Universidade de Brasília - Departamento de Ciência da Computação
Introdução à Inteligência Artificial, Turma A, 1/2020 - Prof. Díbio

Projeto 4 - Rede CNN

Ana Luísa Salvador Alvarez - 16/0048036
Marcella Pantarotto - 13/0143880

Resumo: Neste projeto a ideia é o uso e teste de variações de uma rede CNN (Convolutional Neural Network), ou Rede Convolucional Neural, implementada no framework Keras para reconhecimento de dígitos manuscritos. A partir de um modelo explicativo, o intuito é colocar dados, framework e projeto básico de uma CNN para reconhecer dígitos manuscritos a partir da base do MNIST.

Palavras-chave: Rede Convolucional Neural; Deep Learning; Aprendizado Profundo; Reconhecimento de Dígitos; MNIST.

1 Introdução

Uma Rede Neural Convolucional (Convolutional Neural Network ou CNN) é um algoritmo de Aprendizado Profundo (ou Deep Learning) que pode captar uma imagem de entrada, atribuir importância (pesos e vieses que podem ser aprendidos) a vários aspectos ou objetos da imagem e ser capaz de diferenciar um do outro.

O pré-processamento exigido em uma CNN é muito menor em comparação com outros algoritmos de classificação. Enquanto nos métodos primitivos os filtros são feitos à mão, com treinamento suficiente, as CNN têm a capacidade de aprender esses filtros ou características. A arquitetura de uma CNN é análoga àquela do padrão de conectividade de neurônios no cérebro humano e foi inspirada na organização do Córtex Visual.

MNIST é uma base de dados grande do NIST (National Institute of Standards and Technology), de dígitos manuscritos, comumente usada para treinar sistemas de processamento de imagem. Também é amplamente utilizada para treinar e testar no campo de aprendizado de máquina.

2 Materiais e métodos

O algoritmo utilizado foi baseado num modelo explicativo, obtido em na página 'MNIST - Digits Classification with Keras', de Manish Bhobé, conforme especificação. Algoritmo está na linguagem Python, com uso do Keras, uma biblioteca de rede neural de código aberto, escrita em Python. Ela é projetada para permitir experimentação rápida com redes neurais profundas e roda em cima do Tensorflow, uma biblioteca de código aberto para aprendizado de máquina aplicável a uma ampla variedade de tarefas. É um sistema para criação e treinamento de redes neurais para detectar e decifrar padrões e correlações.

De acordo com os questionamentos propostos na especificação do projeto, variações foram feitas no código para coletar resultados e verificar as alterações nos resultados:

- Algoritmo com e sem normalização dos dados;
- Diferentes proporções entre treinamento e validação (90/10% e 70/30%);
- CNN com uma, duas e três camadas de convolução.

3 Resultados quadro, gráficos e figuras

O algoritmo base utilizado é o com normalização dos dados, com proporção 90% treinamento e 10% validação e com 3 camadas convolucionais, seguindo o modelo explicativo.

```
None
Epoch 1/15
844/844 [==============================] - 43s 51ms/step - loss: 0.1919 - accuracy: 0.9413 - val_loss: 0.0532 - val_accuracy: 0.9835
Epoch 2/15
844/844 [==============================] - 51s 60ms/step - loss: 0.0509 - accuracy: 0.9843 - val_loss: 0.0432 - val_accuracy: 0.9867
Epoch 3/15
844/844 [==============================] - 51s 61ms/step - loss: 0.0359 - accuracy: 0.9886 - val_loss: 0.0355 - val_accuracy: 0.9887
Epoch 4/15
844/844 [==============================] - 52s 61ms/step - loss: 0.0282 - accuracy: 0.9913 - val_loss: 0.0323 - val_accuracy: 0.9905
Epoch 5/15
844/844 [==============================] - 52s 61ms/step - loss: 0.0230 - accuracy: 0.9925 - val_loss: 0.0336 - val_accuracy: 0.9878
Epoch 6/15
844/844 [==============================] - 51s 61ms/step - loss: 0.0182 - accuracy: 0.9940 - val_loss: 0.0319 - val_accuracy: 0.9913
Epoch 7/15
844/844 [==============================] - 52s 62ms/step - loss: 0.0147 - accuracy: 0.9955 - val_loss: 0.0293 - val_accuracy: 0.9907
Epoch 8/15
844/844 [==============================] - 52s 62ms/step - loss: 0.0136 - accuracy: 0.9954 - val_loss: 0.0426 - val_accuracy: 0.9892
Epoch 9/15
844/844 [==============================] - 53s 63ms/step - loss: 0.0108 - accuracy: 0.9965 - val_loss: 0.0274 - val_accuracy: 0.9915
Epoch 10/15
844/844 [==============================] - 53s 62ms/step - loss: 0.0097 - accuracy: 0.9968 - val_loss: 0.0280 - val_accuracy: 0.9925
Epoch 11/15
844/844 [==============================] - 52s 62ms/step - loss: 0.0097 - accuracy: 0.9965 - val_loss: 0.0246 - val_accuracy: 0.9920
Epoch 12/15
844/844 [==============================] - 53s 63ms/step - loss: 0.0066 - accuracy: 0.9978 - val_loss: 0.0271 - val_accuracy: 0.9923
Epoch 13/15
844/844 [==============================] - 52s 62ms/step - loss: 0.0068 - accuracy: 0.9976 - val_loss: 0.0262 - val_accuracy: 0.9933
Epoch 14/15
844/844 [==============================] - 52s 62ms/step - loss: 0.0076 - accuracy: 0.9974 - val_loss: 0.0299 - val_accuracy: 0.9925
Epoch 15/15
844/844 [==============================] - 52s 61ms/step - loss: 0.0057 - accuracy: 0.9980 - val_loss: 0.0443 - val_accuracy: 0.9910
Time taken: 12 mins 52 secs
157/157 [==============================] - 2s 13ms/step - loss: 0.0394 - accuracy: 0.9915
Test loss: 0.0394 accuracy: 0.9915
```

Figura 1 – Dados referentes ao algoritmo com normalização, proporção 90%-10% e 3 camadas convolucionais.
Fonte: Autoria própria.

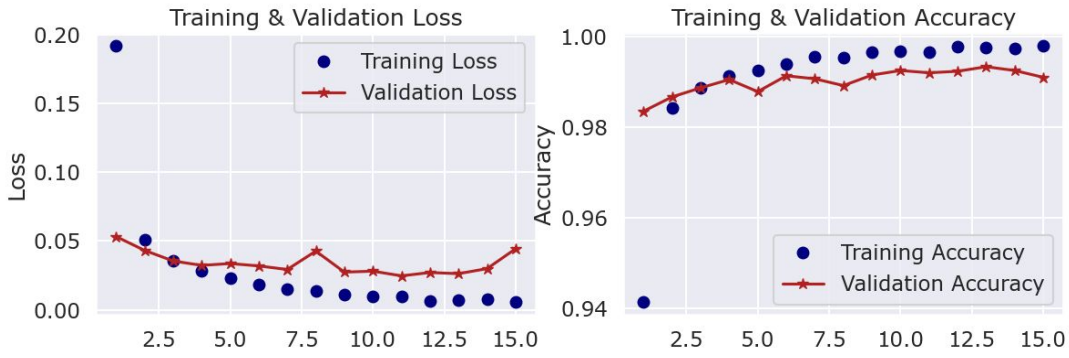

Figura 2 – Gráficos referentes ao algoritmo com normalização, proporção 90%-10% e 3 camadas convolucionais
Fonte: Autoria própria.

Para responder ao primeiro questionamento, o trecho de código abaixo, apresentado na figura 3, foi comentado, para que não fossem normalizados os dados.

```
#normalização dos dados
# re-scale the image data to values between (0.0,1.0]
train_data = train_data.astype('float32') / 255.
test_data = test_data.astype('float32') / 255.
```

Figura 3 – Trecho de código responsável pela normalização dos dados.
Fonte: Autoria própria.

Ao rodar o código sem a normalização, foram obtidos os dados apresentados abaixo, nas figuras 4 e 5.

```
None
Epoch 1/15
844/844 [==============================] - 51s 60ms/step - loss: 0.4098 - accuracy: 0.9268 - val_loss: 0.0835 - val_accuracy: 0.9745
Epoch 2/15
844/844 [==============================] - 52s 62ms/step - loss: 0.0638 - accuracy: 0.9799 - val_loss: 0.1013 - val_accuracy: 0.9727
Epoch 3/15
844/844 [==============================] - 50s 59ms/step - loss: 0.0456 - accuracy: 0.9853 - val_loss: 0.0677 - val_accuracy: 0.9780
Epoch 4/15
844/844 [==============================] - 53s 63ms/step - loss: 0.0396 - accuracy: 0.9875 - val_loss: 0.0486 - val_accuracy: 0.9863
Epoch 5/15
844/844 [==============================] - 52s 61ms/step - loss: 0.0319 - accuracy: 0.9898 - val_loss: 0.0390 - val_accuracy: 0.9883
Epoch 6/15
844/844 [==============================] - 53s 62ms/step - loss: 0.0296 - accuracy: 0.9908 - val_loss: 0.0416 - val_accuracy: 0.9883
Epoch 7/15
844/844 [==============================] - 53s 63ms/step - loss: 0.0285 - accuracy: 0.9906 - val_loss: 0.0429 - val_accuracy: 0.9875
Epoch 8/15
844/844 [==============================] - 51s 61ms/step - loss: 0.0240 - accuracy: 0.9924 - val_loss: 0.0399 - val_accuracy: 0.9890
Epoch 9/15
844/844 [==============================] - 53s 63ms/step - loss: 0.0246 - accuracy: 0.9921 - val_loss: 0.0483 - val_accuracy: 0.9873
Epoch 10/15
844/844 [==============================] - 52s 62ms/step - loss: 0.0226 - accuracy: 0.9929 - val_loss: 0.0421 - val_accuracy: 0.9907
Epoch 11/15
844/844 [==============================] - 54s 64ms/step - loss: 0.0194 - accuracy: 0.9943 - val_loss: 0.0422 - val_accuracy: 0.9888
Epoch 12/15
844/844 [==============================] - 52s 62ms/step - loss: 0.0187 - accuracy: 0.9939 - val_loss: 0.0544 - val_accuracy: 0.9865
Epoch 13/15
844/844 [==============================] - 53s 63ms/step - loss: 0.0153 - accuracy: 0.9956 - val_loss: 0.0467 - val_accuracy: 0.9898
Epoch 14/15
844/844 [==============================] - 52s 61ms/step - loss: 0.0153 - accuracy: 0.9951 - val_loss: 0.0426 - val_accuracy: 0.9915
Epoch 15/15
844/844 [==============================] - 52s 62ms/step - loss: 0.0174 - accuracy: 0.9947 - val_loss: 0.0573 - val_accuracy: 0.9885
Time taken: 13 mins 4 secs
157/157 [==============================] - 2s 14ms/step - loss: 0.0736 - accuracy: 0.9876
Test loss: 0.0736 accuracy: 0.9876
```

Figura 4 – Dados referentes ao algoritmo sem normalização, proporção 90%-10% e 3 camadas convolucionais.
Fonte: Autoria própria.

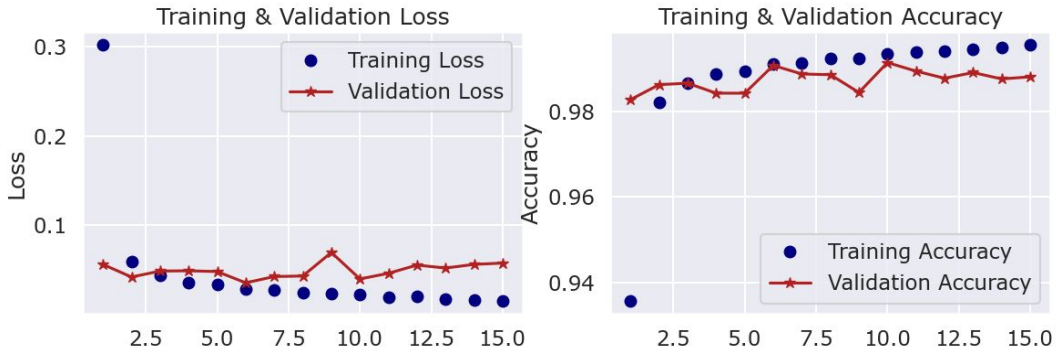

Figura 5 – Gráficos referentes ao algoritmo sem normalização, proporção 90%-10% e 3 camadas convolucionais
Fonte: Autoria própria.

Para responder a segunda questão, voltamos com o trecho de normalização, mas alteramos a proporção dos dados de treinamento e validação para 70%-30%, conforme especificado e de acordo com o trecho de código apresentado na figura 6.

```
#porcentagem de validação é setada
# now set-aside val_perc% (10%) of the train_data/labels as the cross-validation sets
val_perc = 0.30
val_count = int(val_perc * len(train_data))
```

Figura 6 – Trecho de código que estabelece a porcentagem de validação a ser utilizada.
Fonte: Autoria própria.

Ao rodar o código com essa nova porcentagem e com normalização de dados, obtivemos os seguintes dados.

```
None
Epoch 1/15
657/657 [==============================] - 43s 66ms/step - loss: 0.2135 - accuracy: 0.9362 - val_loss: 0.0658 - val_accuracy: 0.9802
Epoch 2/15
657/657 [==============================] - 44s 67ms/step - loss: 0.0550 - accuracy: 0.9831 - val_loss: 0.0546 - val_accuracy: 0.9824
Epoch 3/15
657/657 [==============================] - 42s 65ms/step - loss: 0.0392 - accuracy: 0.9876 - val_loss: 0.0433 - val_accuracy: 0.9868
Epoch 4/15
657/657 [==============================] - 45s 69ms/step - loss: 0.0314 - accuracy: 0.9902 - val_loss: 0.0450 - val_accuracy: 0.9853
Epoch 5/15
657/657 [==============================] - 44s 67ms/step - loss: 0.0223 - accuracy: 0.9929 - val_loss: 0.0411 - val_accuracy: 0.9868
Epoch 6/15
657/657 [==============================] - 45s 68ms/step - loss: 0.0198 - accuracy: 0.9933 - val_loss: 0.0488 - val_accuracy: 0.9856
Epoch 7/15
657/657 [==============================] - 46s 70ms/step - loss: 0.0152 - accuracy: 0.9949 - val_loss: 0.0485 - val_accuracy: 0.9858
Epoch 8/15
657/657 [==============================] - 44s 67ms/step - loss: 0.0139 - accuracy: 0.9958 - val_loss: 0.0372 - val_accuracy: 0.9893
Epoch 9/15
657/657 [==============================] - 45s 69ms/step - loss: 0.0110 - accuracy: 0.9964 - val_loss: 0.0342 - val_accuracy: 0.9905
Epoch 10/15
657/657 [==============================] - 46s 69ms/step - loss: 0.0098 - accuracy: 0.9967 - val_loss: 0.0443 - val_accuracy: 0.9888
Epoch 11/15
657/657 [==============================] - 44s 67ms/step - loss: 0.0091 - accuracy: 0.9969 - val_loss: 0.0429 - val_accuracy: 0.9886
Epoch 12/15
657/657 [==============================] - 46s 70ms/step - loss: 0.0078 - accuracy: 0.9974 - val_loss: 0.0331 - val_accuracy: 0.9911
Epoch 13/15
657/657 [==============================] - 46s 70ms/step - loss: 0.0057 - accuracy: 0.9979 - val_loss: 0.0407 - val_accuracy: 0.9904
Epoch 14/15
657/657 [==============================] - 45s 69ms/step - loss: 0.0083 - accuracy: 0.9968 - val_loss: 0.0544 - val_accuracy: 0.9882
Epoch 15/15
657/657 [==============================] - 46s 70ms/step - loss: 0.0081 - accuracy: 0.9974 - val_loss: 0.0407 - val_accuracy: 0.9917
Time taken: 11 mins 12 secs
157/157 [==============================] - 3s 17ms/step - loss: 0.0352 - accuracy: 0.9918
Test loss: 0.0352 accuracy: 0.9918
```

Figura 7 – Dados referentes ao algoritmo com normalização, proporção 70%-30% e 3 camadas convolucionais.
Fonte: Autoria própria.

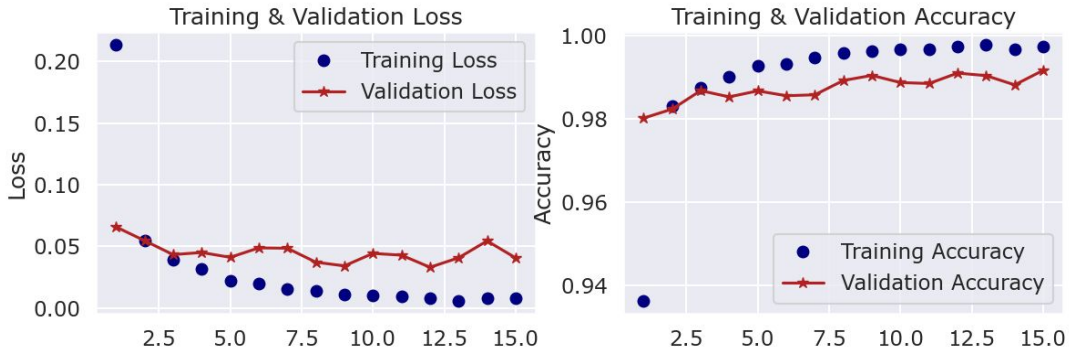

Figura 8 – Gráficos referentes ao algoritmo com normalização, proporção 70%-30% e 3 camadas convolucionais.
Fonte: Autoria própria.

Para responder ao terceiro questionamento, a proporção 90%-10% foi retomada, a normalização seguiu ocorrendo, mas alterações foram feitas no trecho de código abaixo para que fossem testadas 1 e 2 camadas convolucionais, versus a base, que era de 3 camadas.

```
# add Convolutional layers
model.add(Conv2D(filters=32, kernel_size=(3,3), activation='relu', padding='same',      # camada 1
                 input_shape=(image_height, image_width, num_channels)))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(filters=64, kernel_size=(3,3), activation='relu', padding='same'))     # camada 2
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(filters=64, kernel_size=(3,3), activation='relu', padding='same'))     # camada 3
model.add(MaxPooling2D(pool_size=(2,2)))
```

Figura 9 – Trecho de código que declara as 3 camadas convolucionais.
Fonte: Autoria própria.

```
None
Epoch 1/15
844/844 [==============================] - 24s 29ms/step - loss: 0.1961 - accuracy: 0.9420 - val_loss: 0.0861 - val_accuracy: 0.9742
Epoch 2/15
844/844 [==============================] - 24s 29ms/step - loss: 0.0656 - accuracy: 0.9800 - val_loss: 0.0504 - val_accuracy: 0.9848
Epoch 3/15
844/844 [==============================] - 24s 29ms/step - loss: 0.0436 - accuracy: 0.9863 - val_loss: 0.0439 - val_accuracy: 0.9852
Epoch 4/15
844/844 [==============================] - 24s 29ms/step - loss: 0.0302 - accuracy: 0.9909 - val_loss: 0.0391 - val_accuracy: 0.9870
Epoch 5/15
844/844 [==============================] - 24s 29ms/step - loss: 0.0212 - accuracy: 0.9935 - val_loss: 0.0416 - val_accuracy: 0.9858
Epoch 6/15
844/844 [==============================] - 28s 33ms/step - loss: 0.0157 - accuracy: 0.9949 - val_loss: 0.0322 - val_accuracy: 0.9895
Epoch 7/15
844/844 [==============================] - 31s 37ms/step - loss: 0.0111 - accuracy: 0.9964 - val_loss: 0.0374 - val_accuracy: 0.9885
Epoch 8/15
844/844 [==============================] - 47s 56ms/step - loss: 0.0094 - accuracy: 0.9973 - val_loss: 0.0368 - val_accuracy: 0.9893
Epoch 9/15
844/844 [==============================] - 26s 30ms/step - loss: 0.0070 - accuracy: 0.9977 - val_loss: 0.0419 - val_accuracy: 0.9872
Epoch 10/15
844/844 [==============================] - 26s 31ms/step - loss: 0.0061 - accuracy: 0.9982 - val_loss: 0.0401 - val_accuracy: 0.9882
Epoch 11/15
844/844 [==============================] - 29s 35ms/step - loss: 0.0050 - accuracy: 0.9984 - val_loss: 0.0440 - val_accuracy: 0.9870
Epoch 12/15
844/844 [==============================] - 28s 34ms/step - loss: 0.0037 - accuracy: 0.9990 - val_loss: 0.0480 - val_accuracy: 0.9873
Epoch 13/15
844/844 [==============================] - 24s 29ms/step - loss: 0.0046 - accuracy: 0.9985 - val_loss: 0.0434 - val_accuracy: 0.9897
Epoch 14/15
844/844 [==============================] - 24s 29ms/step - loss: 0.0024 - accuracy: 0.9993 - val_loss: 0.0417 - val_accuracy: 0.9877
Epoch 15/15
844/844 [==============================] - 24s 29ms/step - loss: 0.0043 - accuracy: 0.9984 - val_loss: 0.0510 - val_accuracy: 0.9878
Time taken: 6 mins 50 secs
157/157 [==============================] - 2s 11ms/step - loss: 0.0582 - accuracy: 0.9869
Test loss: 0.0582 accuracy: 0.9869
```

Figura 10 – Dados referentes ao algoritmo com normalização, proporção 90%-10% e 1 camada convolucional.
Fonte: Autoria própria.

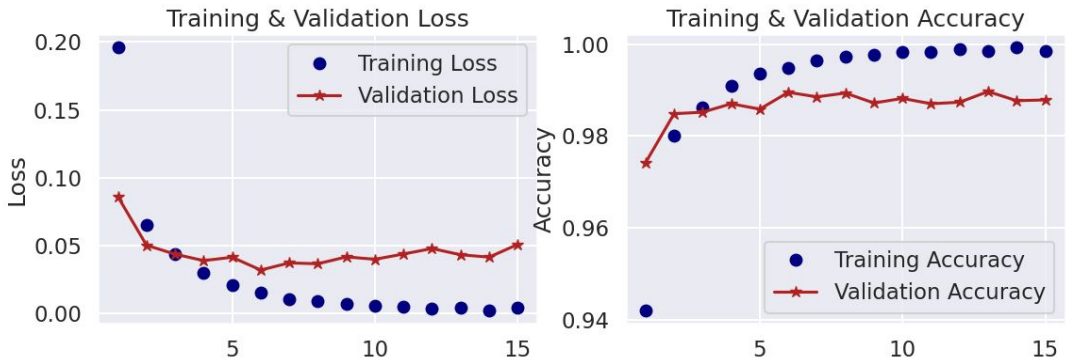

Figura 11 – Gráficos referentes ao algoritmo com normalização, proporção 90%-10% e 1 camada convolucional.
Fonte: Autoria própria.

```
None
Epoch 1/15
844/844 [==============================] - 49s 58ms/step - loss: 0.1558 - accuracy: 0.9525 - val_loss: 0.0729 - val_accuracy: 0.9767
Epoch 2/15
844/844 [==============================] - 46s 54ms/step - loss: 0.0465 - accuracy: 0.9857 - val_loss: 0.0343 - val_accuracy: 0.9895
Epoch 3/15
844/844 [==============================] - 48s 57ms/step - loss: 0.0313 - accuracy: 0.9904 - val_loss: 0.0294 - val_accuracy: 0.9905
Epoch 4/15
844/844 [==============================] - 46s 55ms/step - loss: 0.0233 - accuracy: 0.9929 - val_loss: 0.0280 - val_accuracy: 0.9898
Epoch 5/15
844/844 [==============================] - 46s 54ms/step - loss: 0.0175 - accuracy: 0.9939 - val_loss: 0.0285 - val_accuracy: 0.9908
Epoch 6/15
844/844 [==============================] - 46s 54ms/step - loss: 0.0135 - accuracy: 0.9957 - val_loss: 0.0324 - val_accuracy: 0.9897
Epoch 7/15
844/844 [==============================] - 46s 55ms/step - loss: 0.0108 - accuracy: 0.9963 - val_loss: 0.0311 - val_accuracy: 0.9902
Epoch 8/15
844/844 [==============================] - 45s 54ms/step - loss: 0.0088 - accuracy: 0.9972 - val_loss: 0.0335 - val_accuracy: 0.9895
Epoch 9/15
844/844 [==============================] - 46s 54ms/step - loss: 0.0075 - accuracy: 0.9975 - val_loss: 0.0328 - val_accuracy: 0.9908
Epoch 10/15
844/844 [==============================] - 46s 54ms/step - loss: 0.0077 - accuracy: 0.9976 - val_loss: 0.0330 - val_accuracy: 0.9913
Epoch 11/15
844/844 [==============================] - 46s 54ms/step - loss: 0.0050 - accuracy: 0.9984 - val_loss: 0.0390 - val_accuracy: 0.9902
Epoch 12/15
844/844 [==============================] - 46s 54ms/step - loss: 0.0038 - accuracy: 0.9989 - val_loss: 0.0383 - val_accuracy: 0.9928
Epoch 13/15
844/844 [==============================] - 46s 54ms/step - loss: 0.0057 - accuracy: 0.9981 - val_loss: 0.0406 - val_accuracy: 0.9922
Epoch 14/15
844/844 [==============================] - 46s 54ms/step - loss: 0.0053 - accuracy: 0.9982 - val_loss: 0.0324 - val_accuracy: 0.9918
Epoch 15/15
844/844 [==============================] - 46s 54ms/step - loss: 0.0036 - accuracy: 0.9987 - val_loss: 0.0458 - val_accuracy: 0.9902
Time taken: 11 mins 34 secs
157/157 [==============================] - 3s 20ms/step - loss: 0.0508 - accuracy: 0.9897
Test loss: 0.0508 accuracy: 0.9897
```

Figura 12 – Dados referentes ao algoritmo com normalização, proporção 90%-10% e 2 camadas convolucionais.
Fonte: Autoria própria.

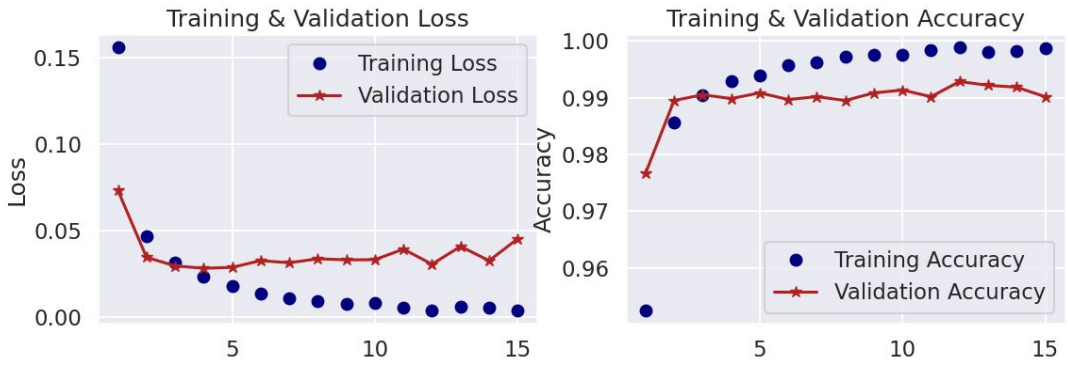

Figura 13 – Gráficos referentes ao algoritmo com normalização, proporção 90%-10% e 2 camadas convolucionais.
Fonte: Autoria própria.

```
None
Epoch 1/15
844/844 [==============================] - 54s 64ms/step - loss: 0.1832 - accuracy: 0.9429 - val_loss: 0.0462 - val_accuracy: 0.9857
Epoch 2/15
844/844 [==============================] - 54s 64ms/step - loss: 0.0482 - accuracy: 0.9853 - val_loss: 0.0295 - val_accuracy: 0.9910
Epoch 3/15
844/844 [==============================] - 54s 64ms/step - loss: 0.0349 - accuracy: 0.9893 - val_loss: 0.0318 - val_accuracy: 0.9887
Epoch 4/15
844/844 [==============================] - 54s 65ms/step - loss: 0.0258 - accuracy: 0.9919 - val_loss: 0.0316 - val_accuracy: 0.9892
Epoch 5/15
844/844 [==============================] - 53s 63ms/step - loss: 0.0212 - accuracy: 0.9933 - val_loss: 0.0249 - val_accuracy: 0.9922
Epoch 6/15
844/844 [==============================] - 54s 64ms/step - loss: 0.0161 - accuracy: 0.9945 - val_loss: 0.0277 - val_accuracy: 0.9918
Epoch 7/15
844/844 [==============================] - 53s 63ms/step - loss: 0.0153 - accuracy: 0.9948 - val_loss: 0.0257 - val_accuracy: 0.9928
Epoch 8/15
844/844 [==============================] - 53s 63ms/step - loss: 0.0122 - accuracy: 0.9959 - val_loss: 0.0283 - val_accuracy: 0.9918
Epoch 9/15
844/844 [==============================] - 55s 65ms/step - loss: 0.0093 - accuracy: 0.9967 - val_loss: 0.0247 - val_accuracy: 0.9933
Epoch 10/15
844/844 [==============================] - 60s 71ms/step - loss: 0.0105 - accuracy: 0.9965 - val_loss: 0.0322 - val_accuracy: 0.9905
Epoch 11/15
844/844 [==============================] - 53s 63ms/step - loss: 0.0073 - accuracy: 0.9976 - val_loss: 0.0277 - val_accuracy: 0.9905
Epoch 12/15
844/844 [==============================] - 54s 64ms/step - loss: 0.0088 - accuracy: 0.9970 - val_loss: 0.0306 - val_accuracy: 0.9928
Epoch 13/15
844/844 [==============================] - 54s 63ms/step - loss: 0.0063 - accuracy: 0.9978 - val_loss: 0.0314 - val_accuracy: 0.9922
Epoch 14/15
844/844 [==============================] - 54s 64ms/step - loss: 0.0073 - accuracy: 0.9976 - val_loss: 0.0269 - val_accuracy: 0.9918
Epoch 15/15
844/844 [==============================] - 56s 66ms/step - loss: 0.0056 - accuracy: 0.9981 - val_loss: 0.0257 - val_accuracy: 0.9933
Time taken: 13 mins 38 secs
157/157 [==============================] - 3s 21ms/step - loss: 0.0396 - accuracy: 0.9908
Test loss: 0.0396 accuracy: 0.9908
```

Figura 14 – Dados referentes ao algoritmo com normalização, proporção 90%-10% e 3 camadas convolucionais.
Fonte: Autoria própria.

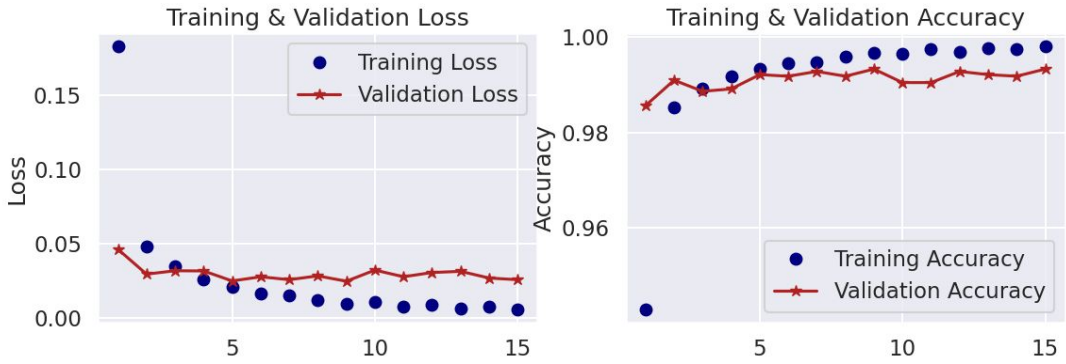

Figura 15 – Gráficos referentes ao algoritmo com normalização, proporção 90%-10% e 3 camadas convolucionais.
Fonte: Autoria própria.

4 Análise de Resultados

Normalização é uma técnica comumente aplicada como parte da preparação dos dados em aprendizado de máquina. O objetivo é mudar os valores para uma escala comum, sem distorcer diferenças em variação. Assim evita-se problemas de estabilidade numérica, obtendo-se melhores resultados de generalização.

Neste caso foi necessária, pois os valores utilizados variam entre 0 e 255. Ao dividir esses valores por 255, todos eles variam entre 0 e 1. Dessa forma, a análise é feita sem que o atributo pareça mais importante como preditor do que realmente é. O que foi possível visualizar nos gráficos, foi a diminuição nos valores de perda de treinamento e perda de validação, assim como uma proporcional aproximação das curvas para esses valores, indicando um melhor encaixe (better fit). Nas curvas de acurácia de validação e teste, foi possível visualizar maior variação quando os dados não foram normalizados.

A separação de dados em conjuntos de treinamento e validação é uma parte importante da avaliação de modelos de redes neurais. Normalmente, a maior parte dos dados é usada para treinamento e uma parte menor dos dados é usada para validação. Depois que um modelo for processado usando o conjunto de treinamentos, o modelo é testado, fazendo previsões contra o conjunto de validação.

O que pôde ser observado é que nesse caso, o modelo com 90% de dados de treinamento e 10% de dados de validação obteve melhores resultados do que o modelo 70%-30%. As curvas de perda de treinamento e validação para o modelo 90%-10% ficaram mais próximas uma da outra, assim como as de acurácia de treinamento e validação. Isso indica um melhor encaixe (better fit).

As camadas de convoluções usadas no algoritmo servem como filtros que divisam as imagens como pequenos quadrados e vão resvalando até que traços mais marcantes da figura sejam definidos. A profundidade da saída de uma convolução é igual a quantidade de filtros aplicados. Quanto mais profundas são as camadas das convoluções, mais detalhados são os traços identificados com a função de ativação. Ela serve para ressalta-se a não-linearidade do sistema, permitindo qualquer tipo de aprendizado sobre funcionalidade do sistema. No presente projeto é utilizado a função Relu, por ser mais eficiente computacionalmente sem grandes diferenças de acurácia.

Assim, foram feitos teste para 1, 2 e 3 camadas, sendo a última a quantidade recomendada para uso e, para isto, as alterações do código para permitir estes testes foram realizadas no trecho de código apresentado na Figura 9. Comparando as Figuras 10, 12 e 14 pode-se comparar a diferença dos resultados para os testes e quanto maior for a quantidade de camadas, maior é o tempo de execução e a acurácia e menor é a perde que se tem nos testes. Já comparando as imagens 11, 13 e 15, é possível notar algumas diferenças entre os gráficos plotados em cada test.

5 Considerações Finais/Conclusões

Com o desenvolvimento deste projeto, foi possível a consolidação do conhecimento adquirido sobre aprendizagem de máquina durante as aulas teóricas e os estudos proferidos, em específico, sobre o assunto de redes neurais convolucionais (CNN). A utilização do framework Keras para reconhecimento de dígitos manuscritos colaborou para o enriquecimento do conhecimento, permitindo a criação de um projeto básico de CNN, para estudar a base de dados MNIST.

Referências Bibliográficas:

Bhobé, Manish. MNIST - Digits Classification with Keras. Disponível em https://medium.com/@mjbhobe/mnist-digits-classification-with-keras-ed6c2374bd0e. Acesso em 22/11/2020.

Brownlee, Jason. How to use Learning Curves to Diagnose Machine Learning Model Performance. Disponível em

https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/. Acesso em 24/11/2020.

Data Science Academy. Deep Learning Book, Capítulo 40 – Introdução às Redes Neurais Convolucionais. Disponível em http://deeplearningbook.com.br/introducao-as-redes-neurais-convolucionais/. Acesso em 23/11/2020.

Jaitley, Urvashi. Why Data Normalization is necessary for Machine Learning Models. Disponível em https://medium.com/@urvashilluniya/why-data-normalization-is-necessary-for-machine-learning-models-681b65a05029. Acesso em 23/11/2020.

Keras. Conteúdo aberto. In: Wikipédia: a enciclopédia livre. Disponível em https://pt.wikipedia.org/wiki/Keras. Acesso em 23/11/2020.

Miyazaki, Caio K. Redes neurais convolucionais para aprendizagem e reconhecimento de objetos 3D. USP-São Carlos. 2017.

MNIST database. Conteúdo aberto. In: Wikipédia: a enciclopédia livre. Disponível em https://en.wikipedia.org/wiki/MNIST_database. Acesso em 23/11/2020.

Ruizendaal, Rutger. Deep Learning #3: More on CNNs & Handling Overfitting. Towards Data Science. Disponível em https://towardsdatascience.com/deep-learning-3-more-on-cnns-handling-overfitting-2bd5d99abe5d. Acesso em 24/11/2020.

Tensorflow. Conteúdo aberto. In: Wikipédia: a enciclopédia livre. Disponível em https://pt.wikipedia.org/wiki/TensorFlow. Acesso em 23/11/2020.