یادگیری عمیق

دکتر امین گلزاری اسکوئی

a.golzari@azaruniv.ac.ir

a.golzari@tabrizu.ac.ir

https://github.com/Amin-Golzari-Oskouei

دانشگاه شهید مدنی آذربایجان

پاییز ۱۴۰۱

فصل ۵

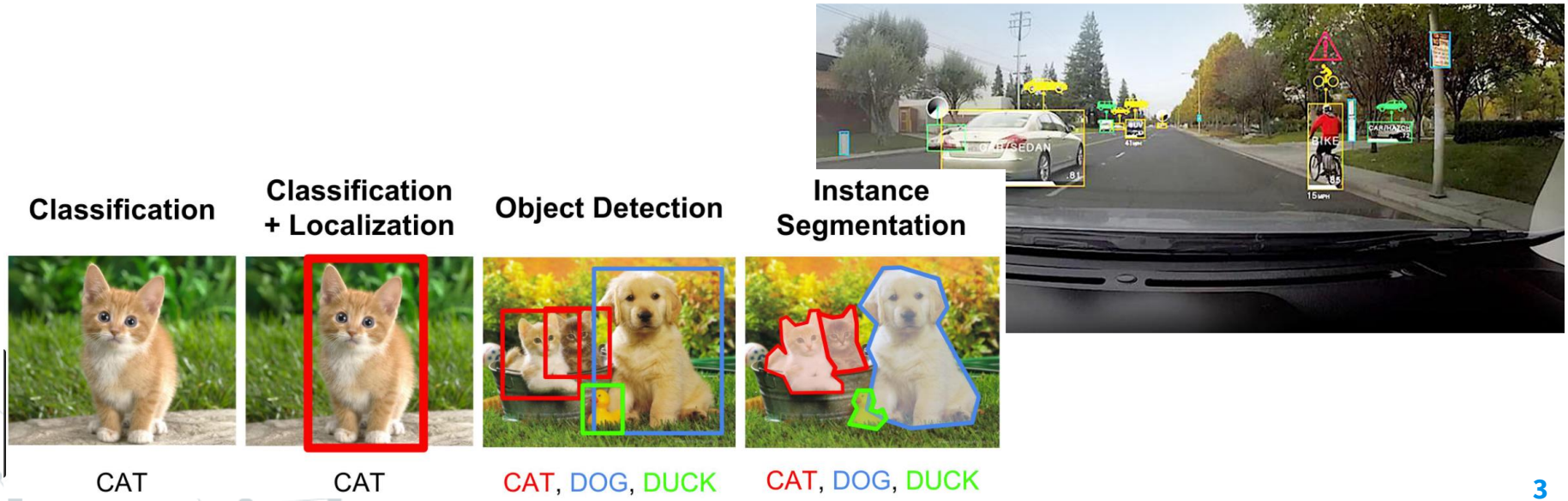
یادگیری عمیق برای بینایی ماشین

مطالب این فصل

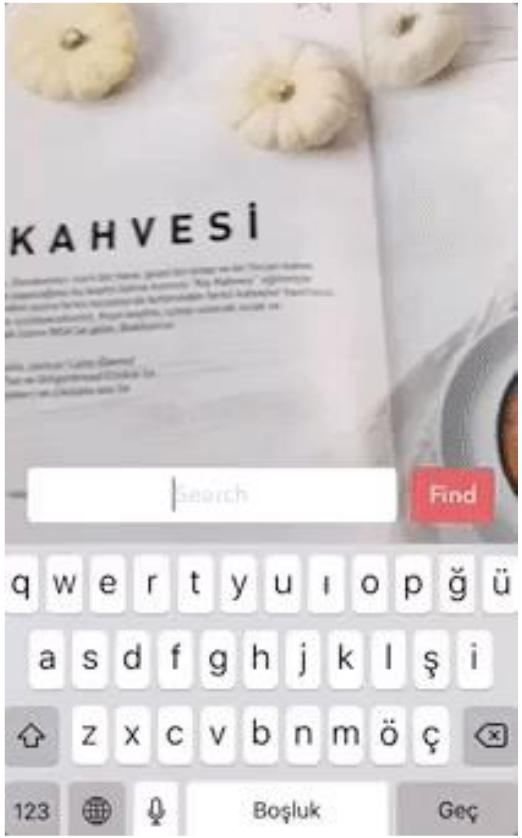
- درک شبکه‌های عصبی کانولوشنی
- استفاده از داده‌افزایی برای کاهش بیش‌برازش
- استفاده از شبکه کانولوشنی از پیش آموزش دیده برای استخراج ویژگی
- تنظیم دقیق شبکه کانولوشنی از پیش آموزش دیده
- مصورسازی آموخته‌های شبکه‌های کانولوشنی و نحوه تصمیم‌گیری آن‌ها در انجام دسته‌بندی

بینایی ماشین

برای خودکار کردن کارهایی که سیستم بینایی انسان می تواند انجام دهد، مانند طبقه بندی تصویر، تشخیص اشیا، تشخیص چهره، شرح تصاویر، تشخیص حالت چهره و غیره.



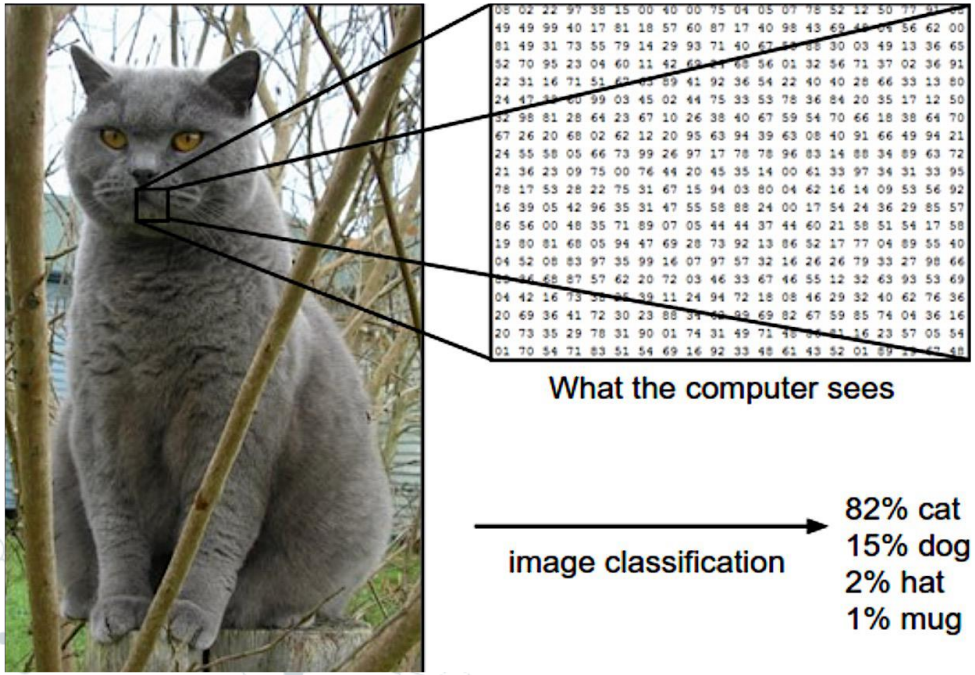
بینایی ماشین



ctrl+f in real world

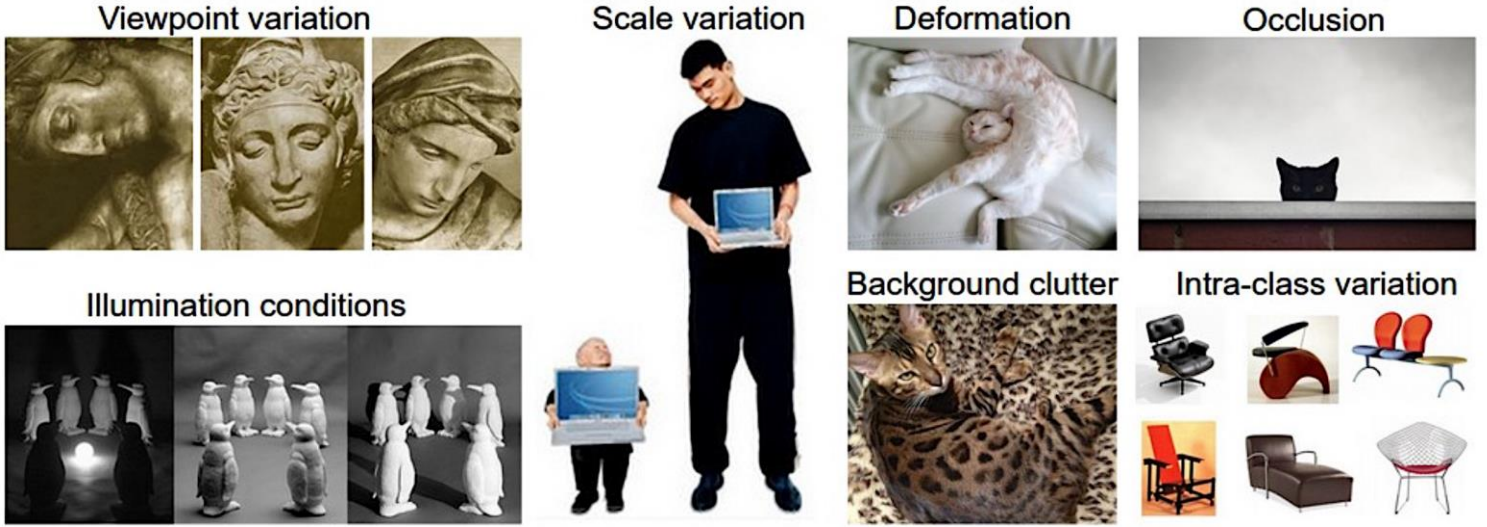
طبقه‌بندی تصاویر

تمرکز ما در اینجا طبقه‌بندی تصاویر به عنوان یک کار اصلی در بینایی ماشین است



http://cs231n.github.io/classification/

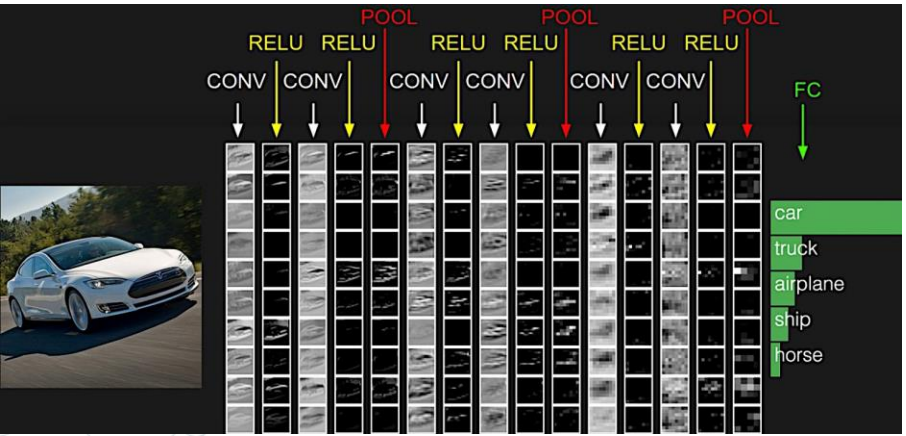
چالش‌ها

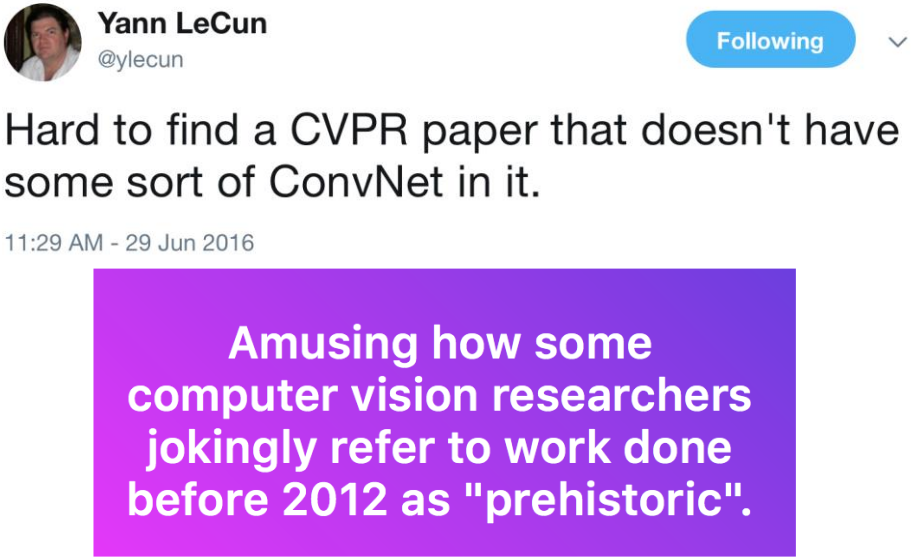


http://cs231n.github.io/classification/

ابزار

در این فصل شبکه‌های عصبی کانولوشنی معرفی می‌شوند. این شبکه‌ها نوعی مدل یادگیری عمیق هستند که عموماً در بینایی ماشین به کار می‌روند.





http://cs231n.github.io/convolutional-networks/

مقدمه‌ای بر شبکه‌های کانولوشنی

- ✓ حال می‌خواهیم شبکه‌های کانولوشنی و دلیل موفقیت آن‌ها در بینایی ماشین را بررسی کنیم؛ اما در ابتدا، نگاه عملی به یک مثال شبکه کانولوشنی ساده خواهیم داشت. در این مثال از شبکه کانولوشنی برای دسته‌بندی ارقام MNIST استفاده می‌شود؛ کاری که در فصل 2 با شبکه تمام متصل (Dense) انجام دادیم (دقت آزمایش ما 97.8 درصد بود).

- ✓ با یک شبکه کانولوشنی ساده حتی برروی داده تست به دقت 99 درصد نیز می‌رسیم.

معماری شبکه

```
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'))
```

این شبکه پشته‌ای از لایه‌های conv2D و MaxPooling2D است.

موضوع مهم، شکل تنسورهای ورودی به شبکه کانولوشنی است (که حاوی بعد دسته نیست). در این مورد، پیکربندی شبکه کانولوشنی به صورتی خواهد بود که ورودی‌هایی را با اندازه (28,28,1) پردازش کند که اندازه تصاویر MNIST است.

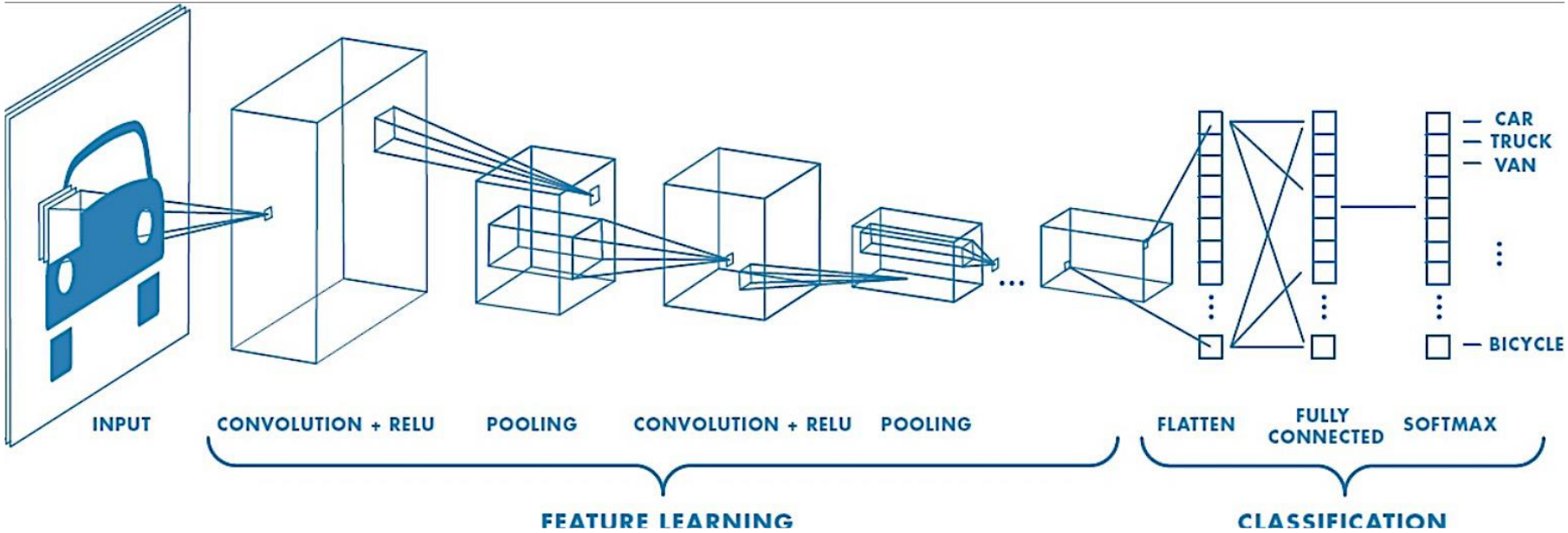
معماری شبکه

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

مرحله بعد وارد کردن آخرین تنسور خروجی (با ابعاد (3, 3, 64)) به دسته‌بند تمام متصل است که قبلاً با آن آشنا شدید: پشته‌ای از لایه‌های تمام متصل. این دسته‌بندها، ورودی خود را به صورت بردار تک بعدی است، دریافت می‌کنند، در حالی که خروجی فعلی تنسور سه بعدی است. ابتدا باید خروجی‌های سه بعدی را به شکل مسطح درآورده و یک بعدی کنیم و سپس چند لایه Denseدر ادامه اضافه کنیم.

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \rightarrow \begin{bmatrix} a \\ b \\ c \\ \vdots \\ h \\ i \end{bmatrix}$$

معماری شبکه



https://www.mathworks.com/discovery/convolutional-neural-network.html

معماری شبکه

```
>>> 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
```

No parameter

$576 \times 64 + 64$

$64 \times 10 + 10$

آموزش شبکه

```
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)
```

```
>>> test_loss, test_acc = model.evaluate(test_images, test_labels)
>>> test_acc
0.9908000000000001
```

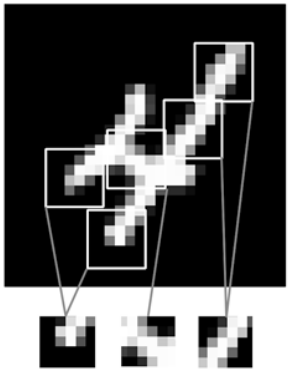
GOING TIME

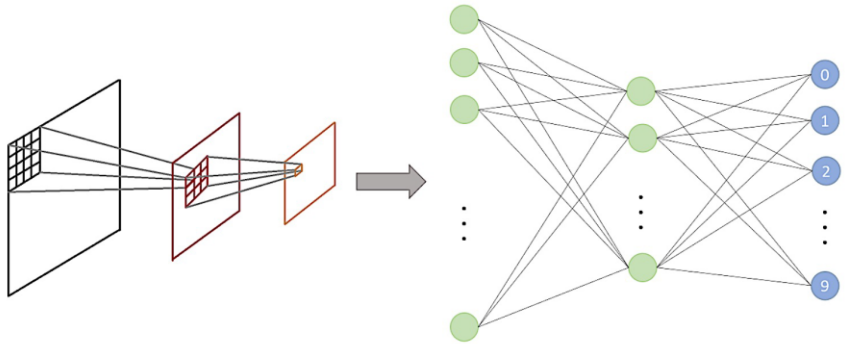
5.1-introduction-to-convnets

عملیات کانولوشن

تفاوت عمده بین لایه تمام متصل و لایه کانولوشن بدین شرح است:

لایه‌های تمام متصل، الگوهای سراسری را در فضای ویژگی ورودی خودشان یاد می‌گیرند (به عنوان مثال، برای یک رقم MNIST، الگوها شامل تمامی پیکسل‌ها هستند)، در حالی که لایه‌های کانولوشن الگوهای محلی را یاد می‌گیرند. در مورد تصاویر، الگوها در پنجره‌های دو بعدی کوچک ورودی‌ها یافت می‌شوند.





عملیات کانولوشن

- الگوهایی که شبکه‌های کانولوشنی یاد می‌گیرند، مستقل از مکان هستند
  - بعد از یاد گرفتن الگو در گوشه سمت راست تصویر، کانولوشن می‌تواند آن را در هر جایی شناسایی کند: به عنوان مثال، در گوشه بالای سمت چپ. در حالی که شبکه تمام متصل در صورت ظاهر شدن الگو در مکان جدید، باید آن را از نو یاد بگیرد.
  - بدین ترتیب، شبکه‌های کانولوشنی برای پردازش تصاویر کارآمد هستند (چرا که جهان مصور عمدتاً مستقل از مکان است):
  - شبکه‌های کانولوشنی برای یاد گرفتن بازنمایی‌هایی که قدرت تعمیم دارند، به نمونه‌های کمتری احتیاج دارند.

Cat

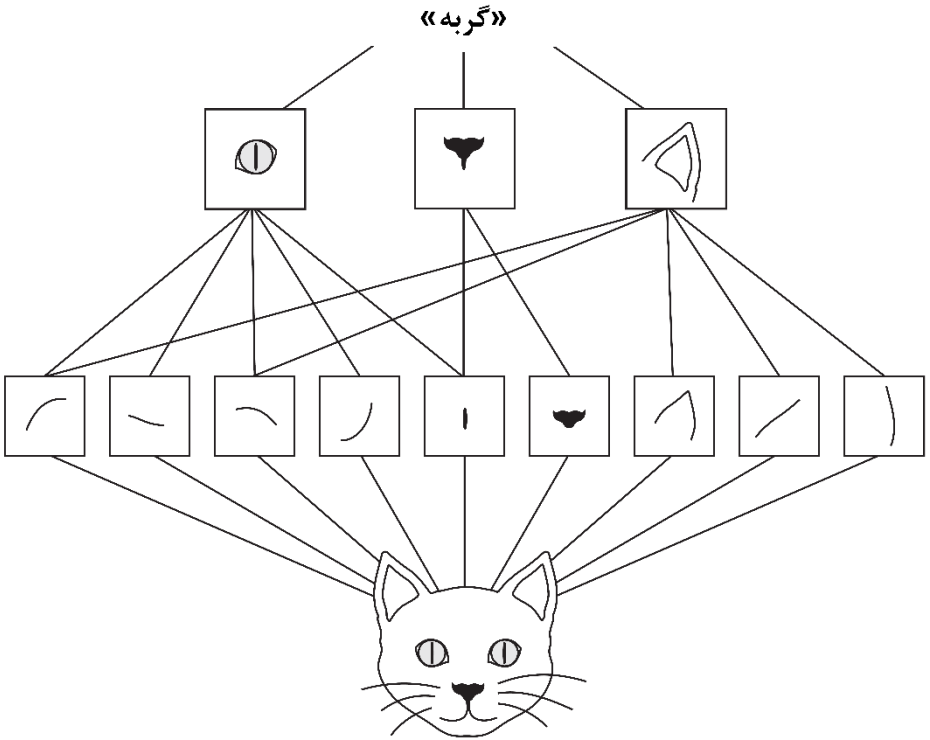
Cat

عملیات کانولوشن

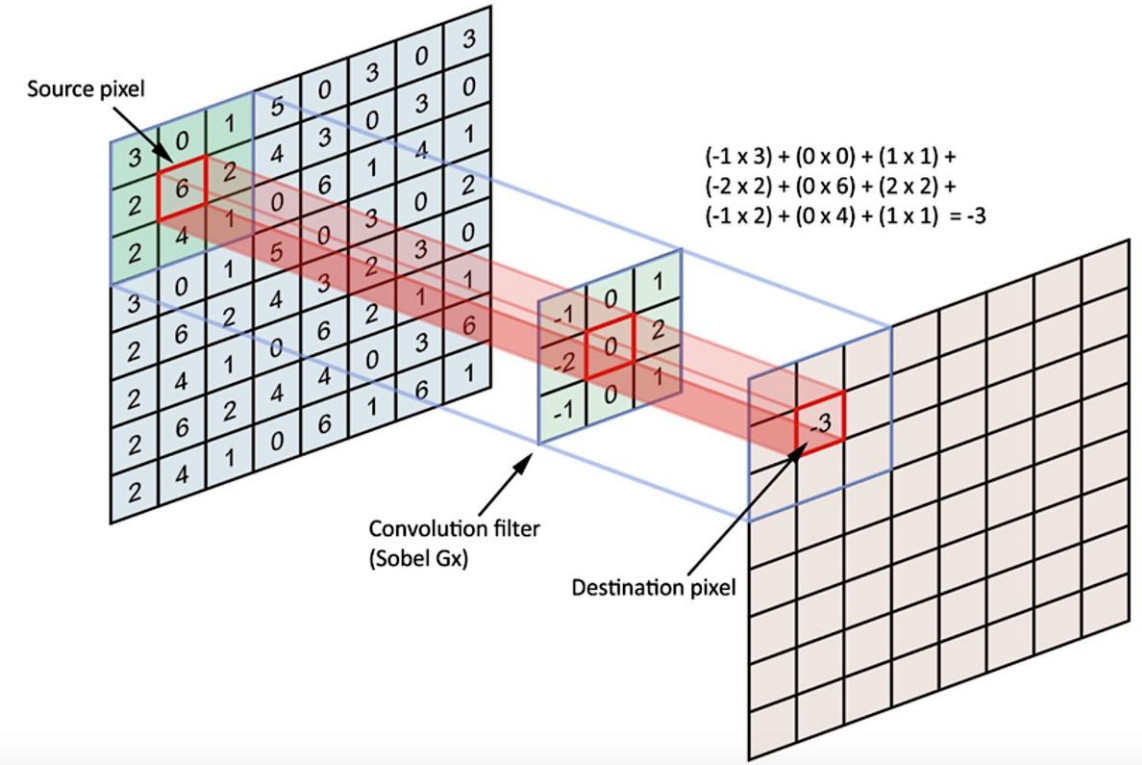
شبکه‌های کانولوشنی سلسله مراتب فضایی الگوها را یاد می‌گیرند

- اولین لایه کانولوشن، الگوهای محلی کوچک مانند لبه‌ها را یاد خواهد گرفت، دومین لایه کانولوشن الگوهای بزرگ‌تر متشکل از ویژگی‌های لایه‌های اول را یاد خواهد گرفت و الی آخر.
- این ویژگی به شبکه‌های کانولوشنی کمک می‌کند که به طور کارآمدی مفاهیم پیچیده و مصورسازی انتزاعی را یاد بگیرند (چرا که جهان مصور عمدتاً دارای سلسله مراتب فضایی است).

عملیات کانولوشن



عملیات کانولوشن



https://github.com/pjreddie/cnn-primer

عملیات کانولوشن

7	2	3	3	8
4	5	3	8	4
3	3	2	8	4
2	8	7	2	7
5	4	4	5	4

*

1	0	-1
1	0	-1
1	0	-1

7x1+4x1+3x1+
2x0+5x0+3x0+
3x-1+3x-1+2x-1
= 6

=

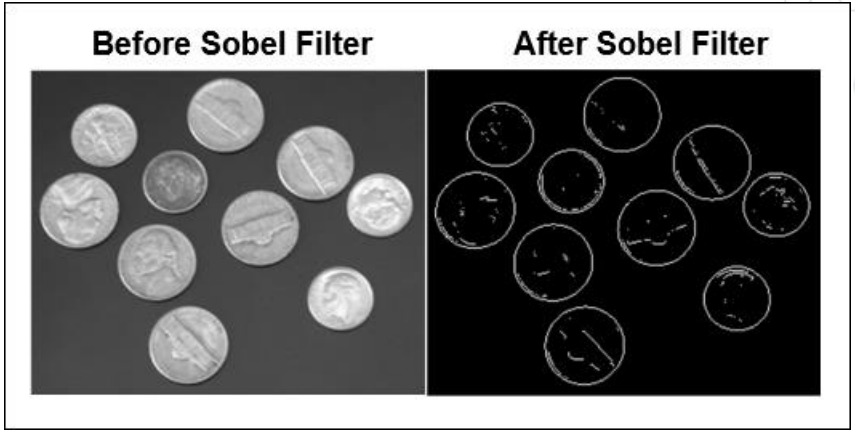
6		

https://aishack.in/tutorials/image-convolution-examples/

عملیات کانولوشن

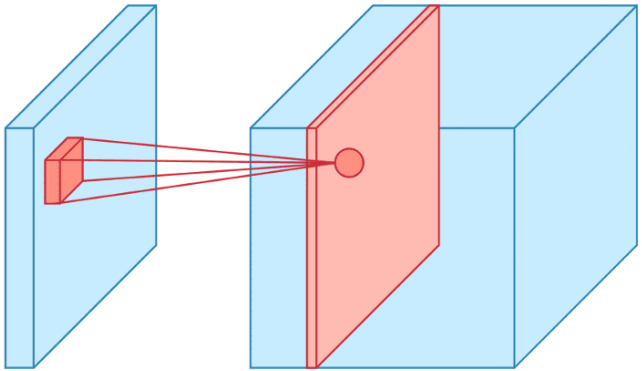
$*$

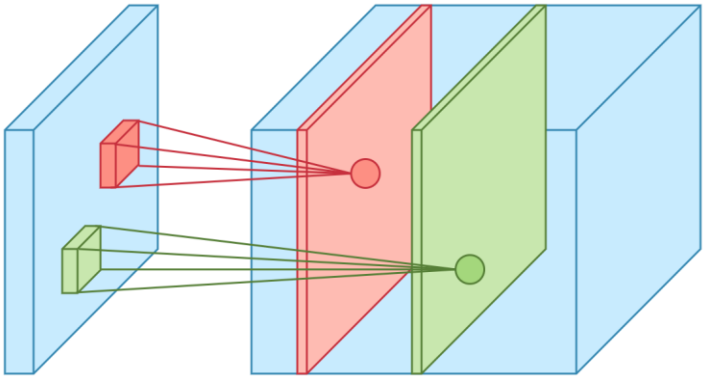
1	0	-1
2	0	-2
1	0	-1



https://aishack.in/tutorials/image-convolution-examples/

عملیات کانولوشن

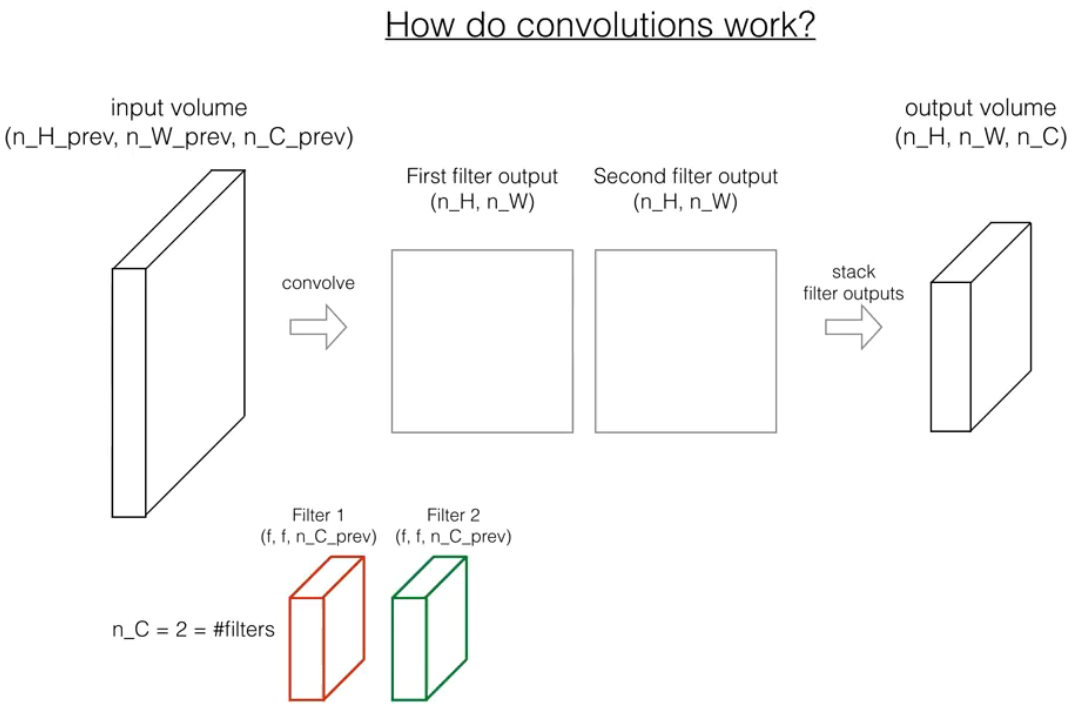




Note that this is a highly time consuming process, which can be controlled by parallelization using GPUs.

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

عملیات کانولوشن



عملیات کانولوشن

```
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'))
```

Windows are typically 3x3 or 5x5

```
_________________________________________________________________
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
=================================================================
```

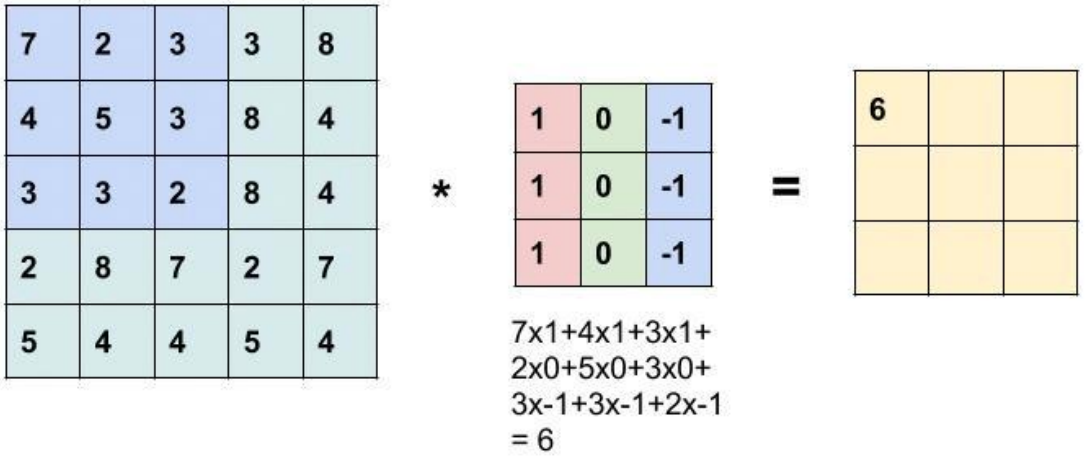
Why 26x26? The border effect.

$32 \times 3 \times 3 \times 1 + 32$

$64 \times 3 \times 3 \times 32 + 64$

$64 \times 3 \times 3 \times 64 + 64$

عملیات کانولوشن



عملیات کانولوشن

چنانچه می‌خواهید ابعاد فضایی نقشه ویژگی خروجی با ابعاد فضایی ورودی یکسان باشند، می‌توانید از پدینگ استفاده کنید.

0	0	0	0	0	0	0
0	60	113	56	139	85	0
0	73	121	54	84	128	0
0	131	99	70	129	127	0
0	80	57	115	69	134	0
0	104	126	123	95	130	0
0	0	0	0	0	0	0

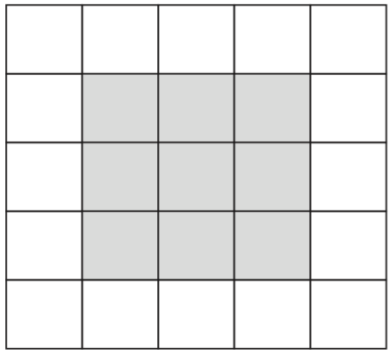
Kernel

0	-1	0
-1	5	-1
0	-1	0

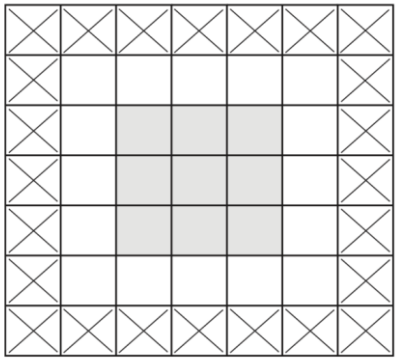
114				

عملیات کانولوشن

در لایه‌های Conv2D، پدینگ از طریق آرگومان padding قابل پیکربندی است که دو مقدار می‌گیرد: "valid" که به معنای عدم پدینگ است (فقط از مکان‌های معتبر پنجره استفاده خواهد شد)؛ و "same"، یعنی هدف، داشتن خروجی‌هایی با عرض و ارتفاع یکسان با ورودی است. پیش فرض آرگومان پدینگ valid است

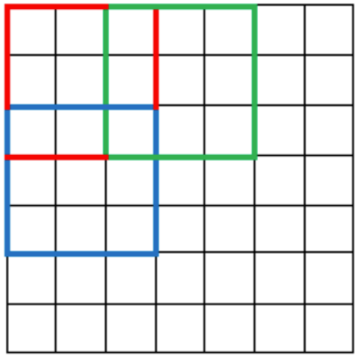


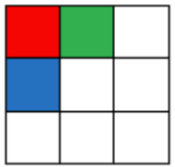
Padding= "valid"



Padding= "same"

عملیات کانولوشن





- فاصله بین دو پنجره متوالی، پارامتر کانولوشن است که گام (stride) نامیده می‌شود. و پیش فرض آن 1 است.
- استفاده از گام 2 بدین معنی است که عرض و ارتفاع نقشه ویژگی با ضریب 2 کاهش اندازه شده‌اند.
- با اینکه کانولوشن‌های گام‌دار در برخی از انواع مدل‌ها می‌توانند مفید واقع شوند، در عمل به ندرت مورد استفاده قرار می‌گیرند.
- برای کاهش اندازه نقشه‌های ویژگی، به جای گام از عملیات ادغام بیشینه استفاده می‌کنیم.

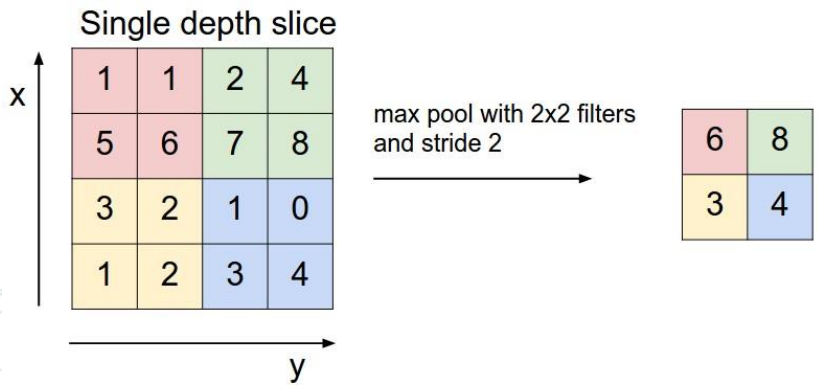
https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/

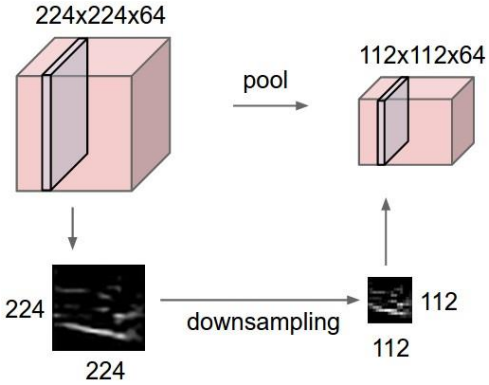
عملیات ادغام بیشینه (max-pooling)

ادغام بیشینه عبارت است از استخراج پنجره‌ها از نقشه‌های ویژگی ورودی و یافتن مقدار بیشینه هر یک از پنجره‌ها به عنوان خروجی.

از نظر مفهومی ادغام بیشینه به کانولوشن شباهت دارد، با این تفاوت که به جای تبدیل بخش‌های محلی از طریق تبدیل خطی (کرنل کانولوشن)، بخش‌ها از طریق عملیات تنسور max تبدیل می‌شوند.

ادغام بیشینه اغلب با پنجره‌های 2 × 2 و گام 2 انجام می‌شود تا نقشه‌های ویژگی را با ضریب 2 کاهش اندازه دهد. تعداد کانال‌ها تغییر نمی‌کند.





عملیات ادغام بیشینه (max-pooling)

```
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
```

چرا نقشه‌های ویژگی را باید به این صورت کاهش اندازه داد؟

- ✓ برای یادگرفتن سلسله مراتب فضایی ویژگی‌ها مساعد نیست. پنجره‌های 3 × 3 در لایه سوم تنها حاوی اطلاعات پنجره‌های 7 × 7 در ورودی‌های اولیه خواهند بود. الگوهای سطح بالایی که شبکه کانولوشنی یاد گرفته است با توجه به ورودی‌های اولیه، بسیار کوچک خواهند بود که ممکن است برای یاد گرفتن دسته‌بندی ارقام کافی نباشند (شناسایی یک رقم فقط با نگاه کردن به آن از طریق پنجره‌هایی با اندازه 7 × 7 پیکسل). ویژگی‌های آخرین لایه کانولوشن باید حاوی اطلاعاتی در مورد کل ورودی باشند.

عملیات ادغام بیشینه (max-pooling)

```
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
```

چرا نقشه‌های ویژگی را باید به این صورت کاهش اندازه داد؟

- ✓ نقشه ویژگی نهایی دارای $22 \times 22 \times 64 = 30976$ درایه برای هر نمونه خواهد بود که بسیار عظیم است. اگر بخواهید آن را مسطح کنید و به یک لایه تمام متصل با اندازه 512 وصل کنید، این لایه 15.8 میلیون پارامتر خواهد داشت. این مقدار برای چنین مدل کوچکی بیش از اندازه بزرگ است و به بیش‌برازش شدید منتهی خواهد شد.

عملیات ادغام بیشینه (max-pooling)

ادغام بیشینه تنها روش برای کاهش اندازه نیست. همانطور که مطلع هستید می‌توانید از گام‌ها در لایه کانولوشن قبلی استفاده کنید؛ همچنین می‌توانید به جای ادغام بیشینه از ادغام میانگین استفاده کنید که در آن، بخش ورودی محلی به جای مقدار بیشینه با مقدار میانگین هر کانال در بخش جایگزین می‌شود؛ اما ادغام بیشینه بهتر از هر کدام از این روش‌ها عمل می‌کند.

منطقی‌ترین راهبرد نمونه‌برداری این است که ابتدا نگاشت‌های متراکمی از ویژگی‌ها (از طریق کانولوشن‌های بدون گام) تولید شود و سپس فعال‌سازی بیشینه ویژگی‌ها در دسته‌های کوچک بررسی شود.

بررسی پنجره‌های ورودی (از طریق کانولوشن‌های گام‌دار) یا میانگین گرفتن از دسته‌های ورودی ممکن است به از دست دادن یا کم‌رنگ شدن اطلاعات ویژگی-حضور منجر شود.

مثال

آموزش مدل دسته‌بندی تصویر با استفاده از نمونه‌های بسیار کم، چالشی است که اگر به صورت حرفه‌ای بینایی ماشین کار کرده باشید، در عمل با آن مواجه شده‌اید. تعداد کمی نمونه، می‌تواند چند صد تا چند ده هزار تصویر را شامل شود. به عنوان یک مثال عملی، روی دسته‌بندی تصاویر به عنوان سگ یا گربه در مجموعه داده‌ای با 4 هزار تصویر گربه و سگ (هر کدام 2 هزار) تمرکز می‌کنیم. از 2000 تصویر برای آموزش، 1000 تصویر را برای اعتبارسنجی و 1000 تصویر را برای آزمایش استفاده خواهیم کرد.

جعبه ابزار

با آموزش ابتدایی شبکه کانولوشنی روی 2000 نمونه آموزشی و بدون هیچ گونه تنظیم، کار را شروع خواهیم کرد تا برای آنچه می‌توان به دست آورد یک حد پایین تعریف کرده باشیم.

- با این کار به دقت دسته‌بندی 71 درصد خواهیم رسید.

سپس داده‌افزایی را معرفی خواهیم کرد که راهبرد نیرومندی برای کاهش بیش‌برازش در بینایی ماشین است. با استفاده از داده‌افزایی، شبکه بهبود داده می‌شود.

- با این کار به دقت دسته‌بندی 82 درصد خواهیم رسید.

در این بخش، دو راهکار ضروری برای اعمال یادگیری عمیق در مجموعه داده‌های کوچک را مرور خواهیم کرد:

- استخراج ویژگی با شبکه از پیش آموزش دیده (که دقت را به 90 درصد تا 96 درصد خواهد رساند)
- تنظیم دقیق شبکه از پیش آموزش دیده (که دقت نهایی 97 درصد را رقم خواهد زد).

داده

مجموعه داده سگ‌ها- درمقابل-گربه‌ها که از آن استفاده خواهید کرد در پکیج کراس موجود نیست. گگل آن را به عنوان بخشی از رقابت بینایی ماشین در اواخر 2013، یعنی زمانی که شبکه‌های کانولوشنی شایع نبودند، در اختیار عموم قرار داد. می‌توانید مجموعه داده اصلی را از سایت گگل دانلود کنید www.kaggle.com/c/dogs-vs-cats/data .

اینکه برنده رقابت 2013 سگ‌ها-در مقابل-گربه‌های گگل شرکت‌کننده‌هایی بودند که از شبکه کانولوشنی استفاده کرده بودند، غافلگیر کننده نبود. بهترین دقت تا 95 درصد بود.

این مجموعه داده شامل 25 هزار تصویر سگ و گربه (12 هزار و پانصد برای هر کلاس) و 543 مگابایت (فشرده) است.

بعد از دانلود و باز کردن، یک مجموعه داده جدید با سه زیرمجموعه خواهیم ساخت: مجموعه آموزشی با 1000 نمونه از هر کلاس، اعتبارسنجی با 500 نمونه از هر کلاس و مجموعه آزمایش با 500 نمونه از هر کلاس.

ساخت شبکه

از آنجایی که در اینجا با تصاویر بزرگ‌تر و مسائل پیچیده‌تری سر و کار داریم، شبکه را نیز به همان نسبت بزرگ‌تر خواهیم کرد. در مثال قبلی یک شبکه کانولوشنی کوچک برای مجموعه داده MNIST ساختیم، بنابراین با چنین شبکه‌های کانولوشنی آشنا هستید. در این بخش از همان ساختار کلی مجدداً استفاده خواهیم کرد.

- این مرحله باعث تقویت ظرفیت شبکه شده
- اندازه نقشه‌های ویژگی خواهد کاست تا هنگام رسیدن به لایه Flatten، نقشه‌ها بیش از اندازه بزرگ نباشند

الگو کلی برای طراحی شبکه های کانالوشنی:

- عمق نقشه‌های ویژگی به طور تصاعدی در شبکه افزایش می‌یابد (از 32 به 128)، در حالی که اندازه آن‌ها کاهش می‌یابد (از 148 × 148 به 7 × 7). تقریباً در تمامی شبکه‌های کانولوشنی این الگو را خواهید دید.

پیش‌پردازش داده‌ها

گام‌ها:

1) فایل‌های تصاویر را بخوانید.

2) محتوای تصاویر JPEGرا به ماتریسی از پیکسل‌های RGB تبدیل کنید.

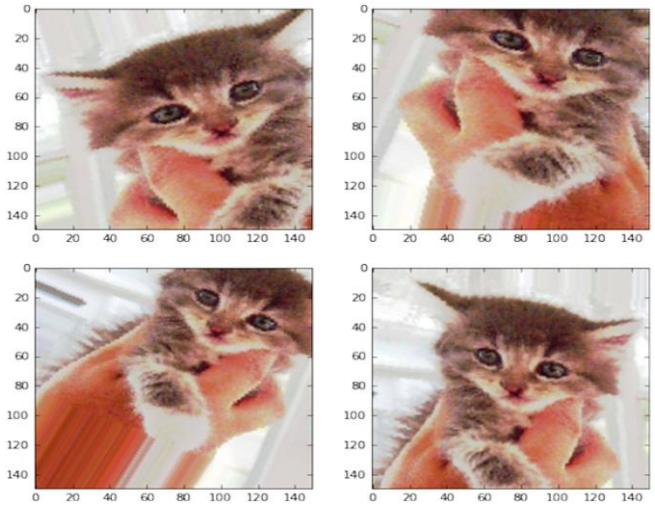
3) این ماتریس‌ها را به تنسورهای ممیز شناور تبدیل کنید.

4) مقادیر پیکسل (بین 0 و 255) را به بازه [1، 0] تغییر دهید (همانطور که می‌دانید شبکه‌های عصبی مدیریت مقادیر ورودی کوچک را ترجیح می‌دهند).

از کلاس ImageDataGenerator از کراس استفاده می کنیم.

داده افزایی

به دلیل داشتن نمونه‌های آموزشی اندک (2000)، بیش‌برازش، اولین دغدغه شما خواهد بود.



داده‌افزایی رویکرد تولید نمونه‌های آموزشی بیشتر از روی نمونه‌های آموزشی موجود است. در این روش، نمونه‌ها از طریق تعدادی تبدیل تصادفی افزایش می‌یابند و تصاویری شبه واقعی تولید می‌شود.

هدف از این کار، دوری جستن از نمایش دوباره یک تصویر برای مدل در هنگام آموزش است. با این کار، مدل با جوانب بیشتری مواجه شده و بهتر تعمیم می‌یابد.

داده افزایی

چند نمونه از گزینه‌های قابل دسترس هستند (برای نمونه‌های بیشتر به مستندات کراس مراجعه کنید):

- rotation_range: مقداری در بازه صفر تا 180 درجه است، دامنه‌ای که تصاویر در آن به طور تصادفی می‌چرخند.

- width_shift و height_shift: دامنه‌هایی هستند (به عنوان کسری از عرض یا ارتفاع کل) که در آن‌ها تصاویر به طور تصادفی به صورت عمودی یا افقی جابه‌جا می‌شوند.

- shear_range: برای اعمال تصادفی تبدیل‌های برشی به کار می‌رود.

- zoom_range: برای بزرگ‌نمایی تصادفی در داخل تصاویر به کار می‌رود.

داده افزایی

horizontal_flip: برای برگرداندن تصادفی نصف تصاویر به صورت افقی به کار می‌رود. زمانی کاربرد دارد که هیچ فرضی برای عدم تقارن افقی وجود ندارد (به عنوان مثال، تصاویر دنیای واقعی).

fill_mode: راهبردی برای پر کردن پیکسل‌هایی است که جدیداً به وجود آمده‌اند. علت ظهور این پیکسل‌ها چرخش یا جابه‌جایی عرض یا ارتفاع است.

```
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')
```

داده افزایی

- اگر شبکه جدیدی را با استفاده از این پیکربندی داده‌افزایی آموزش دهید، شبکه هرگز یک ورودی را دو بار نخواهد دید.

- با این وجود ورودی‌هایی که می‌بیند به شدت همبستگی دارند، چرا که همگی از چند تصویر اصلی مشتق شده‌اند.

  - در نتیجه شاید این راهکار برای خلاص شدن کامل از بیش‌برازش کافی نباشد.

  - برای مقابله بیشتر با بیش‌برازش، درست قبل از دسته‌بند تمام متصل، یک لایه حذف تصادفی به مدل اضافه خواهیم کرد.

GOING TIME

5.2-using-convnets-with-small-datasets

استفاده از شبکه کانولوشنی (از قبل) آموزش دیده

شبکه آموزش دیده، شبکه ذخیره شده‌ای است که قبلاً روی مجموعه داده‌ای بزرگ، به ویژه دسته‌بندی تصویر در مقیاس‌های بزرگ، آموزش دیده است.

در صورتی که مجموعه داده اصلی به اندازه کافی بزرگ و کلی باشد، سلسله مراتب ویژگی‌هایی که شبکه آموزش دیده آموخته است، می‌تواند به عنوان مدلی کلی از دنیای مصور، کارکرد مؤثری داشته باشد. بدین ترتیب، ویژگی‌های آن برای بسیاری از مسائل بینایی ماشین مفید واقع خواهد شد.

- حتی اگر مسائل جدید، حاوی کلاس‌هایی کاملاً متفاوت از مجموعه داده بزرگ اولیه باشند.

- به عنوان مثال، ممکن است شبکه‌ای را روی ایمیجنت آموزش داده باشید (که در آن کلاس‌ها اکثراً حیوانات و اشیای روزمره هستند) سپس این شبکه آموزش دیده را برای کاربردی کاملاً متفاوت مانند تشخیص مبلمان در تصاویر به کار ببرید.

- چنین **قابلیت انتقالی** برای ویژگی‌های یاد گرفته شده بین مسائل متفاوت، مزیت کلیدی یادگیری عمیق در مقایسه با بسیاری از رویکردهای قدیمی و یادگیری-سطحی بوده و یادگیری عمیق را برای مسائلی با مجموعه داده‌های کوچک کارآمد می‌سازد.

استفاده از شبکه کانولوشنی (از قبل) آموزش دیده

- می‌خواهیم یک شبکه کانولوشنی بزرگ (VGG16) را در نظر بگیریم که روی مجموعه داده ایمیجنت آموزش دیده است (1.4 میلیون تصویر برچسب دار و 1000 کلاس متفاوت).

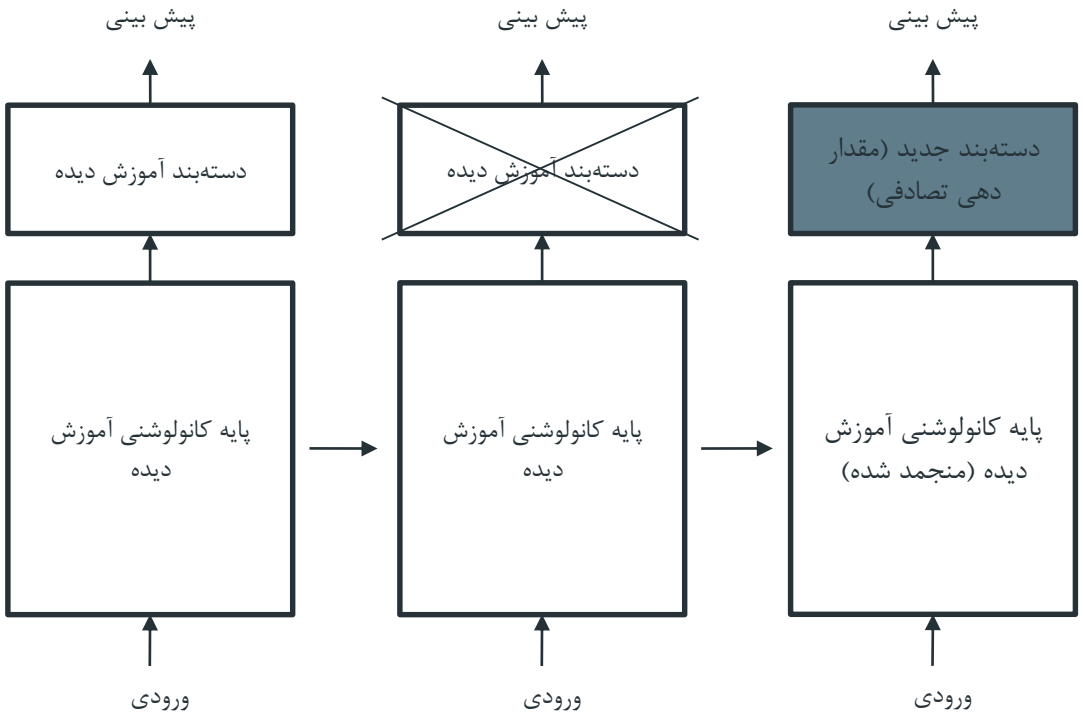
- این معماری ساده بوده و کاربرد زیادی در معماری شبکه کانولوشنی برای ایمیجنت دارد.

- معماری‌های جدیدتری مانند VGG، ResNet، Inception، Inception-ResNet، Xception و غیره نیز وجود دارد.

استفاده از شبکه کانولوشنی (از قبل) آموزش دیده

- دو روش برای استفاده از شبکه آموزش دیده وجود دارد:
- استخراج ویژگی
-
  - با داده‌افزایی
  - بدون داده‌افزایی
- تنظیم دقیق

استخراج ویژگی با استفاده از شبکه آموزش دیده



استخراج ویژگی با استفاده از شبکه آموزش دیده

چرا فقط پایه کانولوشن مجدداً مورد استفاده قرار می‌گیرد؟ آیا می‌توان از دسته‌بند تمام متصل نیز مجدداً استفاده کرد؟ عموماً، باید از چنین کاری دوری کرد؛

- چرا که بازنمایی‌هایی که پایه کانولوشنی یاد گرفته است کلی‌تر هستند و بنابراین قابلیت استفاده مجدد بیشتری دارند؛
- لایه‌هایی که زودتر در مدل قرار می‌گیرند، نقشه‌های ویژگی محلی و کلی‌تری را استخراج می‌کنند (مانند لبه‌های مصور، رنگ‌ها و بافت‌ها)، در حالی که لایه‌هایی که بالاتر قرار گرفته‌اند مفاهیم انتزاعی‌تری را استخراج می‌کنند (مانند «گوش گربه» یا «چشم سگ»)؛
- بنابراین، در صورتی که مجموعه داده جدید شما با مجموعه داده مدل آموزش دیده تفاوت‌های زیادی دارد، بهتر است برای استخراج ویژگی به جای استفاده از کل پایه کانولوشنی فقط از چند لایه اول مدل استفاده کنید.

مدل کانولوشنی VGG16

- اگر input_shape وارد نکنید، شبکه توانایی پردازش ورودی‌ها با هر اندازه‌ای را خواهد داشت.
  - طول و عرض تصویر نباید از 48 کوچک تر باشد.

- اگر include_top، Trueباشد، input_shape باید (3، 224، 224) باشد تا بردار صحیح برای لایه های متراکم ایجاد شود.

```
from keras.applications import VGG16

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

مدل کانولوشنی 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
```

لایه‌های تمام متصل

اجرای پایه کانولوشنی روی مجموعه داده، ذخیره کردن خروجی و سپس استفاده از این داده خروجی به عنوان ورودی دسته‌بند تمام متصل جدید

- اجرای این راه‌حل، سریع و ارزان است، چرا که تنها لازمه آن اجرای پایه کانولوشنی و فقط برای یک بار برای تمامی تصاویر ورودی است و این پایه کانولوشنی با فاصله زیاد پرهزینه‌ترین بخش این فرایند است؛
- اما در این روش نمی‌توانید از داده‌افزایی استفاده کنید.

گسترش دادن مدل موجود (conv_base) با اضافه کردن لایه‌های متراکم و اجرای همه چیز به صورت سر به سر روی نمونه‌های ورودی.

- در این روش می‌توانید از داده‌افزایی استفاده کنید؛ چرا که هر بار تصویری به مدل داده می‌شود از پایه کانولوشنی نیز گذر می‌کند.
- این روش بسیار پرهزینه‌تر از روش اول است.

استخراج ویژگی با استفاده از شبکه آموزش دیده و داده‌افزایی

قبل از کامپایل و آموزش مدل، منجمد کردن پایه کانولوشن از اهمیت شایانی برخوردار است:

- منجمد کردن یک لایه یا مجموعه‌ای از لایه‌ها به معنای ممانعت از به‌روزشدن وزن‌های آن‌ها در طول آموزش است.

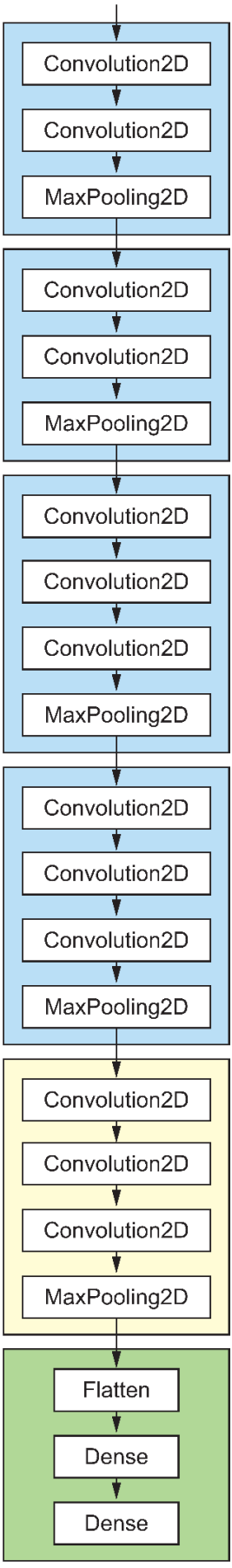
- اگر این کار را انجام ندهید، بازنمایی‌هایی که پایه کانولوشنی قبلاً یاد گرفته است، در طول آموزش مجدداً اصلاح خواهند شد. به دلیل مقداردهی تصادفی لایه‌های متراکمی که جدیداً اضافه شده‌اند، به‌روزرسانی‌های وزن بسیار بزرگی در شبکه انتشار می‌یابد و به شدت بازنمایی‌های یاد گرفته شده قبلی را از بین خواهد برد.

- در کراس، منجمد کردن شبکه با تنظیم ویژگی trainable به False صورت می‌گیرد.

تنظیم دقیق

تنظیم دقیق عبارت است از برداشتن انجماد چند لایه آخر مدل پایه منجمد شده که برای استخراج ویژگی به کار می‌رفت و سپس آموزش همزمان بخش‌های اضافه شده به مدل (در این مورد، دسته‌بند تمام متصل) و این لایه‌های بالایی.

- این روش از این رو تنظیم دقیق خوانده می‌شود که بازنمایی‌های انتزاعی‌تر مدل (مدلی که مجدداً مورد استفاده قرار می‌گیرد) را اندکی تنظیم می‌کند تا آن‌ها را با مسئله موردنظر مرتبط‌تر سازد.


Convolution2D
Convolution2D
MaxPooling2D
Conv block 1: frozen
Convolution2D
Convolution2D
MaxPooling2D
Conv block 2: frozen
Convolution2D
Convolution2D
Convolution2D
MaxPooling2D
Conv block 3: frozen
Convolution2D
Convolution2D
Convolution2D
MaxPooling2D
Conv block 4: frozen
Convolution2D
Convolution2D
Convolution2D
MaxPooling2D
تنظیم دقیق
Conv block 5
Flatten
Dense
Dense
تنظیم دقیق دسته بند
تمام متصل خودمان

تنظیم دقیق

تنظیم دقیق

چرا نباید لایه‌های بیشتری را تنظیم دقیق کرد؟

- لایه‌های اولیه در پایه کانولوشنی، ویژگی‌های قابل استفاده مجدد و کلی‌تری را کدگذاری می‌کنند در حالی که لایه‌های بالاتر ویژگی‌های اختصاصی‌تری را کدگذاری می‌کنند. تنظیم دقیق ویژگی‌های اختصاصی‌تر مفیدتر خواهد بود، چرا که این‌ها ویژگی‌هایی هستند که لازم است در مسئله جدید مجدداً هدف‌گذاری کنید. در تنظیم دقیق لایه‌های پایین‌تر شاهد بازده‌های سریعاً کاهشی خواهیم بود.

- تعداد پارامترهای آموزشی هر چه قدر بیشتر باشد، خطر بیش‌برازش بیشتر خواهد بود. پایه کانولوشنی 15 میلیون پارامتر دارد، بنابراین آموزش آن روی مجموعه داده کوچکتان همراه با خطر بیش‌برازش خواهد بود.

تنظیم دقیق

مراحل تنظیم دقیق شبکه به شرح زیر هستند:

1) اضافه کردن شبکه خودتان به لایه‌های آخر شبکه پایه‌ای که قبلاً آموزش دیده است

2) منجمد کردن شبکه پایه

3) آموزش بخشی که خودتان اضافه کرده‌اید.

4) برداشتن انجماد برخی از لایه‌ها در شبکه پایه

5) آموزش همزمان این لایه‌ها و بخشی که خودتان اضافه نموده‌اید.

GOING TIME

5.3-using-a-pretrained-convnet

مصورسازی آنچه شبکه‌های کانولوشنی یاد می‌گیرند

اغلب گفته می‌شود که مدل‌های یادگیری عمیق «جعبه‌های سیاه» هستند.

- به سختی می‌توان آموخته‌های آن را استخراج کرده و به صورت قابل فهم برای انسان نمایش داد.

- با وجود اینکه این گفته تا حدودی برای انواع خاصی از مدل‌های یادگیری عمیق صحت دارد، مشخصاً در مورد شبکه‌های کانولوشنی صادق نیست. بازنمایی‌هایی که شبکه‌های کانولوشنی یاد می‌گیرند به دلیل آنکه بازنمایی مفاهیم بصری هستند تا حدود زیادی قابلیت مصورسازی دارند.

مصورسازی آنچه شبکه‌های کانولوشنی یاد می‌گیرند

فنون گسترده‌ای برای مصورسازی و تفسیر این بازنمایی‌ها توسعه یافته‌اند. همه این فنون را بررسی نخواهیم کرد، اما به سه مورد از مفیدترین آن‌ها که به راحتی نیز در دسترس هستند اشاره خواهیم کرد:

- مصورسازی خروجی‌های میانی شبکه کانولوشنی (فعال‌سازی‌های میانی)
- مصورسازی فیلترهای شبکه‌های کانولوشنی
- مصورسازی و تولید نقشه‌های حرارتی از فعال‌سازی کلاس در یک تصویر

مصورسازی فعال‌سازی‌های میانی

- مصورسازی فعال‌سازی‌های میانی عبارت است از نمایش نقشه‌های ویژگی خروجی هر یک از لایه‌های کانولوشن به ازای یک ورودی مشخص
  - خروجی یک لایه اغلب فعال‌سازی آن نامیده می‌شود که در حقیقت همان خروجی تابع فعال‌سازی است.
- نقشه‌های ویژگی را با سه بعد یعنی عرض، ارتفاع و عمق (کانال‌ها) مصورسازی می‌کنیم.
  - از آنجا که هر کانال تقریباً ویژگی‌های مستقلی را کدگذاری می‌کند، بنابراین روش مناسب مصورسازی این نقشه‌های ویژگی این است که مفاهیم هر کانال را به طور مستقل به عنوان تصویر دو بعدی مصورسازی کنیم.

مصورسازی فعال‌سازی‌های میانی

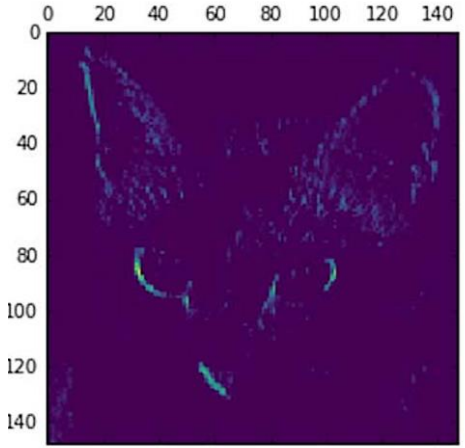
به منظور استخراج نقشه‌های ویژگی که می‌خواهیم نگاهی به آن‌ها بیاندازیم، مدل کراسی می‌سازیم که دسته‌های تصاویر را به عنوان ورودی می‌گیرد و فعال‌سازی تمامی لایه‌های کانولوشن و ادغام را به عنوان خروجی تولید می‌کند.

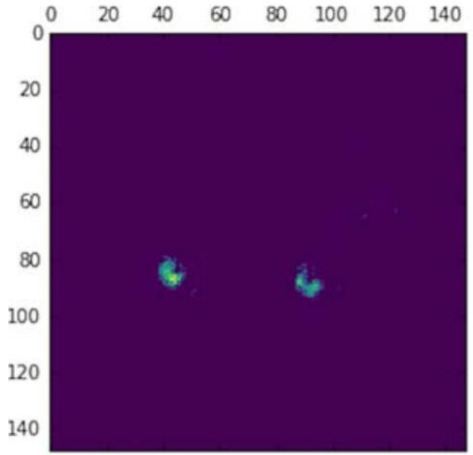
- برای این کار از کلاس Model کراس استفاده می‌کنیم. یک مدل با استفاده از دو آرگومان معرفی می‌شود: تنسور ورودی (لیست تنسورهای ورودی) و تنسور خروجی (لیست تنسورهای خروجی).

- کلاس حاصل، یک مدل کراس است درست مثل مدل‌های Sequential که با آن‌ها آشنا هستید، ورودی‌های خاصی را به خروجی‌های خاصی نگاشت می‌کند.

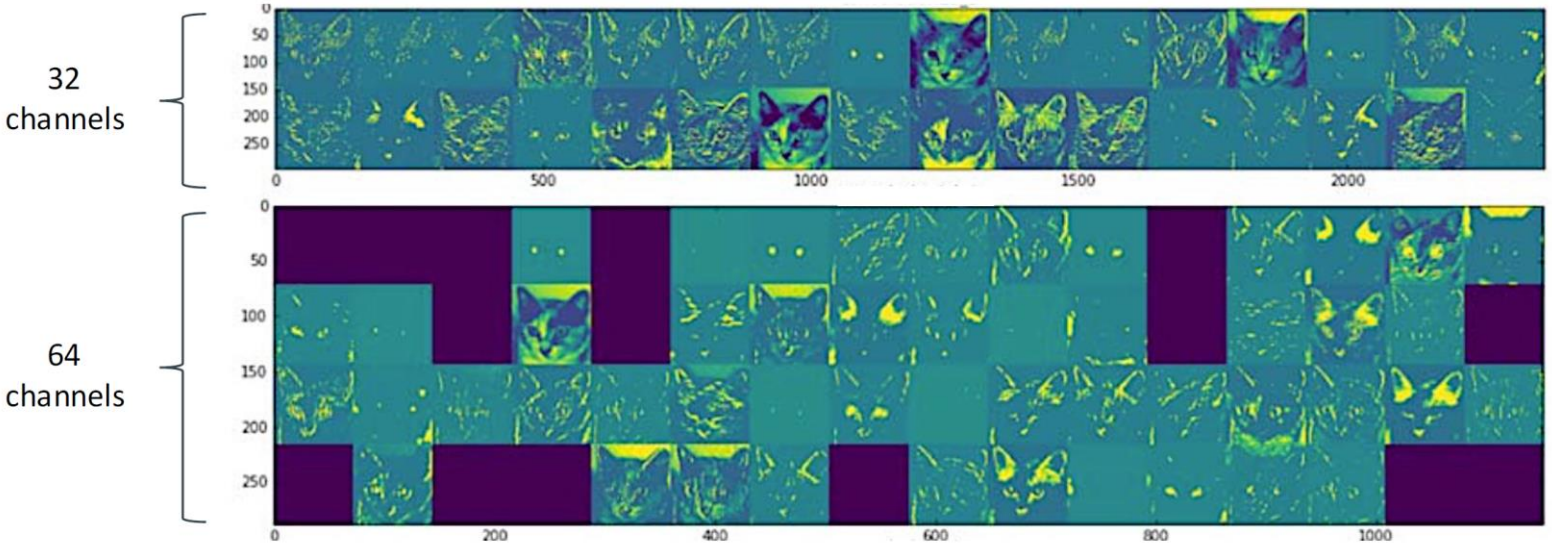
مصورسازی فعال‌سازی‌های میانی

```
import matplotlib.pyplot as plt

plt.matshow(first_layer_activation[0, :, :, 4], cmap='viridis')

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

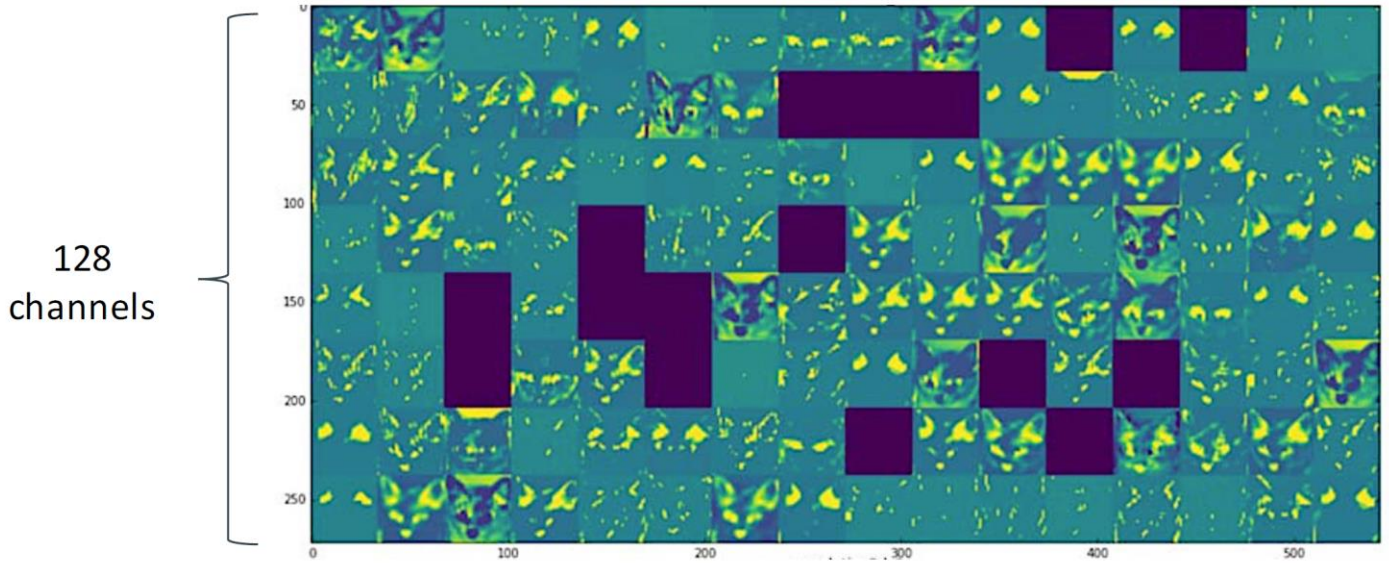




مصورسازی فعال‌سازی‌های میانی



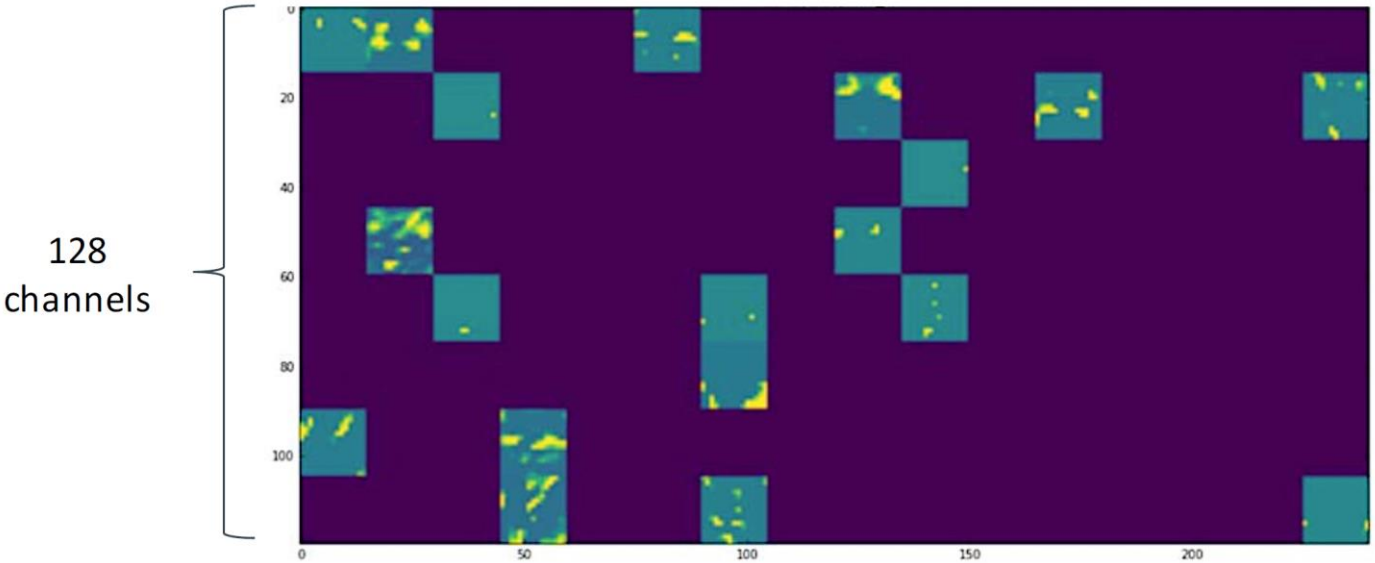
مصورسازی فعال‌سازی‌های میانی



هر چه بالاتر بروید، فعال‌سازی‌ها انتزاعی‌تر می‌شوند و قابلیت تفسیر بصری آن‌ها کم‌تر می‌شود. آن‌ها مفاهیم سطح بالاتری مثل «گوش گربه» و «چشم گربه» را کدگذاری می‌کنند.

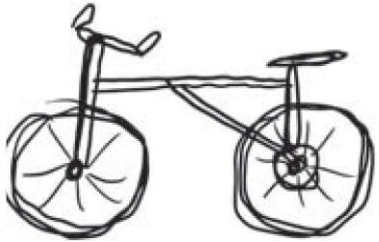
بازنمایی‌های بالاتر اطلاعات کم‌تری را در مورد محتوای بصری تصاویر داشته و اطلاعات مربوط به کلاس تصاویر در آن‌ها بیشتر است.

مصورسازی فعال‌سازی‌های میانی



پراکندگی فعال‌سازی‌ها با عمق لایه افزایش می‌یابد: در اولین لایه، تمامی فیلترها با تصویر ورودی فعال می‌شوند؛ اما در لایه‌های بعدی، فیلترهای بیشتر و بیشتری خالی (سفید) هستند؛ یعنی الگویی که با آن فیلتر کدگذاری شده است در تصویر ورودی یافت نمی‌شود.

مصورسازی فعال‌سازی‌های میانی: خلاصه



هر چه که در عمق شبکه پیش می‌رویم ویژگی‌های استخراج شده به وسیله لایه‌ها انتزاعی‌تر می‌گردند. فعال‌سازی‌های لایه‌های بالاتر حامل اطلاعات کمتر و کمتری در مورد ورودی و حامل اطلاعات بیشتر و بیشتری در مورد هدف هستند.

این روش همانند شیوه ادراک جهان به وسیله انسان‌ها و حیوانات است: بعد از مشاهده یک صحنه، انسان می‌تواند اشیای انتزاعی موجود در صحنه را به خاطر بیاورد (دوچرخه، درخت) اما نمی‌تواند ظاهر این اشیا را به خاطر بیاورد. در واقع، اگر سعی کنید از روی حافظه یک دوچرخه کلی را بِکشید، با وجود اینکه در طول زندگی هزاران دوچرخه را دیده‌اید، احتمالاً حتی نخواهید توانست کمی آن را درست به تصویر بکشید.

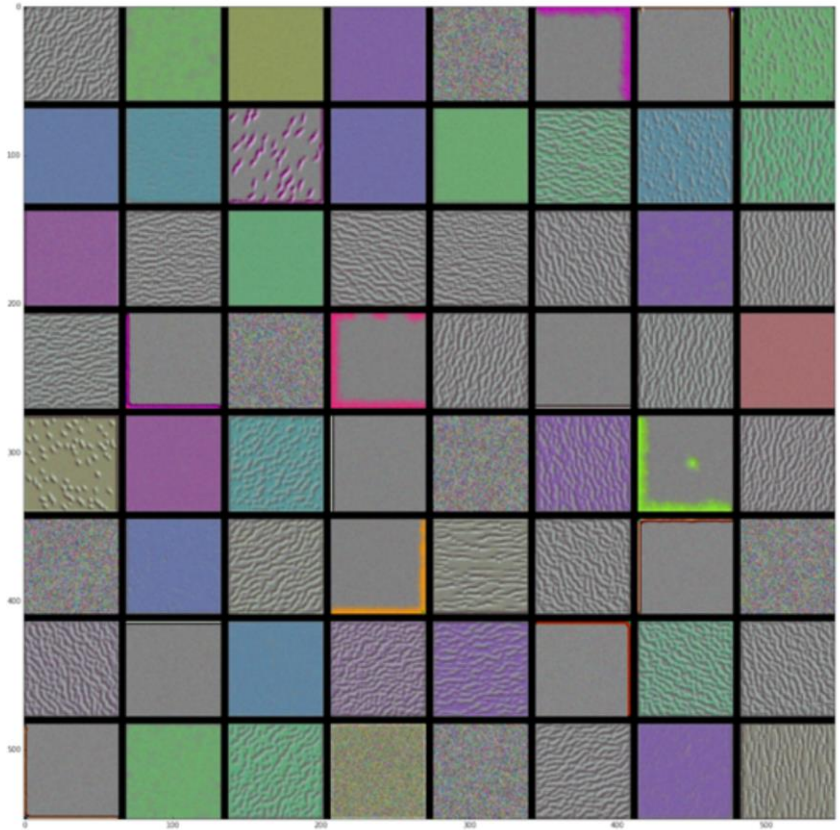
مصورسازی فیلترهای شبکه کانولوشنی

روش ساده دیگری که برای بررسی فیلترهای یاد گرفته شده به وسیله شبکه‌های کانولوشنی وجود دارد نمایش الگوی مصوری است که هر فیلتر مسئول ایجاد آن است.

این کار با اجرای گرادیان صعودی در فضای ورودی قابل انجام است: بدین ترتیب که با شروع از تصویر ورودی خام و اعمال گرادیان نزولی روی آن سعی میشود پاسخ یک فیلتر خاص به حداکثر رسانده شود. تصویر ورودی حاصل، تصویری خواهد بود که فیلتر منتخب حداکثر پاسخ‌گویی به آن را دارد.

مصورسازی فیلترهای شبکه کانولوشنی

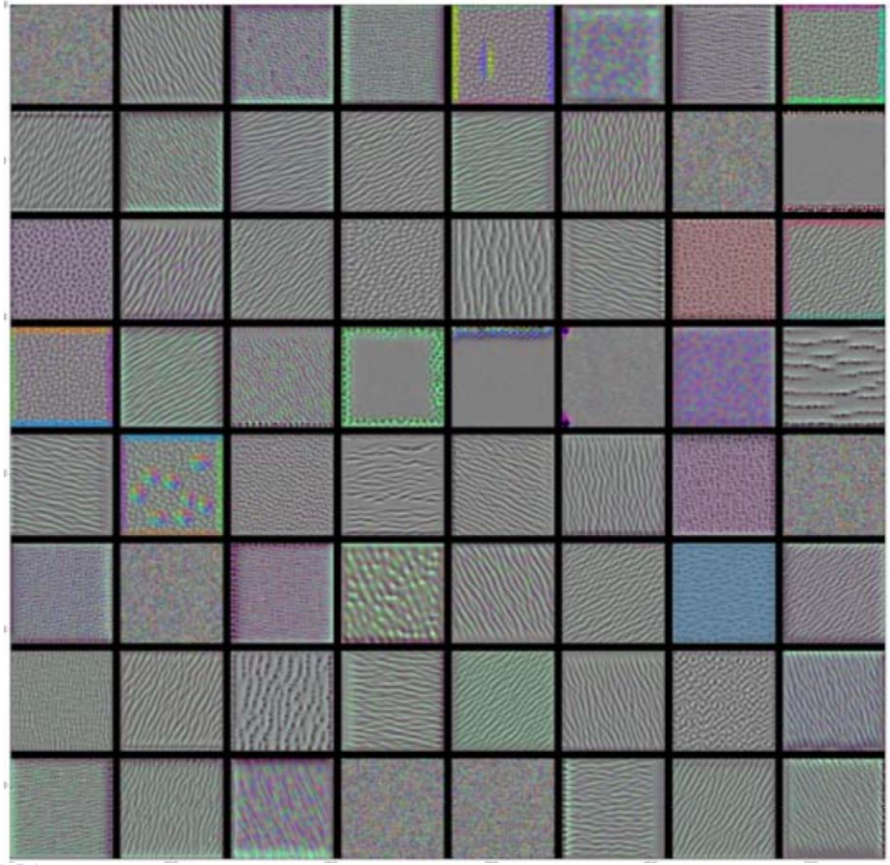
Filter patterns for the first 64 filters of layer block1_conv1



The filters from the first layer in the model encode simple directional edges and colors.

مصورسازی فیلترهای شبکه کانولوشنی

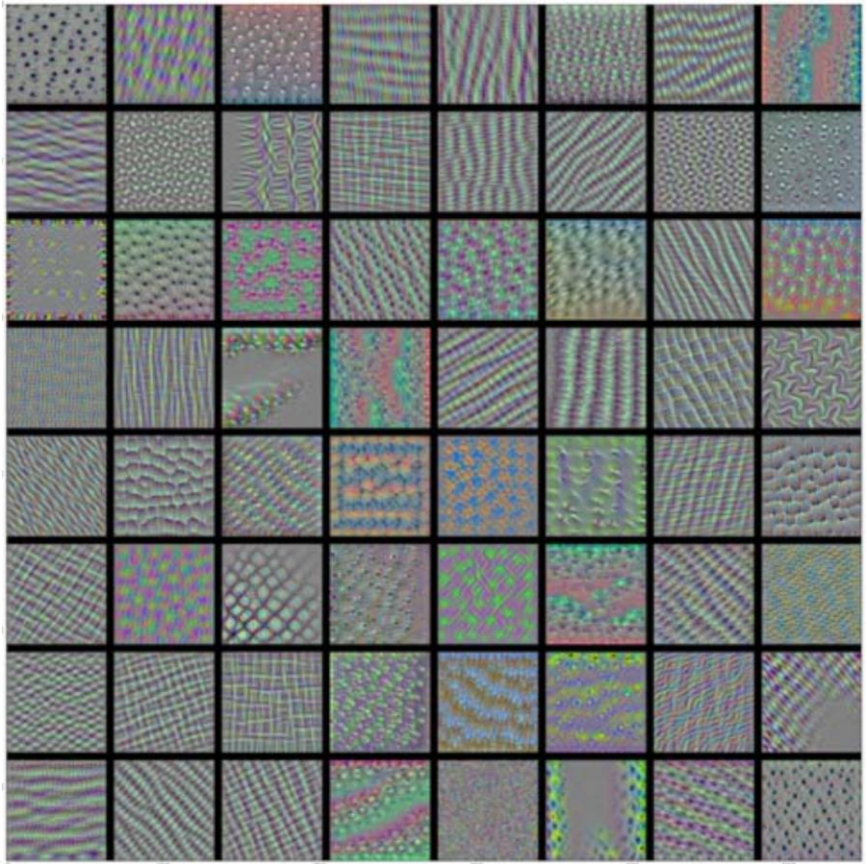
Filter patterns for the first 64 filters of layer block2_conv1



The filters from the second layer encode simple textures made from combinations of edges and colors.

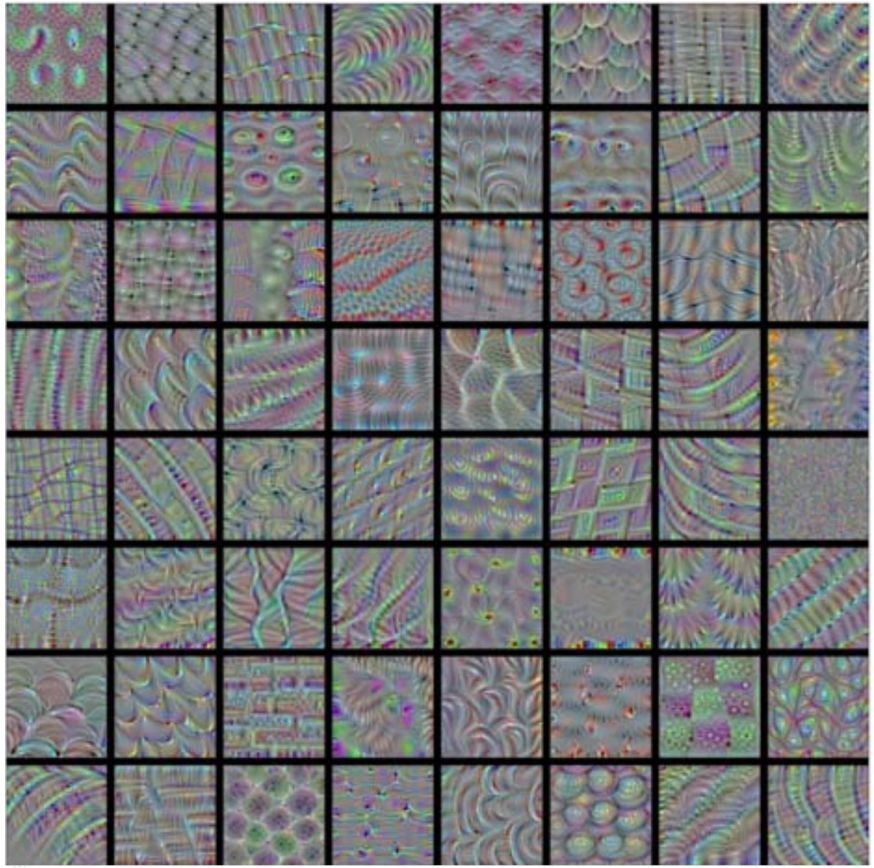
مصورسازی فیلترهای شبکه کانولوشنی

Filter patterns for the first 64 filters of layer block3_conv1



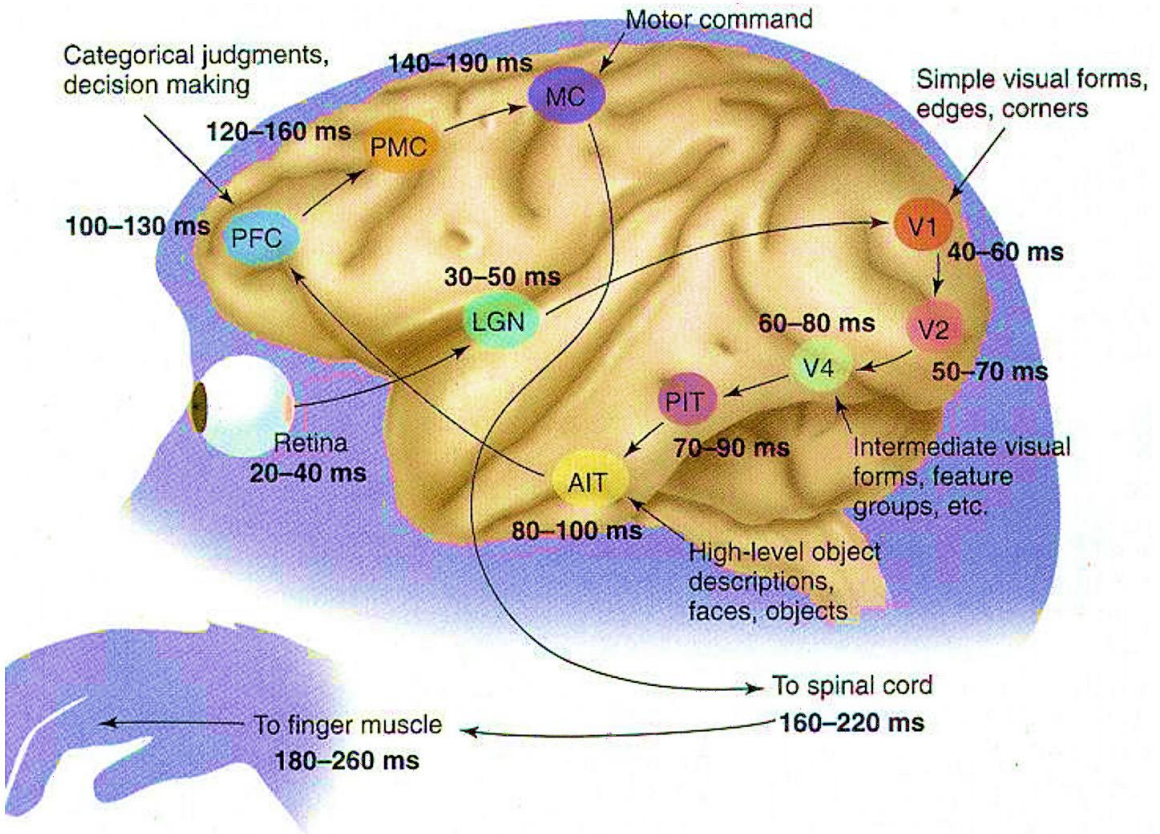
مصورسازی فیلترهای شبکه کانولوشنی

Filter patterns for the first 64 filters of layer block4_conv1



The filters in higher layers begin to resemble textures found in natural images: feathers, eyes, leaves, and so on.

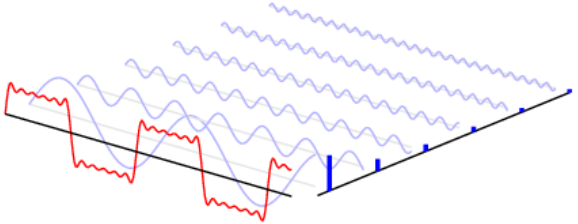
مصورسازی فیلترهای شبکه کانولوشنی: مغز



مصورسازی فیلترهای شبکه کانولوشنی: خلاصه

- هر لایه در شبکه عصبی، مجموعه‌ای از فیلترها را به گونه‌ای یاد می‌گیرد که ورودی‌های آن‌ها را بتوان به عنوان ترکیبی از فیلترها توصیف کرد.

- این روش مشابه با تبدیل فوریه است که در آن سیگنال‌ها به دسته‌ای از توابع کسینوسی تجزیه می‌شوند. این دسته فیلترهای کانولوشنی با پیشروی در مدل رفته رفته پیچیده‌تر شده و پالایش می‌شوند.



http://pgfplots.net/tikz/examples/fourier-transform/

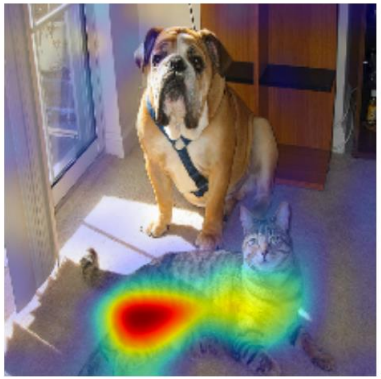
مصورسازی نقشه‌های حرارتی فعال‌سازی کلاس

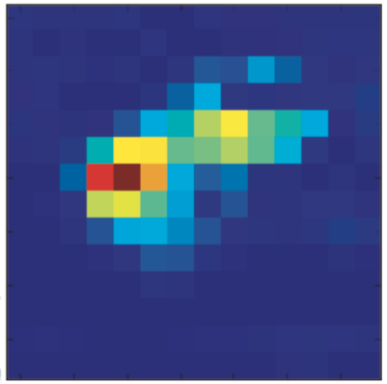
روشی که به شما می‌گوید کدام قسمت از یک تصویر مفروض، شبکه کانولوشنی را به تصمیم دسته‌بندی نهایی هدایت می‌کند.

- این روش برای اشکال‌زدایی فرایند تصمیم‌گیری شبکه کانولوشنی، به ویژه در صورت بروز اشتباه در دسته‌بندی، مفید واقع می‌شود.
- همچنین امکان تشخیص مکان اشیا در یک تصویر را فراهم می‌آورد.

جالب است بدانید که گوش‌های بچه فیل به شدت فعال شده‌اند: احتمالاً شبکه بدین طریق است که می‌تواند تفاوت بین فیل‌های آفریقایی و هندی را تشخیص دهد.

مصورسازی نقشه‌های حرارتی فعال‌سازی کلاس





این دسته‌ی عمومی از فنون، مصورسازی نقشه فعالیت کلاس (CAM) نامیده می‌شود.

نقشه فعالیت کلاس عبارت است از تولید نقشه حرارتی فعال‌سازی کلاس در تصاویر ورودی. نقشه حرارتی فعال‌سازی کلاس، یک ماتریس دو بعدی از امتیازاتی است که با کلاس خروجی خاصی مرتبط است، برای هر مکانی در تصویر ورودی محاسبه می‌شود و اهمیت هر مکان با توجه به کلاس مورد نظر را نشان می‌دهد.

مصورسازی نقشه‌های حرارتی فعال‌سازی کلاس

از پیاده‌سازی که در مقاله Grad-CAM رائه شده است، استفاده خواهیم کرد.

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

این روش بسیار ساده بوده و عبارت است از دریافت نقشه ویژگی خروجی یک لایه کانولوشنی به ازای دریافت یک تصویر و وزن‌دار کردن هر یک از کانال‌های این نقشه ویژگی مبتنی بر گرادیان کلاس نسبت به آن کانال. به عبارتی شما «میزان اهمیت کانال‌های مختلف در تصویر ورودی» را با «اهمیت هر کانال با توجه به کلاس» وزن‌دار می‌کنید و نقشه فضایی حاصل «میزان تعلق تصویر به یک کلاس » است.

مصورسازی نقشه‌های حرارتی فعال‌سازی کلاس

```
('n03788195', 'mosque', 0.99687386)
('n03220513', 'dome', 0.0022577539)
('n03028079', 'church', 0.00034330128)
```

GOING TIME

5.4-visualizing-what-convnets-learn

Project

یک مجموعه داده از کگل دانلود کرده و روی آن کار کنید. موضوعات مختلف آموخته شده در این فصل را امتحان کنید: از جمله آموزش یک شبکه کانولوشنی از ابتدا، استفاده از یک شبکه کانولوشنی از پیش آموزش دیده، و مصورسازی آنچه که شبکه ها می آموزند.

گزارش علمی (قطعات کد، نمودارها، نتایج، بحث) را تا ساعت ۲۳:۵۹، ۸ بهمن ۱۴۰۱ از طریق سامانه جام ارسال کنید. به ازای هر روز تاخیر ۲۰٪ نمره کم خواهد شد.

تشکر

سوال؟

a.golzari@azaruniv.ac.ir

a.golzari@tabrizu.ac.ir

https://github.com/Amin-Golzari-Oskouei