Exercício 13 - CNN

Aluno: Giovanni Martins de Sá Júnior - 2017001850

Neste exercício, foi utilizada uma implementação de uma arquitetura CNN feita em R disponibilizada pelo professor. Nesse sentido, o algoritmo foi ajustado para o Python, e nele, foram experimentadas algumas alterações em sua implementação, pegando como inspiração, alguns dos artigos listados na própria tarefa.

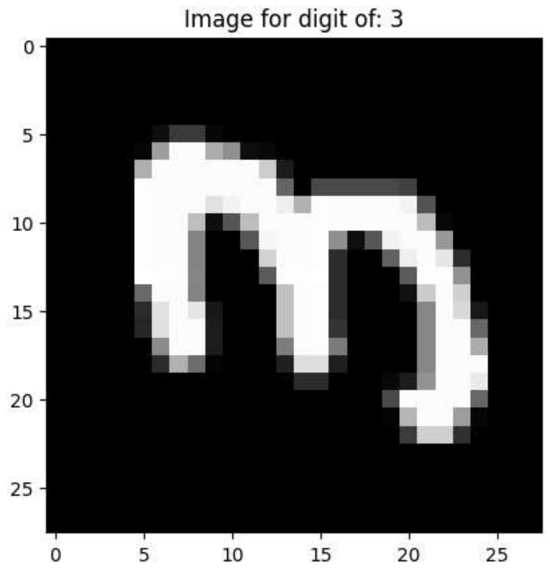
A partir disso, primeiramente foi então modificado a arquitetura da CNN, modificando-se a largura do número de filtros, para permitir que fossem buscadas mais características complexas nos primeiros estágios. Em segundo, aumentou-se a profundidade dos filtros, no intuito de aumentar a capacidade em aprender representações mais ricas. E por último ajustou-se o tamanho do kernel para que a rede capture padrões espaciais mais amplos.

```
import tensorflow as tf
from tensorflow.keras import layers, models
import matplotlib.pyplot as plt
import numpy as np
```

```
# Carregar o conjunto de dados MNIST
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
```

```
# Visualizar uma imagem
index_image = 7  # Altere este índice para ver diferentes imagens
input_matrix = x_train[index_image, :, :]
output_matrix = np.apply_along_axis(np.flip, 1, input_matrix)
output_matrix = np.transpose(output_matrix)
plt.imshow(output_matrix, cmap='gray')
plt.title(f'Image for digit of: {y_train[index_image]}')
plt.show()
```



```
# Parâmetros
batch_size = 64
num_classes = 10
epochs = 5
```

```
# Dimensões da imagem de entrada
img_rows, img_cols = 28, 28
```

```
# Reformular os dados
x_train = x_train.reshape((x_train.shape[0], img_rows, img_cols, 1))
x_test = x_test.reshape((x_test.shape[0], img_rows, img_cols, 1))
input_shape = (img_rows, img_cols, 1)
```

```
# Escala para estabilidade numérica
x_train = x_train / 255.0
x_test = x_test / 255.0


# Converter rótulos para matrizes categóricas
y_train = tf.keras.utils.to_categorical(y_train, num_classes)
y_test = tf.keras.utils.to_categorical(y_test, num_classes)


# Definir a estrutura do modelo CNN
cnn_model = models.Sequential([
    layers.Conv2D(16, kernel_size=(5, 5), activation='relu', input_shape=input_shape),
    # Aumentei a largura para 16, usando kernel_size (5, 5)
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Conv2D(32, kernel_size=(3, 3), activation='relu'),
    # Aumentei a profundidade dos filtros para 32
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Dropout(0.25),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(64, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(num_classes, activation='softmax')
])

cnn_model.summary()
```

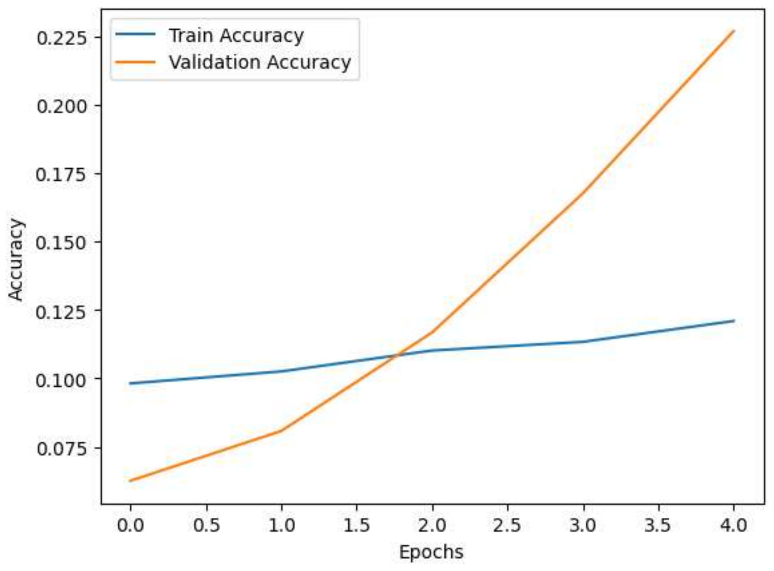
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 24, 24, 16)        416

 max_pooling2d_2 (MaxPoolin  (None, 12, 12, 16)        0
 g2D)

 conv2d_3 (Conv2D)           (None, 10, 10, 32)        4640

 max_pooling2d_3 (MaxPoolin  (None, 5, 5, 32)          0
 g2D)

 dropout_3 (Dropout)         (None, 5, 5, 32)          0

 flatten_1 (Flatten)         (None, 800)               0

 dense_3 (Dense)             (None, 128)               102528

 dropout_4 (Dropout)         (None, 128)               0

 dense_4 (Dense)             (None, 64)                8256

 dropout_5 (Dropout)         (None, 64)                0

 dense_5 (Dense)             (None, 10)                650

=================================================================
Total params: 116490 (455.04 KB)
Trainable params: 116490 (455.04 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
# Compilar o modelo
cnn_model.compile(
    loss=tf.keras.losses.categorical_crossentropy,
    optimizer=tf.keras.optimizers.Adadelta(),
    metrics=['accuracy']
)



# Treinar o modelo
cnn_history = cnn_model.fit(
    x_train, y_train,
    batch_size=batch_size,
    epochs=epochs,
    validation_split=0.2
)
```

```
Epoch 1/5
750/750 [==============================] - 35s 43ms/step - loss: 2.3421 - accuracy: 0.0982 - val_loss: 2.3075 - val_accuracy: 0.062
Epoch 2/5
750/750 [==============================] - 31s 41ms/step - loss: 2.3275 - accuracy: 0.1026 - val_loss: 2.2956 - val_accuracy: 0.080
Epoch 3/5
```

```
     750/750 [==============================] - 35s 47ms/step - loss: 2.3142 - accuracy: 0.1102 - val_loss: 2.2861 - val_accuracy: 0.116
     Epoch 4/5
     750/750 [==============================] - 32s 43ms/step - loss: 2.3068 - accuracy: 0.1134 - val_loss: 2.2780 - val_accuracy: 0.167
     Epoch 5/5
     750/750 [==============================] - 32s 42ms/step - loss: 2.2987 - accuracy: 0.1210 - val_loss: 2.2709 - val_accuracy: 0.226
```

```
# Plotar a curva de aprendizado
plt.plot(cnn_history.history['accuracy'], label='Train Accuracy')
plt.plot(cnn_history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



```
# Avaliar o modelo
cnn_model.evaluate(x_test, y_test)

# Previsões do modelo
cnn_pred = cnn_model.predict(x_test)
print(cnn_pred[:50])
```

```
     313/313 [==============================] - 2s 6ms/step - loss: 2.2709 - accuracy: 0.2275
     313/313 [==============================] - 2s 6ms/step
     [[0.0936554  0.0950572  0.09112116 0.10223257 0.11232971 0.10378505
       0.1029211  0.10721803 0.10046658 0.09121314]
      [0.09725788 0.10594968 0.09637022 0.09751301 0.09778313 0.10430496
       0.10869116 0.10856127 0.09332936 0.09023935]
      [0.10031069 0.10426235 0.09633014 0.09779631 0.09965014 0.10312151
       0.10039826 0.10360531 0.09768352 0.09684183]
      [0.10996898 0.09721649 0.09222211 0.10460191 0.08958466 0.10711527
       0.11287319 0.10113019 0.09820816 0.08707891]
      [0.09821337 0.09662624 0.10006981 0.09984751 0.1067332  0.10593085
       0.09575544 0.10280466 0.10081811 0.0932008 ]
      [0.09736553 0.10416796 0.09449164 0.09737861 0.09902459 0.10509311
       0.09791905 0.1056854  0.1023325  0.09654159]
      [0.09163772 0.09352645 0.09450037 0.10083631 0.11440562 0.1033878
       0.10077804 0.10288773 0.10811178 0.08992808]
      [0.09298588 0.09689637 0.09808706 0.09682368 0.10047119 0.10816243
       0.10025283 0.10730525 0.0985758  0.10043946]
      [0.09887483 0.0933245  0.09741373 0.10299726 0.09931839 0.11028184
       0.11008896 0.10955917 0.09587976 0.08226163]
      [0.09410203 0.09729888 0.09038874 0.10042959 0.1086506  0.10167162
       0.10108674 0.11025819 0.10375471 0.09235881]
      [0.10930987 0.09362999 0.09471197 0.11051055 0.0944471  0.10600865
       0.10620464 0.09740975 0.10020198 0.08756545]
      [0.10017277 0.09852239 0.10101819 0.09989274 0.09523204 0.10252512
       0.10817572 0.10195759 0.09923375 0.09326968]
      [0.09752297 0.09556661 0.0974232  0.09348523 0.11024477 0.10132881
       0.09822094 0.10456018 0.10175792 0.09988936]
      [0.11041364 0.09631962 0.09036238 0.10289398 0.09043644 0.10699376
       0.10908314 0.10209548 0.09911671 0.09228489]
      [0.0997237  0.10369904 0.093425   0.09746355 0.09779833 0.10591251
       0.10144878 0.10502674 0.09989354 0.09560879]
      [0.09848912 0.10253454 0.08875401 0.1101212  0.09505349 0.10507668
       0.10210613 0.10759004 0.10675634 0.08351837]
      [0.0968536  0.09762199 0.09900926 0.10012499 0.10057309 0.10710492
       0.09839983 0.09982194 0.10319673 0.09729364]
      [0.09393201 0.09656238 0.09367266 0.10304973 0.11151774 0.10350496
       0.10501593 0.10474221 0.10000389 0.08799854]
```

```
[0.09206174 0.10295234 0.0916099  0.1038572  0.10297322 0.1030822
 0.10357182 0.10643961 0.10671444 0.08673755]
[0.09492038 0.09812776 0.09591204 0.09494192 0.10360367 0.1055215
 0.09774093 0.10324545 0.10436831 0.10161801]
[0.09754831 0.09893226 0.09157261 0.09375313 0.1043077  0.10359
 0.1017411  0.10729562 0.10149885 0.09976031]
[0.09388619 0.10019437 0.09647558 0.10296693 0.09526909 0.10801692
 0.10089102 0.1063615  0.10488577 0.09105261]
[0.09920872 0.10419083 0.08969613 0.09734436 0.09974384 0.10097201
 0.10468753 0.10651376 0.10488405 0.09275866]
[0.10294428 0.09580094 0.09575922 0.10706638 0.09787127 0.10715722
 0.10791759 0.10523543 0.0931629  0.08708483]
[0.09567549 0.09713066 0.09854095 0.10260088 0.10566771 0.10146091
 0.10119849 0.10241637 0.10011736 0.09519114]
[0.10605599 0.08932025 0.09420925 0.10715887 0.10113387 0.10852827
 0.10968511 0.10561668 0.09395248 0.0843392 ]
[0.09673455 0.09802933 0.09448162 0.1012322  0.10941844 0.10379686
 0.09863862 0.10287382 0.10459509 0.09019941]
[0.0973117  0.09900337 0.09887532 0.09936594 0.1029703  0.10272109
 0.0986376  0.10518485 0.10058323 0.0953466 ]
```

```python
# Obter pesos do modelo
w = cnn_model.get_weights()

# Plotar filtros
plt.imshow(w[0][:, :, 0, 0], cmap='gray')  # Para plotar o filtro 1 da camada 1
plt.show()

plt.imshow(w[0][:, :, 0, 2], cmap='gray')  # Para plotar o filtro 3 da camada 1
plt.show()
```