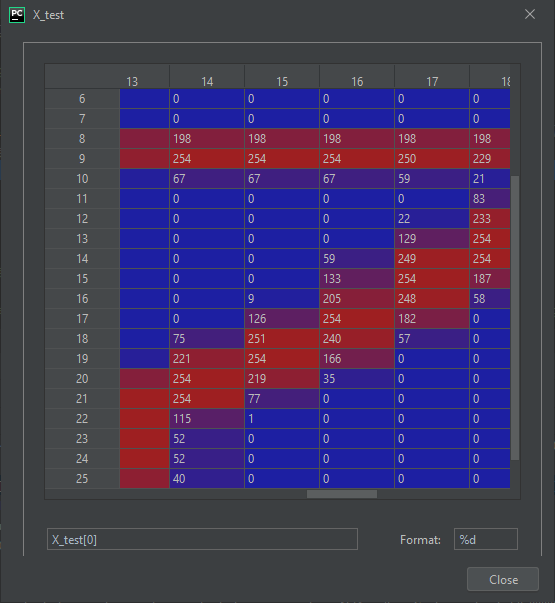
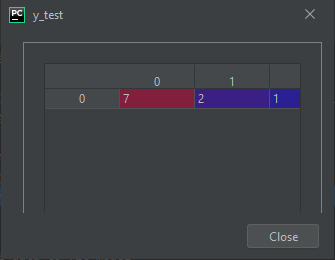
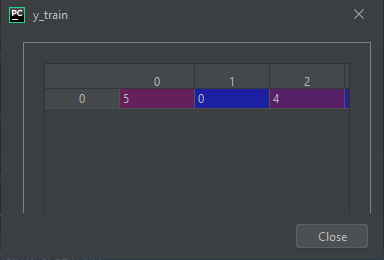
Com o objetivo de melhor compreender CNN, utilizou-se do artigo escrito por Eijaz Allibhai, do fórum de notícias e artigos sobre ciência de dados *Towards Data Science* (INSERIR BIBLIOGRAFIA CERTA) (<https://towardsdatascience.com/building-a-convolutional-neural-network-cnn-in-keras-329fbbadc5f5>). O artigo explica cada passo na utilização da biblioteca Keras para o desenvolvimento de uma CNN. Julga-se relevante citar o processo de análise do código disposto no artigo, o qual foi testado e compreendido pela equipe deste projeto. Tal estudo precede o produto final, cujo código é baseado neste do artigo. Veja que cada segmento do código no bloco XX foi comentado e associado com outros conteúdos de referência deste projeto.

|  |
| --- |
| from keras.datasets import mnist import matplotlib.pyplot as plt from keras.utils import to\_categorical from keras.models import Sequential from keras.layers import Dense, Conv2D, Flatten  """  Fonte: https://towardsdatascience.com/building-a-convolutional-neural-network-cnn-in-keras-329fbbadc5f5 """  """ Carregamento da base de dados da biblioteca Keras. Em mnist são 60.000 imagens para treinamento, e 10.000, para teste; sendo assim 70.000 imagens na totalidade. X\_train e X\_test representam vetores de matrizes com o desenho de números, detalhando em escala de cinza cada pixel das  imagens. y\_train e y\_test representam vetores com os valores corretos correspondentes às matrizes de X. Assim, X têm 3 dimensões, enquanto y têm apenas 1. """ (X\_train, y\_train), (X\_test, y\_test) = mnist.load\_data() # Correção da escala dos valores. Valor máximo deve ser 1, e não 255. X\_train, X\_test = X\_train / 255.0, X\_test / 255.0 # Impressão gráfica de um dos elementos da entrada plt.imshow(X\_train[0])   """ As linhas de código abaixo mudam o arranjo das matrizes contendo as imagens. Isso é necessário por conta do formato ao  qual o modelo espera como entrada. Caso o conjunto de imagens fosse colorido, o último parâmetro do método reshape()  seria diferente de 1. Quanto aos outros parâmetros deste método, tem-se:  -Primeiro parâmetro é a quantidade de imagens;  -Segundo parâmetro é a dimensão das imagens. Vale ressaltar que não houve mudança na quantidade de imagens aqui, nem perda de resolução. """ X\_train = X\_train.reshape(60000,28,28,1) X\_test = X\_test.reshape(10000,28,28,1)   """ Torna as variáveis-resposta em vetores binários. Este passo é essencial para adequação ao modelo, que prevê uma saída  binária, ainda que com múltiplos nós de saída. O autor usa o termo "one-hot encode target column", que significa  basicamente as saídas binárias. """ y\_train = to\_categorical(y\_train) y\_test = to\_categorical(y\_test) y\_train[0]  """ Modelo usado aqui é o sequencial. Permite construir o modelo camada a camada. Sendo assim, o método add() nos permite adicionar tais camadas ao modelo. Kernel size significa o tamanho da matriz de convolução.  Segmento de código para construir e configurar o modelo a ser usado. Utiliza-se uma instância do modelo sequencial. A característica deste modelo é a adição sequencial de camadas. As camadas aqui adicionadas são recomendações do autor do artigo. Vale ressaltar que existe extenso estudo a respeito do número de nós e outros parâmetros aqui utilizados, mas entende-se que sairia do escopo do projeto estudar quais  seriam valores ótimos para cada caso, então segue-se com a recomendação do autor. Quanto aos outros parâmetros de  entrada, bem como as camadas utilizadas, vale listar cada aspecto.  -Método add() da classe Sequential() é aquele que permite a adição das camadas.  -Parâmetro de entrada kernel\_size significa o tamanho da matriz de convolução, em outras palavras, é o tamanho do   filtro que percorrerá cada pixel da imagem.  -Parâmetro de entrada activation significa a curva de da função de ativação; neste caso utilizou-se a ReLU, cuja   função matemática é R(z) = max(0, z), e é a mais utilizada para CNN. Para a última camada, foi utilizada a curva   de ativação softmax, cuja forma é uma transição suave entre 0 e 1, podendo traduzir na probabilidade de dedução do   modelo, que deve variar entre 0e 100%.  -Note que input\_shape representa as dimensões de cada imagem nas matrizes de entrada, após o uso da função   reshape().  -Conv2D() são camadas de matrizes bidimencionais de convolução.  -Flatten() representa uma camada de adequação entre as camadas Conv2D e Dense; em inglês significa achatamento.  -Dense() é uma camada unidimensional frequentemente usada em redes neurais; neste caso oferece 10 saídas possíveis,  cada qual com uma probabilidade associada. Idealmente apenas uma das saídas deve apresentar valor de probabilidade  elevado. """ # Criando uma instância de modelo sequencial (permite adição sequencial de camadas). model = Sequential() # Adicionando as camadas model.add(Conv2D(filters=64, kernel\_size=3, activation='relu', input\_shape=(28,28,1))) model.add(Conv2D(filters=32, kernel\_size=3, activation='relu')) model.add(Flatten()) model.add(Dense(units=10, activation='softmax'))   #TODO terminar essa porra """ Trecho abaixo faz a junção das camadas do modelo. Perceba que a natureza do modelo prevê primeiro a adição de camadas, seguida pela junção através de método da classe respectiva. Os parâmetros aqui utilizados são:  -optimizer: parâmetro responsável pelo controle no ritmo de aprendizado. O autor recomenda o 'adam' e diz que há um  trade-off entre agilidade e acurácia.  -loss:  -metrics:  """  #compile model using accuracy to measure model performance model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])  """ Finalmente se faz o treinamento do modelo. Aqui se utiliza a entrada e a saída providas pela biblioteca mnist do keras. Também se coloca os conjuntos de teste. "epochs" representa o número de iterações para o modelo, cuja recomendação do  autor aqui é 3; o aumento deste valor costuma melhorar o poder preditivo do modelo, mas também com maior tempo de pro- cessamento, até determinado ponto, onde pode ocorrer "overfitting". """ model.fit(X\_train, y\_train, validation\_data=(X\_test, y\_test), epochs=3)  """ A partir daqui o modelo já está treinado e pronto para uso. Fazendo predições para as 4 primeiras imagens no conjunto de imagens separadas para teste. O método abaixo devolve um  vetor de predição. """ model.predict(X\_test[:4])  """ Averiguando o conjunto de 4 imagens. """ y\_test[:4] |



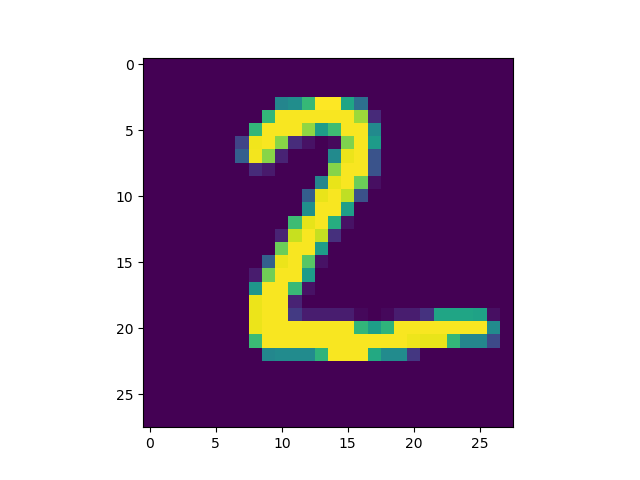
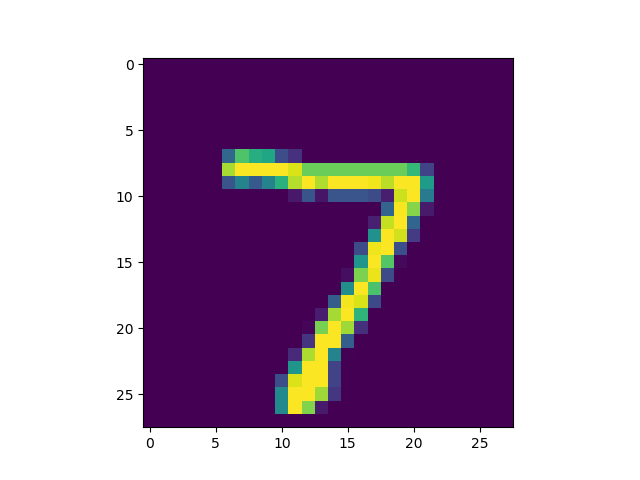
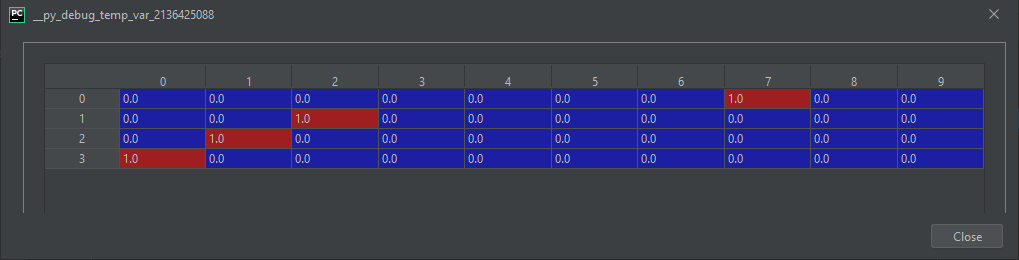


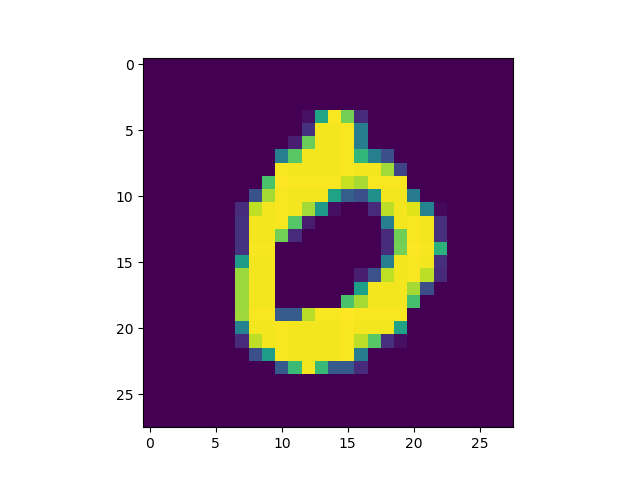
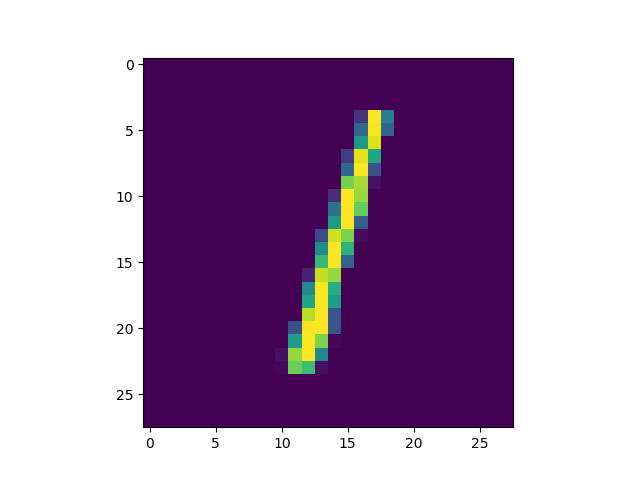




Saída da linha de predição

Gabarito





|  |
| --- |
| from keras.datasets import mnist import matplotlib.pyplot as plt from keras.utils import to\_categorical from keras.models import Sequential from keras.layers import Dense, Conv2D, Flatten  """ https://towardsdatascience.com/building-a-convolutional-neural-network-cnn-in-keras-329fbbadc5f5 """  """ Carregamento da base de dados da biblioteca Keras. Em mnist são 60.000 imagens para treinamento, e 10.000, para teste; sendo assim 70.000 imagens na totalidade. X\_train e X\_test representam vetores de matrizes com o desenho de números, detalhando em escala de cinza cada pixel das  imagens. y\_train e y\_test representam vetores com os valores corretos correspondentes às matrizes de X. Assim, X têm 3 dimensões, enquanto y têm apenas 1. """ (X\_train, y\_train), (X\_test, y\_test) = mnist.load\_data() X\_train, X\_test = X\_train / 255.0, X\_test / 255.0 # Impressão gráfica de um dos elementos da entrada plt.imshow(X\_train[0])   """ As linhas de código abaixo mudam o arranjo das matrizes contendo as imagens. Isso é necessário por conta do formato ao  qual o modelo espera como entrada. Caso o conjunto de imagens fosse colorido, o último parâmetro do método reshape()  seria diferente de 1. Quanto aos outros parâmetros deste método, tem-se:  -Primeiro parâmetro é a quantidade de imagens;  -Segundo parâmetro é a dimensão das imagens. Vale ressaltar que não houve mudança na quantidade de imagens aqui, nem perda de resolução. """ X\_train = X\_train.reshape(60000,28,28,1) X\_test = X\_test.reshape(10000,28,28,1)   """ Torna as variáveis-resposta em vetores binários. Este passo é essencial para adequação ao modelo, que prevê uma saída  binária, ainda que com múltiplos nós de saída. O autor usa o termo "one-hot encode target column", que significa  basicamente as saídas binárias. """ y\_train = to\_categorical(y\_train) y\_test = to\_categorical(y\_test) y\_train[0]  """ Modelo usado aqui é o sequencial. Permite construir o modelo camada a camada. Sendo assim, o método add() nos permite adicionar tais camadas ao modelo. Kernel size significa o tamanho da matriz de convolução.  Segmento de código para construir e configurar o modelo a ser usado. Utiliza-se uma instância do modelo sequencial. A característica deste modelo é a adição sequencial de camadas. As camadas aqui adicionadas são recomendações do autor do artigo. Vale ressaltar que existe extenso estudo a respeito do número de nós e outros parâmetros aqui utilizados, mas entende-se que sairia do escopo do projeto estudar quais  seriam valores ótimos para cada caso, então segue-se com a recomendação do autor. Quanto aos outros parâmetros de  entrada, bem como as camadas utilizadas, vale listar cada aspecto.  -Método add() da classe Sequential() é aquele que permite a adição das camadas.  -Parâmetro de entrada kernel\_size significa o tamanho da matriz de convolução, em outras palavras, é o tamanho do   filtro que percorrerá cada pixel da imagem.  -Parâmetro de entrada activation significa a curva de da função de ativação; neste caso utilizou-se a ReLU, cuja   função matemática é R(z) = max(0, z), e é a mais utilizada para CNN. Para a última camada, foi utilizada a curva   de ativação softmax, cuja forma é uma transição suave entre 0 e 1, podendo traduzir na probabilidade de dedução do   modelo, que deve variar entre 0e 100%.  -Note que input\_shape representa as dimensões de cada imagem nas matrizes de entrada, após o uso da função   reshape().  -Conv2D() são camadas de matrizes bidimencionais de convolução.  -Flatten() representa uma camada de adequação entre as camadas Conv2D e Dense; em inglês significa achatamento.  -Dense() é uma camada unidimensional frequentemente usada em redes neurais; neste caso oferece 10 saídas possíveis,  cada qual com uma probabilidade associada. Idealmente apenas uma das saídas deve apresentar valor de probabilidade  elevado. """ # Criando uma instância de modelo sequencial (permite adição sequencial de camadas). model = Sequential() # Adicionando as camadas model.add(Conv2D(filters=64, kernel\_size=3, activation='relu', input\_shape=(28,28,1))) model.add(Conv2D(filters=32, kernel\_size=3, activation='relu')) model.add(Flatten()) model.add(Dense(units=10, activation='softmax'))   #*TODO terminar essa porra* """ Trecho abaixo faz a junção das camadas do modelo. Perceba que a natureza do modelo prevê primeiro a adição de camadas, seguida pela junção através de método da classe respectiva. Os parâmetros aqui utilizados são:  -optimizer: parâmetro responsável pelo controle no ritmo de aprendizado. O autor recomenda o 'adam' e diz que há um  trade-off entre agilidade e acurácia.  -loss:  -metrics:  """ #compile model using accuracy to measure model performance model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])  """ Finalmente se faz o treinamento do modelo. Aqui se utiliza a entrada e a saída providas pela biblioteca mnist do keras. Também se coloca os conjuntos de teste. "epochs" representa o número de iterações para o modelo, cuja recomendação do  autor aqui é 3; o aumento deste valor costuma melhorar o poder preditivo do modelo, mas também com maior tempo de pro- cessamento, até determinado ponto, onde pode ocorrer "overfitting". """ model.fit(X\_train, y\_train, validation\_data=(X\_test, y\_test), epochs=3)  """ A partir daqui o modelo já está treinado e pronto para uso. Fazendo predições para as 4 primeiras imagens no conjunto de imagens separadas para teste. O método abaixo devolve um  vetor de predição. """ model.predict(X\_test[:4])  """ Averiguando o conjunto de 4 imagens. """ y\_test[:4] |