماً متصل) و هم این لایه‌های بالایی را توأماً مورد آموزش قرار می‌دهیم.

این تکنیک، بازنمایی‌های انتزاعی‌تر مدل مورد استفاده‌ی مجدد را اندکی تنظیم می‌کند تا آنها را به مسئله‌ی مورد نظر مربوط‌تر نماید.

# شبکه‌های عصبی کانوولوشنال

## روش‌های استفاده از یک مدل پیش‌آموزش دیده: تنظیم دقیق




منجمد کردن پایه‌ی کانوولوشنال ضروری است
تا بتوان طبقه‌بندی‌کننده‌ی بالایی را که به‌صورت تصادفی مقداردهی اولیه شده است را مورد آموزش قرار داد.
به همین دلیل، تنظیم دقیق لایه‌های بالایی پایه‌ی کانوولوشنال تنها در زمانی امکان‌پذیر است که طبقه‌بندی‌کننده‌ی بالایی از قبل آموزش **یافته** باشد.

اگر این طبقه‌بندی‌کننده از قبل آموزش یافته باشد،
آن‌گاه سیگنال خطایی که در حین آموزش در شبکه منتشر می‌شود، بسیار بزرگ خواهد بود و بازنمایی‌هایی که قبلاً توسط لایه‌های دقیقاً تنظیم‌شده یاد گرفته شده‌اند، نابود خواهند شد.

بر این اساس، مراحل تنظیم دقیق یک شبکه به‌صورت زیر است:

**مراحل تنظیم دقیق (Fine-Tuning) یک شبکه**

۱) اضافه کردن شبکه‌ی سفارشی خود به بالای شبکه‌ی پایه‌ی هم‌اکنون آموزش دیده
۲) منجمد کردن شبکه‌ی پایه
۳) آموزش بخشی که اخیراً اضافه شده است
۴) خارج کردن برخی لایه‌های شبکه‌ی پایه از انجماد
۵) آموزش توأم این لایه‌ها و بخش سفارشی

Fine-tuning the last convolutional block of the VGG16 network

# مسئله‌ی طبقه‌بندی تصویر سگ/گربه

## تنظیم دقیق: جزئیات مدل پایه‌ی کانوولوشنال

```
>>> conv_base.summary()
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 150, 150, 3)       0
_________________________________________________________________
block1_conv1 (Convolution2D)  (None, 150, 150, 64)     1792
_________________________________________________________________
block1_conv2 (Convolution2D)  (None, 150, 150, 64)     36928
_________________________________________________________________
block1_pool (MaxPooling2D)    (None, 75, 75, 64)       0
_________________________________________________________________
block2_conv1 (Convolution2D)  (None, 75, 75, 128)      73856
_________________________________________________________________
block2_conv2 (Convolution2D)  (None, 75, 75, 128)      147584
_________________________________________________________________
block2_pool (MaxPooling2D)    (None, 37, 37, 128)      0
_________________________________________________________________
block3_conv1 (Convolution2D)  (None, 37, 37, 256)      295168
_________________________________________________________________
block3_conv2 (Convolution2D)  (None, 37, 37, 256)      590080
_________________________________________________________________
block3_conv3 (Convolution2D)  (None, 37, 37, 256)      590080
_________________________________________________________________
block3_pool (MaxPooling2D)    (None, 18, 18, 256)      0
_________________________________________________________________
block4_conv1 (Convolution2D)  (None, 18, 18, 512)      1180160
_________________________________________________________________
block4_conv2 (Convolution2D)  (None, 18, 18, 512)      2359808
_________________________________________________________________
block4_conv3 (Convolution2D)  (None, 18, 18, 512)      2359808
_________________________________________________________________
block4_pool (MaxPooling2D)    (None, 9, 9, 512)        0
_________________________________________________________________
block5_conv1 (Convolution2D)  (None, 9, 9, 512)        2359808
_________________________________________________________________
block5_conv2 (Convolution2D)  (None, 9, 9, 512)        2359808
_________________________________________________________________
block5_conv3 (Convolution2D)  (None, 9, 9, 512)        2359808
_________________________________________________________________
block5_pool (MaxPooling2D)    (None, 4, 4, 512)        0
=================================================================
Total params: 14714688
```

**مراحل تنظیم دقیق (Fine-Tuning) یک شبکه**

۱) اضافه کردن شبکه‌ی سفارشی خود به بالای شبکه‌ی پایه‌ی هم‌اکنون آموزش دیده
۲) منجمد کردن شبکه‌ی پایه
۳) آموزش بخشی که اخیراً اضافه شده است
۴) خارج کردن برخی لایه‌های شبکه‌ی پایه از انجماد
۵) آموزش توأم این لایه‌ها و بخش سفارشی

از اینجا ادامه می‌دهیم

# مسئله‌ی طبقه‌بندی تصویر سگ/گربه

تنظیم دقیق: غیرمنجمدسازی مدل پایه و سپس انجماد برخی لایه‌های آن

سه لایه‌ی کانوولوشنال آخر را مورد **تنظیم دقیق** قرار می‌دهیم:
یعنی همه‌ی لایه‌ها تا `block4_pool` باید منجمد شوند و
لایه‌های `block5_conv1`، `block5_conv2` و `block5_conv3` باید آموزش‌پذیر باشند.

چرا تعداد لایه‌ی بیشتری را مورد تنظیم دقیق قرار نمی‌دهیم؟
چرا کل پایه‌ی کانوولوشنال را مورد تنظیم دقیق قرار نمی‌دهیم؟
اگر چه می‌توانیم، اما دو موضوع وجود دارد:

- **لایه‌های اول در پایه‌ی کانوولوشنال، ویژگی‌های عام‌تر و قابل استفاده‌ی مجدد را کدگذاری می‌کنند، درحالی‌که لایه‌های بالاتر ویژگی‌های خاص‌تر را کدگذاری می‌کنند.**
  مفیدتر است که ویژگی‌های خاص‌تر را تنظیم دقیق کنیم، زیرا آنها هستند که نیاز به تغییر منظور برای مسئله‌ی جدید دارند.
  در تنظیم دقیق لایه‌های پایین‌تر افت بازدهی سریعی وجود دارد.

- **هرچه تعداد پارامتر بیشتری وارد آموزش شود، ریسک بیش‌برازش بیشتر می‌شود.**
  پایه‌ی کانوولوشنال ۱۵ میلیون پارامتر دارد، پس تلاش برای آموزش آن روی مجموعه‌داده‌ی کوچک ریسک زیادی دارد.

## مسئله‌ی طبقه‌بندی تصویر سگ/گربه

تنظیم دقیق: منجمدسازی تا یک لایه‌ی خاص

تنها دو تا سه لایه‌ی بالایی در پایه‌ی کانوولوشنال را مورد تنظیم دقیق قرار می‌دهیم.

از ادامه‌ی مثال قبلی کار را دنبال می‌کنیم:

### Freezing all layers up to a specific one

```
conv_base.trainable = True

set_trainable = False
for layer in conv_base.layers:
    if layer.name == 'block5_conv1':
        set_trainable = True
   if set_trainable:
        layer.trainable = True
   else:
        layer.trainable = False
```

# مسئله‌ی طبقه‌بندی تصویر سگ/گربه

تنظیم دقیق: تنظیم دقیق مدل

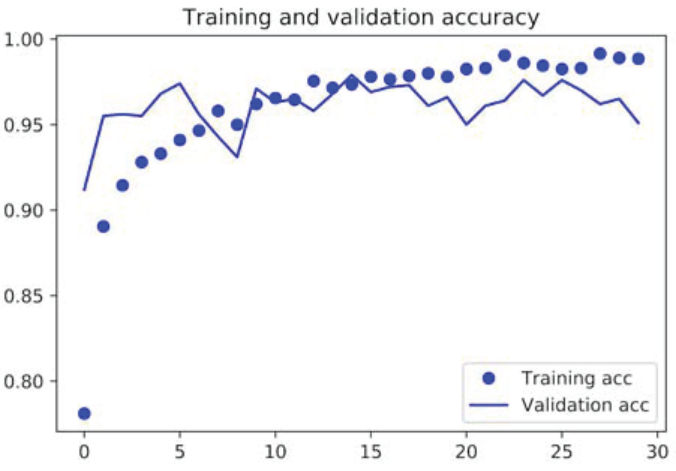
از بهینه‌ساز RMSprop با یک نرخ یادگیری بسیار کوچک برای تنظیم دقیق استفاده می‌کنیم.
**دلیل:** محدود کردن بزرگی تغییراتی که روی بازنمایی‌های سه لایه‌ی آخر انجام می‌شود.
اگر به‌روزرسانی‌ها بسیار بزرگ باشند، ممکن است به این بازنمایی‌ها صدمه بزنند.

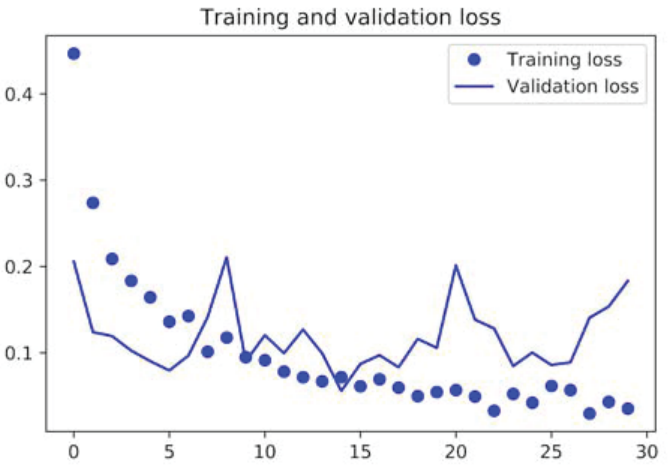
## Fine-tuning the model

```
model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-5),
              metrics=['acc'])

history = model.fit_generator(
      train_generator,
      steps_per_epoch=100,
      epochs=100,
      validation_data=validation_generator,
      validation_steps=50)
```

# مسئله‌ی طبقه‌بندی تصویر سگ/گربه

## تنظیم دقیق: نمودارهای اتلاف و دقت روی داده‌های آموزشی و اعتبارسنجی




منحنی‌ها بسیار نویزی به‌نظر می‌رسند.
برای خواناتر کردن آنها، می‌توانیم آنها را هموار کنیم:
با جایگزین کردن هر مقدار اتلاف و دقت با متوسط متحرک نمایی آن کمیت‌ها.

# مسئله‌ی طبقه‌بندی تصویر سگ/گربه

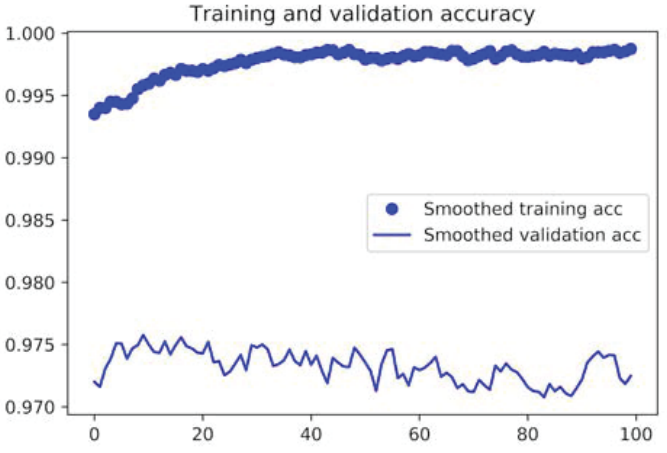
تنظیم دقیق: تابعی برای هموارسازی منحنی‌ها

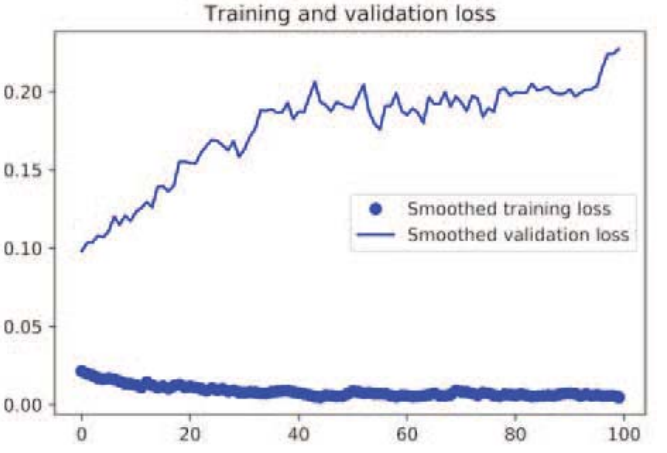
## Smoothing the plots

```
def smooth_curve(points, factor=0.8):
  smoothed_points = []
  for point in points:
    if smoothed_points:
       previous = smoothed_points[-1]
       smoothed_points.append(previous * factor + point * (1 - factor))
    else:
       smoothed_points.append(point)
  return smoothed_points

plt.plot(epochs, smooth_curve(acc), 'bo', label='Smoothed training acc')
plt.plot(epochs, smooth_curve(val_acc), 'b', label='Smoothed validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, smooth_curve(loss), 'bo', label='Smoothed training loss')
plt.plot(epochs, smooth_curve(val_loss), 'b', label='Smoothed validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```

# مسئله‌ی طبقه‌بندی تصویر سگ/گربه

## تنظیم دقیق: نمودارهای اتلاف و دقت روی داده‌های آموزشی و اعتبارسنجی (پس از هموارسازی)




۱% بهبود در دقت دیده می‌شود: از ۹۶% به ۹۷%.

منحنی اتلاف هیچ بهبود واقعی را نشان نمی‌دهد (در واقع، رو به بدتر شدن است)،
اما چرا دقت پایدار شده است یا بهبود می‌یابد اگر اتلاف در حال کاهش نیست؟
پاسخ ساده است:
آنچه نمایش داده می‌شود، متوسط نقطه به نقطه‌ی مقادیر اتلاف است،
اما آنچه برای دقت مهم است، توزیع مقادیر اتلاف است، نه مقادیر متوسط آن.
زیرا مقدار دقت نتیجه‌ی حاصل از یک سطح آستانه‌ی دودویی از احتمال کلاسی است که توسط مدل پیش‌بینی شده است. ممکن است مدل همچنان در حال بهبود باشد حتی اگر موضوع این در متوسط اتلاف منعکس نشده باشد.

# مسئله‌ی طبقه‌بندی تصویر سگ/گربه

## تنظیم دقیق: ارزیابی مدل با داده‌های آزمایشی

```
test_generator = test_datagen.flow_from_directory(
        test_dir,
        target_size=(150, 150),
        batch_size=20,
        class_mode='binary')

test_loss, test_acc = model.evaluate_generator(test_generator, steps=50)
print('test acc:', test_acc)
```

در اینجا به دقت ۹۷% روی داده‌های آزمایشی رسیده‌ایم.

در مسابقه‌ی اصلی kaggle با این مجموعه‌داده،
این یکی از نتایج برتر بوده است.
اما ما در اینجا با استفاده از تکنیک‌های پیشرفته‌ی یادگیری عمیق،
توانستیم با استفاده از بخش کوچکی از داده‌های آموزشی موجود (حدود ۱۰% آنها) به این نتیجه برسیم؛
که بسیار باارزش است.

# یادگیری عمیق برای بینایی کامپیوتری

خلاصه

## WRAPPING UP

Here's what you should take away from the exercises in the past two sections:

- **Convnets** are the best type of machine-learning models for computer-vision tasks.
  It's possible to train one from scratch even on a very small dataset, with decent results.
- On a small dataset, overfitting will be the main issue.
  Data augmentation is a powerful way to fight overfitting when you're working with image data.
- It's easy to reuse an existing convnet on a new dataset via feature extraction.
  This is a valuable technique for working with small image datasets.
- As a complement to feature extraction, you can use fine-tuning, which adapts to a new problem some of the representations previously learned by an existing model. This pushes performance a bit further.

یادگیری عمیق

یادگیری عمیق برای بینایی کامپیوتری

# ۴

# مصورسازی آنچه convnetها یاد می‌گیرند

# مصورسازی آنچه convnetها یاد می‌گیرند

VISUALIZING WHAT CONVNETS LEARN

اغلب گفته می‌شود مدل‌های یادگیری عمیق، «جعبه سیاه» هستند:
یادگیری بازنمایی‌هایی که استخراج آنها دشوار است و ارائه‌ی آنها به شکل خوانا برای انسان دشوار است.

این موضوع تا حدی برای برخی مدل‌های یادگیری عمیق درست است،
اما برای شبکه‌های عصبی کانوولوشنال قطعاً درست نیست:

بازنمایی‌های یادگرفته شده توسط convnetها برای مصورسازی بسیار مناسب است،
عمدتاً به این دلیل که آنها بازنمایی‌های مفاهیم دیداری هستند.

# مصورسازی آنچه convnetها یاد می‌گیرند

## روش‌های مصورسازی

فهرست گسترده‌ای از تکنیک‌ها برای مصورسازی و تفسیرگری بازنمایی‌های convnet ارائه شده است.

**سه روش مفیدتر و دردسترس‌تر برای مصورسازی بازنمایی‌های شبکه‌های عصبی کانوولوشنال**

۱. **مصورسازی خروجی‌های میانی شبکه (فعالیت‌های میانی)**
*Visualizing intermediate convnet outputs (intermediate activations)*
مفید برای فهم اینکه لایه‌های متوالی شبکه چگونه ورودی خود را تبدیل می‌کنند؛ برای اخذ ایده‌های اولیه در مورد معنای فیلترهای منفرد شبکه

۲. **مصورسازی فیلترهای شبکه**
*Visualizing convnets filters*
مفید برای فهم دقیق اینکه هر فیلتر در شبکه، چه مفاهیم یا چه الگوهای دیداری را می‌پذیرد

۳. **مصورسازی نقشه‌های گرمایی فعالیت کلاس در یک تصویر**
*Visualizing heatmaps of class activation in an image*
مفید برای فهم اینکه کدام بخش‌های یک تصویر به‌عنوان تعلق به یک کلاس داده شده شناسایی می‌شوند ⇐ امکان مکان‌یابی اشیا در تصویر

# روش‌های مصورسازی: مصورسازی فعالیت‌های میانی

VISUALIZING INTERMEDIATE ACTIVATIONS

**مصورسازی فعالیت‌های میانی**:
نمایش نقشه‌های ویژگی خروجی
که توسط لایه‌های گوناگون کانوولوشن و پولینگ در شبکه
به‌ازای یک ورودی خاص ایجاد می‌شوند.
(خروجی یک لایه اغلب فعالیت/activation نامیده می‌شود: خروجی تابع فعالیت activation function)

این روش به ما نمایی در مورد چگونگی تجزیه‌ی ورودی به فیلترهای مختلف یادگیری شده توسط شبکه می‌دهد.

می‌خواهیم نقشه‌های ویژگی دارای سه بعد را مصورسازی کنیم:
width, height, and depth (channels)
هر کانال ویژگی‌های نسبتاً مستقلی را کدگذاری می‌کند؛
بنابراین، روش مناسب برای مصورسازی این نقشه‌های ویژگی،
رسم مستقل محتوای هر یک از آنها به‌صورت یک تصویر دو-بعدی است.

# روش‌های مصورسازی: مصورسازی فعالیت‌های میانی

## مثال

از مدل ذخیره شده در مثال‌های قبلی استفاده می‌کنیم:

```
>>> from keras.models import load_model
>>> model = load_model('cats_and_dogs_small_2.h5')
>>> model.summary() <1> As a reminder.
_________________________________________________________________
Layer (type) Output Shape Param #
=================================================================
conv2d_5 (Conv2D) (None, 148, 148, 32) 896
_________________________________________________________________
maxpooling2d_5 (MaxPooling2D) (None, 74, 74, 32) 0
_________________________________________________________________
conv2d_6 (Conv2D) (None, 72, 72, 64) 18496
_________________________________________________________________
maxpooling2d_6 (MaxPooling2D) (None, 36, 36, 64) 0
_________________________________________________________________
conv2d_7 (Conv2D) (None, 34, 34, 128) 73856
_________________________________________________________________
maxpooling2d_7 (MaxPooling2D) (None, 17, 17, 128) 0
_________________________________________________________________
conv2d_8 (Conv2D) (None, 15, 15, 128) 147584
_________________________________________________________________
maxpooling2d_8 (MaxPooling2D) (None, 7, 7, 128) 0
_________________________________________________________________
flatten_2 (Flatten) (None, 6272) 0
_________________________________________________________________
dropout_1 (Dropout) (None, 6272) 0
_________________________________________________________________
dense_3 (Dense) (None, 512) 3211776
_________________________________________________________________
dense_4 (Dense) (None, 1) 513
=================================================================
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0
```

# روش‌های مصورسازی: مصورسازی فعالیت‌های میانی

## پیش‌پردازش یک تصویر تنها

یک تصویر (تصویر یک گربه) را باز می‌کنیم (از مجموعه‌ی آزمایشی نه آموزشی)

### Preprocessing a single image

```
img_path = '/Users/fchollet/Downloads/cats_and_dogs_small/test/cats/cat.1700.jpg'

from keras.preprocessing import image
import numpy as np

img = image.load_img(img_path, target_size=(150, 150))
img_tensor = image.img_to_array(img)
img_tensor = np.expand_dims(img_tensor, axis=0)
img_tensor /= 255.

<1> Its shape is (1, 150, 150, 3)
print(img_tensor.shape)
```

Preprocesses the image into a 4D tensor

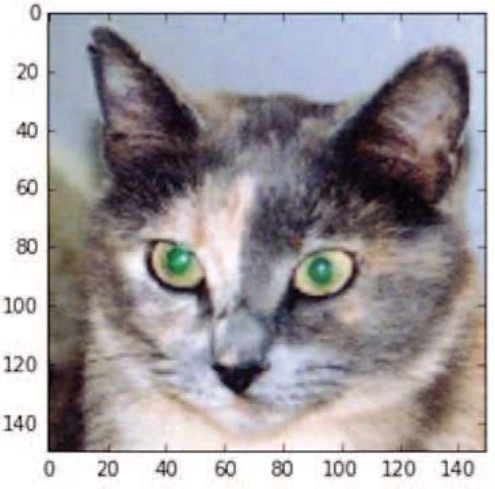
Remember that the model was trained on inputs that were preprocessed this way.

# روش‌های مصورسازی: مصورسازی فعالیت‌های میانی

نمایش تصویر آزمایشی

## Displaying the test picture

```
import matplotlib.pyplot as plt

plt.imshow(img_tensor[0])
plt.show()
```



# روش‌های مصورسازی: مصورسازی فعالیت‌های میانی

نمونه‌سازی یک مدل از یک تانسور ورودی و لیستی از تانسورهای خروجی

برای استخراج نقشه‌های ویژگی که می‌خواهیم به آنها نگاه کنیم،
یک مدل Keras درست می‌کنیم که دسته‌ای از تصاویر را به‌عنوان ورودی دریافت می‌کند،
و فعالیت‌های تمام لایه‌های کانوولوشنی و پولینگ را در قالب خروجی ثبت می‌کند.

از کلاس `Model` در Keras استفاده می‌کنیم.
یک نمونه از مدل با استفاده از دو آرگومان ساخته می‌شود:
۱) یک تانسور ورودی (یک لیست از تانسورهای ورودی) ۲) یک تانسور خروجی (یک لیست از تانسورهای خروجی)
مدل حاصل، ورودی‌های مشخص‌شده را به خروجی مشخص شده نگاشت می‌دهد.
یک مدل می‌تواند چند خروجی داشته باشد.

## Instantiating a model from an input tensor and a list of output tensors

```
from keras import models

layer_outputs = [layer.output for layer in model.layers[:8]]

activation_model = models.Model(inputs=model.input, outputs=layer_outputs)
```

Extracts the outputs of the top eight layers

Creates a model that will return these outputs, given the model input

# روش‌های مصورسازی: مصورسازی فعالیت‌های میانی

اجرای مدل برای تعیین مقادیر فعالیت‌ها

وقتی یک ورودی تصویری به این مدل داده می‌شود، این مدل مقادیر فعالیت لایه‌ها در مدل اصلی را برمی‌گرداند.
(در اینجا با یک مدل دارای یک ورودی و ۸ خروجی سروکار داریم)

## Running the model in predict mode

```
activations = activation_model.predict(img_tensor)
```

Returns a list of five Numpy arrays:
one array per layer activation

به‌عنوان مثال، فعالیت اولیه‌ی لایه‌ی کانوولوشنی برای ورودی تصویر گربه
به‌صورت زیر است:

```
>>> first_layer_activation = activations[0]
>>> print(first_layer_activation.shape)
(1, 148, 148, 32)
```

که یک نقشه‌ی ویژگی ۱۴۸×۱۴۸ با تعداد ۳۲ کانال است.

# روش‌های مصورسازی: مصورسازی فعالیت‌های میانی

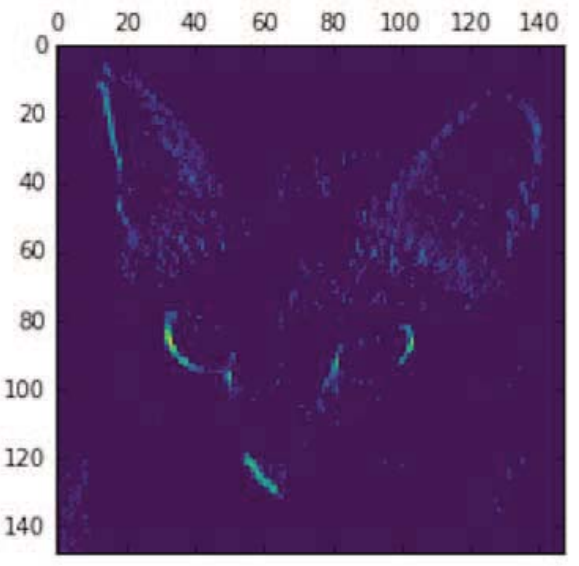
مصورسازی کانال چهارم فعالیت لایه‌ی اول کانوولوشنی

## Visualizing the fourth channel

```
import matplotlib.pyplot as plt
plt.matshow(first_layer_activation[0, :, :, 4], cmap='viridis')
```

کانال ۴ از یک نقشه‌ی ویژگی ۱۴۸×۱۴۸

به نظر می‌رسد این کانال یک آشکارساز لبه‌ی قطری
(diagonal edge detector)
را کدگذاری می‌کند.



# روش‌های مصورسازی: مصورسازی فعالیت‌های میانی

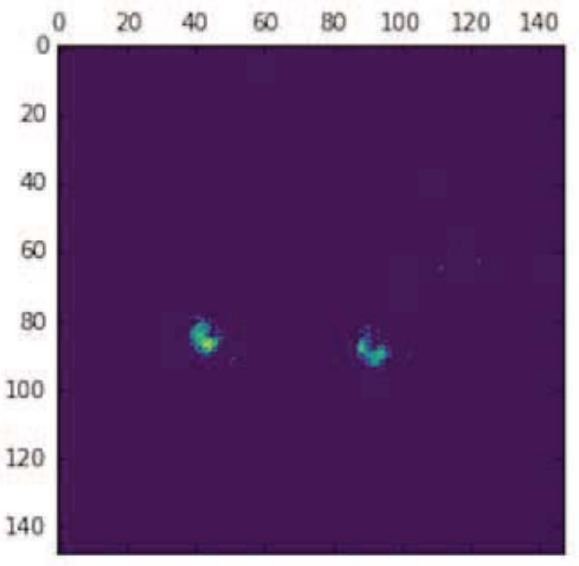
مصورسازی کانال هفتم فعالیت لایه‌ی اول کانوولوشنی

## Visualizing the seventh channel

```
plt.matshow(first_layer_activation[0, :, :, 7], cmap='viridis')
```

کانال ۷ از یک نقشه‌ی ویژگی ۱۴۸×۱۴۸

به نظر می‌رسد این کانال یک آشکارساز نقاط سبز روشن (bright green dot) را کدگذاری می‌کند.
(مفید برای کدگذاری چشم‌های گربه)



# روش‌های مصورسازی: مصورسازی فعالیت‌های میانی

## رسم مصورسازی کامل همه‌ی کانال‌ها در همه‌ی فعالیت‌های میانی در شبکه (۱ از ۲)

همه‌ی کانال‌ها در هر یک از هشت نقشه‌ی فعالیت شبکه را
استخراج و رسم می‌کنیم.

### Visualizing every channel in every intermediate activation

```
layer_names = []
for layer in model.layers[:8]:
    layer_names.append(layer.name)

images_per_row = 16

for layer_name, layer_activation in zip(layer_names, activations):
    n_features = layer_activation.shape[-1]

    size = layer_activation.shape[1]

    n_cols = n_features // images_per_row
    display_grid = np.zeros((size * n_cols, images_per_row * size))
```

Names of the layers, so you can have them as part of your plot

Displays the feature maps

Number of features in the feature map

The feature map has shape (1, size, size, n_features).

Tiles the activation channels in this matrix

# روش‌های مصورسازی: مصورسازی فعالیت‌های میانی

رسم مصورسازی کامل همه‌ی کانال‌ها در همه‌ی فعالیت‌های میانی در شبکه (۲ از ۲)

```
    for col in range(n_cols):
        for row in range(images_per_row):
            channel_image = layer_activation[0,
                                             :, :,
                                             col * images_per_row + row]
            channel_image -= channel_image.mean()
            channel_image /= channel_image.std()
            channel_image *= 64
            channel_image += 128
            channel_image = np.clip(channel_image, 0, 255).astype('uint8')
            display_grid[col * size : (col + 1) * size,
                         row * size : (row + 1) * size] = channel_image
scale = 1. / size
plt.figure(figsize=(scale * display_grid.shape[1],
                    scale * display_grid.shape[0]))
plt.title(layer_name)
plt.grid(False)
plt.imshow(display_grid, aspect='auto', cmap='viridis')
```

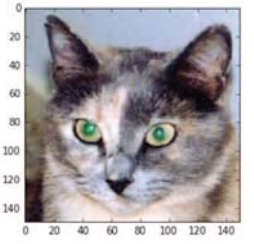
Tiles each filter into a big horizontal grid

Post-processes the feature to make it visually palatable

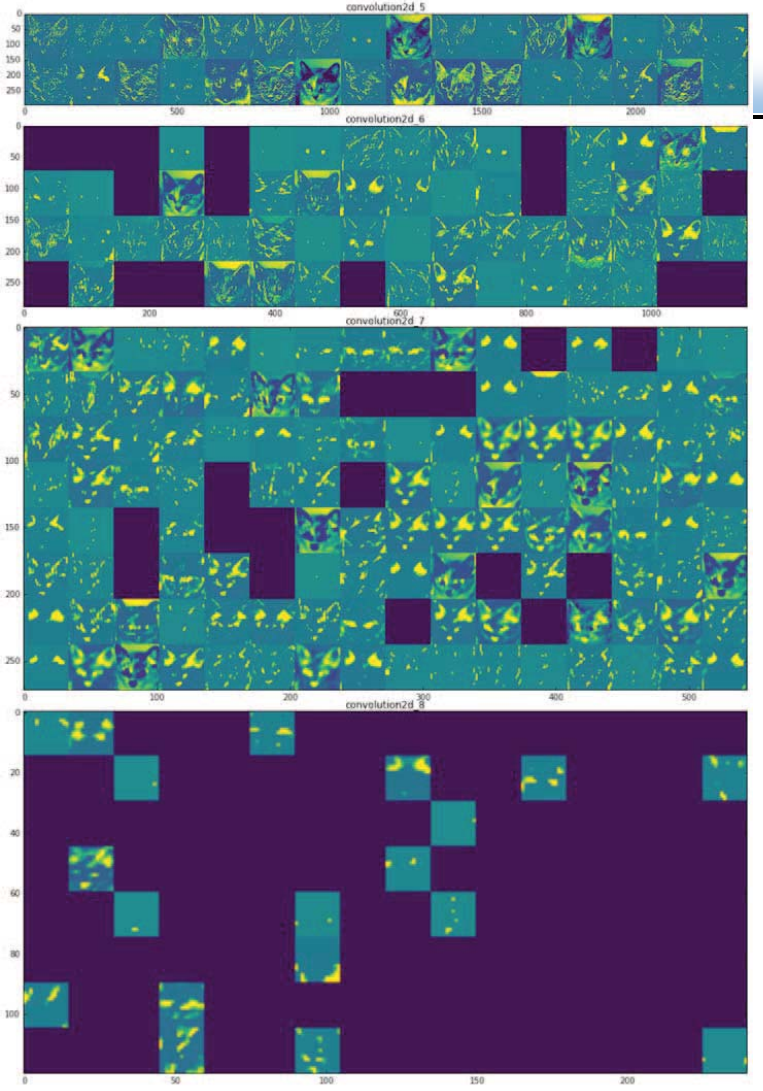
Displays the grid

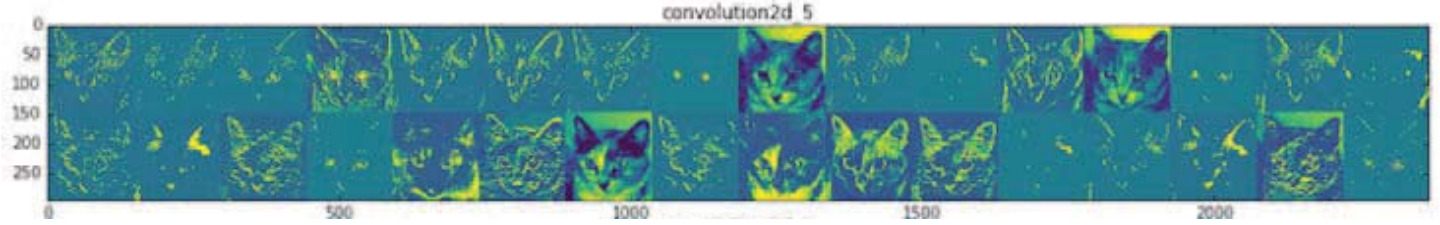
# مصورسازی فعالیت‌های میانی

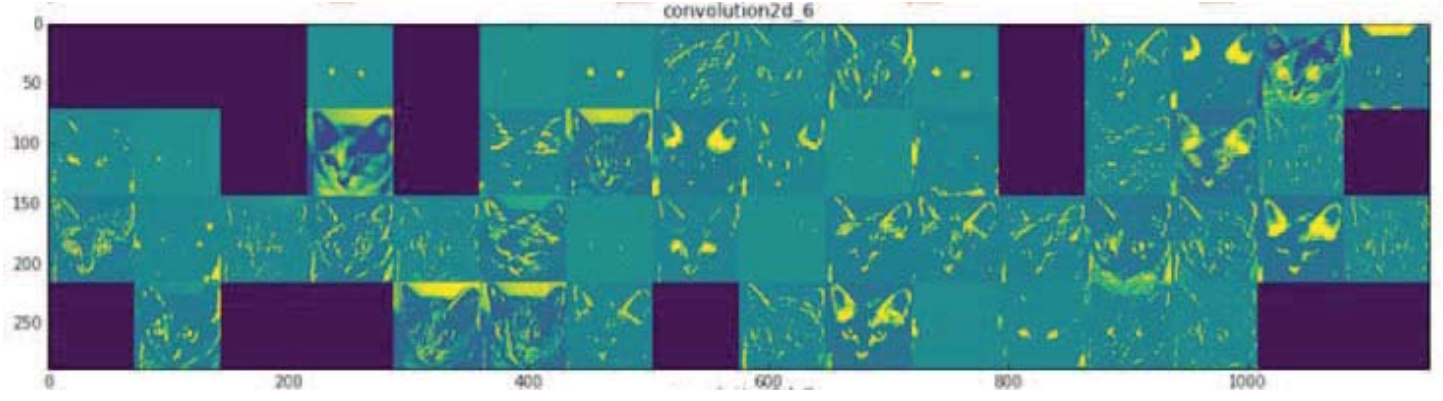
## همه‌ی کانال‌های همه‌ی فعالیت‌ها

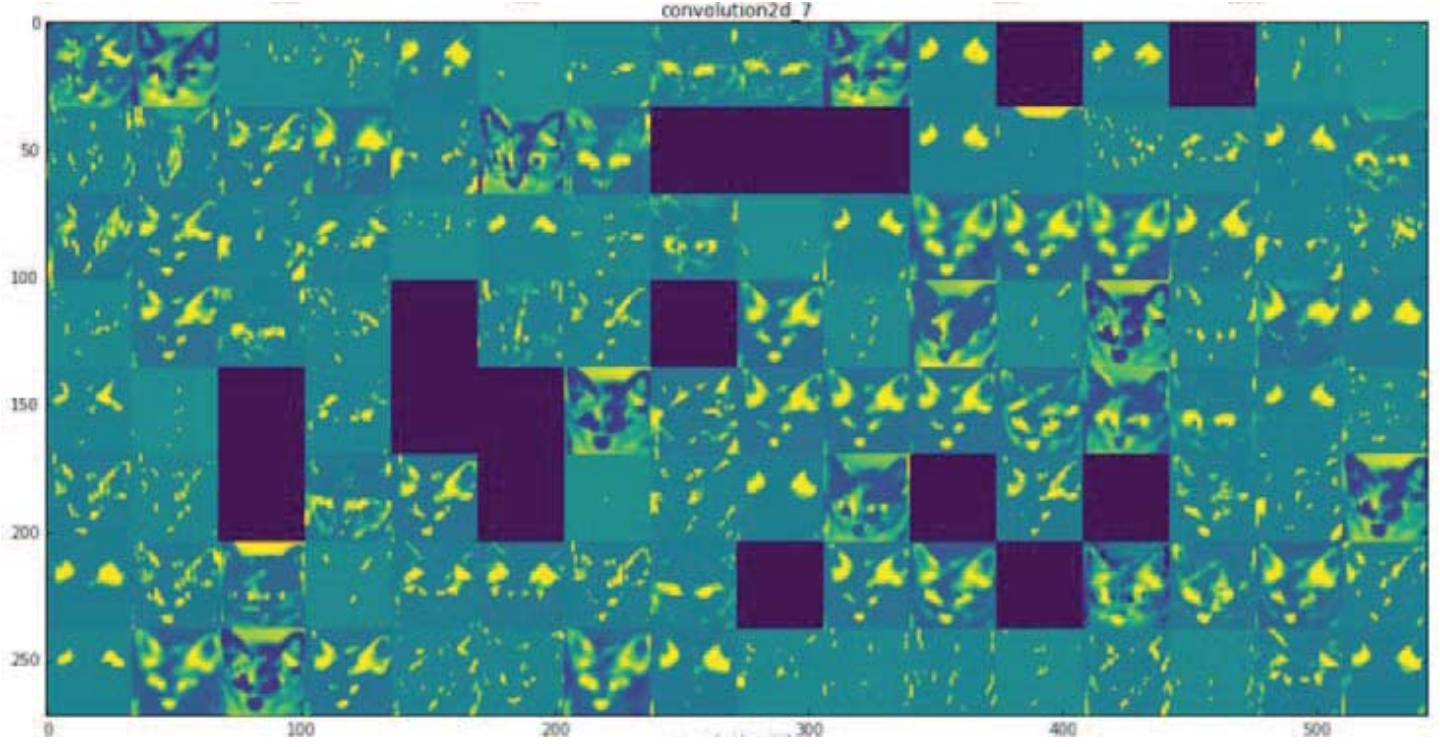


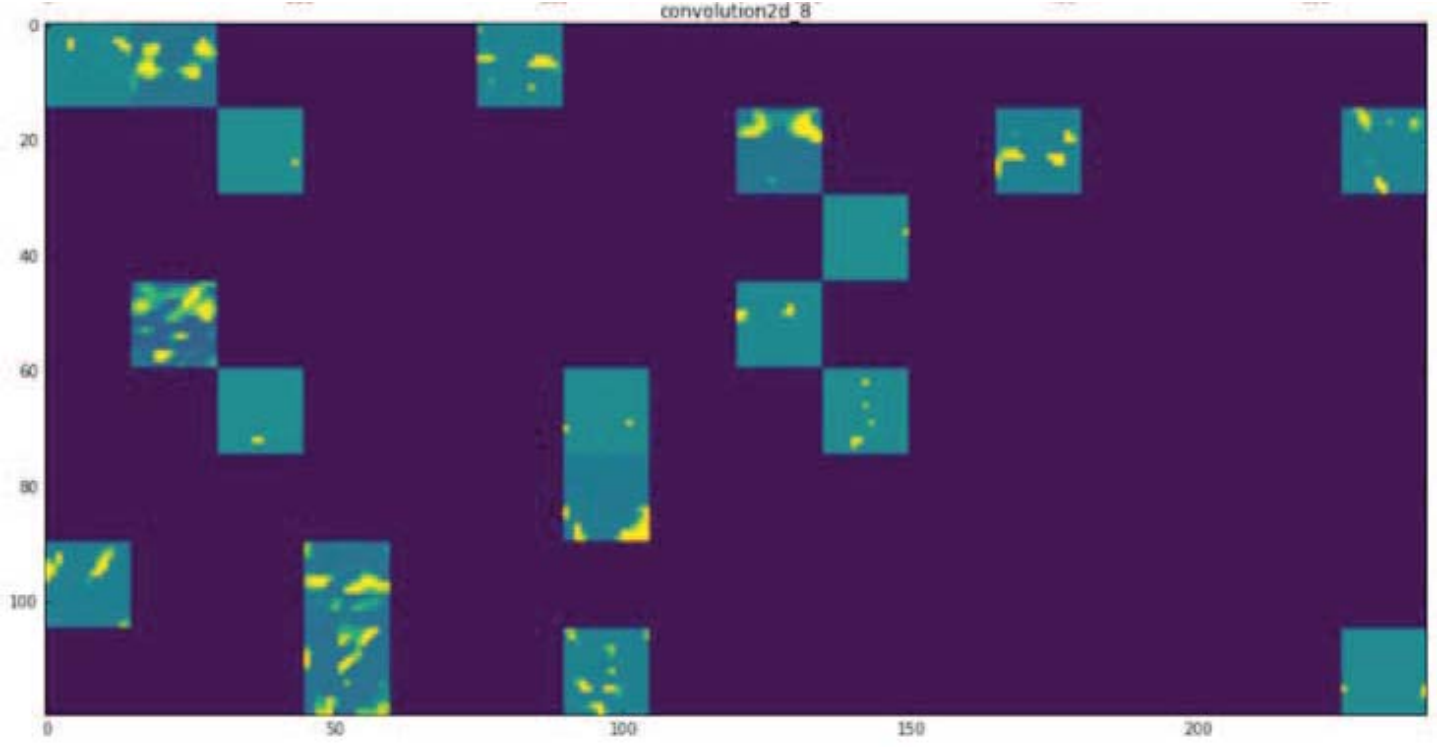
برای این تصویر ورودی




convolution2d_5
0
50
100
150
200
250
0
500
1000
1500
2000


convolution2d_6
0
50
100
150
200
250
0
200
400
600
800
1000


convolution2d_7
0
50
100
150
200
250
0
100
200
300
400
500


convolution2d_8
0
20
40
60
80
100
0
50
100
150
200

# روش‌های مصورسازی: مصورسازی فعالیت‌های میانی

نکات

نکات قابل توجه:

- **لایه‌ی اول به‌عنوان گرادیه‌ای از انواع آشکارسازهای لبه عمل می‌کند.**
  در این مرحله، فعالیت‌ها تقریباً همه‌ی اطلاعات موجود در تصویر اولیه را حفظ می‌کنند.

- **هرچه بالاتر می‌رویم، فعالیت‌ها به‌طور افزایشی انتزاعی‌تر می‌شوند و به‌طور بصری کمتر تفسیرپذیرند.**
  فعالیت شروع به کدگذاری مفاهیم سطح بالاتر مانند «گوش گربه» و «چشم گربه» می‌کنند.
  بازنمایی‌های بالاتر به‌طور افزایشی اطلاعات کمتری در مورد محتوای بصری تصویر حمل می‌کنند،
  بلکه اطلاعات بیشتری مرتبط با کلاس آن تصویر حمل می‌کنند.

- **خلوت شدن فعالیت‌ها با عمق لایه افزایش می‌یابند.**
  در اولین لایه، همه‌ی فیلترها به‌وسیله‌ی تصویر ورودی فعال می‌شوند.
  اما در لایه‌های بعدی، تعداد بیشتری از فیلترها خالی می‌شوند.
  (یعنی: الگوی کد شده توسط آن فیلتر در تصویر ورودی یافت نمی‌شود.)

# روش‌های مصورسازی: مصورسازی فعالیت‌های میانی

## نتایج

یک مشخصه‌ی سراسری مهم بازنمایی‌های یادگیری شده توسط شبکه‌های عصبی عمیق را مشاهده کردیم:
**ویژگی‌های استخراج شده توسط یک لایه، متناسب با عمق آن لایه، انتزاعی‌تر می‌شوند.**

فعالیت‌های لایه‌های بالاتر حامل اطلاعات کمتر و کمتری در مورد ورودی خاص مشاهده شده هستند.
فعالیت‌های لایه‌های بالاتر حامل اطلاعات بیشتر و بیشتری در مورد تارگت (کلاس) ورودی خاص هستند.

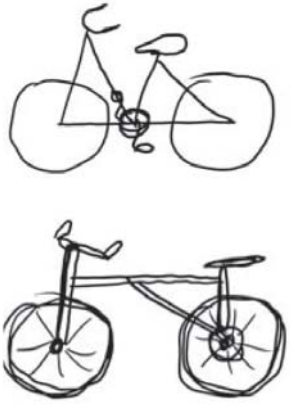
یک شبکه‌ی عصبی عمیق، به‌صورت مؤثر به‌عنوان یک
خط لوله‌ی تقطیر اطلاعات
(information distillation pipeline)
عمل می‌کند.
داده‌های خام (مثلاً تصاویر RGB) وارد آن می‌شوند
و به‌طور مکرر تبدیل می‌شوند تا اطلاعات نامربوط (مثل ظاهر بصری خاص تصویر) از آن فیلتر شود
و اطلاعات مفید بزرگ‌نمایی و تصفیه می‌شود (مثل کلاس تصویر).

# روش‌های مصورسازی: مصورسازی فعالیت‌های میانی

مقایسه با ادراک بصری انسان از جهان

یک دوچرخه‌ی شماتیک
باید شبیه این باشد.

تلاش‌ها برای رسم
یک دوچرخه از حافظه



عملکرد خط لوله‌ی تقطیر اطلاعات قابل مقایسه با شیوه‌ی ادراک بصری انسان‌ها از جهان است:
یک انسان پس از مشاهده‌ی یک صحنه به‌مدت چند ثانیه،
می‌تواند به‌خاطر بیاورد که کدام اشیای انتزاعی در آن صحنه موجود بوده‌اند (دوچرخه، درخت، ...)
اما نمی‌تواند ظاهر خاص (جزئیات) این اشیا را به‌خاطر بیاورد.

مغز ما یاد گرفته است که ورودی‌های بصری خود را کاملاً انتزاعی کند
(تا آن را به مفاهیم بصری سطح بالا تبدیل کند و همزمان جزئیات بصری نامربوط را فیلتر و حذف کند)
و باعث می‌شود یادآوری کیفیت چیزهای اطراف ما بسیار دشوار شود.

## روش‌های مصورسازی: مصورسازی فیلترهای شبکه

VISUALIZING CONVNET FILTERS

روش ساده‌ی دیگر برای معاینه‌ی فیلترهای یادگیری شده توسط شبکه‌های عصبی کانوولوشنال:
**نمایش الگوی بصری‌ای که هر فیلتر باید به آن پاسخ دهد.**

این کار را می‌توان با جستجوی افزایش گرادیان در فضای ورودی
با هدف ماکزیمم‌سازی پاسخ به یک فیلتر خاص انجام داد:
از یک تصویر ورودی خالی شروع می‌کنیم،
تصویر ورودی حاصل، آن است که فیلتر انتخاب شده ماکزیمم پاسخ را به آن بدهد.

لازم است یک تابع اتلاف بسازیم که مقدار یک فیلتر داده شده در یک لایه‌ی کانوولوشنی داده شده را ماکزیمم کند؛
و سپس از الگوریتم کاهش گرادیان تصادفی استفاده کنیم تا
مقادیر تصویر ورودی را به‌گونه‌ای تنظیم کند که این مقدار فعالیت را ماکزیمم کند.

# روش‌های مصورسازی: مصورسازی فیلترهای شبکه

## تعریف تابع اتلاف

یک تابع اتلاف برای برای فعالیت فیلتر صفر در لایه‌ی `block3_conv1` از شبکه‌ی VGG16
که بر روی ImageNet پیش‌آموزش دیده است.

## Defining the loss tensor for filter visualization

```
from keras.applications import VGG16
from keras import backend as K

model = VGG16(weights='imagenet',
              include_top=False)

layer_name = 'block3_conv1'
filter_index = 0

layer_output = model.get_layer(layer_name).output
loss = K.mean(layer_output[:, :, :, filter_index])
```

# روش‌های مصورسازی: مصورسازی فیلترهای شبکه

## محاسبه‌ی گرادیان اتلاف نسبت به ورودی

برای پیاده‌سازی الگوریتم کاهش گرادیان،
به گرادیان تابع اتلاف بر حسب ورودی مدل نیاز داریم.
برای این کار، از تابع `gradients` که در ماژول `backend` در Keras موجود است استفاده می‌کنیم.

### Obtaining the gradient of the loss with regard to the input

```
grads = K.gradients(loss, model.input)[0]
```

**The call to gradients returns a list of tensors (of size 1 in this case). Hence, you keep only the first element —which is a tensor.**

# روش‌های مصورسازی: مصورسازی فیلترهای شبکه

## ترفند نرمال‌سازی گرادیان

یک ترفند غیربدیهی برای کمک به پیشروی هموار فرآیند کاهش گرادیان،
نرمال‌سازی تانسور از طریق تقسیم آن بر نُرم L2 آن (جذر میانگین مربعات مقادیر موجود در تانسور) است.
* این کار تضمین می‌کند که بزرگی به‌روزرسانی‌ها تصویر ورودی همیشه در بازه‌ی ثابتی بماند.

### Gradient-normalization trick

```
grads /= (K.sqrt(K.mean(K.square(grads))) + 1e-5)
```

**Add 1e-5 before dividing to avoid accidentally dividing by 0.**

# روش‌های مصورسازی: مصورسازی فیلترهای شبکه

## واکشی مقادیر خروجی با توجه به مقادیر ورودی

به روشی برای محاسبه‌ی مقدار تانسور اتلاف و تانسور گرادیان آن با توجه به تصویر ورودی نیاز داریم.

یک تابع backend در Keras می‌نویسیم:
تابع `iterate` تابعی است که یک تانسور Numpy (لیستی از تانسورها با اندازه‌ی ۱) را دریافت می‌کند و لیستی از دو تانسور Numpy (مقداراتلاف و مقدار گرادیان آن) را برمی‌گرداند.

### Fetching Numpy output values given Numpy input values

```
iterate = K.function([model.input], [loss, grads])

import numpy as np
loss_value, grads_value = iterate([np.zeros((1, 150, 150, 3))])
```

# روش‌های مصورسازی: مصورسازی فیلترهای شبکه

## ماکزیمم‌سازی محلی با استفاده از کاهش گرادیان تصادفی

در این مرحله، یک حلقه‌ی پایتون برای انجام کاهش گرادیان تصادفی می‌نویسیم:

### Loss maximization via stochastic gradient descent

```
Starts from a gray image with some noise
input_img_data = np.random.random((1, 150, 150, 3)) * 20 + 128.

step = 1.  Magnitude of each gradient update
for i in range(40):
    loss_value, grads_value = iterate([input_img_data])
    Computes the loss value and gradient value

    input_img_data += grads_value * step
                      Adjusts the input image in the direction that maximizes the loss
```

Runs gradient ascent for 40 steps

# روش‌های مصورسازی: مصورسازی فیلترهای شبکه

## تبدیل تانسور به تصویر

تانسور تصویر حاصل، یک تانسور ممیز شناور با شکل (1, 150, 150, 3) است.
ممکن است مقادیر آن اعداد صحیح بازه‌ی $[0,255]$ نباشد.
پس باید این تانسور را پس‌پردازش کنیم تا به یک تصویر قابل نمایش تبدیل شود.

## Utility function to convert a tensor into a valid image

```
def deprocess_image(x):
    x -= x.mean()
    x /= (x.std() + 1e-5)
    x *= 0.1

    x += 0.5
    x = np.clip(x, 0, 1)

    x *= 255
    x = np.clip(x, 0, 255).astype('uint8')
    return x
```

Normalizes the tensor: centers on 0, ensures that std is 0.1

Clips to [0, 1]

Converts to an RGB array

# روش‌های مصورسازی: مصورسازی فیلترهای شبکه

تابع کامل برای دریافت نام لایه + اندیس فیلتر و تولید تصویر الگوی ماکزیمم‌کننده‌ی آن فیلتر

## Function to generate filter visualizations

Builds a loss function that maximizes the activation of the nth filter of the layer under consideration

```
def generate_pattern(layer_name, filter_index, size=150):
    layer_output = model.get_layer(layer_name).output
    loss = K.mean(layer_output[:, :, :, filter_index])

    grads = K.gradients(loss, model.input)[0]

    grads /= (K.sqrt(K.mean(K.square(grads))) + 1e-5)

    iterate = K.function([model.input], [loss, grads])

    input_img_data = np.random.random((1, size, size, 3)) * 20 + 128.

    step = 1.
    for i in range(40):
        loss_value, grads_value = iterate([input_img_data])
        input_img_data += grads_value * step

    img = input_img_data[0]
    return deprocess_image(img)
```

Computes the gradient of the input picture with regard to this loss

Normalization trick: normalizes the gradient

Returns the loss and grads given the input picture

Starts from a gray image with some noise

Runs gradient ascent for 40 steps

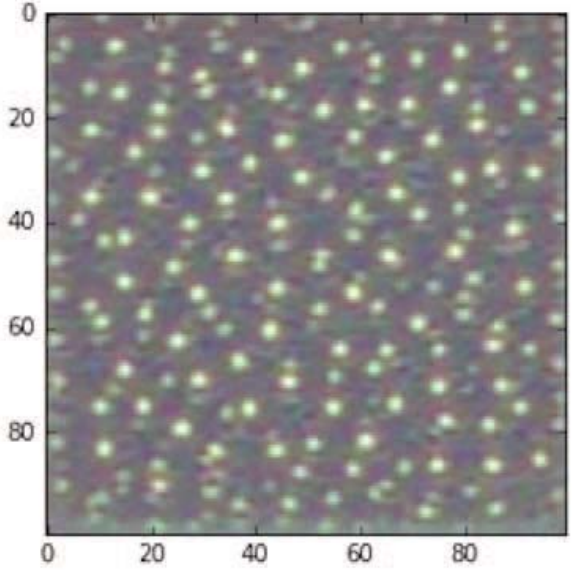
# روش‌های مصورسازی: مصورسازی فیلترهای شبکه

مثال

```
>>> plt.imshow(generate_pattern('block3_conv1', 0))
```

الگویی که کانال صفر در لایه‌ی
`block3_conv1`
ماکزیمم پاسخ را به آن می‌دهد.

به‌نظر می‌رسد فیلتر صفر در این لایه،
مسئول الگوی
polka-dot pattern
است.



# روش‌های مصورسازی: مصورسازی فیلترهای شبکه

تولید یک توری از همه‌ی الگوهای پاسخ فیلتر در یک لایه

برای سادگی، فقط ۶۴ فیلتر اول در هر لایه و اولین لایه‌ی هر بلوک کانوولوشنی را بررسی می‌کنیم.

## Generating a grid of all filter response patterns in a layer

```
layer_name = 'block1_conv1'
size = 64
margin = 5

results = np.zeros((8 * size + 7 * margin, 8 * size + 7 * margin, 3))

for i in range(8):
    for j in range(8):
        filter_img = generate_pattern(layer_name, i + (j * 8), size=size)
        horizontal_start = i * size + i * margin
        horizontal_end = horizontal_start + size
        vertical_start = j * size + j * margin
        vertical_end = vertical_start + size
        results[horizontal_start: horizontal_end,
                vertical_start: vertical_end, :] = filter_img

plt.figure(figsize=(20, 20))
plt.imshow(results)
```

Empty (black) image to store results

Iterates over the rows of the results grid

Iterates over the columns of the results grid

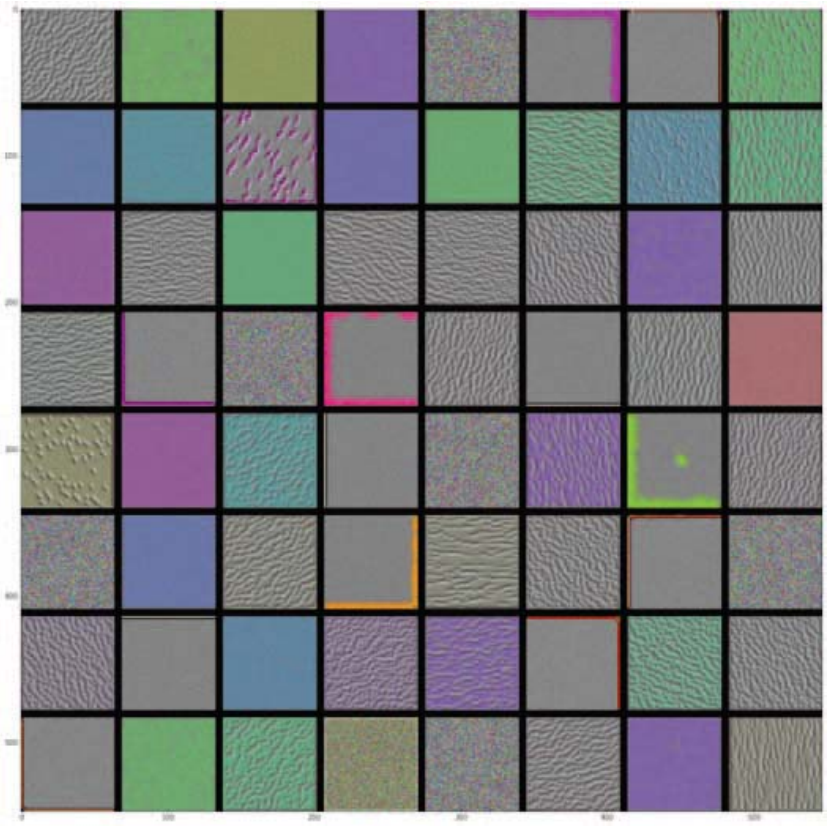
Generates the pattern for filter i + (j * 8) in layer_name

Puts the result in the square (i, j) of the results grid

Displays the results grid

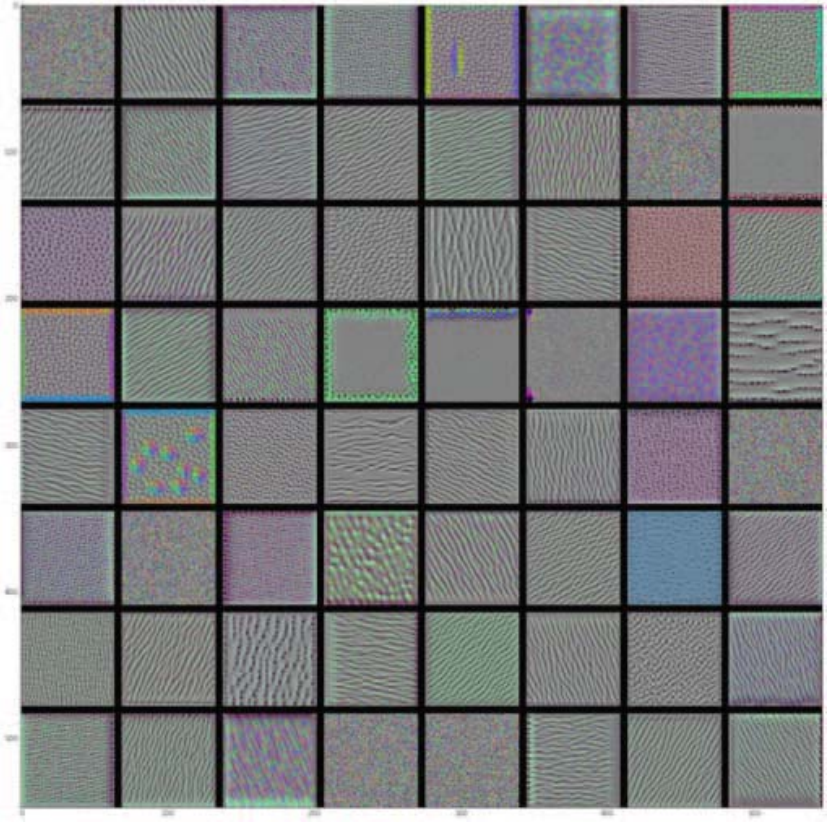
# روش‌های مصورسازی: مصورسازی فیلترهای شبکه

الگوهای فیلتر برای لایه‌ی block1_conv1



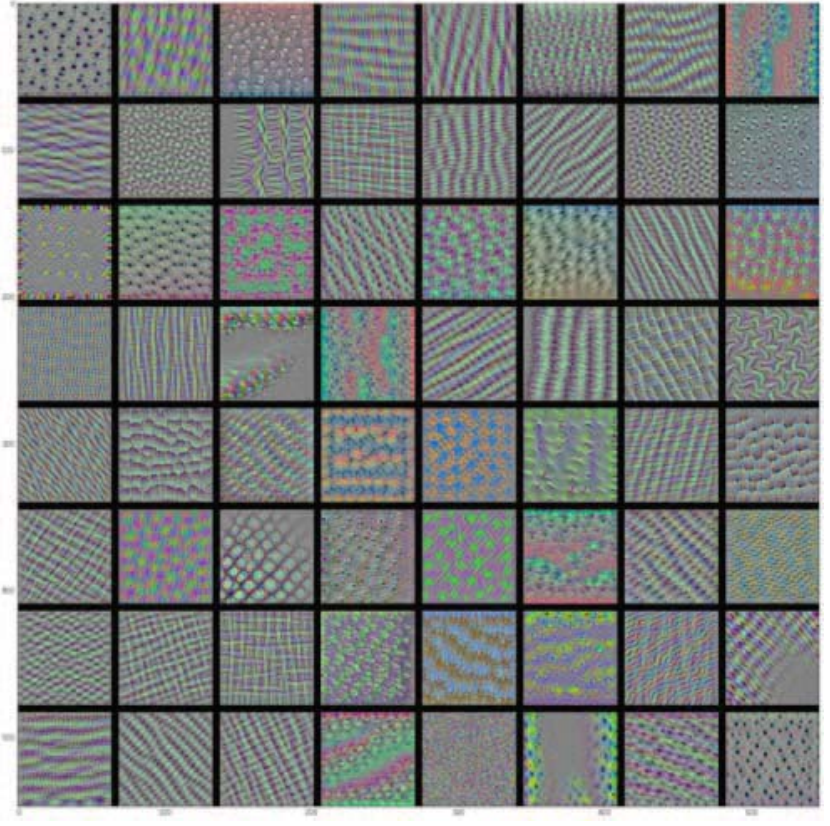
# روش‌های مصورسازی: مصورسازی فیلترهای شبکه

الگوهای فیلتر برای لایه‌ی block2_conv1



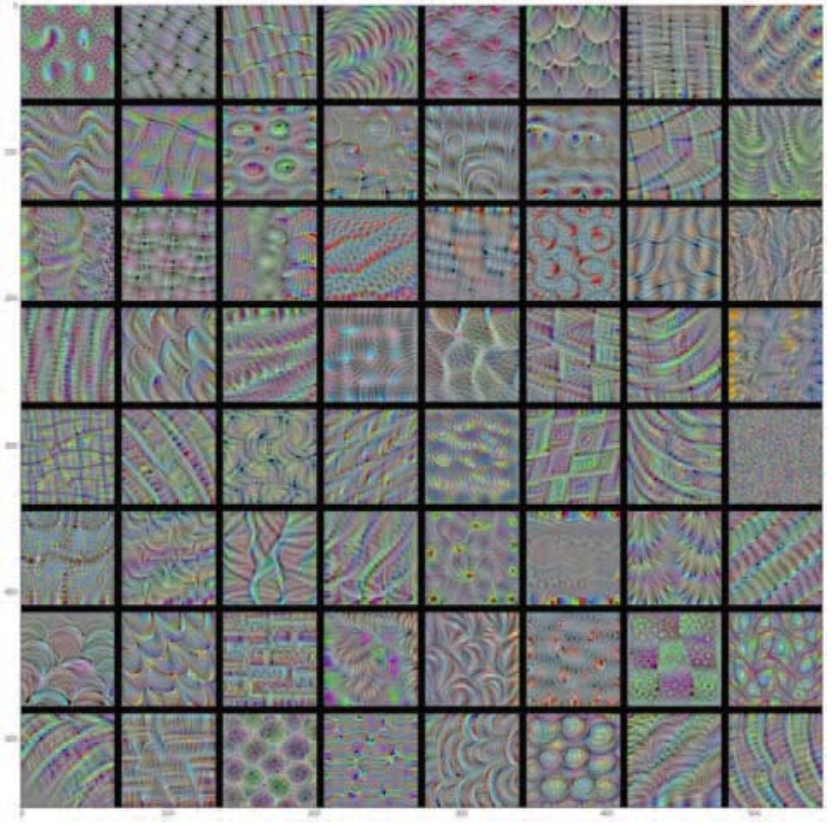
# روش‌های مصورسازی: مصورسازی فیلترهای شبکه

الگوهای فیلتر برای لایه‌ی block3_conv1



# روش‌های مصورسازی: مصورسازی فیلترهای شبکه

الگوهای فیلتر برای لایه‌ی block4_conv1



# روش‌های مصورسازی: مصورسازی فیلترهای شبکه

نکات

مصورسازی فیلترها، به ما چیزهای زیادی در مورد اینکه لایه‌های convnet دنیا را چگونه می‌بینند، می‌گوید.

هر لایه در یک convnet گردایه‌ای از فیلترها را یاد می‌گیرد؛
به‌گونه‌ای که ورودی‌های آنها را می‌توان به‌صورت ترکیبی از آن فیلترها بیان کرد.
(مشابه تبدیل فوریه که سیگنال ورودی را به مجموعه‌ای از توابع سینوسی تجزیه می‌کند.)

فیلترها در این بانک‌های فیلتر convnet، به‌صورت افزایشی پیچیده و پالایش‌شده می‌شوند:

- **فیلترهای مربوط به لایه‌ی اول** در مدل (block1_conv1) به‌کدگذاری لبه‌های جهت‌دار ساده و رنگ‌ها می‌پردازند (در برخی موارد، لبه‌های رنگی).
- **فیلترهای مربوط به بلوک دوم** در مدل (block2_conv1) به‌کدگذاری بافت‌های ساده‌ی مرکب از لبه‌ها و رنگ‌ها می‌پردازند.
- **فیلترهای مربوط به لایه‌های بالاتر** شروع به شبیه‌سازی بافت‌های یافت شده در تصاویر طبیعی (مانند پرها، چشم‌ها، برگ‌ها و ...) می‌کنند.

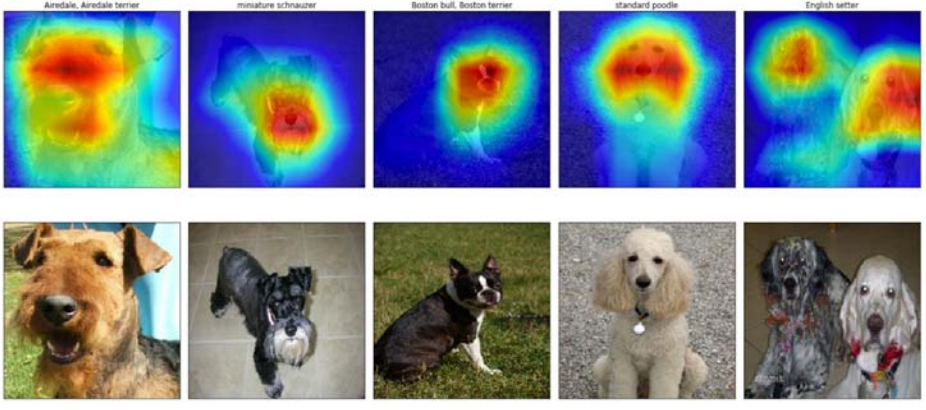
# روش‌های مصورسازی: مصورسازی نقشه‌های گرمایی فعالیت کلاس

## VISUALIZING HEATMAPS OF CLASS ACTIVATION

تکنیک **نقشه‌های گرمایی**:
به ما کمک می‌کند تا متوجه شویم کدام‌یک از بخش‌های یک تصویر داده شده
باعث می‌شود تا یک شبکه‌ی عصبی کانوولوشنال تصمیم‌گیری نهایی برای طبقه‌بندی را انجام دهد.

**کاربردهای نقشه‌های گرمایی**

- **اشکال‌زدایی** از فرآیند تصمیم‌گیری یک convnet (به‌ویژه در مورد خطای طبقه‌بندی)
- **مکان‌یابی** اشیای خاص در یک تصویر



# روش‌های مصورسازی: مصورسازی نقشه‌های گرمایی فعالیت کلاس

مصورسازی نقشه‌ی فعالیت کلاسی

## CLASS ACTIVATION MAP (CAM) VISUALIZATION

رده‌ی عمومی تکنیک‌ها: **مصورسازی نقشه‌ی فعالیت کلاسی (CAM)**:
تهیه‌ی نقشه‌های گرمایی فعالیت کلاسی بر روی تصاویر ورودی.

**نقشه‌ی فعالیت کلاسی**
*Class Activation Map (CAM)*

یک توری دو-بعدی از امتیازهای مرتبط با یک کلاس خروجی خاص،
که برای هر مکان در هر تصویر ورودی محاسبه می‌شود
و نشان می‌دهد که هر مکان نسبت به کلاس مورد بررسی چه‌قدر اهمیت دارد.

برای نمونه:
با داشتن تصویری که به یک شبکه‌ی بازشناسی سگ/گربه وارد می‌شود،
مصورسازی CAM این امکان را می‌دهد که
یک نقشه‌ی گرمایی برای کلاس «گربه» تولید کنیم: نشان‌دهنده‌ی اینکه بخش‌های گربه-مانند تصویر کدام‌ها هستند و
یک نقشه‌ی گرمایی برای کلاس «سگ» تولید کنیم: نشان‌دهنده‌ی اینکه بخش‌های سگ-مانند تصویر کدام‌ها هستند.

# روش‌های مصورسازی: مصورسازی نقشه‌های گرمایی فعالیت کلاس

## مصورسازی نقشه‌ی فعالیت کلاسی: پیاده‌سازی با الگوریتم Grad-CAM

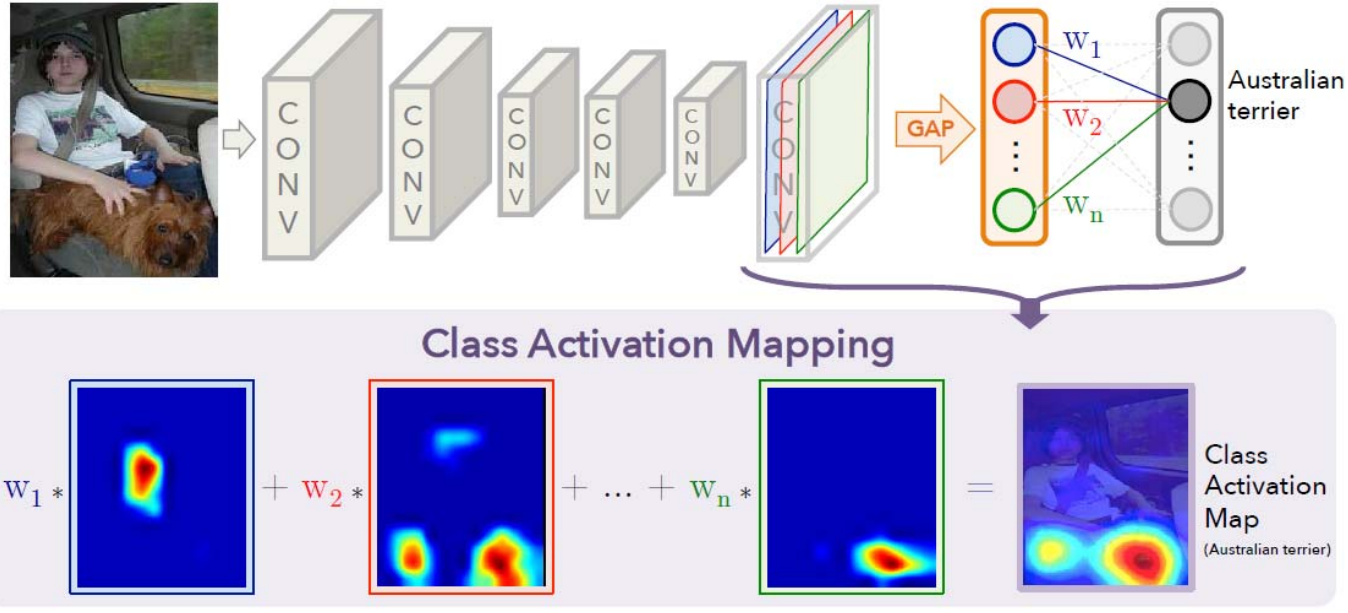
## CLASS ACTIVATION MAP (CAM) VISUALIZATION: GRAD-CAM ALGORITHM

پیاده‌سازی خاص مصورسازی نقشه‌ی فعالیت کلاسی: الگوریتم Grad-CAM:

نقشه‌ی ویژگی خروجی یک لایه‌ی کانوولوشنی برای یک تصویر ورودی دریافت می‌شود،
هر کانال در آن نقشه‌ی ویژگی به‌وسیله‌ی گرادیان آن کلاس نسبت به آن کانال وزن‌دهی می‌شود.

یک نقشه‌ی فضایی، این مسئله که
«تصویر ورودی، کانال‌های مختلف را با چه شدتی فعال می‌کند؟»
را بر اساس زیر وزن‌دهی می‌کند:
«اهمیت هر کانال در رابطه با هر کلاس چه‌قدر است»
که در نتیجه یک نقشه‌ی فضایی از این کمیت حاصل می‌شود:
«تصویر ورودی، آن کلاس را با چه شدتی فعال می‌کند؟»

**Ref:** R.R. Selvaraju, et al. "Grad-cam: Visual explanations from deep networks via gradient-based localization," https://arxiv.org/abs/1610.02391, (2016)


CONV
CONV
CONV
CONV
CONV
CONV
GAP
$w_1$
$w_2$
$w_n$
Australian terrier
Class Activation Mapping
$w_1$ *
+ $w_2$ *
+ … + $w_n$ *
=
Class Activation Map
(Australian terrier)

# روش‌های مصورسازی: مصورسازی نقشه‌های گرمایی فعالیت کلاس

مثال

مثال پیاده‌سازی مصورسازی نقشه‌ی فعالیت کلاسی: الگوریتم Grad-CAM:

با استفاده از شبکه‌ی پیش‌آموزش دیده‌ی VGG16

## Loading the VGG16 network with pretrained weights

```
from keras.applications.vgg16 import VGG16

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

**Note that you include the densely connected classifier on top; in all previous cases, you discarded it.**

# روش‌های مصورسازی: مصورسازی نقشه‌های گرمایی فعالیت کلاس

مثال: تصویر آزمایشی ورودی

تصویر دو فیل آفریقایی

# روش‌های مصورسازی: مصورسازی نقشه‌های گرمایی فعالیت کلاس

مثال: پیش‌پردازش تصویر آزمایشی ورودی

باید تصویر به تصویری تبدیل شود که برای مدل VGG16 قابل خواندن باشد:
تصویر را بارگذاری می‌کنیم، اندازه‌ی آن را به ۲۲۴×۲۲۴ تبدیل می‌کنیم
و آن را به یک تانسور ممیزشناور ۳۲ بیتی Numpy تبدیل می‌کنیم.

## Preprocessing an input image for VGG16

```
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input, decode_predictions
import numpy as np

img_path = '/Users/fchollet/Downloads/creative_commons_elephant.jpg'

img = image.load_img(img_path, target_size=(224, 224))

x = image.img_to_array(img)

x = np.expand_dims(x, axis=0)

x = preprocess_input(x)
```

Local path to the target image

Python Imaging Library (PIL) image of size 224 × 224

float32 Numpy array of shape (224, 224, 3)

Adds a dimension to transform the array into a batch of size (1, 224, 224, 3)

Preprocesses the batch (this does channel-wise color normalization)

# روش‌های مصورسازی: مصورسازی نقشه‌های گرمایی فعالیت کلاس

مثال: اجرای شبکه روی تصویر آزمایشی

شبکه‌ی پیش‌آموزش دیده را روی تصویر آزمایشی اجرا می‌کنیم
و بردار پیش‌بینی آن را در قالب قابل خواندن برای انسان کدگذاری می‌کنیم:

```
>>> preds = model.predict(x)
>>> print('Predicted:', decode_predictions(preds, top=3)[0])
Predicted:', [(u'n02504458', u'African_elephant', 0.92546833),
(u'n01871265', u'tusker', 0.070257246),
(u'n02504013', u'Indian_elephant', 0.0042589349)]
```

سه کلاس برتر پیش‌بینی شده برای این تصویر:
فیل آفریقایی، فیل عاج‌دار، فیل هندی

- African elephant (with 92.5% probability)
- Tusker (with 7% probability)
- Indian elephant (with 0.4% probability)

این شبکه، این تصویر را حاوی تعداد نامعینی فیل آفریقایی بازشناسی کرده است.
درایه‌ای در بردار پیش‌بینی که دارای ماکزیمم فعال‌شدگی است، آن است که متناظر با «فیل آفریقایی» است (در اندیس ۳۸۶).

```
>>> np.argmax(preds[0])
386
```

# روش‌های مصورسازی: مصورسازی نقشه‌های گرمایی فعالیت کلاس

مثال: اجرای فرآیند Grad-CAM

برای مصورسازی بخش‌هایی از تصویر که بیشترین شباهت را به «فیل آفریقایی» دارند، یک فرآیند Grad-CAM را اجرا می‌کنیم.

## Setting up the Grad-CAM algorithm

```
african_e66lephant_output = model.output[:, 386]

last_conv_layer = model.get_layer('block5_conv3')

grads = K.gradients(african_elephant_output, last_conv_layer.output)[0]

pooled_grads = K.mean(grads, axis=(0, 1, 2))

iterate = K.function([model.input],
                     [pooled_grads, last_conv_layer.output[0]])

pooled_grads_value, conv_layer_output_value = iterate([x])

for i in range(512):
    conv_layer_output_value[:, :, i] *= pooled_grads_value[i]

heatmap = np.mean(conv_layer_output_value, axis=-1)
```

"African elephant" entry in the prediction vector

Output feature map of the block5_conv3 layer, the last convolutional layer in VGG16

Gradient of the "African elephant" class with regard to the output feature map of block5_conv3

Vector of shape (512,), where each entry is the mean intensity of the gradient over a specific feature-map channel

Lets you access the values of the quantities you just defined: pooled_grads and the output feature map of block5_conv3, given a sample image

Values of these two quantities, as Numpy arrays, given the sample image of two elephants

Multiplies each channel in the feature-map array by "how important this channel is" with regard to the "elephant" class

The channel-wise mean of the resulting feature map is the heatmap of the class activation.

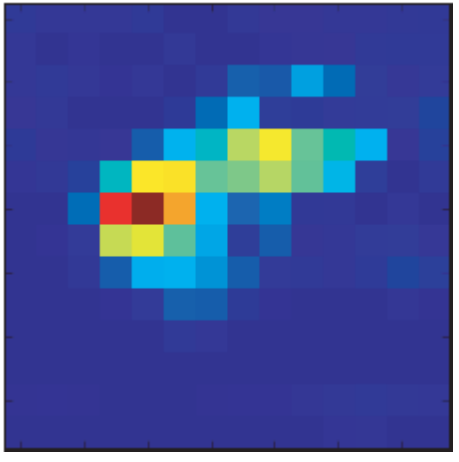
# روش‌های مصورسازی: مصورسازی نقشه‌های گرمایی فعالیت کلاس

مثال: پس‌پردازش نقشه‌ی گرمایی

به‌منظور مصورسازی، نقشه‌ی گرمایی را بین صفر و یک نرمال‌سازی می‌کنیم.

## Heatmap post-processing

```
heatmap = np.maximum(heatmap, 0)
heatmap /= np.max(heatmap)
plt.matshow(heatmap)
```



# روش‌های مصورسازی: مصورسازی نقشه‌های گرمایی فعالیت کلاس

مثال: قرار دادن نقشه‌ی گرمایی روی تصویر اصلی

از OpenCV برای تولید تصویری که نقشه‌ی گرمایی را روی تصویر اصلی قرار می‌دهد، استفاده می‌کنیم.

## Superimposing the heatmap with the original picture

```
import cv2

img = cv2.imread(img_path)

heatmap = cv2.resize(heatmap, (img.shape[1], img.shape[0]))

heatmap = np.uint8(255 * heatmap)

heatmap = cv2.applyColorMap(heatmap, cv2.COLORMAP_JET)

superimposed_img = heatmap * 0.4 + img

cv2.imwrite('/Users/fchollet/Downloads/elephant_cam.jpg', superimposed_img)
```

Uses cv2 to load the original image

Resizes the heatmap to be the same size as the original image

Converts the heatmap to RGB

Applies the heatmap to the original image

0.4 here is a heatmap intensity factor.

Saves the image to disk

# روش‌های مصورسازی: مصورسازی نقشه‌های گرمایی فعالیت کلاس

مثال: قرار دادن نقشه‌ی گرمایی روی تصویر اصلی: نتیجه

# روش‌های مصورسازی: مصورسازی نقشه‌های گرمایی فعالیت کلاس

مثال: قرار دادن نقشه‌ی گرمایی روی تصویر اصلی: نتیجه

این تکنیک مصورسازی به دو پرسش مهم پاسخ می‌دهد:

- **چرا شبکه فکر کرده است این تصویر حاوی «فیل آفریقایی» است؟**
- **فیل آفریقایی در چه مکانی از این تصویر واقع شده است؟**

به‌طور خاص، جالب توجه است که گوش‌های فیل کوچک قوی‌ترین فعال‌شدگی را دارند؛
احتمالاً این دلیلی است که شبکه بر اساس آن می‌تواند بگوید فرق فیل آفریقایی با فیل هندی در چیست.

# یادگیری عمیق برای بینایی کامپیوتری

## خلاصه

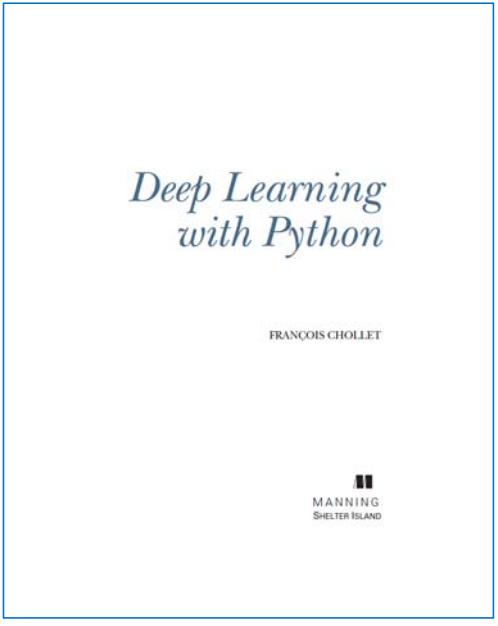
- **Convnets** are the best tool for attacking visual-classification problems.
- **Convnets** work by learning a hierarchy of modular patterns and concepts to represent the visual world.
- The representations they learn are easy to inspect—convnets are the opposite of black boxes!
- You're now capable of training your own convnet from scratch to solve an image-classification problem.
- You understand how to use visual data augmentation to fight overfitting.
- You know how to use a pretrained convnet to do feature extraction and fine-tuning.
- You can generate visualizations of the filters learned by your convnets, as well as heatmaps of class activity

یادگیری عمیق برای بینایی کامپیوتری

# ۵

# منابع

# منبع اصلی




François Chollet,
**Deep Learning with Python,**
Manning Publications, 2018.

**Chapter 5**