«به نام خدا»

Convolutional Neural Networks

CNN

شبکه‌های عصبی کانولوشنال

ارائه دهنده : مریم دولتشاهی

استاد محترم : آقای دکتر آرشی

شبکه‌های عصبی کانولوشنال (CNN یا Convolutional Neural Networks) یکی از پرکاربردترین معماری‌های یادگیری عمیق برای پردازش داده‌های با ساختار شبکه‌ای مانند تصاویر، ویدیوها و حتی سیگنال‌های صوتی هستند. این شبکه‌ها به ویژه در بینایی کامپیوتر (Computer Vision) عملکردی فوق‌العاده دارند.

1. ساختار کلی CNN

یک CNN معمولاً از سه نوع لایه اصلی تشکیل شده است:

1. لایه‌های کانولوشن (Convolutional Layers)
2. لایه‌های Pooling (Pooling Layers)
3. لایه‌های Fully Connected (Fully Connected Layers)

1. لایه کانولوشن (Convolutional Layer)

- عملکرد: این لایه با اعمال فیلترها (kernels) روی تصویر ورودی، ویژگی‌های محلی مانند لبه‌ها، رنگ، و بافت را استخراج می‌کند.

- مفاهیم کلیدی:

- فیلتر (Kernel): یک ماتریس کوچک که روی تصویر حرکت می‌کند و با ضرب عناصر تصویر در وزن‌های خود، یک نقشه ویژگی (Feature Map) تولید می‌کند.

- Stride: تعداد پیکسل‌هایی که فیلتر در هر حرکت جابه‌جا می‌شود. اگر stride=1 باشد، فیلتر یک پیکسل به راست یا پایین حرکت می‌کند.

- Padding: اضافه کردن پیکسل‌های صفر به اطراف تصویر برای حفظ ابعاد خروجی (same padding) یا کاهش ابعاد (valid padding).

2. لایه Pooling

- هدف: کاهش ابعاد داده و حفظ مهم‌ترین ویژگی‌ها برای کاهش محاسبات و جلوگیری Overfitting.

- انواع Pooling:

- Max Pooling: بیشترین مقدار در هر ناحیه را انتخاب می‌کند (رایج‌ترین نوع).

- Average Pooling: میانگین مقادیر ناحیه را محاسبه می‌کند.

3.لایه Fully Connected (FC)

- عملکرد: پس از استخراج ویژگی‌ها توسط لایه‌های کانولوشن و Pooling، این لایه‌ها برای طبقه‌بندی نهایی استفاده می‌شوند.

- مشابه شبکه‌های عصبی معمولی (MLP): هر نورون به تمام نورون‌های لایه بعدی متصل است.

- کاربرد: تبدیل ویژگی‌های استخراج‌شده به برچسب‌های خروجی (مثلاً تشخیص رقم در MNIST).

2. مزایای CNN نسبت به شبکه‌های عصبی معمولی (MLP)

1. حفظ ساختار فضایی تصاویر:

- در MLP، تصویر به یک بردار مسطح تبدیل می‌شود و اطلاعات مکانی از بین می‌رود. اما CNN با فیلترهای کانولوشن، روابط محلی پیکسل‌ها را حفظ می‌کند.

2. پارامترهای کمتر:

- در MLP، هر پیکسل به تمام نورون‌های لایه بعدی متصل است (پارامترهای زیاد). اما CNN با اشتراک گذاری وزن‌ها ،پارامترها را کاهش می‌دهد.

3. انتقال‌پذیری (Transfer Learning):

- CNNهای آموزش‌دیده روی یک دیتاست را ،می‌توان برای کارهای دیگر تنظیم مجدد کرد.

3. معماری‌های معروف CNN

1. LeNet-5 (1998): اولین CNN موفق برای تشخیص ارقام دستنوشته.

2. AlexNet (2012): پیروز در مسابقه ImageNet با استفاده از ReLU و Dropout.

3. VGG (2014): با لایه‌های x33 کانولوشن و عمق بیشتر.

4. ResNet (2015): معرفی بلوک‌های باقیمانده (Residual Blocks) برای آموزش شبکه‌های بسیار عمیق.

مثال کاربردی: تشخیص ارقام MNIST با CNN

مرحله ۱: بارگذاری کتابخانه‌های لازم

```
from google.colab import files

uploaded = files.upload()

for fn in uploaded.keys():
  print(f'فایل "{fn}" توسط کاربر با حجم {len(uploaded[fn])} بایت آپلود شد.')
```

chinese_mnist.csv(text/csv) - 234341 bytes, last modified: ۲۸/۳/۱۴۰۴ - 100% done

Saving chinese_mnist.csv to chinese_mnist.csv

فایل "chinese_mnist.csv" با حجم 234341 بایت آپلود شد توسط کاربر.

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt
```

مرحله ۲: بارگذاری داده‌ها

داده‌های آموزشی و آزمایشی را از فایل‌های .npy بارگذاری می کنیم . اگر فایل‌ها وجود نداشته باشند، از دیتاست MNIST به عنوان جایگزین استفاده می‌شود.

```
try:
    X_train = np.load('A1_train_images.npy')
    y_train = np.load('A1_train_labels.npy')
    X_test = np.load('A2_test_images.npy')
    y_test = np.load('A2_test_labels.npy')
    print("بارگذاری شدند npy. داده‌ها با موفقیت از فایل‌های!")
    (X_train, y_train), (X_test, y_test) = tf.keras.datasets.mnist.load_data()
```

مرحله ۳: پیش‌پردازش داده‌ها

-نرمال‌سازی: مقادیر پیکسل‌ها از بازه [0,255] به [0,1] تبدیل می‌شوند.

-تبدیل برچسب‌ها به One-Hot Encoding: برای استفاده در مدل طبقه‌بندی.

- تغییر شکل داده‌ها: تصاویر به بردارهای یک‌بعدی (برای شبکه عصبی ساده) تبدیل می‌شوند.

```
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0
y_train = to_categorical(y_train, num_classes=10)
y_test = to_categorical(y_test, num_classes=10)
```

مرحله ۴: ساخت مدل شبکه عصبی

مدل از سه لایه تشکیل شده است:

```
    Flatten(input_shape=(28, 28)),
```

یک لایه پنهان با ۱۲۸ نرون و تابع فعال‌سازی ReLU

```
    Dense(128, activation='relu'),
```

لایه خروجی با ۱۰ نرون (برای ارقام ۰-۹) و تابع فعال‌سازی softmax برای احتمالات

```
Dense(10, activation='softmax')
```

مرحله ۵: کامپایل و آموزش مدل

-کامپایل: از Adam بهینه‌ساز و categorical_crossentropy به عنوان تابع هزینه استفاده می‌شود.

-آموزش: مدل با ۱۰ دوره (epochs) و اندازه دسته‌ی ۳۲ آموزش می‌بیند. ۲۰٪ داده‌ها برای اعتبارسنجی (validation_split) جدا می‌شوند.

```
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
history = model.fit(X_train, y_train, epochs=10, batch_size=32,
validation_split=0.2)
```

مرحله ۶: ارزیابی مدل

مدل روی داده‌های آزمایشی ارزیابی می‌شود و دقت (accuracy) و تابع هزینه (loss) چاپ می‌شوند.

```
loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
print(f"Test Loss: {loss:.4f}")
print(f"Test Accuracy: {accuracy:.4f}")
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/reshaping/flatten.py:37:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer.
When using Sequential models, prefer using an `Input(shape)` object as the first
layer in the model instead.
  super().__init__(**kwargs)
Epoch 1/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12s 7ms/step -
accuracy: 0.8601 - loss: 0.4803 - val_accuracy: 0.9540 - val_loss: 0.1605
```

Epoch 2/10

1500/1500 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 17s 5ms/step - accuracy: 0.9597 - loss: 0.1386 - val_accuracy: 0.9620 - val_loss: 0.1269

Epoch 3/10

1500/1500 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9731 - loss: 0.0899 - val_accuracy: 0.9703 - val_loss: 0.1009

Epoch 4/10

1500/1500 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 11s 4ms/step - accuracy: 0.9812 - loss: 0.0657 - val_accuracy: 0.9712 - val_loss: 0.0959

Epoch 5/10

1500/1500 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9854 - loss: 0.0491 - val_accuracy: 0.9758 - val_loss: 0.0858

Epoch 6/10

1500/1500 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 9s 4ms/step - accuracy: 0.9899 - loss: 0.0354 - val_accuracy: 0.9745 - val_loss: 0.0904

Epoch 7/10

1500/1500 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.9913 - loss: 0.0290 - val_accuracy: 0.9752 - val_loss: 0.0929

Epoch 8/10

1500/1500 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9928 - loss: 0.0232 - val_accuracy: 0.9744 - val_loss: 0.0899

Epoch 9/10

1500/1500 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.9939 - loss: 0.0202 - val_accuracy: 0.9760 - val_loss: 0.0882

Epoch 10/10

1500/1500 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9962 - loss: 0.0138 - val_accuracy: 0.9756 - val_loss: 0.1000

Test Loss: 0.0929

Test Accuracy: 0.9745

مرحله ۷: پیش‌بینی روی یک تصویر تکی

تابع classify_single_digit برای پیش‌بینی رقم در یک تصویر دلخواه استفاده می‌شود:

1.تصویر نرمال‌سازی و تغییر شکل می‌یابد.

2.مدل پیش‌بینی را انجام می‌دهد.

3.رقم پیش‌بینی شده و میزان اطمینان نمایش داده می‌شود.

```
def classify_single_digit(model, image_array):
    if image_array.max() > 1.0:
     image_array = image_array.astype('float32') / 255.0
  input_image = np.expand_dims(image_array, axis=0)
  predictions = model.predict(input_image)
  predicted_digit = np.argmax(predictions[0])
  confidence = np.max(predictions[0])

  print(f"مدل پیش‌بینی کرد: {predicted_digit}")
  print(f"اطمینان (Confidence): {confidence:.2f}")
  plt.imshow(image_array.reshape(28, 28), cmap='gray')
  plt.title(f"پیش‌بینی شده: {predicted_digit} (اطمینان: {confidence:.2f})")
  plt.axis('off')
  plt.show()

  return predicted_digit, confidence
```

مرحله ۸: استفاده از مدل

کد سه گزینه برای تست مدل ارائه می‌دهد:

1.استفاده از تصاویر آزمایشی: مثلاً اولین تصویر در X_test.

2.ساخت تصویر ساختگی: مانند یک مربع ساده .

3.بارگذاری تصویر سفارشی: مانند یک فایل digit.png .

با تغییر این ایندکس تصاویر مختلف رو تست می کنیم ← sample_image_index = 0

--- طبقه‌بندی یک تصویر از مجموعه آزمایشی ---

برچسب واقعی برای این تصویر: 7

1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ **0s** 162ms/step

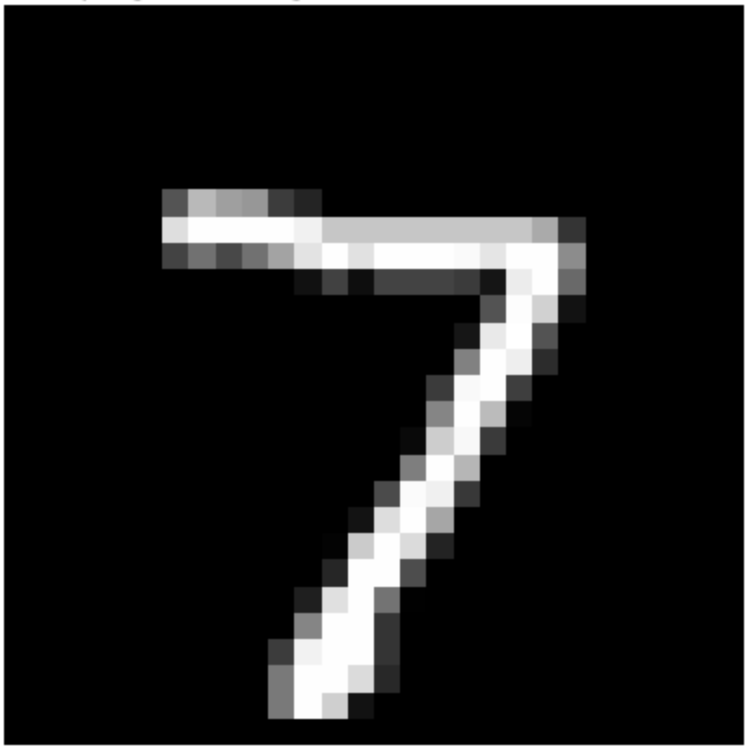
مدل پیش‌بینی کرد: 7

اطمینان (Confidence): 1.00

نتیجه اجرا

خروجی نشان می‌دهد که مدل با دقت ۹۷٫۴۵٪ روی داده‌های آزمایشی عمل کرده است. برای مثال، یک تصویر آزمایشی با برچسب واقعی ۷ به درستی با اطمینان تقریبا ۱٫۰۰ پیش‌بینی کرده است.

پیش‌بینی شده: 7 (اطمینان: 1.00)



«پایان»