Uma introdução as redes neurais convolucionais utilizando o Keras

Saiba como funciona uma CNN através desse exemplo com o dataset MNIST

Alan Melo Clappis
Jul 12, 2019 · 6 min read

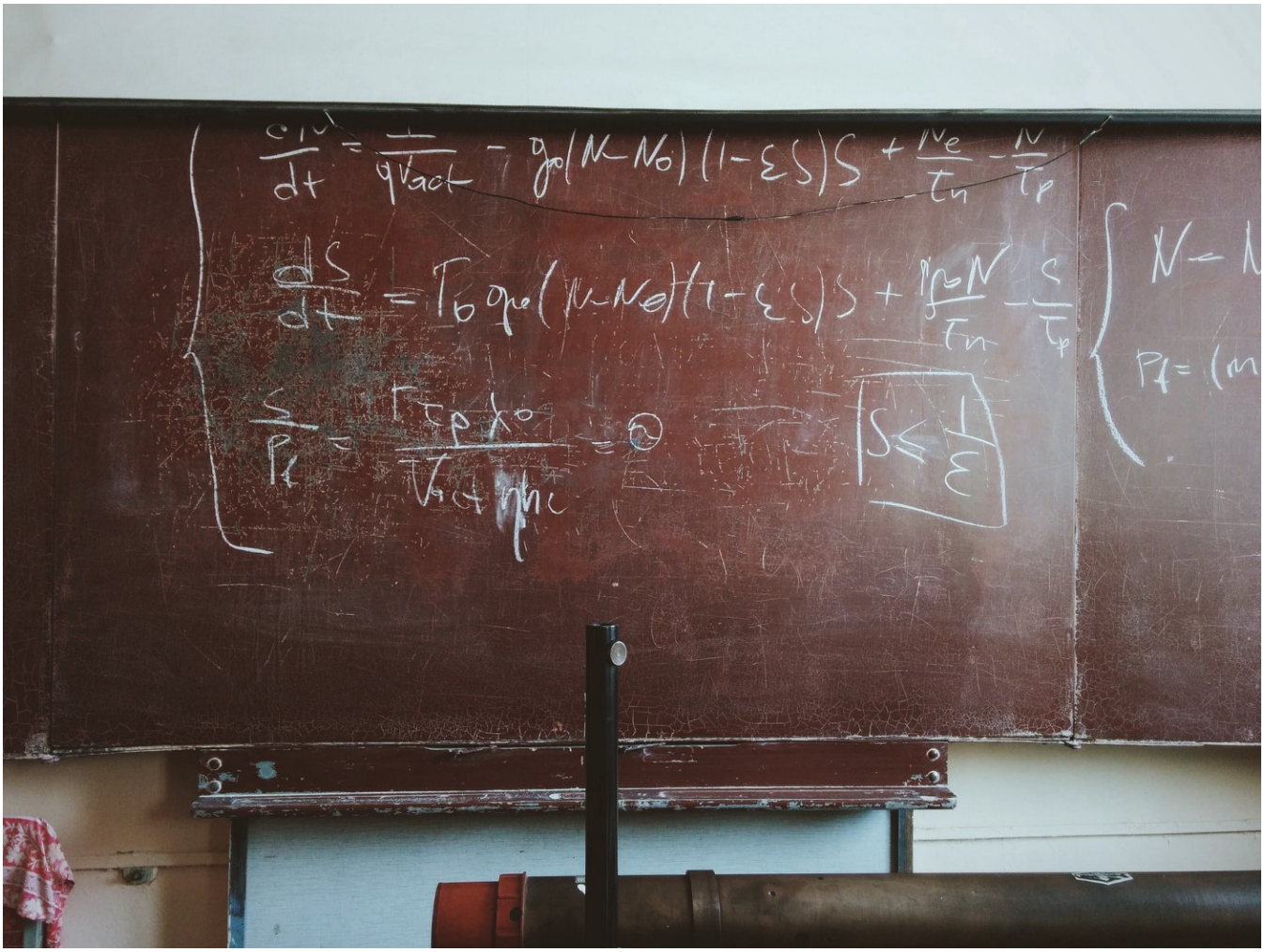


Photo by Roman Mager on Unsplash

Nós, aqui da EurekaLabs, decidimos reunir nosso time de cientistas de dados para resolvermos alguns problemas do Kaggle. As soluções terão um viés didático e serão compartilhadas aqui no Medium, então fique atento :).

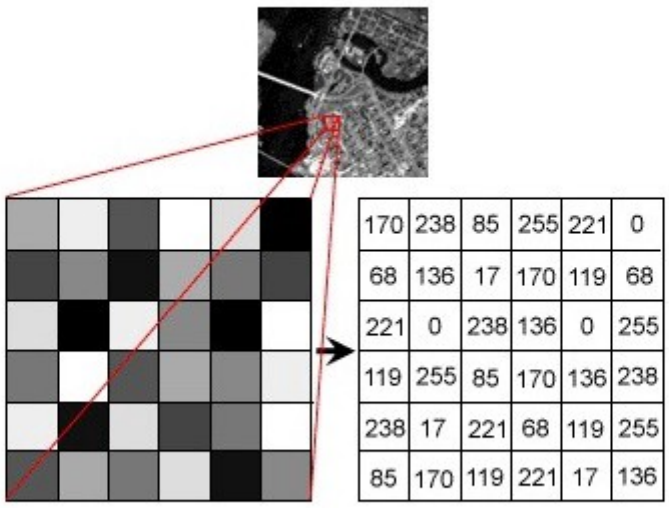
Iniciaremos por um problema simples e gradualmente iremos aumentar a complexidade. Dessa vez, iremos resolver o digit-recognizer, que é basicamente um desafio de reconhecimento de dígitos.

Mas antes de detalhar o problema, vamos entender a teoria de uma rede neural convolucional (CNN), que vai ser o algoritmo adotado para resolver esse problema.

. . .

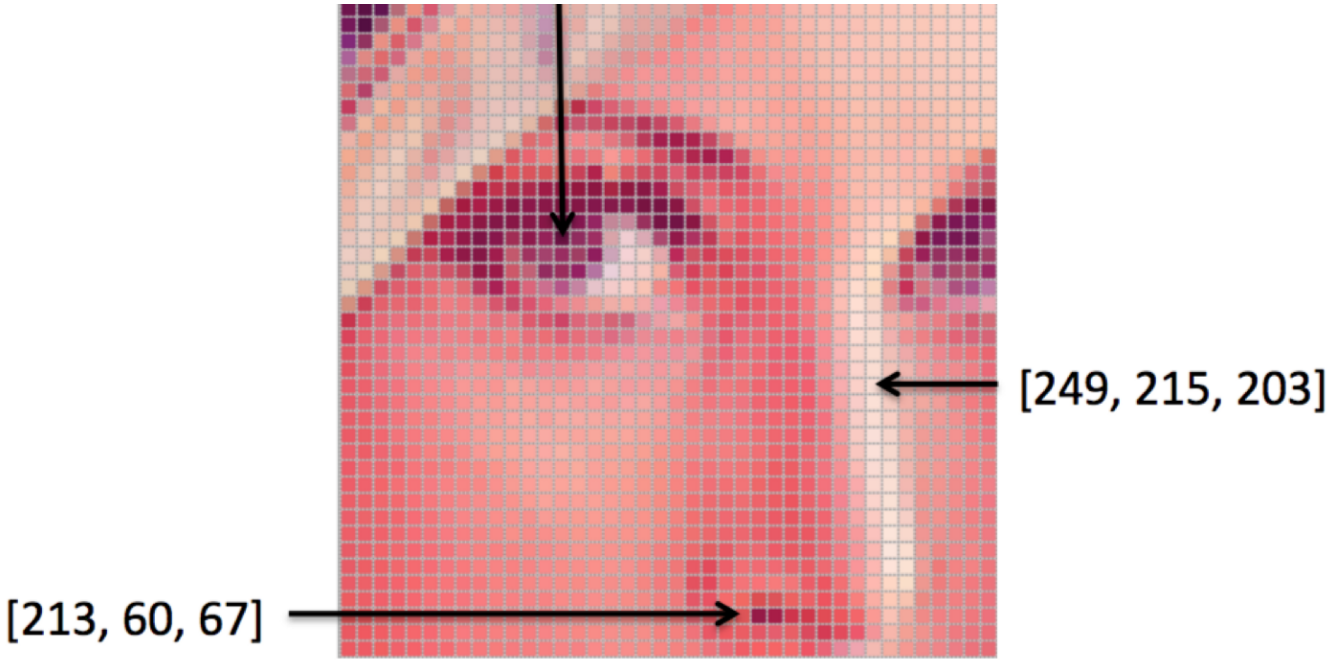
Uma CNN é um tipo específico de rede neural normalmente utilizada para classificação de imagens. Mas para entendermos uma CNN, é preciso compreender como uma imagem é representada computacionalmente.

Uma imagem preta e branca (*grayscale*) é representada como uma *matrix* 2D, em que cada posição da *matrix* representa um pixel da imagem. Os valores para cada elemento variam entre 0 (preto) até 255 (branco), conforme o exemplo abaixo.



Representação grayscale. Fonte: Stanford

Já uma imagem colorida, é normalmente representada por uma matrix 3D de forma que seja possível armazenar uma combinação das cores vermelho, verde e azul (RGB do inglês).



Representação RGB. Fonte: Stanford

Uma CNN pode ser dividida em duas partes: extração de características (*Conv*, *Padding*, *Relu*, *Pooling*) e uma rede neural tradicional.

Convoluções

Matematicamente, uma convolução é uma operação linear que a partir de duas funções, gera uma terceira (normalmente chamada de *feature map*). No contexto de imagens, podemos entender esse processo como um filtro/*kernel* que transforma uma imagem de entrada.

Um *kernel* é uma *matrix* utilizada para uma operação de multiplicação de matrizes. Esta operação é aplicada diversas vezes em diferentes regiões da imagem. A cada aplicação, a região é alterada por um parâmetro conhecido como *stride*. Normalmente o stride possui o valor 1, o que significa que a transformação será aplicada em todos os *pixels* da imagem. Um exemplo dessa transformação é ilustrado abaixo.

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0

1	0	1
0	1	0
1	0	1

0	1	1	0	0

Imagem de entrada a esquerda, filtro a direita. Fonte: Towards

1x1	1x0	1x1	0	0
0x0	1x1	1x0	1	0
0x1	0x0	1x1	1	1
0	0	1	1	0
0	1	1	0	0

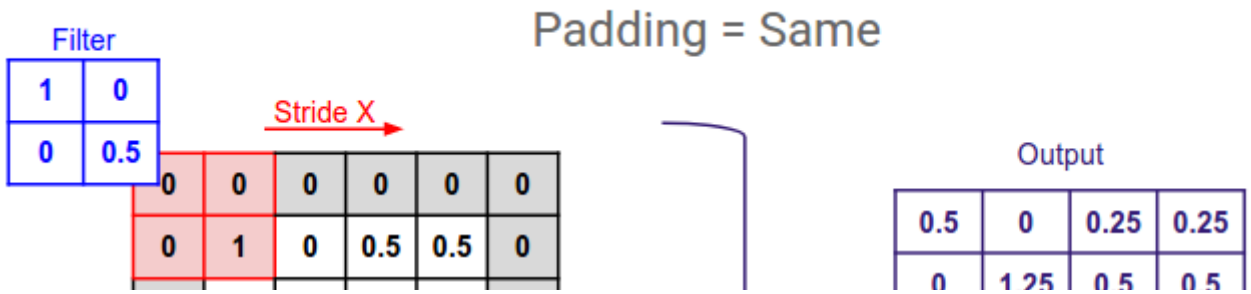
4		

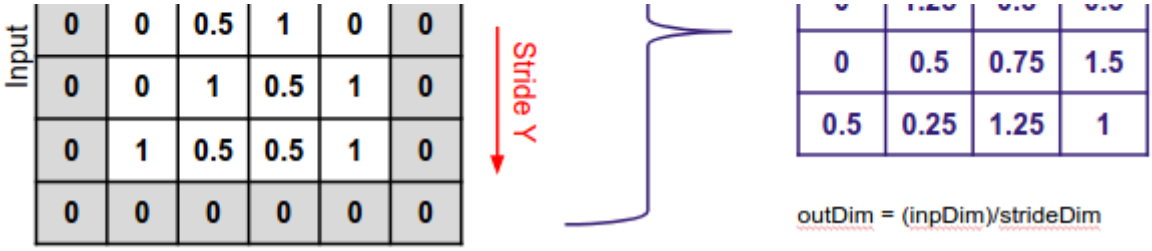
Convolução em uma imagem: Fonte: Towards

Como você pode notar, a *matrix* resultante desse exemplo possui uma dimensionalidade menor que a imagem de origem. Muitas convoluções podem impactar na assertividade da CNN se o tamanho da imagem for muito reduzido. Para contornar esse cenário, normalmente é utilizado o conceito de *Padding*.

Padding

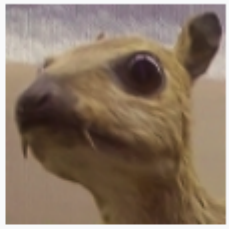
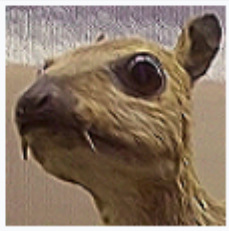
Padding é um processo em que alguns pixels são adicionados ao redor da imagem antes da operação de convolução, de forma a manter a dimensionalidade na imagem resultante durante a operação.





Padding — Fonte: Medium

Algumas das operações de convolução mais utilizadas com imagens são apresentadas abaixo.

Operation	Kernel ω	Image result g(x,y)
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	

Convoluções em imagens — Fonte: Wikipedia

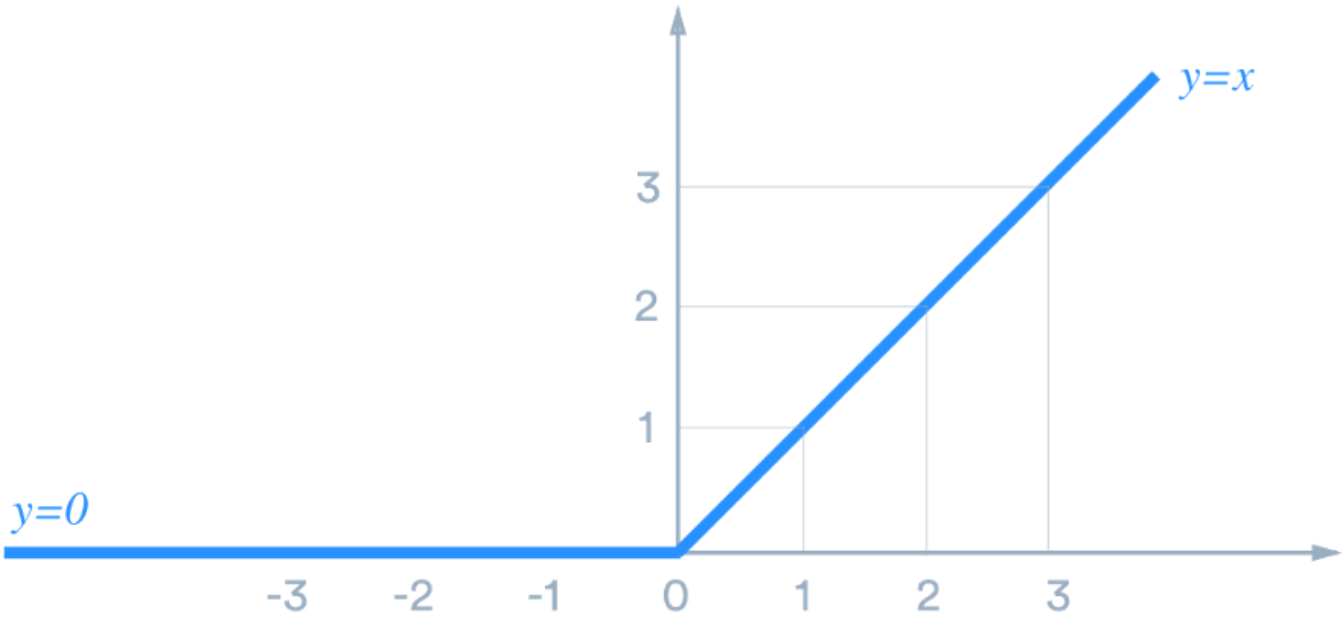
Esse processo é utilizado porque essas imagens resultantes podem conter elementos que facilitam a identificação da classe alvo para a rede. Uma CNN utiliza esse processo nas suas camadas iniciais, com a diferença de que a *matrix* de *kernel* não possui valores

"chumbados" como no exemplo acima, ou seja: eles são parâmetros treinados pelo algoritmo.

ReLU

Uma rede neural sem função de ativação torna-se um modelo linear. Se o seu problema é linear, existem outros modelos mais simples que te atenderão tão bem quanto uma rede neural. Infelizmente a maioria dos problemas complexos não são lineares. Portanto, para adicionar a não linearidade a rede, utilizamos as funções de ativação. Nos dias de hoje, e principalmente no contexto de imagens, a mais utilizada é a função ReLU.

Matematicamente a função ReLU é definida como $y = max(0, x)$. O gráfico a seguir é a ilustração desta função.



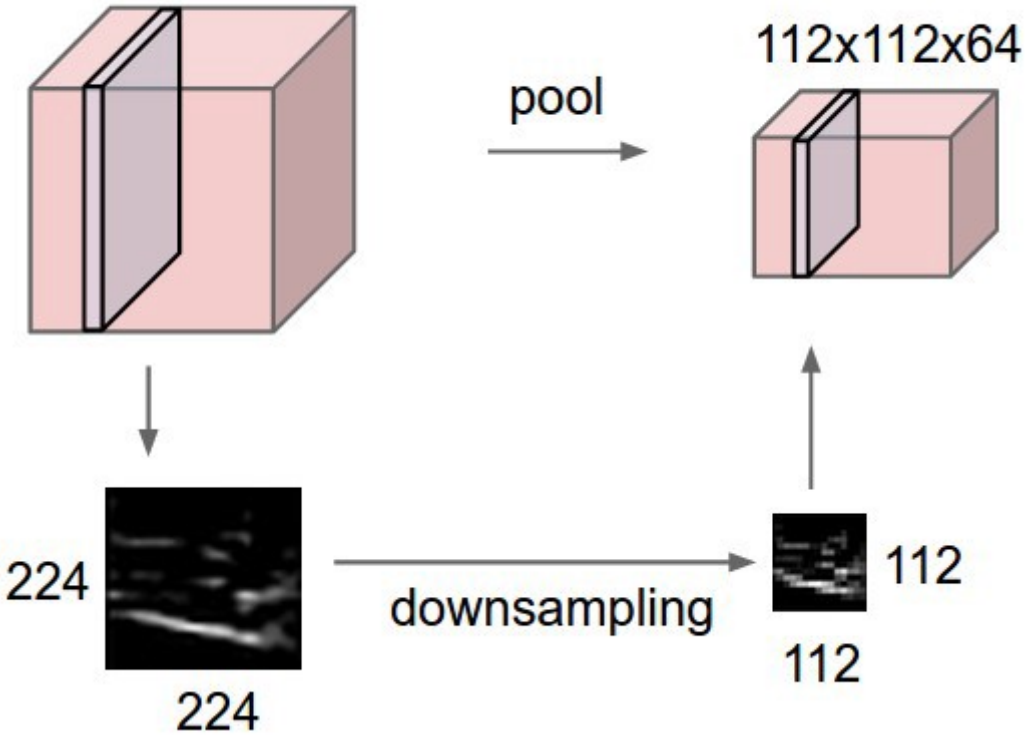
Função de Ativação ReLU: Fonte — Medium

Pooling

Pooling é um processo de *downsamping*. É um processo simples de redução da dimensionalidade/*features maps*. Em uma forma leviana de pensar, podemos entender essa transformação como uma redução do tamanho da imagem.

A principal motivação dessa operação no modelo, é de diminuir sua variância a pequenas alterações e também de reduzir a quantidade de parâmetros treinados pela rede.

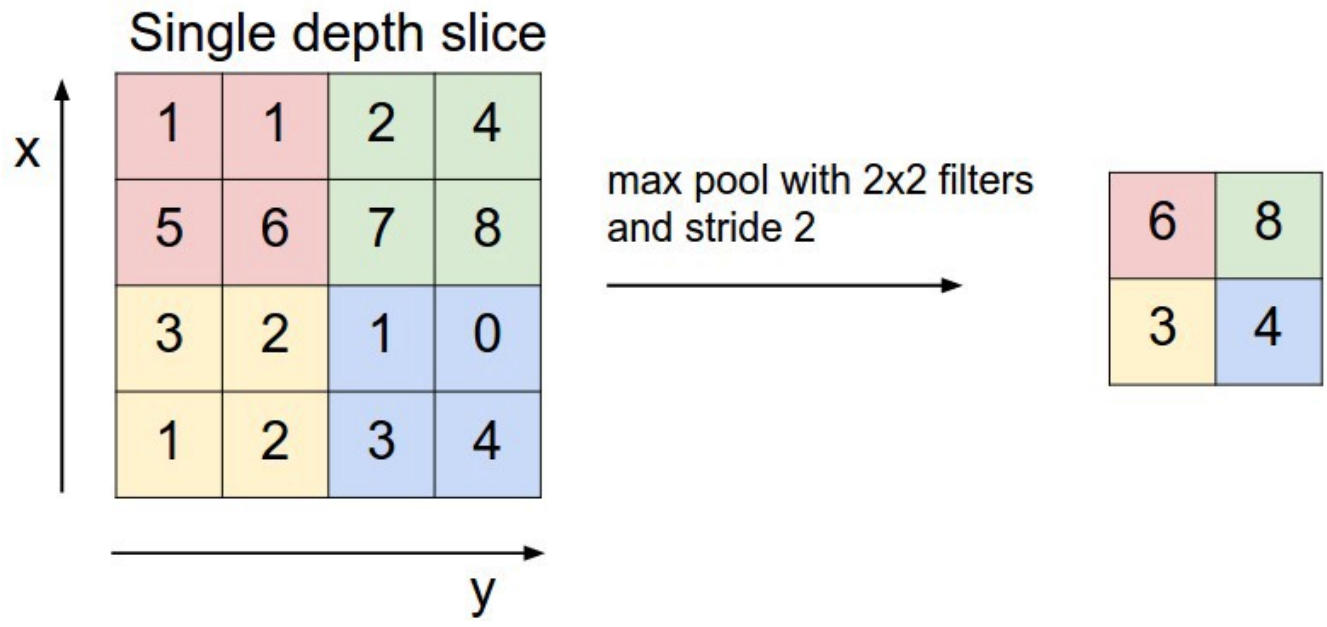
224x224x64



Exemplo de downsampling — Fonte: CS231n

Existem 3 operações diferentes de *Pooling* (*MaxPooling*, *SumPooling*, *AvaragePooling*). Todas elas seguem o mesmo princípio e só se diferem na forma como calculam o valor final. A mais utilizada nos dias de hoje é a *MaxPooling*.

A operação de *MaxPooling* retira o maior elemento de determinada região da *matrix* (considerando o tamanho do *pool* aplicado). Posteriormente, é feito um deslizamento considerando um parâmetro de *stride (*similar a a operação de convolução) para aplicação de uma nova operação.



MaxPooling — Fonte: CS231n

Dropout

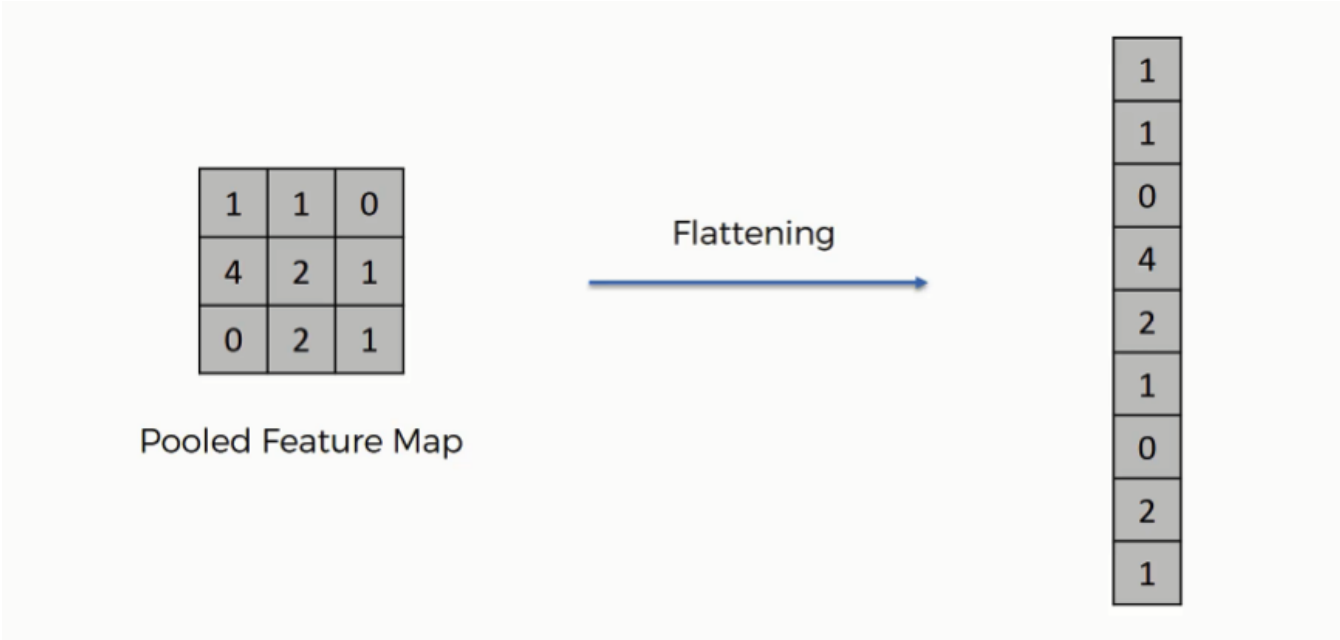
Dropout não é uma especificidade de uma CNN, porém a utilizaremos em nossa implementação técnica, portanto abordaremos seu funcionamento.

Em resumo, a camada de *Dropout* é utilizada para evitar que determinadas partes da rede neural tenham muita responsabilidade e consequentemente, possam ficar muito sensíveis a pequenas alterações.

Essa camada recebe um hyper-parâmetro que define uma probabilidade de "desligar" determinada área da rede neural durante o processo de treinamento.

Flatten

Essa camada normalmente é utilizada na divisão das 2 partes da CNN (extração de características / rede neural tradicional). Ela basicamente opera uma transformação na *matrix* da imagem, alterando seu formato para um array. Por exemplo, uma imagem em *grayscale* de 28x28 será transformada para um array de 784 posições. A imagem abaixo ilustra essa operação.



Operação de Flatten — Fonte: SuperDataScience

Rede Tradicional (*Dense Layers*)

Rede neural é um modelo computacional baseado no sistema nervoso central humano. Elas são capazes de reconhecer padrões em uma massa de dados de forma a classificá-los em alguma categoria ou fazer a regressão de algum valor. Por já existir um material

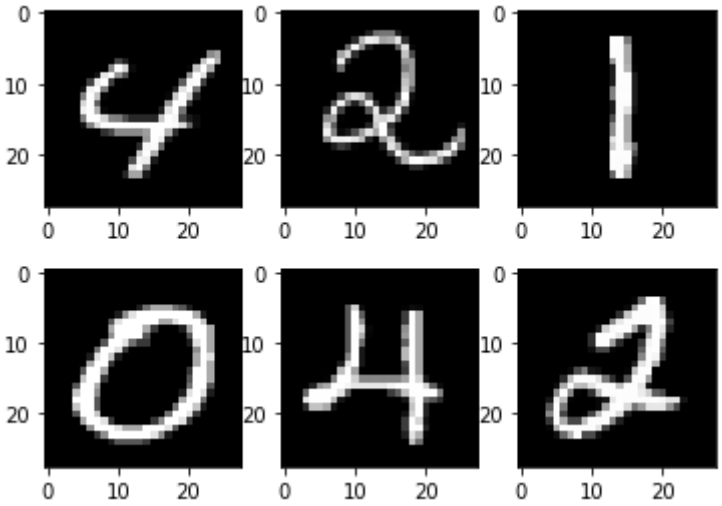
muito bom sobre redes neurais aqui no DataHackers, se você não tiver familiaridade com o assunto, aconselho dar uma olhada nesse artigo aqui — Introdução teórica a neural network.

Implementação

O problema abordado neste artigo é um desafio de reconhecimento de dígitos escritos a punho por humanos. Tais imagens do desafio encontram-se no formato *grayscale* e são representados por uma *matrix* 28x28. Vamos importar a massa de dados do *Kaggle* e visualizar algumas imagens.

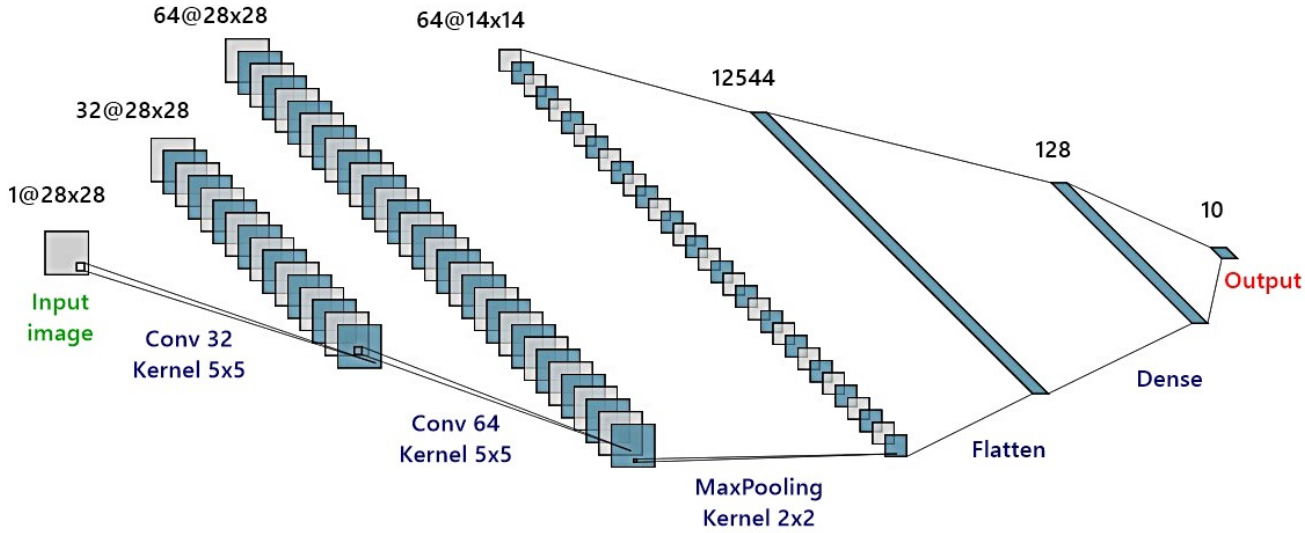
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from keras.utils import np_utils

#setamos o seed para reprodução do experimento
np.random.seed(2)

df_train = pd.read_csv('train.csv')
df_test = pd.read_csv('test.csv')

#retiramos a informação do digito
x_train = df_train.drop(["label"], axis=1).values
#apesar do dataset ja estar no formato 28x28, o framework do keras espera que seja
#informado a terceira dimensão,portanto já redimensionamentos para 28x28x1.
x_train = x_train.reshape((x_train.shape[0], 28, 28, 1))
x_test = df_test.values.reshape((df_test.shape[0], 28, 28, 1))

# utilizamos a função to_categorial do utils do keras para fazermos o one-hot-encoder d
y_train = df_train["label"].values
y_train = np_utils.to_categorical(y_train)

#visualizando randomicamente algumas imagens
for i in range(0, 6):
    random_num = np.random.randint(0, len(x_train))
    img = x_train[random_num]
    plt.subplot(3,2,i+1)
    plt.imshow(img.reshape(28, 28), cmap=plt.get_cmap('gray'))
plt.subplots_adjust(top=1.4)
plt.show()
```

digit-recognizer-import-data.py hosted with ❤ by **GitHub** **view raw**



Visualização randômica das imagens

Vamos implementar uma CNN com a topologia abaixo.



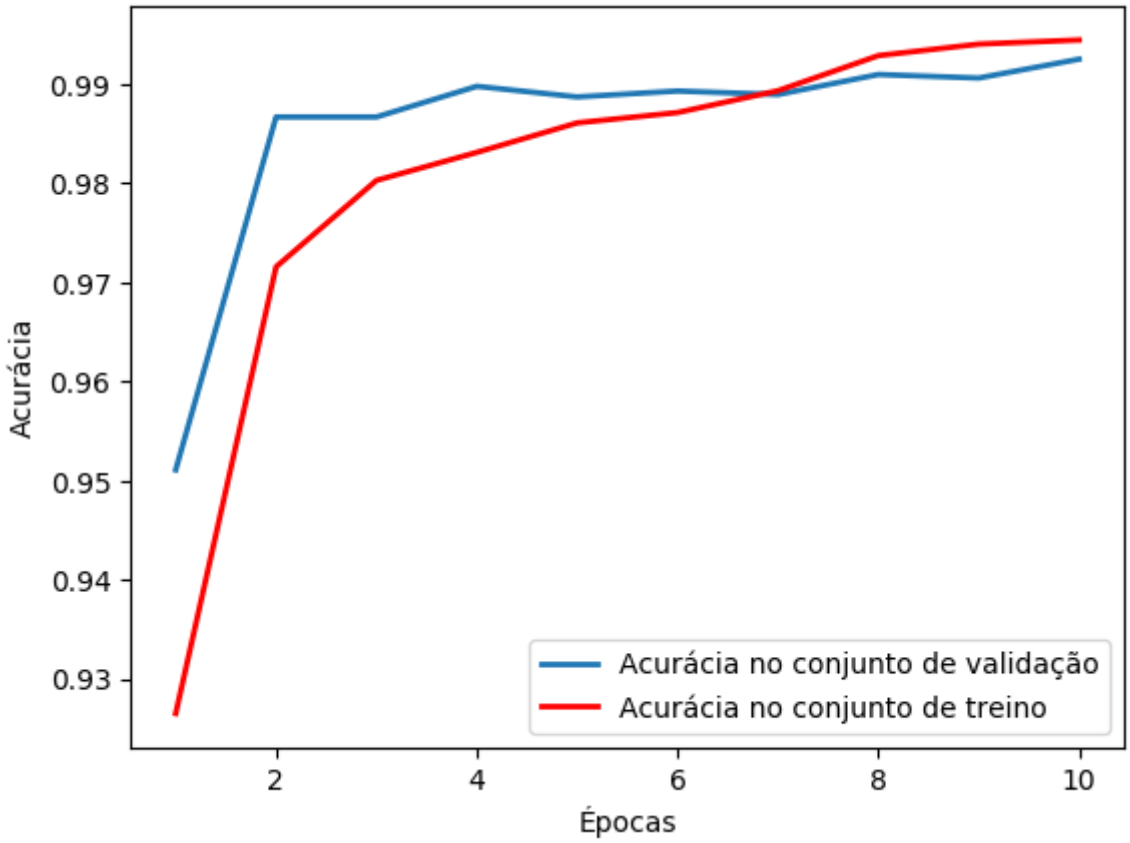
Topologia CNN

Inicialmente é aplicado um processo de convolução com 32 filtros na imagem de entrada. Podemos entender esse processo como a geração de 32 novas imagens a partir da entrada. Analogamente, é aplicado outro processo de convolução com 64 filtros. Na camada subsequente, é aplicado a operação de *MaxPooling*, que reduzirá o tamanho da imagem em 14x14. Finalmente, esses valores são unificados em um *array* pela operação *Flatten* (64x14x14 = 12544) para serem a entrada de uma rede neural tradicional.

```
1  from keras.models import Sequential
2  from keras.layers import Dense, Dropout, Flatten
3  from keras.layers import Conv2D, MaxPooling2D
```

```
from keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ReduceLROnPlateau

## dado que o range de valores possível pra um pixél vai de 0-255
## escalonamos os valores entre 0-1
## esse processo torna nosso modelo menos variante a pequenas alterações.
x_train = x_train / 255
x_test = x_test / 255


model = Sequential()
model.add(Conv2D(32, (5,5), activation='relu', padding='same', input_shape=(28, 28,1)))
model.add(Conv2D(64, (5,5), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))

optimizer = Adam()
model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy
print(model.summary())

# reduz o parâmetro de learning rate se não houver
# melhoras em determinado número de epocas
# útil para encontrar o mínimo global.
learning_rate_reduction = ReduceLROnPlateau(monitor='val_acc',
                                            patience=3,
                                            verbose=1,
                                            factor=0.5,
                                            min_lr=0.00001)

batch_size = 32
epochs = 10

history = model.fit(x_train,
                        y_train,
                        batch_size = batch_size,
                        epochs = epochs,
                        validation_split=0.2,
                        verbose = 1,
                        callbacks=[learning_rate_reduction])

history_dict = history.history
acc = history_dict['acc']
val_acc = history_dict['val_acc']
```

```
range_epochs = range(1, len(acc) + 1)

plt.style.use('default')
accuracy_val = plt.plot(range_epochs, val_acc, label='Acurácia no conjunto de validaçã
accuracy_train = plt.plot(range_epochs, acc, label='Acurácia no conjunto de treino', c
plt.setp(accuracy_val, linewidth=2.0)
plt.setp(accuracy_train, linewidth=2.0)
plt.xlabel('Épocas')
plt.ylabel('Acurácia')
plt.legend(loc="lower right")
plt.show()
```



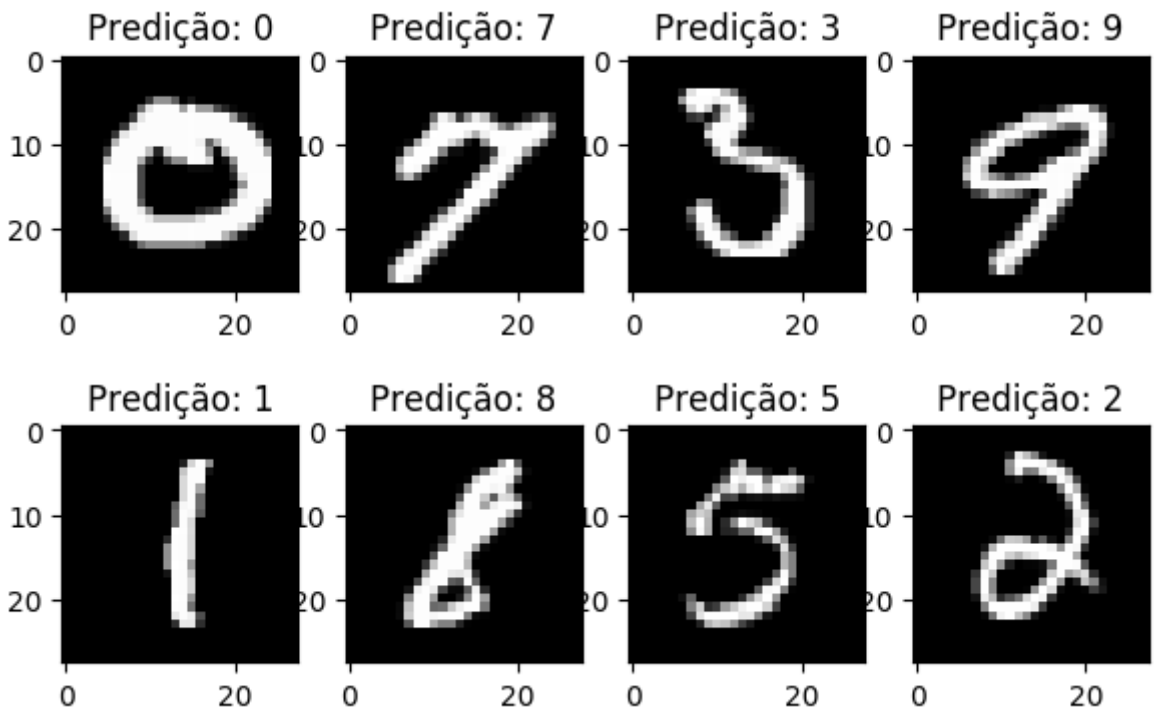
Acurácia durante as épocas

Finalmente, vamos visualizar algumas predições por curiosidade e depois gerar a resposta para o Kaggle.

```
predictions = model.predict_classes(x_test)

plt.figure(figsize=(7,14))
for i in range(0, 8):
    random_num = np.random.randint(0, len(x_test))
    img = x_test[random_num]
    plt.subplot(6,4,i+1)
```

```
    plt.margins(x = 20, y = 20)
    plt.title('Predição: ' + str(predictions[random_num]))
    plt.imshow(img.reshape(28, 28), cmap=plt.get_cmap('gray'))
plt.show()

submission = pd.DataFrame({'ImageID' : pd.Series(range(1,28001)), 'Label' : predictions
submission.to_csv("submission.csv",index=False)
```

digit-recognizer-submit.py hosted with ❤ by **GitHub** **view raw**



Apresentação de predição para algumas imagens.

Poderíamos melhorar essa solução com a utilização de técnicas como *Data Augmentation* e alguns outros truques, mas esse papo vai ficar pra outro artigo :).

Eai, gostou? Não deixe de acompanhar esse *Medium* e as redes sociais da EurekaLabs (Linkedin, Facebook) para acompanhar os novos artigos! Até lá!

Thanks to Isabela Vendramini.

Machine Learning Convolutional Network Image Processing Data Science

Artificial Intelligence