Code Explanation: CNN Training with CIFAR-10 and Carbon Emission Tracking

The provided code demonstrates how to train a **Convolutional Neural Network (CNN)** on the **CIFAR-10** dataset while tracking the carbon emissions produced during the training process using the **CodeCarbon** library.

---

1. **Importing Necessary Libraries**

```python
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D
from tensorflow.keras.utils import to_categorical
from codecarbon import EmissionsTracker
import matplotlib.pyplot as plt
import time
```

- **TensorFlow and Keras**: These libraries are used for building and training the neural network. Keras provides a high-level API that simplifies the implementation of the CNN.
- **CIFAR-10**: This dataset is loaded using `tensorflow.keras.datasets.cifar10`. It contains 60,000 32x32 color images in 10 different classes (e.g., airplanes, cars, birds, etc.).

- **CodeCarbon**: This library tracks the carbon emissions of the hardware during model training.

- **Matplotlib**: Used for visualizing the training and validation accuracy over time.

- **Time**: This is used to calculate the total training time.

---

2. **Loading and Preprocessing the CIFAR-10 Dataset**

```python
(X_train, y_train), (X_test, y_test) = cifar10.load_data()

# Normalize pixel values
X_train = X_train.astype('float32') / 255
X_test = X_test.astype('float32') / 255

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

- **Loading the Dataset**: The CIFAR-10 dataset is split into a **training set** (50,000 images) and a **test set** (10,000 images).

- **Normalization**: The pixel values of the images are divided by 255 to normalize them into a range of [0, 1], which is a standard preprocessing step for neural networks to improve training stability.

- **One-hot Encoding**: The labels are one-hot encoded, meaning each class label is converted into a vector of size 10, where the index corresponding to the class is set to 1, and all other indices are 0. This is required for multi-class classification.

---

3. **Initializing CodeCarbon Tracker**

```python
tracker = EmissionsTracker(project_name="CIFAR10_CNN_Carbon_Tracking")
tracker.start()  # Start tracking carbon emissions
```

- **CodeCarbon Tracker**: The `EmissionsTracker()` initializes the carbon tracker for the training session, and `tracker.start()` begins monitoring the energy consumption and calculating CO2 emissions.
- **Project Name**: The `project_name` parameter is used to label the specific experiment being tracked, making it easier to distinguish between multiple runs or different experiments.

---

4. **Defining the CNN Model**

```python
model = Sequential([
    Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(32, 32, 3)),
```

```
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(64, kernel_size=(3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(128, kernel_size=(3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(256, activation='relu'),
    Dense(10, activation='softmax')
])
```

- **Sequential Model**: The CNN model is built using a **Sequential** API, meaning layers are stacked one after another.

- **Convolutional Layers (Conv2D)**:

  - The first layer has 32 filters, the second has 64, and the third has 128. These layers are used to automatically learn spatial features from the images.

  - A `kernel_size=(3, 3)` defines the filter size as 3x3 pixels, and the activation function is ReLU (Rectified Linear Unit), which helps the network learn nonlinear features.

- **MaxPooling Layers**: Each convolutional layer is followed by a max-pooling layer, which reduces the spatial dimensions of the image and helps reduce computational cost while retaining important features.

- **Flatten Layer**: After the convolutional layers, the 3D output is flattened into a 1D vector.

- **Dense Layers**: The fully connected (Dense) layers process the flattened data. The final layer has 10 units (one for each class in CIFAR-10) and uses the **softmax** activation function to produce a probability distribution over the classes.

---

5. **Compiling the Model**

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

- **Optimizer**: The Adam optimizer is used for efficient gradient-based optimization.
- **Loss Function**: **Categorical Crossentropy** is used as the loss function, which is standard for multi-class classification problems.
- **Metrics**: The model will be evaluated based on **accuracy** during training.

---

6. **Training the Model**

```python
start_time = time.time()
history = model.fit(X_train, y_train, epochs=10, batch_size=64, validation_data=(X_test, y_test))
end_time = time.time()
```

- **Model Training**: The model is trained using the **fit()** function. Training runs for 10 epochs with a batch size of 64 (i.e., 64 images are passed through the network before updating the weights).

- **Validation**: The model's performance is evaluated after each epoch using the test data `(X_test, y_test)` to check how well it generalizes to unseen data.

- **Timing**: `time.time()` is used to measure the total time taken for training, allowing us to correlate the carbon emissions with the training duration.

---

7. **Evaluating the Model**

```python
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f"Test accuracy: {test_acc * 100:.2f}%")
```

- After training, the model is evaluated on the test set to check its performance in terms of accuracy and loss. The test accuracy is printed as a percentage.

---

8. **Stopping the CodeCarbon Tracker and Displaying Results**

```python
emissions = tracker.stop()
print(f"Total CO2 emissions: {emissions:.4f} kg")
print(f"Training duration: {end_time - start_time:.2f} seconds")
```

```

- **Stop Tracking**: The `tracker.stop()` method stops the CodeCarbon tracker and returns the total carbon emissions generated during the training in kilograms of CO2.

- The emissions and total training time are printed, providing insights into the environmental cost of running the model.

---

### 9. **Visualizing Model Performance**

```python

plt.plot(history.history['accuracy'], label='train_accuracy')

plt.plot(history.history['val_accuracy'], label='val_accuracy')

plt.title('Training and Validation Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.show()
```

- **Plotting Accuracy**: The accuracy for both the training and validation sets is plotted over the number of epochs. This allows for visualizing how the model's performance evolves over time, providing insights into the learning process and potential overfitting.

```

Summary of Key Points:

1. **CNN Architecture**: The CNN is constructed to classify CIFAR-10 images, with 3 convolutional layers followed by dense layers for classification.

2. **Carbon Tracking**: **CodeCarbon** is used to measure the environmental impact of training the model, expressed as the total CO2 emissions.

3. **Training and Validation**: The model is trained for 10 epochs with accuracy and validation metrics visualized.

4. **Results**: The final accuracy and emissions are reported, giving insights into both the performance and environmental footprint of the ML model.

This code demonstrates how to integrate **carbon emission tracking** into a machine learning workflow, promoting sustainable AI practices.