Reconhecimento computacional de letras

Introdução

O relatório é composto pelo grupo Tobias Aguiar e Leandro Rodrigues, onde discute-se sobre o seguinte desafio : dada uma letra qualquer escrita à mão, como reconhecer qual letra do alfabeto esta pertence, ou mais se aproxima? Assim, a base de dados apresentada conta com um número considerável de letras escrita com diferentes formas, a missão do algoritmo é identificar ou prever qual letra é, independente da forma na qual está escrita.

Metodologia

O modelo utilizado no desenvolvimento do experimento é a rede neural convolucional (*Convolutional Neural Network* - CNN), uma rede neural arquitetada especificamente para trabalhar na área de processamento de imagens, especificamente na parte de processamento de pixels na qual apresenta a seguinte característica : uma imagem de entrada é colocada na rede, sofre a convolução e produz, por conseguinte, um rótulo de saída para poder classificar a imagem. Assim, o algoritmo utilizado para trabalhar com a CNN são os filtros de convolução, são operações utilizadas para, dada uma imagem, ser gerada uma nova com re-dimensionalidade e, a depender do tipo de filtro utilizado, a nova imagem gerada aparece com detecção de padrões, que variam de acordo com o interesse do usuário para identificar o que se deseja. Logo, eis como funciona este algoritmo convolucional :

- Posicione a detector de característica (DC) no canto superior esquerdo da imagem de entrada (IE) ;
- Multiplique cada elemento de DC por cada elemento equivalente em IE considerando sua posição. Some este resultado e guarde no mapa de característica em uma posição equivalente à imagem de entrada;
- Avance uma posição no em IE;

- Volte para o passo 2 enquanto não chegar no canto inferior direito da IE ;

É necessário, também, reduzir a imagem(pooling layer) para que os dados da imagem estejam mais compactados. Isso possibilita que a matriz seja transformada num vetor e seja a entrada do método de aprendizagem de máquina.

Códigos

Algumas funções são essenciais no método CNN, como visto abaixo. No primeiro momento, é usado funções para separar os dados em entrada e saída para o treinamento e teste.Depois é feito o processo de obtenção do resultado da rede neural: convoluciona a imagem, compacta e cria o vetor de entrada das densas camadas.

```
(X_train, y_train), (X_test, y_test)= mnist.load_data()

X_train = X_train.reshape(60000, 28, 28, 1)

X_test = X_test.reshape(10000, 28, 28, 1)

y_train = to_categorical(y_train, 10)

y_test = to_categorical(y_test, 10)

 model.add(Conv2D(30, (5,5), input_shape=(28,28,1), activation='relu'))
#convoluçao

  model.add(MaxPooling2D(pool_size=(2,2)))              #diminuir tamanho

  model.add(Conv2D(15,(3,3), activation='relu'))

  model.add(MaxPooling2D(pool_size=(2,2)))

  model.add(Flatten())                                  #vetor

  model.add(Dense(500, activation='relu'))           #camada input

  model.add(Dense(num_classes, activation='softmax'))

  model.compile(Adam(lr=0.01), loss='categorical_crossentropy',
metrics=['accuracy'])
```

O código da CNN possui muitas conversões e funções para gerar os resultados. É importante usar as imagens em tons de cinza(Figura 01), com objetivo que o sistema reconheça o formato da letra em destaque da cor do fundo. Outro trecho significante(figura 02), é a o treinamento do deep learning, no caso há 500 neurônios de entrada, 120 na segunda camada que foi utilizado. A última parte consiste no teste(predict) e exibição do resultado.

Figura 01- Modificação no tom das imagens

```
[4]  #Este trecho do código trabalha com  a captura das imagens utilizando o OpenCV

     def read_image(file_path):
         image = cv2.imread(file_path)
         # converte para tons de cinza
         gray_scale = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
         # inverte a cor
         image = cv2.bitwise_not(gray_scale)
         return image

     #O trecho de código a seguir faz parte da chamada principal para a identificação dos arquivos dentro da pasta do conjunto de

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

         return images, labels
```

```
[5]  #Finalmente, eis a chamada da função para a captura das imagens no diretório composto pelo dataset

     images, labels = extract_data('./letras')
     labels = np.reshape(labels, [-1])
```

Figura 02- Treinamento da deep learning

```
[ ]  num_classes = 26
     def create_model():
       model = Sequential()
       model.add(Dense(500, input_dim=num_pixels, activation='relu'))
       model.add(Dense(120, activation='relu'))
       model.add(Dense(num_classes, activation='softmax'))
       model.compile(Adam(lr=0.01), loss='categorical_crossentropy', metrics=['accuracy'])
       return model
```

```
[ ]  #Criação do modelo de treinamento
     model = create_model()
     print(model.summary())
```

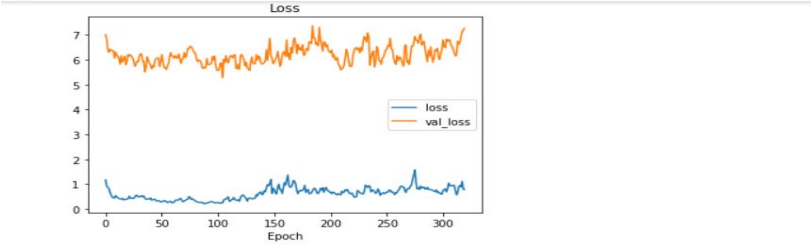
```
[ ]  #Treinamento em si, aqui são selecionadas a proporção de validação utilizada, a quantidade de épocas e o batch size, esses são os mais importantes
     history = model.fit(X_train, y_train, validation_split=0.1, epochs=320, batch_size = 75, verbose = 1, shuffle = 15)
```

Experimentos

Os testes executados foram obtidos de uma base de dados de imagens que representam letras. Percebe-se que esse banco de dados tem poucas imagens e muitas não representam bem o modelo representativo da letra, o que pôde influenciar no resultado. O parâmetro que identifica o resultado desse código de deep learning é a acurácia; nesse caso foi de 60%(figura 03). Isso indica um resultado é razoavelmente bom, porém poderia melhorar com um banco de dados maior e melhor escrito.

Outra coisa que se pode observar da figura 03, é que mesmo mudando os parâmetros da rede(epoch), não há mudança perceptível no erro de validação. Assim, conclui-se que a mudança interna na rede de camadas testada não provoca melhores resultados.

Figura 03- Principais resultados




```
#Aqui é possível ver a acurácia do teste
score = model.evaluate(X_test, y_test, verbose=0)
print(type(score))
print('Test score:',score[0])
print('Test accuracy:',score[1])
```

```
<class 'list'>
Test score: 6.392599096634901
Test accuracy: 0.6009852216748769
```