CNN para Dataset de Digitos Manuscritos MNIST

Disponível em:https://www.geeksforgeeks.org/applying-convolutional-neural-network-on-mnist-dataset/

Adaptado por: Dr. Arnaldo de Carvalho Junior – junho 2025

1. Introdução

CNN é um modelo conhecido por ser um Rede Neural Convolucional (convolutional neural network) e nos últimos tempos ganhou muita popularidade por causa de sua utilidade. A CNN usa perceptrons multicamadas para fazer trabalho computacional. A CNN usa relativamente pouco pré-processamento em comparação com outros algoritmos de classificação de imagens. Isso significa que a rede aprende por meio de filtros que, nos algoritmos tradicionais, foram projetados manualmente. Assim, para as tarefas de processamento de imagens, as CNNs são a opção mais adequada.

2. Aplicando Rede Neural Convolucional no conjunto de dados mnist

Aplicar uma CNN no conjunto de dados MNIST é uma maneira popular de aprender e demonstrar os recursos das CNNs para tarefas de classificação de imagens. O conjunto de dados MNIST consiste em imagens em escala de cinza 28x28 de dígitos manuscritos (0-9), com um conjunto de treinamento de 60.000 exemplos e um conjunto de teste de 10.000 exemplos.

2.1. Abordagem Básica com Keras

Aqui está uma abordagem básica para aplicar uma CNN no conjunto de dados MNIST usando a linguagem de programação Python e a biblioteca Keras:

1. Carregar e pré-processar os dados: O conjunto de dados MNIST pode ser carregado usando a biblioteca Keras, e as imagens podem ser normalizadas para ter valores de pixel entre 0 e 1.
2. Defina a arquitetura do modelo: A CNN pode ser construída usando a API Sequencial Keras, que permite a fácil construção de modelos sequenciais camada por camada. A arquitetura normalmente deve incluir camadas convolucionais, camadas de agrupamento e camadas totalmente conectadas.

3. Compilar o modelo: O modelo precisa ser compilado com uma função de perda, um otimizador e uma métrica para avaliação.
4. Treine o modelo: O modelo pode ser treinado no conjunto de treinamento usando a função Keras fit(). É importante monitorar a precisão e a perda do treinamento para garantir que o modelo esteja convergindo adequadamente.
5. Avalie o modelo: O modelo treinado pode ser avaliado no conjunto de testes usando a função Keras avalie(). A métrica de avaliação normalmente usada para tarefas de classificação é a precisão.

2.2 Recomendações

A seguir são apresentadas algumas recomendações de melhores práticas a ter em mente ao aplicar uma CNN no conjunto de dados do MNIST:

1. Comece com uma arquitetura simples e aumente gradualmente a complexidade, se necessário.
2. Experimente diferentes funções de ativação, otimizadores, taxas de aprendizado e tamanhos de lote para encontrar a combinação ideal para sua tarefa específica.
3. Use técnicas de regularização, como abandono ou queda de peso, para evitar overfitting.
4. Visualize os filtros e mapas de recursos aprendidos pelo modelo para obter insights sobre seu funcionamento interno.
5. Compare o desempenho da CNN com outros algoritmos de aprendizado de máquina, como Support Vector Machines ou Random Forests, para ter uma noção de seu desempenho relativo.

2.3. Referências Utilizadas

1. Conjunto de dados MNIST: http://yann.lecun.com/exdb/mnist/
2. Documentação Keras: https://keras.io/
3. "Aprendizado Profundo com Python" de François Chollet (https://www.manning.com/books/deep-learning-with-python)

2.4. **Conjunto de dados MNIST**

O conjunto de dados Mnist é um conjunto de dados de imagens manuscritas conforme mostrado na Figura 1.

Figura 1 – Exemplo de digitos manuscritos do dataset mnist
Fonte: Adaptado de [1]

Pode-se obter 99,06% de precisão usando CNN (Convolutional Neural Network) com um modelo funcional. A razão para usar um modelo funcional é manter a facilidade ao conectar as camadas.

3. Descrição do Código

3.1. Em primeiro lugar, incluir todas as bibliotecas necessárias

```
import numpy as np
```

```
import keras
from keras.datasets import mnist
from keras.models import Model
from keras.layers import Dense, Input
from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten
from keras import backend as k
```

3.2. Crie os dados do trem e os dados de teste

a) **Dados de teste:** Usado para testar o modelo de como nosso modelo foi treinado.
b) **Dados do treino:** Usado para treinar nosso modelo.

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

3.3. Formatação dos dados

Enquanto prossegue, **img_rows** e **img_cols** são usados como dimensões da imagem. No conjunto de dados “mnist”, é 28 e 28. Também é necessário verificar o formato dos dados ou seja, 'canais_primeiro' ou 'canais_último'. Na CNN, pode-se normalizar os dados

antes de lidar com grandes quantidades de termos e cálculos possam ser reduzidos a termos menores. Tipo, pode-se normalizar os dados x_train e x_test dividindo-os por 255.

3.4. Verificando o formato dos dados

```
img_rows, img_cols=28, 28
```

```
if k.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
    inpx = (1, img_rows, img_cols)

else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
    x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
    inpx = (img_rows, img_cols, 1)

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
```

3.5. Descrição das classes de saída

Como a saída do modelo pode compreender qualquer um dos dígitos entre 0 e 9, são necessárias 10 classes na saída. Para produzir 10 classes, use a função “keras.utils.to_categorical”, que fornecerá as 10 colunas. Destas 10 colunas, apenas um valor será um e o resto 9 será zero e este valor da saída denota a classe do dígito.

```
y_train = keras.utils.to_categorical(y_train)
```

```
y_test = keras.utils.to_categorical(y_test)
```

Agora, o conjunto de dados está pronto, então pode-se avançar para o modelo CNN :

3.5. Modelo de CNN

a) layer1 é a camada Conv2d que convoluciona a imagem usando 32 filtros cada um de tamanho (3*3).

b) layer2 é novamente uma camada Conv2 D que também é usada para convolucionar a imagem e está usando 64 filtros cada um de tamanho (3*3).

c) layer3 é a camada MaxPooling2D que escolhe o valor máximo de uma matriz de tamanho (3*3).

d) layer4 está mostrando Dropout a uma taxa de 0,5.

e) A camada 5 está achatando a saída obtida da camada 4 e esta saída achata é passada para a camada 6.

f) A camada 6 é uma camada oculta de uma rede neural contendo 250 neurônios.

g) A camada 7 é a camada de saída com 10 neurônios para 10 classes de saída que utiliza a função softmax.

```
inpx = Input(shape=inpx)
layer1 = Conv2D(32, kernel_size=(3, 3), activation='relu')(inpx)
layer2 = Conv2D(64, (3, 3), activation='relu')(layer1)
layer3 = MaxPooling2D(pool_size=(3, 3))(layer2)
layer4 = Dropout(0.5)(layer3)
layer5 = Flatten()(layer4)
layer6 = Dense(250, activation='sigmoid')(layer5)
layer7 = Dense(10, activation='softmax')(layer6)
```

3.6. Chamando a função de compilação e ajuste:

```
model = Model([inpx], layer7)
```

```
model.compile(optimizer=keras.optimizers.Adadelta(),
              loss=keras.losses.categorical_crossentropy,
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=12, batch_size=500)
```

Saída:

```
Época 1/12
120/120 ━━━━━━━━━━━━━━━━━━━━ 154s 1s/etapa - precisão: 0,0968 - perda:
2,4955
Época 2/12
120/120 ━━━━━━━━━━━━━━━━━━━━ 201s 1s/etapa - precisão: 0,0957 - perda:
2,4752
Época 3/12
120/120 ━━━━━━━━━━━━━━━━━━━━ 203s 1s/etapa - precisão: 0,0995 - perda:
2,4479
Época 12/04
120/120 ━━━━━━━━━━━━━━━━━━━━ 151s 1s/etapa - precisão: 0,0984 - perda:
2,4262
Época 5/12
```

```
120/120 ━━━━━━━━━━━━━━━━━━━━ 204s 1s/etapa - precisão: 0,0980 - perda:
2,4085
Época 6/12
120/120 ━━━━━━━━━━━━━━━━━━━━ 201s 1s/etapa - precisão: 0,0970 - perda:
2,3864
Época 7/12
120/120 ━━━━━━━━━━━━━━━━━━━━ 202s 1s/etapa - precisão: 0,0982 - perda:
2,3699
Época 8/12
120/120 ━━━━━━━━━━━━━━━━━━━━ 152s 1s/etapa - precisão: 0,0972 - perda:
2,3520
Época 12/09
120/120 ━━━━━━━━━━━━━━━━━━━━ 201s 1s/etapa - precisão: 0,0975 - perda:
2,3324
Época 10/12
120/120 ━━━━━━━━━━━━━━━━━━━━ 203s 1s/etapa - precisão: 0,0966 - perda:
2,3151
Época 11/12
120/120 ━━━━━━━━━━━━━━━━━━━━ 153s 1s/etapa - precisão: 0,0960 - perda:
2,3017
Época 12/12
120/120 ━━━━━━━━━━━━━━━━━━━━ 201s 1s/etapa - precisão: 0,0972 - perda:
2,2847
<keras.src.callbacks.history.History em 0x7b04c491e200>
```

Primeiramente, foi feito um objeto do modelo conforme mostrado nas linhas fornecidas acima, onde [inpx] é a entrada no modelo e a layer7 é a saída do modelo. Compilou-se o modelo usando o otimizador necessário, função de perda e impresso a precisão e no último modelo.fit foi chamado junto com parâmetros como x_train (significando vetores de imagens), y_train (significando o rótulo), número de épocas (*epochs*) e tamanho do lote. Usando a função de ajuste (*fit*) x_train, o conjunto de dados y_train é alimentado para modelar em tamanho de lote específico.

3.7. Avaliar o modelo

A função “model.evaluate” fornece a pontuação para os dados de teste, ou seja, forneceu os dados de teste para o modelo. Agora, o modelo preverá a classe dos dados, e a classe prevista será combinada com o rótulo y_test para dar a precisão.

```
score = model.evaluate(x_test, y_test, verbose=0)
```

```
print('perda =', score[0])
print('acurácia =', score[1])
```

Saída:

```
perda = 2.269895553588867
acurácia = 0,09950000047683716
```

Código completo no Anexo I.

Exemplo disponível em:
<https://colab.research.google.com/drive/1b7HxqbNaCIu-6OiUJyWk_LsKk56-YZWj#scrollTo=Cb03NkIWkWzt&uniqifier=1>

Referências

[1] GEEKSFORGEEKS, Applying Convolutional Neural Network on mnist dataset, geeksforgeeks.org, 2024. Disponível em: <https://www.geeksforgeeks.org/applying-convolutional-neural-network-on-mnist-dataset/>. Acessado em Junho 02, 2025.

Anexo I

```
# Ref: https://www.geeksforgeeks.org/applying-convolutional-neural-network-on-mnist-dataset/

# Importando as bibliotecas necessárias
import numpy as np
import keras
from keras.datasets import mnist # banco de dados de digitos manuscritos "mnist"
from keras.models import Model
from keras.layers import Dense, Input
from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten
from keras import backend as k

# Criando dados de treinamento e dados de teste
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Verificando o formato do arquivo
img_rows, img_cols=28, 28

if k.image_data_format() == 'channels_first':
  x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
  x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
  inpx = (1, img_rows, img_cols)

else:
  x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
  x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
  inpx = (img_rows, img_cols, 1)

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255

# A função keras.utils.to_categorical fornece 10 colunas. A saida dessas 10
# colunas, somente uma será 1 e as demais zero e esse valor 1 de saida
# denotará a classe do digito.
y_train = keras.utils.to_categorical(y_train)
y_test = keras.utils.to_categorical(y_test)
# Agora o dataset está pronto

# Criando o modelo de CNN
# layer1 é a camada Conv2d que convoluciona a imagem usando 32 filtros, cada de
# dimensão (3*3).
# layer2 é novamente uma camada Conv2D que também é usada para convolucionar a
# imagem e está usando 64 filtros, cada de dimensão (3*3).
# layer3 é uma camada MaxPooling2D layer que pega o valor de saída máximo
# de uma matriz de dimensão (3*3).
# layer4 está mostrando a saída em uma taxa de 0.5.
# layer5 está achatando a saída obtida da camada 4 e seu achatamento é entregue
# para a camada 6.
```

```
# layer6 é uma camada oculta de rede neural com 250 neurônios.
# layer7 é a camada de saída com 10 neurônios para 10 classes de saída que está
# usando função softmax.
inpx = Input(shape=inpx)
layer1 = Conv2D(32, kernel_size=(3, 3), activation='relu')(inpx)
layer2 = Conv2D(64, (3, 3), activation='relu')(layer1)
layer3 = MaxPooling2D(pool_size=(3, 3))(layer2)
layer4 = Dropout(0.5)(layer3)
layer5 = Flatten()(layer4)
layer6 = Dense(250, activation='sigmoid')(layer5)
layer7 = Dense(10, activation='softmax')(layer6)

# Chamando as funções de compilação (compile) e ajuste (fit)
model = Model([inpx], layer7)
model.compile(optimizer=keras.optimizers.Adadelta(),
        loss=keras.losses.categorical_crossentropy,
        metrics=['accuracy'])

model.fit(x_train, y_train, epochs=12, batch_size=500)

# Avaliando o modelo
score = model.evaluate(x_test, y_test, verbose=0)
print('perda =', score[0])
print('acurácia =', score[1])
```