Convolutional Neural Networks (CNNs) in Python with Keras

Bisma Fajar

August 2024

1 Introduction to Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) are a class of deep learning models widely used for analyzing visual data. Unlike traditional neural networks, CNNs utilize a grid-like topology, making them especially suitable for image processing tasks. They are composed of several layers, such as convolutional layers, pooling layers, and fully connected layers, which allow the model to learn spatial hierarchies of features from input data.

1.1 Key Components of CNNs

- **Convolutional Layers**: These layers apply a set of learnable filters to the input to extract features. The filters are small squares (e.g., 3x3 or 5x5 pixels) that slide over the input image to produce feature maps.
- **Pooling Layers**: Pooling layers reduce the spatial dimensions of the feature maps, thereby reducing the number of parameters and computation in the network. A common operation is MaxPooling, which selects the maximum value within a window (e.g., 2x2) to down-sample the input.
- **Fully Connected Layers**: These layers are standard neural network layers where each neuron is connected to every neuron in the previous layer. They are usually placed at the end of the CNN to make predictions based on the features extracted by the convolutional and pooling layers.
- **Activation Functions**: Commonly used activation functions in CNNs include ReLU (Rectified Linear Unit) and softmax. ReLU introduces non-linearity into the model, while softmax is often used in the output layer for multi-class classification tasks.

2 Python Implementation of CNN using Keras

Below is the Python code to implement a CNN for image classification using the CIFAR-10 dataset. CIFAR-10 is a widely-used dataset that contains 60,000 32x32 color images in 10 different classes.

2.1 Python Code Explanation

1. **Import Libraries**: The necessary libraries such as TensorFlow, Keras, and NumPy are imported to set up the CNN model.
2. **Load and Preprocess Data**: The CIFAR-10 dataset is loaded, and the pixel values are normalized to be between 0 and 1. The labels are one-hot encoded to be compatible with the categorical cross-entropy loss function.
3. **Build the Model**: The model is built using Keras' Sequential API. It consists of three convolutional layers with ReLU activation, followed by max-pooling layers. The output is flattened and passed through two fully connected layers.
4. **Compile the Model**: The model is compiled with the Adam optimizer and categorical cross-entropy loss function.
5. **Train the Model**: The model is trained on the training dataset for 10 epochs with a batch size of 64.
6. **Evaluate the Model**: The trained model is evaluated on the test dataset to determine its accuracy.
7. **Save and Load the Model**: The model is saved to an '.h5' file, which includes the architecture, weights, and optimizer state, allowing for easy reloading and continued training or inference.

3 Python Code for CNN

```
import sys
import io
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
    Dense, Dropout
from tensorflow.keras.utils import to_categorical

# Set the default encoding to UTF-8
sys.stdout = io.TextIOWrapper(sys.stdout.buffer, encoding='utf-8')

# Load and preprocess the CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

```
# Normalize the pixel values to be between 0 and 1
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

# One-hot encode the labels
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# Build the CNN model
model = Sequential()

# 1st Convolutional Layer
model.add(Conv2D(32, (3, 3), padding='same', activation='relu',
    input_shape=(32, 32, 3)))
model.add(MaxPooling2D(pool_size=(2, 2)))

# 2nd Convolutional Layer
model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# 3rd Convolutional Layer
model.add(Conv2D(128, (3, 3), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# Flatten the layers
model.add(Flatten())

# Fully Connected Layers
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))  # Dropout to prevent overfitting
model.add(Dense(10, activation='softmax'))  # Output layer for 10
    classes

# Compile the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
model.fit(x_train, y_train,
          validation_data=(x_test, y_test),
          epochs=10,
          batch_size=64)

# Evaluate the model
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print(f"Test␣Accuracy:␣{test_accuracy␣*␣100:.2f}%")

# Save the model to an .h5 file
model.save('cnn_cifar10_model.h5')

# Load the model from the .h5 file (optional)
loaded_model = tf.keras.models.load_model('cnn_cifar10_model.h5')
```

4 Explanation of .h5 Files

The `.h5` file extension stands for **Hierarchical Data Format version 5** (HDF5), which is a file format and set of tools for managing complex data. In the context of deep learning with Keras, the `.h5` file format is used to save an entire model, including:

- The **architecture** of the model, allowing for re-creation of the model.
- The **weights** of the model, learned during training.
- The **training configuration** (loss, optimizer).
- The **state** of the optimizer, allowing training to be resumed from where it left off.

4.1 Where to Open .h5 Files

- **Keras/TensorFlow**: The `.h5` file can be loaded directly in Keras or TensorFlow using the `load_model` function, allowing you to resume training or perform inference using the saved model.
- **HDF5 Viewer/Editor**: Tools like **HDFView** (available for Windows, macOS, and Linux) or the **h5py** Python library can be used to view or manipulate the contents of an HDF5 file.
- **Python Libraries**:
  - **h5py**: A Python library that provides a Pythonic interface to the HDF5 binary data format, allowing you to read/write data from/to `.h5` files.