19032024_Card14

Teoria de redes neurais convolucionais

Técnica utilizada para implementar algoritmos de visão computacional, para fazer detecção de objetos, corpos, etc. É mais eficiente que o algoritmo SVM, que era cnosiderado o mais eficiente antes das redes neurais convolucionais.

Pixel: Menor unidade de informação digital que compõe uma imagem. Cada pixel é composto por 3 canais de cores RGB (red, green blue), que são combinados para formar uma cor.

Para classifcar uma imagem, é necessário agrupar as características específicas que a descreve, como o formato, cores e grupos de cores, e definir essas características em bancos de dados que as representam numericamente.

Os dados que representam a imagem numericamente são justamente os pixels, que podem ser agrupados em valores binários no caso de imagens preto e branco, ou valores que representam diferentes intensidades para cada um dos canais de cor. Em um modelo de rede neural densa composto por neurônios de entrada, camadas ocultas e neurônios de saída, cada pixel é uma entrada na rede para que o modelo processe toda a imagem, com a resposta no final para fins de validação.

Complicações desta abordagem;

- Saber quais características utilizar ;
- Definir quais são as características mais importantes que descrevem a imagem;

Quanto maior a imagem, maior a quantidade de pixels para processar no modelo, maior a quantidade de características para especificar e por consequência, maior o tempo e poder de proessamento demandado para a tarefa. Para resolver esses problemas são utilizadas as redes neurais convolucionais.

Vantagens da rede neural convolucional:

1. **Não usa todas os pixels da imagem como entrada;** a rede seleciona os pixels que representam as características que melhor definem a imagem.
2. **Usa uma rede neural tradicional;** Tem toda a estrutura da rede densa, mas trata os dados utilizados na camada de entrada.
3. **Define as características principais:** A própria rede neural define quais são as características principais através do CNN (Convolutional Neural Network).

Etapas para a aplicação de CNN:

1. Operador de convolução:

O operador de convolução é utlizado para a extração de características visuais de imagens. Através da aplicação de kernels(filtros), a convolução permite identificar padrões e características importantes na imagem. O Kernel é uma matriz bidimensional de pesos que define quais características serão detectadas na imagem, e o seu tamanho determina a área da imagem que será analisada na operação.

O cálculo do operador de convolução:

O kernel é deslizado pela imagem, fazendo a multiplicação de seus elementos por cada um dos pixels da imagem e somando o resultado, que é guardado em uma nova matriz chamada de mapa de características.

- Com o mapa de características, a imagem fica menor para facilitar o processamento;
- Informações da imagem original podem ser perdidas, mas o propósito é extrair apenas as características principais;
- **Função Relu:** É muito utilizada para processar o mapa de características, por possuir a propridade de zerar valores negativos e manter valores positivos.
- **Camada de convolução:** É a aplicação dos mapas de características em multiplas camadas, para descobrir quais os melhores parâmetros para o kernel.
- A própria rede neural faz a seleção dos melhores valores para o kernel

2. Pooling:

Serve para enfatizar ainda mais as características da imagem, independentemente de como ela é alimentada ao algoritmo; se está nas mesmas condições, em cores ou formas diferentes, etc. Isso é feito reduzindo a dimensionalidade dos mapas de características gerados pelas camadas convolucionais, o que reduz o custo computacinoal e também ajuda a prevenir overfitting.

- **Poolmax:** O max pooling seleciona o valor máximo dentro de uma região específica do mapa de características. Essa operação captura a característica mais dominante dentro daquela região, e é comumente utilizada para extrair caraterísticas como bordas duras e cantos. O max pooling é aplicado a cada um dos mapas de características.

3. Flattening:

É a transformação do mapa de características após a passagem pelo maxpooling, para um vetor linear, para que esse então possa ser passado para a rede neural densa. Os elementos de cada mapa de características são concatenados linha por linha, ou coluna por coluna, dependendo da implementação.

4. Rede neural densa:

A última camada da rede neural convolucional é uma rede neural densa tradicional, que vai recever cada elemento do vetor gerado pelo processo de flarrening, em seus neurôinios da camada de entrada. A partir daí é feito o treinamento da rede neural, atualizando os pesos de suas conexoes para obter o resultado que melhor representa a imagem de entrada. Com o resultado, a rede neural vai poder decidir qual mapa de características gerou o melhor resultado, e atualizará também os seus pesos nas camadas densas.

Prática: Classificação de dígitos escritos à mão

Base de dados MNIST:

MNIST (Modified National Institute of Standards and Technology) é um banco de dados amplamente utilizado para o reconhecimento de dígitos manuscritos. Ele contém 70.000 imagens de dígitos manuscritos, divididas em 60.000 imagens de treinamento e 10.000 imagens de teste. As imagens são divididas igualmente entre 10 classes, de 0 a 9.

Pré processamento dos dados:

Os seguintes procedimentos foram feitos para tratar as imagens do banco de dados MNIST:

- As imagens foram redimensionadas para um formato adequado para a rede neural (28x28 pixels).
- Os dados previsores foram convertidos para o tipo float32.
- Os valores dos pixels foram normalizados dividindo-os por 255.
- Os rótulos das classes foram convertidos para o formato categórico, que é utilizado pela rede neural.

Estrutura da rede neural:

A estrutura do modelo é composta por duas partes principais: uma seção convolucional e uma rede neural densa.

A seção convolucional aplica 32 filtros de tamanho 3x3 às imagens de entrada, extraindo características relevantes. O pooling subsequente com um poolsize de 2x2 reduz a dimensionalidade da saída da camada convolucional.

A rede neural densa, composta por uma camada oculta com 128 neurônios e ReLU como função de ativação, processa o vetor flattened da camada anterior. A última camada da rede densa possui 10 neurônios e utiliza a função softmax para gerar probabilidades de classificação em 10 classes.

Resultados:

```
Epoch 1/5
469/469 [==============================] - 88s 186ms/step - loss: 1.0209 - accuracy: 0.7380 - val_loss: 0.4442 - val_accuracy: 0.8784
Epoch 2/5
469/469 [==============================] - 81s 173ms/step - loss: 0.3981 - accuracy: 0.8862 - val_loss: 0.3426 - val_accuracy: 0.9027
Epoch 3/5
469/469 [==============================] - 69s 147ms/step - loss: 0.3385 - accuracy: 0.9002 - val_loss: 0.3101 - val_accuracy: 0.9108
Epoch 4/5
469/469 [==============================] - 70s 149ms/step - loss: 0.3077 - accuracy: 0.9096 - val_loss: 0.2819 - val_accuracy: 0.9189
Epoch 5/5
469/469 [==============================] - 65s 140ms/step - loss: 0.2833 - accuracy: 0.9169 - val_loss: 0.2605 - val_accuracy: 0.9246
<keras.src.callbacks.History at 0x7baa8d521f90>
```

```
[ ] resultado = classificador.evaluate(previsores_teste, classe_teste)
```

```
313/313 [==============================] - 6s 18ms/step - loss: 0.2605 - accuracy: 0.9246
```

Melhorias na rede neural convolucional:

1. Além da camada de entrada, também é feita a normalização dos dados para as camadas de convolução com a função BatchNormalization, diminuindo significativamente o tempo de treinamento.
2. Adição de mais uma camada de convolução com filtro de características 3x3 e função de ativação 'relu'
3. Adição de uma segunda camada de batch normalization
4. Adição de uma segunda camada de Pooling
5. Adição de mais uma camada de Flatten

Melhorias na rede neural densa:

1. Dropout de 20% nas camada de entrada
2. Adição de uma segunda camada oculta
3. Dropout de 20% na segunda camada oculta

Resultados pós melhoria:

```
Epoch 1/5
469/469 [==============================] - 58s 119ms/step - loss: 0.2393 - accuracy: 0.9262 - val_loss: 0.1108 - val_accuracy: 0.9694
Epoch 2/5
469/469 [==============================] - 54s 116ms/step - loss: 0.0735 - accuracy: 0.9796 - val_loss: 0.0475 - val_accuracy: 0.9861
Epoch 3/5
469/469 [==============================] - 54s 115ms/step - loss: 0.0528 - accuracy: 0.9852 - val_loss: 0.0352 - val_accuracy: 0.9895
Epoch 4/5
469/469 [==============================] - 54s 115ms/step - loss: 0.0410 - accuracy: 0.9883 - val_loss: 0.0299 - val_accuracy: 0.9906
Epoch 5/5
469/469 [==============================] - 53s 113ms/step - loss: 0.0343 - accuracy: 0.9903 - val_loss: 0.0378 - val_accuracy: 0.9896
<keras.src.callbacks.History at 0x7be695dbcf10>
```

```
[20] resultado = classificador.evaluate(previsores_teste, classe_teste)
```

MNIST com Cross Validation

Configuração:

Modelo com configuração parecida com a anterior com duas camadas convolucionais, cada uma com 32 filtros e kernel de tamanho 3x3. A camada densa tem 128 neurônios ativados pela função ReLU, conectada a uma camada de saída com 10 neurônios ativados pela função Softmax, para classificação final.

A Validação cruzada é utilizada para avaliar o desempenho do modelo em diferentes subconjuntos do conjunto de dados, utilizando a função StratifiedKFold para dividir o conjunto de dados em 5 folds; 4 para treinamento e 1 para teste.

Resultados:

```
Epoch 1/5
375/375 [==============================] - 31s 78ms/step - loss: 1.3049 - accuracy: 0.9057
Epoch 2/5
375/375 [==============================] - 23s 62ms/step - loss: 0.1083 - accuracy: 0.9698
Epoch 3/5
375/375 [==============================] - 23s 61ms/step - loss: 0.0596 - accuracy: 0.9824
Epoch 4/5
375/375 [==============================] - 23s 61ms/step - loss: 0.0368 - accuracy: 0.9886
Epoch 5/5
375/375 [==============================] - 22s 59ms/step - loss: 0.0315 - accuracy: 0.9905
375/375 [==============================] - 3s 7ms/step - loss: 0.1089 - accuracy: 0.9762
```

MNIST com augmentation

Técnica utilizada para aumentar a quantidade de imagens de um conjunto de dados com poucas imagens, mudando aspectos como rotação, direção dos pixels, zoom, etc. Isso aumenta significativamente o tamaho da base de dados, e o modelo consegue se adaptar melhor e previnir o overfitting.

Resultados:

```
Epoch 1/10
<ipython-input-18-fad1c8de25ec>:1: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use
  classificador.fit_generator(base_treinamento, steps_per_epoch = 60000/128,
468/468 [==============================] - 46s 97ms/step - loss: 0.8102 - accuracy: 0.7255 - val_loss: 0.6713 - val_accuracy: 0.7733
Epoch 2/10
468/468 [==============================] - 44s 94ms/step - loss: 0.7601 - accuracy: 0.7432 - val_loss: 0.6228 - val_accuracy: 0.7896
Epoch 3/10
468/468 [==============================] - 46s 98ms/step - loss: 0.7211 - accuracy: 0.7548 - val_loss: 0.5811 - val_accuracy: 0.8070
Epoch 4/10
468/468 [==============================] - 44s 94ms/step - loss: 0.6756 - accuracy: 0.7688 - val_loss: 0.5408 - val_accuracy: 0.8190
Epoch 5/10
468/468 [==============================] - 45s 96ms/step - loss: 0.6358 - accuracy: 0.7831 - val_loss: 0.5232 - val_accuracy: 0.8183
Epoch 6/10
468/468 [==============================] - 44s 94ms/step - loss: 0.6037 - accuracy: 0.7942 - val_loss: 0.4835 - val_accuracy: 0.8340
Epoch 7/10
468/468 [==============================] - 44s 93ms/step - loss: 0.5741 - accuracy: 0.8073 - val_loss: 0.4631 - val_accuracy: 0.8362
Epoch 8/10
468/468 [==============================] - 44s 94ms/step - loss: 0.5498 - accuracy: 0.8159 - val_loss: 0.4289 - val_accuracy: 0.8585
Epoch 9/10
468/468 [==============================] - 46s 98ms/step - loss: 0.5298 - accuracy: 0.8234 - val_loss: 0.4227 - val_accuracy: 0.8583
Epoch 10/10
468/468 [==============================] - 44s 93ms/step - loss: 0.5157 - accuracy: 0.8291 - val_loss: 0.3918 - val_accuracy: 0.8721
<keras.src.callbacks.History at 0x79c8c1c23c10>
```

Curiosamente, os resultados com augmentation tiveram um resultado inferior aos outros métodos, mesmo rodando 10 épocas.