Reconhecimento de caracteres alfabéticos manuscritos utilizando uma Multilayer Perceptron (MLP)

Introdução

Este trabalho foi desenvolvido por Igor Carvalho de Brito Batista, Rony de Sena Lourenço e Thatiana Jéssica da Silva Ribeiro.

O objetivo se trata de analisar uma rede neural para reconhecer caligrafia, ou seja, letras escritas à mão que foram digitalizados em imagens no computador. Para realizar essa, foi utilizado uma base de dados que foi criada pelos alunos de um professor da UFRN.

Metodologia

O modelo de aprendizado de máquina utilizado neste trabalho foi o Multilayer Perceptron, algoritmo responsável pelo aprendizado de rede.

A partir do funcionamento dos neurônios biológicos do sistema nervoso animal, foi estabelecido na área da Inteligência Artificial um modelo computacional de um neurônio. Com apenas um neurônio, pouca coisa se pode fazer, mas os combinando em uma estrutura em camadas, em cada uma terá um número diferente de neurônios, teremos assim a formação de uma rede.

O vetor de valores de uma determinada entrada passa por uma camada inicial chamada input layer, que é a responsável por encaminhar esse vetor para uma outra camada chamada head layer, responsável por produzir um número de hiperplanos no espaço. Logo após, é encaminhado para a camada de saída chamada output layer, responsável por mostrar o resultado.

Para que esse modelo funcione, assim como qualquer outro modelo em rede, é necessário passar pela etapa do treinamento. Essa etapa tem por objetivo fazer com que o modelo aprenda os padrões a partir de uma determinada amostra, dessa forma, quando um dado desconhecido for fornecido à rede, ela será apta à designar a qual classe a amostra pertence. Depois, foi possível fazer a validação do resultado deste algoritmo com os dados de teste, durante esta fase, foi possível determinar o funcionamento da rede com alguns dados que não foram utilizados na etapa de treinamento.

Códigos

Para realizar este trabalho, foi necessário fazer o uso de algumas bibliotecas, como a OpenCV, que é a responsável por fazer a manipulação de imagens. Ainda, foi preciso fazer o uso da biblioteca NumPy, que foi a responsável para realizar operações envolvendo matrizes e vetores. E, para rodar a rede neural, foi necessário importar a biblioteca Keras.

O trecho de código a seguir, é responsável por fazer a identificação e das imagens e labels e, logo após, armazenar em duas listas distintas. Para isso, foi necessário fazer o uso da função *traverse_dir()* para entrar nas pastas.

```
images = []
labels = []
def traverse_dir(path): #Esta função basicamente entra no diretório
    for file_or_dir in os.listdir(path): #De forma recursiva, ele e
        abs_path = os.path.abspath(os.path.join(path, file_or_dir))
        print(abs_path)
        if os.path.isdir(abs_path):  # dir
            traverse_dir(abs_path)
        else:                        # file
            if file_or_dir.endswith('.jpg'):
                image = read_image(abs_path)
                images.append(image)
                labels.append(path[len(path)-1])

    return images, labels
```

Figura 01 - identificação de imagens e labels

Fazendo o uso da biblioteca OpenCV, foi possível fazer a captura das imagens, convertendo-as para a cor cinza e, logo após, fazendo a chamada principal para a identificação dos arquivos dentro da pasta do conjunto de dados

```
def read_image(file_path):
    image = cv2.imread(file_path)

    gray_scale = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    image = cv2.bitwise_not(gray_scale)
    return image


def extract_data(path):
    images, labels = traverse_dir(path)
    images = np.array(images)

    return images, labels
```

Figura 02 - captura de imagens

Para que esse modelo funcione, assim como qualquer outro modelo em rede, é necessário passar pela etapa do treinamento. Neste trabalho, 70% dos dados foram utilizados para treinamento, o trecho de código a seguir é responsável por essa etapa. Logo após, é realizado a etapa de *flattening*, já que para que os dados possam ser usados pela rede neural, é necessário que estejam na forma de *array.*

```
X_train, X_test, y_train, y_test = train_test_split(images, labels, test_size=0.3, random_state=1)
```

```
X_train.flatten()
X_test.flatten()
y_train.flatten()
y_test.flatten()
```

Figura 03 - separação de treinamento e teste

Nas figura a seguir, é possível observar o plot das imagens da base de dados que foram utilizadas durante o processo de treinamento e um gráfico com a distribuição desses dados.

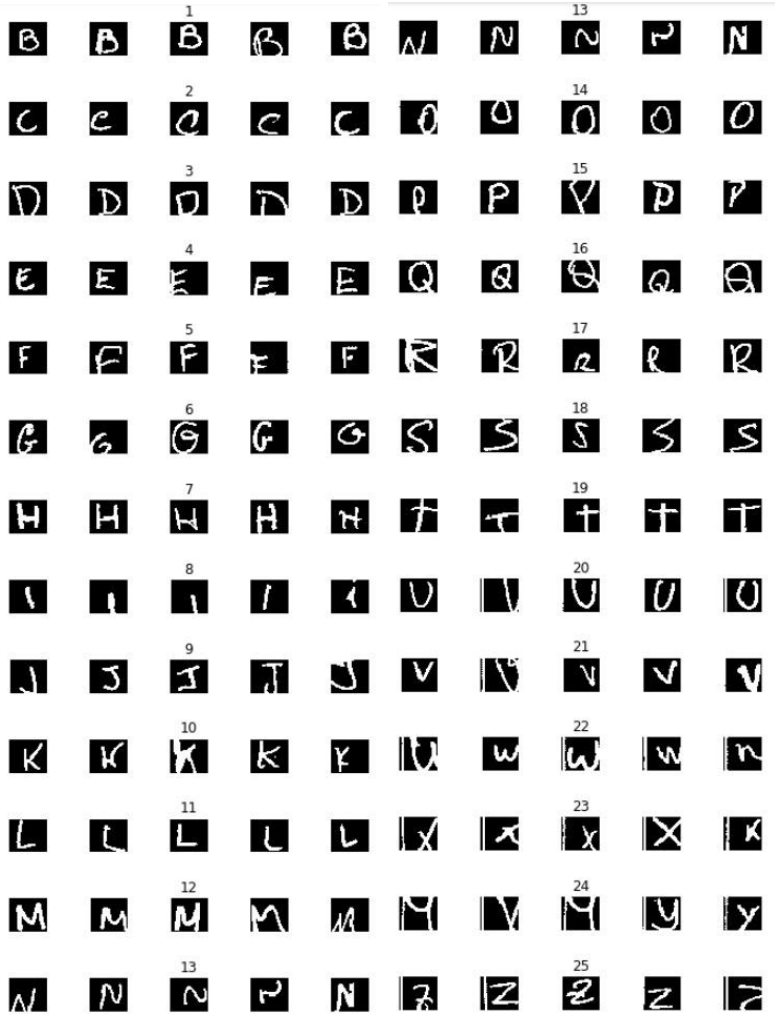


Figura 04 - Imagens capturadas para o treinamento.

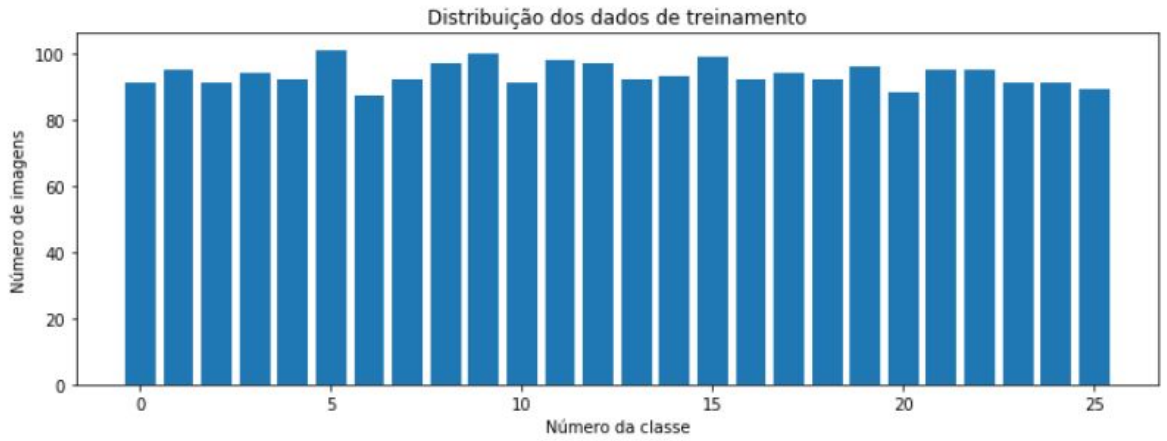


Figura 05 - Distribuição das imagens do treinamento.

As especificações a seguir são referentes a rede neural, sendo possível observar a seleção dos parâmetros. Primeiramente, foi criado as camadas, dentro de cada camada é possível inserir a quantidade de neurônios que serão utilizados para fazer a filtragem dos dados bem como a função de ativação a ser utilizada. Foi adicionado 2 camadas com a função de ativação do tipo *ReLu*, a qual a primeira possui 500 neurônios e a segunda 100. A rede possui uma camada de saída com a função de ativação *softmax*, fazendo com que a saída ser de uma das classes definidas.

```
[ ]  num_classes = 10
     def create_model():
       model = Sequential()
       model.add(Dense(500, input_dim=num_pixels, activation='relu'))
       model.add(Dense(200, activation='relu'))
       model.add(Dense(num_classes, activation='softmax'))
       model.compile(Adam(lr=0.01), loss='categorical_crossentropy', metrics=['accuracy'])
       return model
```

Figura 06 - configurações da rede

A seguir são especificados os parâmetros para realizar a etapa de treinamento, são selecionadas a proporção de validação utilizada, a quantidade de épocas e o batch size.

```
[ ]  history = model.fit(X_train, y_train, validation_split=0.1, epochs=50, batch_size = 15, verbose = 1, shuffle = 1)
```

Figura 07 - dados de treinamento

Experimentos

O experimento consistiu na passagem de todo o processo de seleção e separação e preparação do conjunto de dados. Após as operações realizadas no conjunto de dados, foi feita uma análise de magnitude do conjunto de dados a fim de dimensionar os parâmetros necessários para a execução do treinamento.

Para realizar o treinamento foi escolhida uma proporção de validação de 10% do conjunto de dados, 50 épocas e um batch size de 15, de modo que não fossem treinados muitos dados simultaneamente e também o treinamento não durasse muito tempo. A densidade das camadas de neurônios se mantiveram inalteradas, pois inicialmente não se fez necessário a modificação destes parâmetros.

```
[17] #Treinamento em si, aqui são selecionadas a proporção de validação utilizada, a quantidade de épocas e o batch size, esses são os mais importantes
     history = model.fit(X_train, y_train, validation_split=0.1, epochs=200, batch_size = 75, verbose = 1, shuffle = 15)
     Epoch 174/200
     2189/2189 [==============================] - 0s 150us/step - loss: 0.7558 - acc: 0.9497 - val_loss: 5.9351 - val_acc: 0.6230
     Epoch 175/200
     2189/2189 [==============================] - 0s 160us/step - loss: 0.7424 - acc: 0.9520 - val_loss: 6.1332 - val_acc: 0.6066
     Epoch 176/200
     2189/2189 [==============================] - 0s 154us/step - loss: 0.7032 - acc: 0.9529 - val_loss: 6.3491 - val_acc: 0.6025
     Epoch 177/200
     2189/2189 [==============================] - 0s 159us/step - loss: 1.0685 - acc: 0.9315 - val_loss: 6.4215 - val_acc: 0.5984
     Epoch 178/200
     2189/2189 [==============================] - 0s 153us/step - loss: 1.3605 - acc: 0.9141 - val_loss: 6.0629 - val_acc: 0.6025
     Epoch 179/200
     2189/2189 [==============================] - 0s 156us/step - loss: 1.1309 - acc: 0.9269 - val_loss: 6.6781 - val_acc: 0.5738
     Epoch 180/200
     2189/2189 [==============================] - 0s 156us/step - loss: 1.2709 - acc: 0.9187 - val_loss: 6.6227 - val_acc: 0.5820
     Epoch 181/200
     2189/2189 [==============================] - 0s 161us/step - loss: 0.7079 - acc: 0.9534 - val_loss: 6.4405 - val_acc: 0.5861
     Epoch 182/200
     2189/2189 [==============================] - 0s 155us/step - loss: 0.7599 - acc: 0.9502 - val_loss: 6.2154 - val_acc: 0.6066
     Epoch 183/200
     2189/2189 [==============================] - 0s 156us/step - loss: 0.9061 - acc: 0.9424 - val_loss: 6.6821 - val_acc: 0.5820
     Epoch 184/200
     2189/2189 [==============================] - 0s 154us/step - loss: 0.8282 - acc: 0.9466 - val_loss: 6.8087 - val_acc: 0.5738
     Epoch 185/200
     2189/2189 [==============================] - 0s 156us/step - loss: 0.8995 - acc: 0.9415 - val_loss: 6.3571 - val_acc: 0.6025
     Epoch 186/200
     2189/2189 [==============================] - 0s 152us/step - loss: 1.1704 - acc: 0.9242 - val_loss: 6.9520 - val_acc: 0.5656
     Epoch 187/200
     2189/2189 [==============================] - 0s 154us/step - loss: 1.3223 - acc: 0.9159 - val_loss: 7.0630 - val_acc: 0.5615
     Epoch 188/200
     2189/2189 [==============================] - 0s 150us/step - loss: 1.3684 - acc: 0.9118 - val_loss: 6.5924 - val_acc: 0.5861
     Epoch 189/200
     2189/2189 [==============================] - 0s 156us/step - loss: 0.8078 - acc: 0.9470 - val_loss: 6.0008 - val_acc: 0.6107
     Epoch 190/200
     2189/2189 [==============================] - 0s 151us/step - loss: 0.9810 - acc: 0.9365 - val_loss: 6.6322 - val_acc: 0.5820
```

Figura 08 - Imagens do treinamento.

Como pode ser observado na figura 08, ao final do treinamento foram obtidos como resultado uma acurácia na casa de 90%. No entanto, a acurácia obtida no teste não foi satisfatória como esperada.

Na figura 09, nota-se uma oscilação muito grande no erro da validação, não sendo possível notar uma tendência de decrescimento.

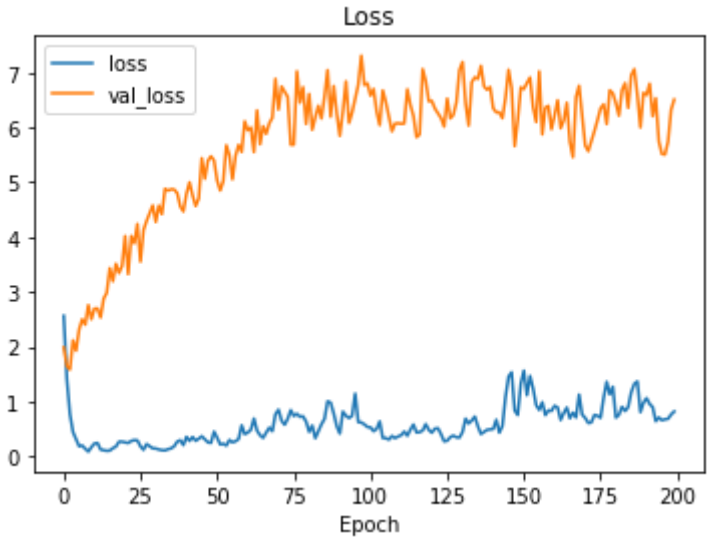


Figura 09 - Erro conforme o aumento das épocas.

Embora o treinamento não tenha se mostrado satisfatório, ao realizar testes com imagens fora do conjunto de treinamento, a rede foi capaz de reconhecer os caracteres. Para realizar o reconhecimento, foi necessário redimensionar a imagem colhida na internet e também normalizá-la assim como feito nas imagens usadas para treinamento.

Nas figuras abaixo, seguem dois exemplos de como foi feito o reconhecimento de uma letra externa ao conjunto de treinamento.

```
[20] #Carregar uma imagem de letra manuscrita da internet
     import requests
     from PIL import Image
     url = 'https://st.depositphotos.com/1077338/3315/i/950/depositphotos_33152079-stock-photo-greek-letter-tau-hand-written.jpg'
     response = requests.get(url, stream=True)
     print(response)
     img = Image.open(response.raw)
     plt.imshow(img)
```

```
<Response [200]>
<matplotlib.image.AxesImage at 0x7f154f625b00>
```

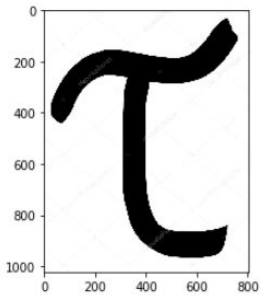


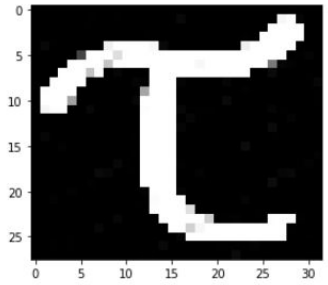
Figura 10 - Reconhecimento de um carácter não presente na base de dados.

```
[21] #Agora é preciso redimensionar a imagem e colocá-la em escala de cinza para que a predição seja feita da maneira menos custosa.
     import cv2

     img_array = np.asarray(img)

     resized = cv2.resize(img_array, (32,28))
     gray_scale = cv2.cvtColor(resized, cv2.COLOR_BGR2GRAY)
     image = cv2.bitwise_not(gray_scale)
     print(gray_scale.shape)
     plt.imshow(image,cmap=plt.get_cmap("gray") )
```

```
(28, 32)
<matplotlib.image.AxesImage at 0x7f154f586b00>
```




```
[22] #Normalizando a matriz RGB
     image = image/255
     image = image.reshape(1,896)
```

```
#A predição é feita
prediction = model.predict_classes(image)
print("Predicted letter:", str(prediction))
```

```
Predicted letter: [19]
```

Figura 11 - Reconhecimento de um carácter não presente na base de dados.

```
#Transformação inversa de número para letra (codificação)
print(le.inverse_transform([prediction]))
```

```
['T']
```

Figura 12 - Reconhecimento de um carácter não presente na base de dados.

O exemplo citado acima, retornou um resultado esperado que era a letra T. Nas figuras abaixo, é possível verificar um exemplo em que não foi possível obter sucesso.

```
[ ] #Carregar uma imagem de letra manuscrita da internet
    import requests
    from PIL import Image
    url = 'https://st2.depositphotos.com/5624298/11071/i/950/depositphotos_110715642-stock-photo-r-letter-written-on-coffee.jpg'
    response = requests.get(url, stream=True)
    print(response)
    img = Image.open(response.raw)
    plt.imshow(img)
```

```
<Response [200]>
<matplotlib.image.AxesImage at 0x7f2a124f5ac8>
```

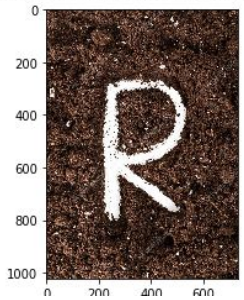


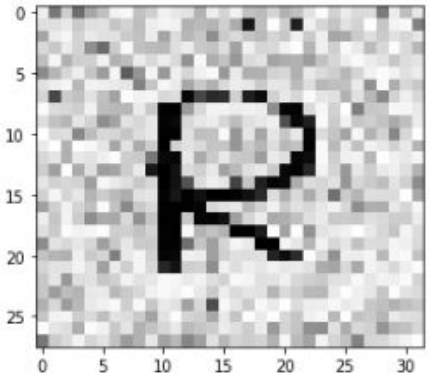
Figura 13 - Reconhecimento de um caractere.

```
[ ] import cv2

    img_array = np.asarray(img)

    resized = cv2.resize(img_array, (32,28))
    gray_scale = cv2.cvtColor(resized, cv2.COLOR_BGR2GRAY)
    image = cv2.bitwise_not(gray_scale)
    print(gray_scale.shape)
    plt.imshow(image,cmap=plt.get_cmap("gray") )
```

```
(28, 32)
<matplotlib.image.AxesImage at 0x7f2a16203a20>
```




```
[ ] #Normalizando a matriz RGB
    image = image/255
    image = image.reshape(1,896)
```

```
[ ] #A predição é feita
    prediction = model.predict_classes(image)
    print("Predicted letter:", str(prediction))
```

```
Predicted letter: [25]
```

```
print(le.inverse_transform([prediction]))
```

```
['Z']
```

Figura 14 - Reconhecimento de um caractere.

Conforme pode ser observado nas imagens acima, foi inserido o caractere “R”, porém a rede reconheceu como letra “Z”. Com a baixa acurácia que foi obtida durante o processo de validação, já era de se esperar que fosse obtido um resultado desse tipo, e por mais que houvesse a alteração dos parâmetros de rede, a predicação foi mal sucedida.

Conclusão

Com base no treinamento realizado e também nas diversas tentativas anteriores de adaptar o processo de treinamento com ênfase nos parâmetros das camadas da rede, na quantidade de épocas e *batch size*, não foi possível obter bons resultados utilizando a rede MLP para o reconhecimento de letras escritas à mão. Embora a rede consiga acertar o reconhecimento de alguns caracteres, os baixos números de acurácia na validação e o alto erro não garantem que haja uma boa generalização na hora de predizer resultados. As diversas tentativas sugerem que o conjunto de dados do treinamento não está eficiente o suficiente para que se trabalhe com uma rede MLP.