Luiz Felipe Nagatani Rabello

Relatório LAMIA - Módulo 14

Seção 9

Redes Convolucionais

Redes convolucionais se diferenciam de redes neurais normais por serem adaptadas para receber entradas em estrutura de “grid”, como imagens e vídeos, tornando-as ideais para aplicações como reconhecimento de imagens e detecção de objetos.

A estrutura muda drasticamente, onde para uma rede normal seria necessário um nodo de entrada para cada pixel da imagem, sendo necessário que multiplique esse número por 3 caso seja uma imagem colorida, enquanto isto a rede convolucional processa imagens com base em características ao invés de pixels.

Para isto é necessário o operador convolucional, onde se multiplica a matriz de entrada por um kernel, que é outra matriz que permitirá realçar as características desejadas, o kernel utilizado depende da característica a ser encontrada. Isto resulta numa feature map, que fornece uma matriz menor e mais fácil de ser processada.

O conceito de camadas também pode ser aplicado na feature map, podendo obter-se vários feature maps e deixar o modelo decidir qual o mais adequado para a situação

Pooling

O pooling é uma operação realizada na feature map onde se reduz às características mantendo somente as mais importantes, evitando overfitting e facilitando o processo de reconhecimento da mesma figura em diferentes contextos

Flattening

Por fim tem-se o flattening, que é uma operação que transforma a feature map em uma estrutura unidimensional, permitindo que ela seja organizada em nodos de entrada para ser inserida em uma rede neural.

Seção 10

Prática

Nesta seção é apresentado um projeto de leitura de digitos escritos a mão utilizando redes convolucionais, para isto se criou uma camada utilizando Conv2D com 32 canais de

convolução na entrada, concluindo a etapa de aplicar um operador convolucional, tendo em seguida o passo 2: Pooling.

```
clas = Sequential()
clas.add(Conv2D(32, (3,3), input_shape = (28, 28, 1), activation = 'relu'))
```

Para o Pooling foi utilizado uma camada de MaxPooling2d, com pool_size de 2x2, ou seja, cada partição da matriz original seria uma matriz 2x2.

```
clas.add(MaxPooling2D(pool_size =(2, 2)))
```

Para o flattening utilizou-se a função Flatten, da biblioteca keras, que não necessitou de argumentos, por ser apenas uma função que transformaria o feature map em uma matriz unidimensional.

```
clas.add(Flatten())
```

Por fim teve-se a Etapa 4: criação da rede neural, foram adicionadas duas camadas Densas, uma com 128 nodos(podem e irão ser mudados futuramente para fins de experimentação) e uma com 10 nodos para a saída de 0-9.

Finalmente compilou-se a rede utilizando categorical cross entropy para medir o erro, e treinar o modelo.

```
clas.compile(loss = 'categorical_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
```

```
clas.fit(prev_train, clas_train, batch_size = 128, epochs = 5, validation_data=(prev_test, clas_test))
```

Melhorias

O modelo resultou em 0.24 de perda e 92% de precisão, apresentando um resultado bom, porém deixando espaço para melhorias.

A primeira melhoria foi a adição de uma camada de normalização depois do operador convolucional, e em seguida a repetição de todas as etapas. A função da normalização é eliminar covariância das ativações, aumentando a eficiência do treinamento, já o motivo de se ter múltiplas iterações da estrutura da rede, ajuda a assimilar características mais complexas que não poderiam ser processadas devidamente no grupo inicial de camadas.

Em seguida foi aplicado Dropout nas camadas da rede neural densa para evitar overfitting por conta do grande volume de dados existentes nas redes convolucionais, e aplicado mais uma camada oculta na rede densa para assimilar as características mais complexas, exigindo mais tempo de execução porém aumentando a precisão para 99%.

Por iniciativa do aluno houve a tentativa de aumentar para 6 épocas, o que abaixou a precisão para 98%.

```
resul = clas.evaluate(prev_test, clas_test)
resul
```

```
313/313 [==============================] - 3s 9ms/step - loss: 0.0318 - accuracy: 0.9899
[0.03179524093866348, 0.9898999929428101]
```

Rede Cruzada

Na aula seguinte é apresentado o mesmo projeto utilizando validação cruzada, para isto foi optado a abordagem do StratifiedKFold, por executar o mesmo processo que o KFold porém com balanceamento dos rótulos.

```
kfold = StratifiedKFold(n_splits=5, shuffle = True, random_state = 5)
resultados = []
```

A aplicação da validação cruzada se provou muito mais eficiente para o modelo, garantindo uma precisão de 99% sem a necessidade de um grande número de camadas, mantendo um bom equilíbrio entre precisão e tempo necessário para execução.

Augmenting e teste

A penúltima seção envolve testar outra rede convolucional dedicada a determinar o objeto na foto é um cachorro ou um gato, com o incremento final sendo buscas individuais em fotos especificas ao invés do dataset inteiro.

Também foi apresentado o data augmentation através do ImageDataGenerator, que distorce as imagens presentes no dataset afim de criar variações das imagens e melhorar o treinamento do modelo.

```
gera_train = ImageDataGenerator(rescale = 1./225,
                                rotation_range = 7,
                                horizontal_flip = True,
                                shear_range = 0.2,
                                height_shift_range = 0.0,
                                zoom_range = 0.2)
gera_test = ImageDataGenerator(rescale = 1./225)
base_train = gera_train.flow_from_directory('dataset/training_set',
                                            target_size = (64, 64),
                                            batch_size = 32,
                                            class_mode = 'binary')
base_test = gera_test.flow_from_directory('dataset/test_set',
                                          target_size = (64,64),
                                          batch_size = 32,
                                          class_mode = 'binary')
```

Aproveitando que havia finalizado o conteúdo de testes, resolvi realizar o exercício 7, o resultado obtido da predição foi 8, acredito que por conta da má qualidade da foto e da caligrafia ruim do escritor.

```
imagem = image.load_img('numero.jpeg').convert('L')
imagem = imagem.resize((28,28))
```

```
imagem = image.img_to_array(imagem)
imagem /= 255
imagem =np.expand_dims(imagem, axis=0)
```

```
prev = clas.predict(imagem)
resul = np.argmax(prev)
resul
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step
8
```