Image Classification

Created By:M.Yeswanthkumar

A convolutional neural network (CNN) for classifying handwritten digits from the MNIST dataset using TensorFlow and Keras .

1. Import Libraries:

- tensorflow as tf: Imports the TensorFlow library, which is the core machine learning framework.
- tensorflow.keras: Imports the Keras API, which is a high-level API for building neural networks within TensorFlow.
- tensorflow.keras.layers: Imports specific layers needed for building the CNN (convolutional, pooling, dense, dropout).
- matplotlib.pyplot as plt: Imports the Matplotlib library for plotting the training history.

2. Load and Preprocess the MNIST Dataset:

- keras.datasets.mnist.load_data(): Loads the MNIST dataset, which consists of 60,000 training images and 10,000 test images of handwritten digits (0-9).
- x_train, y_train, x_test, y_test: The loaded data is split into training and testing sets, with x_ containing the images and y_ containing the labels.
- x_train / 255.0, x_test / 255.0: Normalizes the pixel values of the images to the range [0, 1] by dividing them by 255. This improves training stability.
- x_train.reshape(-1, 28, 28, 1), x_test.reshape(-1, 28, 28, 1): Reshapes the images to the format expected by a CNN: (number of samples, height, width, channels). In this case, the images are 28x28 pixels with a single channel (grayscale).

3. Build the CNN Model:

- keras.Sequential([...]): Creates a sequential model, where layers are added one after another.
- layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)): Adds a 2D convolutional layer with 32 filters, a 3x3 kernel size, ReLU activation, and specifies the input shape.
- layers.MaxPooling2D((2, 2)): Adds a max pooling layer with a 2x2 pool size to reduce spatial dimensions.
- layers.Conv2D(64, (3, 3), activation='relu'): Adds another convolutional layer with 64 filters.
- layers.MaxPooling2D((2, 2)): Another max pooling layer.
- layers.Conv2D(128, (3, 3), activation='relu'): A third convolutional layer with 128 filters.
- layers.Flatten(): Flattens the output of the convolutional layers into a 1D vector.
- layers.Dense(128, activation='relu'): Adds a fully connected dense layer with 128 neurons and ReLU activation.
- layers.Dropout(0.5): Adds a dropout layer with a 50% dropout rate to prevent overfitting.

- layers.Dense(10, activation='softmax'): Adds the final output layer with 10 neurons (one for each digit) and softmax activation to produce probabilities for each class.

4. Compile the Model:

- model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy']): Compiles the model with the Adam optimizer, sparse categorical cross-entropy loss (suitable for integer labels), and accuracy as the evaluation metric.[1]

5. Train the Model:

- model.fit(x_train, y_train, epochs=5, validation_data=(x_test, y_test)): Trains the model for 5 epochs using the training data and validates it on the test data. The history object stores the training progress.

6. Evaluate the Model:

- model.evaluate(x_test, y_test, verbose=2): Evaluates the trained model on the test data and prints the test loss and accuracy.

7. Plot Training History:

- matplotlib.pyplot: Used to generate two plots:
    - One plot shows the training and validation accuracy over epochs.
    - The other plot shows the training and validation loss over epochs.
- These plots visualize the model's learning progress and help identify potential issues like overfitting.

In summary, this code creates, trains, and evaluates a CNN for handwritten digit recognition, and provides visual feedback on the training process.