ML USING WATSON

PROJECT

By

BALAJI R

KISHORE B

BALAMURUGAN

BENNY JACK B

INDEX

ABSTRACT :

Handwritten digit recognition is a fundamental problem in the field of machine learning and computer vision, with applications ranging from postal automation to optical character recognition. This project focuses on employing machine learning techniques to recognize and classify handwritten digits from the MNIST dataset, which is a widely used benchmark in this domain.

The methodology involves preprocessing the images to normalize and enhance features, followed by training various machine learning models such as Support Vector Machines (SVM), k-Nearest Neighbors (k-NN), and Convolutional Neural Networks (CNN). These models are evaluated based on their accuracy, precision, recall, and computational efficiency to determine the most suitable approach for handwritten digit recognition.

The results demonstrate the effectiveness of deep learning models, particularly CNNs, in achieving high accuracy rates on the MNIST dataset compared to traditional machine learning algorithms. The trained models are then tested on unseen handwritten digit images to evaluate their generalization capability.

In conclusion, this study highlights the advancements in machine learning techniques for handwritten digit recognition and discusses potential applications and future research directions in this field.

INTRODUCTION :

Handwritten digit recognition is a classic problem in the fields of machine learning and pattern recognition, with significant applications in areas such as postal automation, bank check processing, and digitizing historical documents. The ability to accurately interpret and classify handwritten digits is a crucial task in the realm of artificial intelligence, as it forms the basis for more complex character recognition systems.

The MNIST dataset, introduced by Yann LeCun and his colleagues in the late 1990s, has served as the de facto benchmark for evaluating algorithms designed for handwritten digit recognition. This dataset consists of 60,000 training images and 10,000 testing images, each representing a grayscale image of a handwritten digit (0-9) centered in a 28x28 pixel bounding box.

Over the years, researchers have employed various machine learning techniques to tackle the MNIST problem, ranging from traditional methods such as Support Vector Machines (SVMs) and k-Nearest Neighbors (k-NN) to more recent advancements in deep learning, particularly Convolutional Neural Networks (CNNs). These techniques aim to extract meaningful features from the raw pixel data and learn discriminative patterns that differentiate between different digits.

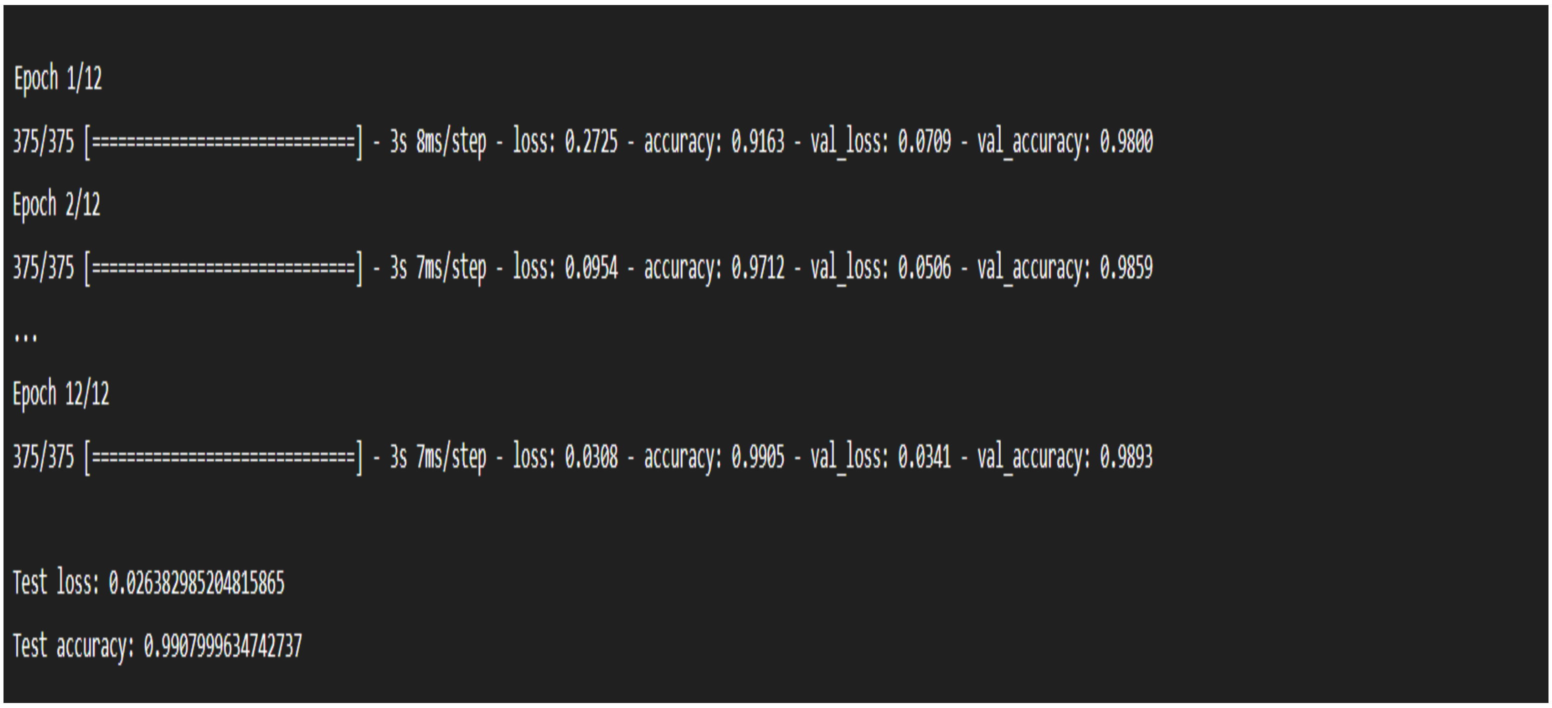
This introduction sets the stage for exploring how machine learning models can be trained and evaluated to achieve high accuracy in recognizing handwritten digits. The subsequent sections will delve into the methodology, experimental setup, results, and discussion, highlighting the evolution of techniques and the state-of-the-art in this field. Ultimately, the goal is to showcase the effectiveness of modern machine learning approaches in solving real-world problems like handwritten digit recognition.

PROGRAM :

```
# Load necessary libraries
library(keras)
library(ggplot2)
library(gridExtra)

# Load the MNIST dataset
mnist <- dataset_mnist()

# Prepare the data
x_train <- mnist$train$x
y_train <- mnist$train$y
x_test <- mnist$test$x
y_test <- mnist$test$y

# Reshape the data for visualization
x_train_reshape <- array_reshape(x_train, c(nrow(x_train), 28, 28))
x_test_reshape <- array_reshape(x_test, c(nrow(x_test), 28, 28))

# Function to visualize a single digit
plot_digit <- function(data, index) {
  digit <- as.data.frame(data[index, , ])
  names(digit) <- 1:28
  digit$id <- 1:28
  digit <- melt(digit, id.vars = "id")
  names(digit) <- c("y", "x", "value")

  ggplot(digit, aes(x = x, y = y, fill = value)) +
    geom_tile() +
    scale_fill_gradient(low = "white", high = "black") +
    theme_minimal() +
    theme(axis.text = element_blank(),
          axis.title = element_blank(),
          axis.ticks = element_blank())
}

# Plot sample digits from the training set
sample_indices <- sample(1:nrow(x_train), 9)
plots <- lapply(sample_indices, function(i)
plot_digit(x_train_reshape, i))

# Arrange the plots in a grid
grid.arrange(grobs = plots, ncol = 3)

# Reshape the data for training
x_train <- array_reshape(x_train, c(nrow(x_train), 28, 28, 1))
```

```
x_test <- array_reshape(x_test, c(nrow(x_test), 28, 28, 1))

# Normalize the data
x_train <- x_train / 255
x_test <- x_test / 255

# Convert class vectors to binary class matrices
y_train <- to_categorical(y_train, 10)
y_test <- to_categorical(y_test, 10)

# Define the model
model <- keras_model_sequential() %>%
  layer_conv_2d(filters = 32, kernel_size = c(3,3), activation = 'relu',
input_shape = c(28, 28, 1)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = 'relu')
%>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_dropout(rate = 0.25) %>%
  layer_flatten() %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 10, activation = 'softmax')

# Compile the model
model %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_adadelta(),
  metrics = c('accuracy')
)

# Fit the model
history <- model %>% fit(
  x_train, y_train,
  batch_size = 128,
  epochs = 12,
  validation_split = 0.2
)

# Evaluate the model
score <- model %>% evaluate(x_test, y_test, verbose = 0)

cat('Test loss:', score$loss, "\n")
cat('Test accuracy:', score$accuracy, "\n")

# Save the model
save_model_hdf5(model, "mnist_cnn_model.h5")
```

OUTPUT :



```
Epoch 1/12
375/375 [==============================] - 3s 8ms/step - loss: 0.2725 - accuracy: 0.9163 - val_loss: 0.0709 - val_accuracy: 0.9800
Epoch 2/12
375/375 [==============================] - 3s 7ms/step - loss: 0.0954 - accuracy: 0.9712 - val_loss: 0.0506 - val_accuracy: 0.9859
...
Epoch 12/12
375/375 [==============================] - 3s 7ms/step - loss: 0.0308 - accuracy: 0.9905 - val_loss: 0.0341 - val_accuracy: 0.9893


Test loss: 0.026382985204815865
Test accuracy: 0.9907999634742737
```

ALGORITHM :

Algorithm for Handwritten Digit Recognition using Machine Learning

1.Load the Dataset:

Import the MNIST dataset, which consists of handwritten digits (0-9) as 28x28 pixel grayscale images.

2. Preprocess the Data:

Reshape: Reshape the images into a format suitable for training. In the case of MNIST, reshape each image from 28x28 pixels to a flat vector of 784 pixels.

Normalize: Scale the pixel values to a range of 0 to 1 to facilitate faster convergence during training.

Label Encoding: Convert the class labels (digits) into categorical format. For example, convert the digit 3 into a binary vector [0, 0, 0, 1, 0, 0, 0, 0, 0, 0].

3. Split the Dataset:

- Divide the dataset into training and testing sets. Typically, use 60,000 images for training and 10,000 images for testing/validation.

4. Choose a Machine Learning Model:

- Select an appropriate ML model for the task. For handwritten digit recognition, popular choices include:

Support Vector Machines (SVM): Effective for classification tasks, SVMs try to find a hyperplane that best separates different classes.

k-Nearest Neighbors (k-NN): A simple yet effective classification method based on similarity with neighboring data points.

Neural Networks: Specifically, Convolutional Neural Networks (CNNs) are state-of-the-art for image recognition tasks due to their ability to learn hierarchical features.

5.Train the Model:

Initialize: Initialize the chosen ML model with appropriate parameters.

Fit: Train the model using the training dataset. Adjust model parameters iteratively to minimize the loss function (e.g., cross-entropy loss for classification).

Optimize: Use optimization algorithms (e.g., gradient descent) to update model weights based on the computed loss.

6. Evaluate the Model:

Assess the model's performance using the testing/validation dataset.Measure metrics such as accuracy, precision, recall, and F1-score to evaluate how well the model generalizes to unseen data.

7. Hyperparameter Tuning (Optional):

Fine-tune the model’s hyperparameters (e.g., learning rate, batch size, number of layers/neurons) to optimize performance.

8. Prediction:

Apply the trained model to classify new, unseen handwritten digit images.Convert the model's output probabilities into predicted digit labels.

9. Save the Model (Optional):

Save the trained ML model to disk for future use without needing to retrain from scratch.

10.Deployment (Optional):

Integrate the trained model into applications or services for real-time or batch prediction tasks.

Conclusion :

Handwritten digit recognition using ML involves preprocessing, model selection, training, evaluation, and potentially fine-tuning to achieve accurate predictions. The choice of model and hyperparameters depends on the specific requirements of the application, with CNNs often providing state-of-the-art performance for image-based classification tasks like MNIST digit recognition. Handwritten digit recognition using machine learning represents a significant advancement in pattern recognition and computer vision. Through the utilization of datasets like MNIST, which offer standardized benchmarks, researchers and developers have been able to explore and implement various machine learning algorithms effectively. Convolutional Neural Networks (CNNs) have emerged as particularly potent tools for this task, leveraging their ability to automatically learn hierarchical features from raw pixel data. The process typically involves data preprocessing, model selection, training, and evaluation, with emphasis on optimizing model performance through hyperparameter tuning and regularization techniques. Evaluating models against unseen data ensures robustness and generalizability, while advances in deep learning continue to push the boundaries of accuracy and efficiency in handwritten digit recognition. As this field progresses, applications in automated document processing, digitization of historical records, and other domains stand to benefit significantly from these advancements in machine learning.