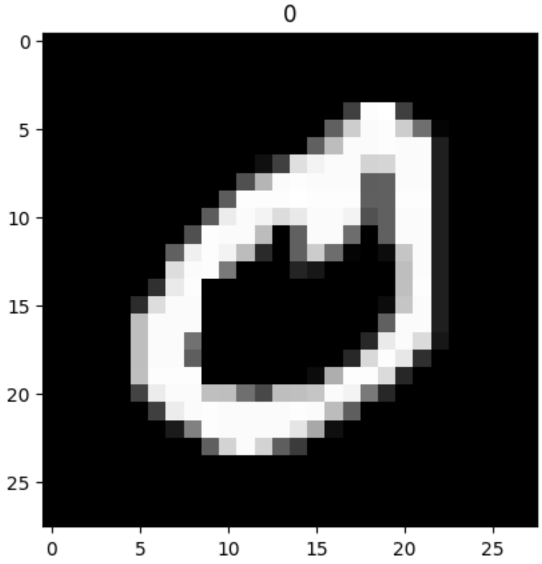
```
# trabalho MNIST
# Afonso Brandão 20230403


# Bibliotecas (no COLAB o tensorflow já está instalado)
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.utils import np_utils
import numpy as np
from sklearn.metrics import confusion_matrix
from keras.datasets import mnist


# Dados e divisão de testes
(X_treinamento, y_treinamento), (X_teste, y_teste) = mnist.load_data()

# Teste de imagem baixada
plt.imshow(X_treinamento[21], cmap = 'gray')
plt.title(y_treinamento[21])
```

```
Text(0.5, 1.0, '0')
```



```
# Redimensionamento para 784
X_treinamento = X_treinamento.reshape((len(X_treinamento), np.prod(X_treinamento.shape[1:])))
X_teste = X_teste.reshape((len(X_teste), np.prod(X_teste.shape[1:])))
X_teste[0]
```

```
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,  84, 185, 159, 151,  60,  36,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0, 222, 254, 254, 254,
254, 241, 198, 198, 198, 198, 198, 198, 198, 198, 170,  52,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  67, 114,
 72, 114, 163, 227, 254, 225, 254, 254, 254, 250, 229, 254, 254,
140,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,  17,  66,  14,  67,  67,  67,  59,  21,
236, 254, 106,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,  83, 253, 209,  18,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,  22, 233, 255,  83,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0, 129, 254, 238,  44,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,  59, 249, 254,  62,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 126, 254, 182,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  75, 251,
240,  57,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  19,
221, 254, 166,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  3, 203, 254, 219,  35,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,  38, 254, 254,  77,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,  31, 224, 254, 115,   1,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0, 133, 254, 254,  52,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,  61, 242, 254, 254,  52,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0, 121, 254, 254, 219,  40,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 121, 254, 207,
 18,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0], dtype=uint8)
```

```
# Transformação e normalização dos dados
X_treinamento = X_treinamento.astype('float32')
X_teste = X_teste.astype('float32')
X_treinamento /= 255
X_teste /= 255

# Dummy de 10 classes
y_treinamento = np_utils.to_categorical(y_treinamento, 10)
y_teste = np_utils.to_categorical(y_teste, 10)
y_teste[0]
```

```
    array([0., 0., 0., 0., 0., 0., 0., 1., 0., 0.], dtype=float32)
```

```
# Estrutura da rede neural: 784 - 64 - 64 - 64 - 10
# Dropout é utilizado para zerar uma porcentagem dos neurônios, para evitar o overfitting
modelo = Sequential()
modelo.add(Dense(units = 64, activation = 'relu', input_dim = 784))
modelo.add(Dropout(0.2))
modelo.add(Dense(units = 64, activation = 'relu'))
modelo.add(Dropout(0.2))
modelo.add(Dense(units = 64, activation = 'relu'))
modelo.add(Dropout(0.2))
#camada de saida, softmax probabilidade
modelo.add(Dense(units = 10, activation = 'softmax'))

# Visualização da estrutura da rede neural
modelo.summary()
```

```
    Model: "sequential_1"
    _________________________________________________________________
     Layer (type)                Output Shape              Param #   
    =================================================================
     dense_4 (Dense)             (None, 64)                50240     
                                                                     
     dropout_3 (Dropout)         (None, 64)                0         
                                                                     
     dense_5 (Dense)             (None, 64)                4160      
                                                                     
     dropout_4 (Dropout)         (None, 64)                0         
                                                                     
     dense_6 (Dense)             (None, 64)                4160      
                                                                     
     dropout_5 (Dropout)         (None, 64)                0         
                                                                     
     dense_7 (Dense)             (None, 10)                650       
                                                                     
    =================================================================
    Total params: 59,210
    Trainable params: 59,210
    Non-trainable params: 0
    _________________________________________________________________
```

```
# Configuração dos parâmetros da rede neural e treinamento (utilizando base de dados de validação)
# Na variável historico temos os histórico das execuções (erro e accuracy)
modelo.compile(optimizer = 'adam', loss = 'categorical_crossentropy',
               metrics = ['accuracy'])
historico = modelo.fit(X_treinamento, y_treinamento, epochs = 20,
                       validation_data = (X_teste, y_teste))
```

```
Epoch 1/20
1875/1875 [==============================] - 10s 3ms/step - loss: 1.0293 - accuracy: 0.6378 - val_loss: 0.5331 - val_acc
Epoch 2/20
1875/1875 [==============================] - 10s 5ms/step - loss: 0.5449 - accuracy: 0.8373 - val_loss: 0.3562 - val_acc
Epoch 3/20
1875/1875 [==============================] - 7s 4ms/step - loss: 0.4316 - accuracy: 0.8752 - val_loss: 0.3047 - val_accu
Epoch 4/20
1875/1875 [==============================] - 6s 3ms/step - loss: 0.3672 - accuracy: 0.8934 - val_loss: 0.2448 - val_accu
Epoch 5/20
1875/1875 [==============================] - 7s 4ms/step - loss: 0.3322 - accuracy: 0.9043 - val_loss: 0.2219 - val_accu
Epoch 6/20
1875/1875 [==============================] - 7s 4ms/step - loss: 0.3038 - accuracy: 0.9114 - val_loss: 0.2065 - val_accu
Epoch 7/20
1875/1875 [==============================] - 7s 4ms/step - loss: 0.2822 - accuracy: 0.9185 - val_loss: 0.1949 - val_accu
Epoch 8/20
1875/1875 [==============================] - 9s 5ms/step - loss: 0.2686 - accuracy: 0.9226 - val_loss: 0.1789 - val_accu
Epoch 9/20
1875/1875 [==============================] - 10s 5ms/step - loss: 0.2547 - accuracy: 0.9270 - val_loss: 0.1689 - val_acc
Epoch 10/20
1875/1875 [==============================] - 10s 6ms/step - loss: 0.2476 - accuracy: 0.9289 - val_loss: 0.1611 - val_acc
Epoch 11/20
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2370 - accuracy: 0.9316 - val_loss: 0.1599 - val_accu
Epoch 12/20
1875/1875 [==============================] - 7s 4ms/step - loss: 0.2310 - accuracy: 0.9343 - val_loss: 0.1576 - val_accu
Epoch 13/20
1875/1875 [==============================] - 8s 4ms/step - loss: 0.2219 - accuracy: 0.9346 - val_loss: 0.1490 - val_accu
Epoch 14/20
1875/1875 [==============================] - 7s 4ms/step - loss: 0.2178 - accuracy: 0.9367 - val_loss: 0.1459 - val_accu
Epoch 15/20
1875/1875 [==============================] - 7s 4ms/step - loss: 0.2114 - accuracy: 0.9366 - val_loss: 0.1441 - val_accu
Epoch 16/20
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2061 - accuracy: 0.9396 - val_loss: 0.1499 - val_accu
Epoch 17/20
1875/1875 [==============================] - 7s 4ms/step - loss: 0.2028 - accuracy: 0.9406 - val_loss: 0.1431 - val_accu
Epoch 18/20
1875/1875 [==============================] - 6s 3ms/step - loss: 0.1995 - accuracy: 0.9413 - val_loss: 0.1409 - val_accu
Epoch 19/20
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1968 - accuracy: 0.9426 - val_loss: 0.1366 - val_accu
Epoch 20/20
1875/1875 [==============================] - 6s 3ms/step - loss: 0.1930 - accuracy: 0.9423 - val_loss: 0.1381 - val_accu
```

```
# Gráfico para visualizar os erros e accuracy
historico.history.keys()
#evolução do erro, azul
plt.plot(historico.history['val_loss'])
#performance da rede
plt.plot(historico.history['val_accuracy'])

# Obtenção das previsões
previsoes = modelo.predict(X_teste)
previsoes
```

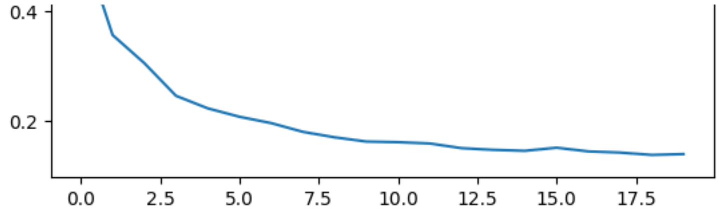
```
    313/313 [==============================] - 1s 2ms/step
    array([[9.32211253e-11, 6.91008563e-06, 2.67957017e-04, ...,
            9.99656439e-01, 2.13234046e-07, 1.00167608e-05],
           [3.20645199e-08, 5.58556167e-05, 9.98079538e-01, ...,
```

```
# Matriz confusão
y_teste_matriz = [np.argmax(t) for t in y_teste]
y_previsoes_matriz = [np.argmax(t) for t in previsoes]
confusao = confusion_matrix(y_teste_matriz, y_previsoes_matriz)
confusao

# Previsão com um novo registro, convertendo o array para o formato de matriz
#número 4
y_treinamento[20]

#passo a mesma posição para o modelo prever
novo = X_treinamento[20]
#de matriz para vetor
novo = np.expand_dims(novo, axis = 0)
#previsao
pred = modelo.predict(novo)
#maior valor
pred = [np.argmax(pred) for t in pred]
pred
```

```
    1/1 [==============================] - 0s 23ms/step
    [4]
```



Produtos pagos do Colab - Cancelar contratos

✓ 0s conclusão: 00:46