

Deep learning – redes convolucionais

Conceitos básicos e exemplos de aplicação
com *keras* e *tensorflow*

Exemplos/ figuras adaptados de F. Chollet, “Deep Learning with Python” e Andrew Ng (deeplearning.ai)

O que são convoluções

Uma convolução é uma operação matemática de duas matrizes:

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

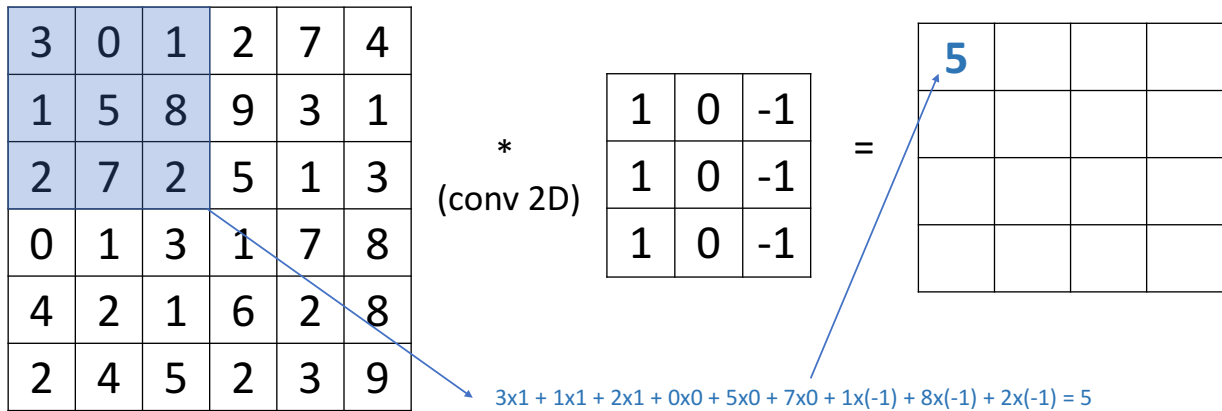
$*$
(conv 2D)

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

$=$

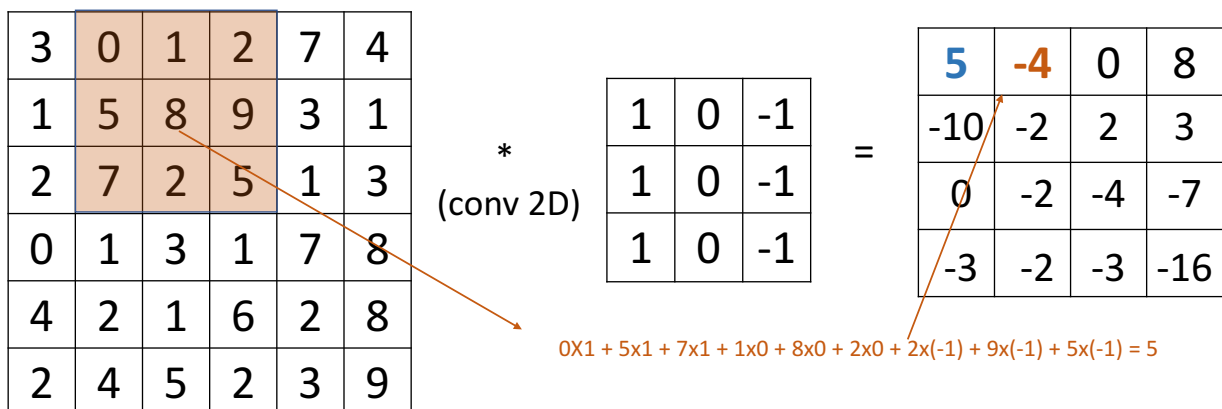
O que são convoluções

Sobrepõe a 2ª matriz (filtro) à primeira em cada posição e multiplica os elementos em cada posição somando o resultado das multiplicações



O que são convoluções

Sobrepõe a 2ª matriz (filtro) à primeira em cada posição e multiplica os elementos em cada posição somando o resultado das multiplicações



Convoluções como filtros de imagens

No processamento de imagens as convoluções, aplicadas a diversas zonas da imagem, funcionam como filtros que podem detectar determinados padrões

Exemplos são convoluções que pode funcionar como detector de “edges” verticais ou horizontais

Convoluções como filtros de imagens: edges verticais

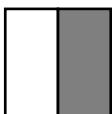

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

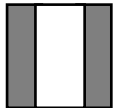
 $*$

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

 $=$

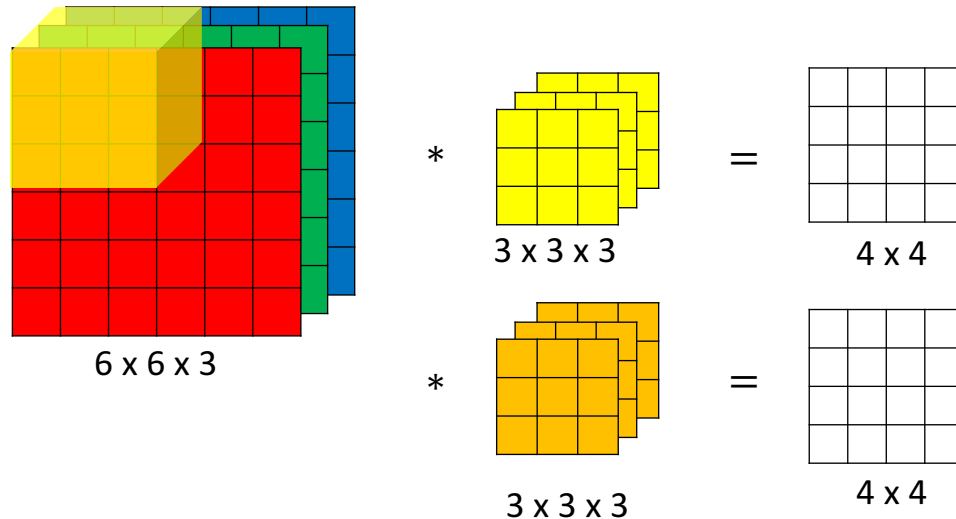
0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0


 $*$


$=$


Exemplo de Andrew Ng

Convoluções sobre imagens 3D



Exemplo de Andrew Ng

Redes Convolucionais

CNN – Convolutional neural networks – são modelos de *deep learning* em que as diversas camadas da rede são diferentes das completamente ligadas, consistindo em camadas convolucionais e de “pooling” que recebem inputs e geram outputs que são tensores 3D para cada exemplo

São modelos tipicamente aplicados sobre imagens onde os tensores 3D representam altura, largura e profundidade (ou “canal”) – na camada inicial, estas podem ser as dimensões de uma imagem (e.g. RGB terá 3 canais); nas camadas subsequentes cada canal funciona como filtro sobre a camada anterior

A introdução destes modelos representou uma melhoria significativa na forma como se processam e classificam imagens

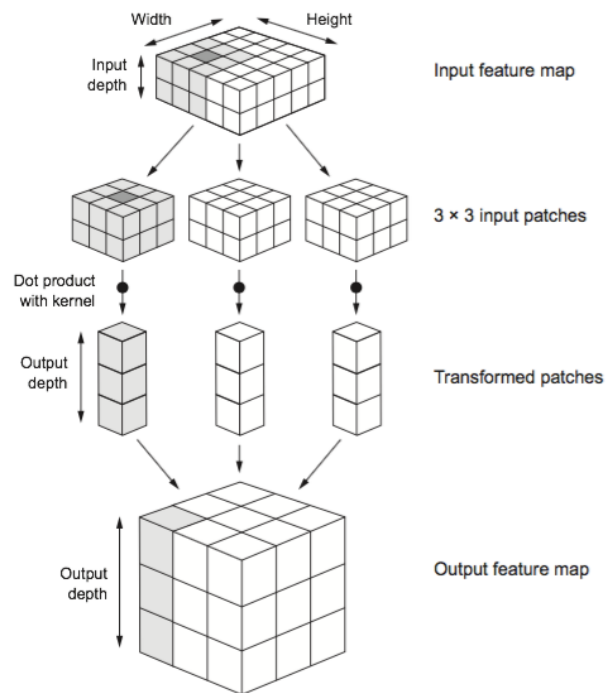
Camadas convolucionais

Trabalham sobre “*feature maps*” – tensores 3D: *height x width x depth*

São seleccionados todos os “patches” de dada dimensão (e.g. 3x3 ou 5x5) e aplicada uma transformação a cada um deles (e.g. tensores 3D 3x3xdepth)

Transformação consta de multiplicação pelos pesos de “convolutional kernels”, que são aprendidos via gradiente descendente (mesmo kernel para todas as janelas; nº kernels = profundidade output)

Transformação cria vetor 1D de tamanho igual à profundidade do output



Camadas de pooling

Usadas como forma de reduzir dimensão das camadas convolucionais através de um processo de amostragem

Extraí janelas da camada anterior e calcula um único valor para cada janela, tipicamente considerando a aplicação da função **max** (fica apenas com o maior valor na janela)

Outra alternativa pode ser a média dos valores da janela (menos usado)

Valores típicos para as janelas são 2 x 2

Estas camadas não têm parâmetros a aprender

Camadas promovem aprendizagem “hierarquizada”

1	3	2	1
2	9	1	1
1	3	2	3
5	6	1	2



9	2
6	3

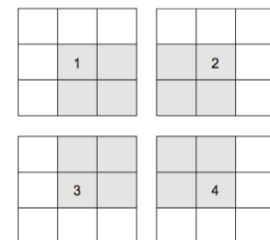
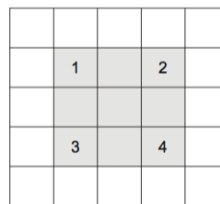
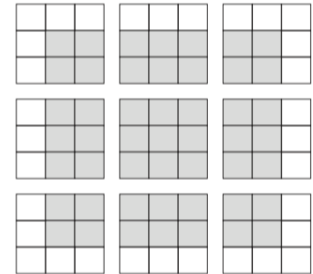
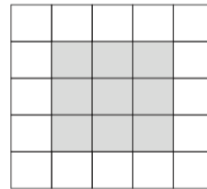
Dimensões das camadas: padding e stride

Dimensões das camadas convolucionais (comprimento, largura) tendem a diminuir dadas as posições válidas da janela

Pode usar-se *padding* para manter as mesmas dimensões (considerar linhas e colunas extra com valores iguais a zero)

O parâmetro *stride* controla os "intervalos" entre as convoluções

Exemplo com stride = 2 =====>
Usado por omissão nas camadas de max pooling

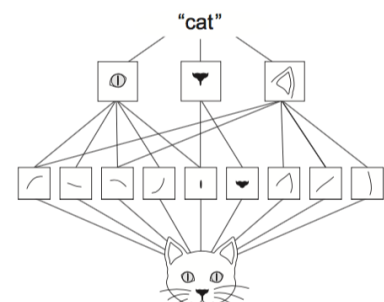
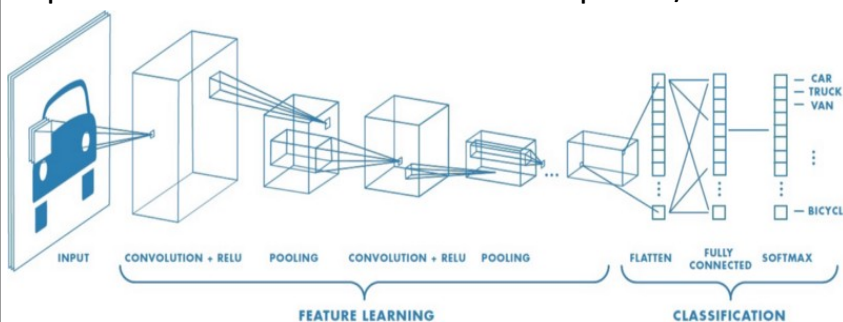


Aprendizagem em redes convolucionais

Processamento de dados nas camadas convolucionais trabalham sobre regiões da imagem, i.e. uma matriz de pixels de pequenas dimensões (e.g. 3x3)

Permite aprender padrões que podem ser reconhecidos em várias partes da matriz (e.g. imagem) – invariante a translações

Permitem aprender de forma hierárquica, com camadas sucessivas a aprenderem conceitos cada vez complexos/ abstratos



<https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148>

Conjunto de dados MNIST

Como primeiro exemplo, vamos usar o conjunto de dados de reconhecimento de dígitos (MNIST) já usado na aula anterior

Inputs: imagens de 28 x 28 pixels

Output: classe representando o dígito (10 classes, dígitos 0-9)

Disponível como dataset do keras

60k imagens de treino e 10k imagens de teste

Conjunto de dados MNIST

```
from keras.datasets import mnist
(train_images, train_labels), (test_images, test_labels) =
    mnist.load_data()
print(train_images.shape, test_images.shape)
print(len(train_labels), len(test_labels))
```

Carregar os dados

Verificar dimensões

```
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)
```

- Ajustando as dimensões para tornar as entradas num tensor 4D
(nº exemplos x largura x comprimento x canais)
- Standardizar valores dividindo por 255

Conjunto de dados MNIST - modelo

```
from keras import layers, models
```

```
model = models.Sequential()
```

32 canais; janelas 3 x 3

```
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'))
```

Alternando camadas convolucionais com pooling

```
model.add(layers.MaxPooling2D((2, 2)))
```

```
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

```
model.add(layers.Flatten())
```

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

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

Camadas para permitir a classificação:
Flatten – passa tensor 3D para 1D
Dense – camada intermédia e de output semelhantes às usadas anteriormente

Conjunto de dados MNIST - modelo

```
print(model.summary())
```

Outputs das camadas convolucionais e de pooling são tensores 3D

Nº parâmetros numa camada convolucional:

[Largura janela x
Altura da janela x
Nº filtros do input (depth) + 1] x
Nº de filtros output (depth)

Exemplos: conv2d_1:

$(3 \times 3 \times 1 + 1) \times 32 = 320$

Conv2d_2

$(3 \times 3 \times 32 + 1) \times 64 = 18496$

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d_1 (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_2 (Conv2D)	(None, 11, 11, 64)	18496
max_pooling2d_2 (MaxPooling2D)	(None, 5, 5, 64)	0
conv2d_3 (Conv2D)	(None, 3, 3, 64)	36928
flatten_1 (Flatten)	(None, 576)	0
dense_80 (Dense)	(None, 64)	36928
dense_81 (Dense)	(None, 10)	650
Total params: 93,322		
Trainable params: 93,322		
Non-trainable params: 0		

Conjunto de dados MNIST – avaliação do modelo

```
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)  
print(test_loss, test_acc)
```

Conjunto de dados cães e gatos (Kaggle)

Versão adaptada do exemplo 5.2 do site:

<https://github.com/fchollet/deep-learning-with-python-notebooks>

Dataset já organizado (zip) e python notebook modificado no elearning

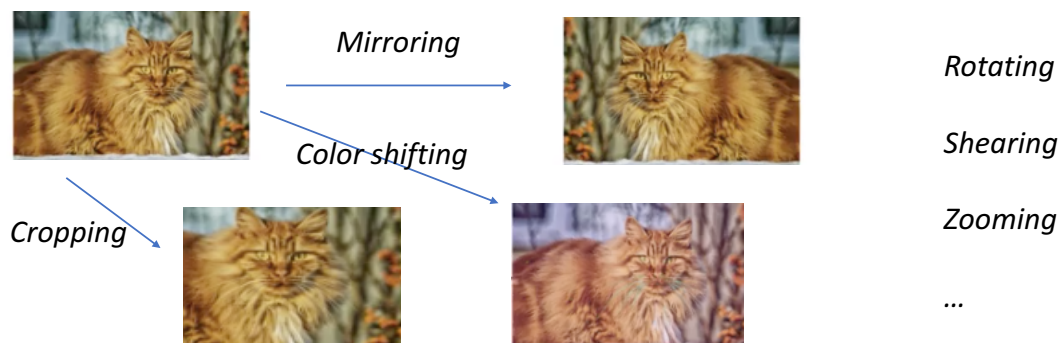
Dataset do exemplo com 2000 imagens para treino, 1000 para validação e 1000 para teste

Inclui exemplos de como usar geradores de dados de treino/ teste a partir de ficheiros JPG

Nesta primeira fase, vamos usar apenas estes dados e verificar o desempenho: rede semelhante à usada no MNIST com mais layers; saída binária

Extensão dos dados de treino (*data augmentation*)

Uma das soluções quando temos sobre-ajustamento devido a uma pequena quantidade de dados é criar exemplos de treino com base nos existentes, através de modificações na imagem que não alterem a sua essência, continuando a representar exemplos da mesma classe



Conjunto de dados cães e gatos: extensão dos dados de treino

Ainda no exemplo 5.2, é dado um exemplo de extensão de dados (*data augmentation*) com base nos dados existentes, para combater o sobre-ajustamento

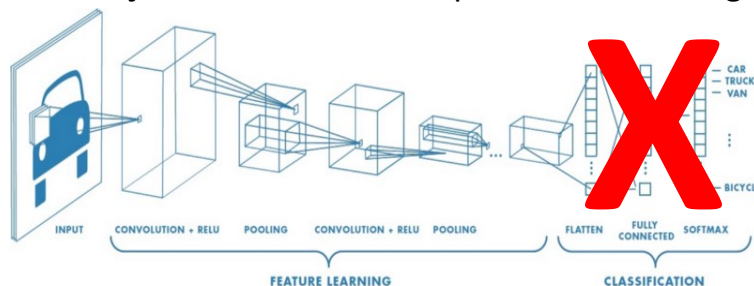
Esta extensão pode ser conseguida pelo processamento das imagens existentes com diversas operações: rotações, translações horizontais e verticais, *zooming*, *flipping*, ...

Combinando esta extensão com outras medidas para combater sobre-ajustamento (e.g. *Dropout*) é possível melhorar o desempenho da rede

Usar modelos pré-treinados (*transfer learning*)

Uma alternativa para poder melhorar modelos quando temos poucos dados passa pelo uso de redes pré-treinadas com conjuntos de dados de maior dimensão (criados para outros problemas) e que podem ser refinados com o nosso conjunto de dados

Passo inicial: identificar uma rede existente adequada, importar a sua topologia e os seus pesos, remover partes que possam não se adaptar (e.g. parte de classificação nas camadas completamente interligadas)



Usar modelos pré-treinados (*transfer learning*)

Depois temos duas alternativas:

- Calcular as saídas da rede anterior para os nossos casos de treino, e usar estas saídas como entradas numa rede complementamente ligada e treinar esta rede
- Juntar camadas completamente ligadas à rede original e treinar esta rede de forma global com os nossos exemplos:
 - Pode-se re-treinar toda a rede usando os pesos como valores iniciais – normalmente não recomendado a não ser que existam muitos dados
 - Pode-se treinar apenas a parte “nova”, “congelando” os pesos da anterior
 - Pode-se treinar a parte nova e uma parte da “anterior” mantendo a parte inicial com pesos inalterados

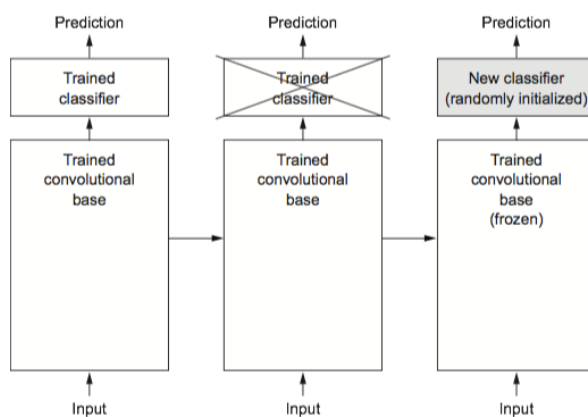
Conjunto de dados cães e gatos: pré-treino

Exemplo 5.3 – mostra como usar CNNs pré-treinadas com outros datasets maiores, re-treinando a parte de classificação do modelo para este dataset

Nesta caso, é usada uma CNN pré-treinada (VGG16) no conjunto de dados ImageNet com mais de 1 milhão de imagens de diversos objetos (mais de 1000 classes)

São seguidas as várias alternativas descritas anteriormente; inicialmente, usa-se a base convolucional da VGG para calcular os inputs para uma rede completamente interligada

Conjunto de dados cães e gatos: pré-treino



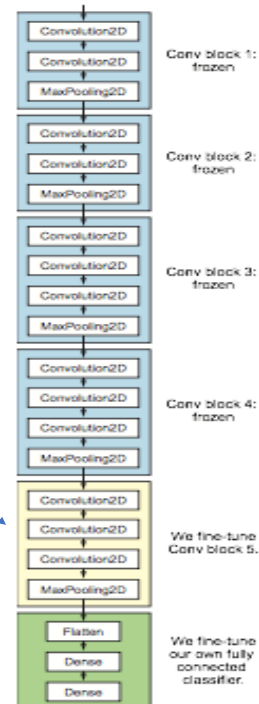
Base convolucional treinada no dataset maior e depois mantida (pesos "congelados")

Usar a rede convolucional base com pesos congelados, juntando-lhe a parte de previsão com camadas Dense, tem vantagens em relação a usar simplesmente as suas saídas para treinar uma DNN, pois permite usar extensão dos dados para prevenir sobre-ajustamento

Conjunto de dados cães e gatos: fine-tuning

“Descongelar” uma ou mais camadas e re-treinar os seus pesos

...



Visualização do “funcionamento” da rede

Há várias estratégias para tentar perceber o que cada camada/ kernel da CNN está a aprender. Quando trabalhamos com imagens, estas estratégias podem ajudar a perceber quais as “features” aprendidas em cada camada/kernel da rede.

Para exemplo, ver notebook 5.4; exemplos de mapas de ativação de várias camadas:

Features de mais “alto nível” / mais abstratas

