|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | | **Hope Foundation’s**  **Finolex Academy of Management and Technology, Ratnagiri** | | | | | |
| **Department of Computer Science and Engineering (AIML)** | | | | | |
| Subject name: Machine Learning | | | | | | Subject Code: CSL604 | |
| Class | | TE CSE | | | Semester –VI (CBCGS) | Academic year: 2024-25 | |
| Name of Student | |  | | | | **QUIZ Score :6** | |
| Roll No | |  | | Experiment No. | | 10 | |
| Title: **To perform classification on MNIST dataset.** | | | | | | | |
|  | | | | | | | |
| **1. Lab objectives applicable:**  **LOB1:**To introduce platforms such as Anaconda, COLAB suitable to Machine Learning.  **LOB3:** To develop Neural Network based learning models. | | | | | | | |
| **2. Lab outcomes applicable:**  **LO2 :** Apply suitable Machine learning models for a given problem | | | | | | | |
| **3. Learning Objectives:**   1. To develop machine learning model performing classification. | | | | | | | |
| **4. Practical applications of the assignment/experiment:**  To demonstrate the classification using the real-world data. | | | | | | | |
| **5. Prerequisites**:   1. Python language | | | | | | | |
| **6. Minimum Hardware Requirements**:-  I series processor, RAM 4GB,  **7. Software Requirements:-**  Colab or Visual Studio or Jupyter notebook (Anaconda) | | | | | | | |
| **8. Quiz Questions :** [**https://docs.google.com/forms/d/e/1FAIpQLSeA-Y\_smf8YomaKH2yFdt\_HS3n-CVujdVO8XijXtera-epBZw/viewform?usp=dialog**](https://docs.google.com/forms/d/e/1FAIpQLSeA-Y_smf8YomaKH2yFdt_HS3n-CVujdVO8XijXtera-epBZw/viewform?usp=dialog) | | | | | | | |
| **9. Experiment Evaluation:** | | | | | | | |
| **Sr. No.** | **Parameters** | | | | | **Marks obtained** | **Out of** |
| **1** | Technical Understanding (Assessment may be done based on Q & A **or** any other relevant method.) Teacher should mention the other method used - | | | | | 6 | 6 |
| **2** | Lab Performance | | | | |  | 2 |
| **3** | Punctuality | | | | |  | 2 |
| **Date of performance (DOP)** | | |  | | **Total marks obtained** |  | **10** |

**Signature of Faculty**

**10. Theory:**

The MNIST (Modified National Institute of Standards and Technology) dataset is a widely used benchmark dataset in machine learning, specifically for image classification tasks. It consists of 60,000 training images and 10,000 test images of handwritten digits (0-9), each represented as a 28×28 grayscale image.

The goal of classification on the MNIST dataset is to train a machine learning model to recognize and correctly classify handwritten digits.

**Methodology:**

1. **Data Preprocessing:**
   * Each image is converted into a numerical matrix of pixel values ranging from 0 to 255.
   * Normalization is applied to scale pixel values between 0 and 1 for better model performance.
2. **Model Selection:**
   * Traditional machine learning models like Support Vector Machines (SVM) and k-Nearest Neighbors (k-NN) can be used for classification.
   * Deep learning models, particularly Convolutional Neural Networks (CNNs), are highly effective for MNIST classification due to their ability to capture spatial features in images.
3. **Training and Evaluation:**
   * The dataset is split into training and testing sets.
   * The model learns patterns in handwritten digits through optimization techniques like stochastic gradient descent (SGD) and backpropagation.
   * Accuracy is measured using metrics like confusion matrix, precision, recall, and F1-score.

**Important Phases in MNSIT Classification:**

**1. Model Building**

**Steps:**

1. Initialize a Sequential model.
2. Add convolutional layers to extract spatial features from the input images.
3. Add pooling layers to reduce dimensionality while retaining important features.
4. Add dropout layers to prevent overfitting.
5. Flatten the feature map to pass it into fully connected layers.
6. Add dense layers for classification, with softmax activation for the final layer.

**Code Snippet:**

model = Sequential() # Initialize the model

model.add(Conv2D(filters=32, kernel\_size=(3,3), activation='relu', input\_shape=(28,28,1)))

model.add(Conv2D(filters=64, kernel\_size=(3,3), activation='relu'))

model.add(MaxPooling2D(pool\_size=(2,2)))

model.add(Dropout(rate=0.25))

model.add(Flatten())

model.add(Dense(units=256, activation='relu'))

model.add(Dropout(rate=0.5))

model.add(Dense(units=10, activation='softmax')) # Output layer with 10 classes

### **Functions and Parameters:**

| **Function** | **Parameters** | **Description** |
| --- | --- | --- |
| Sequential() | - | Initializes a linear stack of layers. |
| Conv2D() | filters, kernel\_size, activation, input\_shape | Adds a convolutional layer to extract features. |
| MaxPooling2D() | pool\_size | Downsamples feature maps to reduce dimensionality. |
| Dropout() | rate | Randomly drops neurons to prevent overfitting. |
| Flatten() | - | Converts multi-dimensional data into a single vector. |
| Dense() | units, activation | Creates a fully connected layer, with softmax for classification. |

**2. Model Compilation**

**Steps:**

1. Define the loss function to measure the error between predicted and actual labels.
2. Choose an optimizer to adjust model weights for better accuracy.
3. Define evaluation metrics to track model performance.

**Code Snippet:**

model.compile(loss=keras.losses.categorical\_crossentropy,

optimizer=keras.optimizers.Adadelta(),

metrics=['accuracy'])

**Functions and Parameters:**

| **Function** | **Parameters** | **Description** |
| --- | --- | --- |
| compile() | loss, optimizer, metrics | Configures the model for training. |
| loss | 'categorical\_crossentropy' | Loss function for multi-class classification. |
| optimizer | Adadelta() | Optimizer to adjust model parameters. |
| metrics | ['accuracy'] | Evaluation metric for model performance. |

**3. Model Training (Fitting the Model)**

**Steps:**

1. Provide training data (features and labels).
2. Set batch size to determine the number of samples processed before model updates.
3. Set the number of epochs to define how many times the model will see the entire dataset.
4. Optionally provide validation data to check performance on unseen data.

**Code Snippet:**

model.fit(X\_train, y\_train, batch\_size=128, epochs=10, verbose=1, validation\_data=(X\_test, y\_test))

**Functions and Parameters:**

| **Function** | **Parameters** | **Description** |
| --- | --- | --- |
| fit() | X\_train, y\_train, batch\_size, epochs, verbose, validation\_data | Trains the model using the given dataset. |
| X\_train, y\_train | - | Input data and corresponding labels. |
| batch\_size | 128 | Number of training samples processed before updating weights. |
| epochs | 10 | Number of times the dataset is passed through the model. |
| verbose | 1 | Display training progress (1 for detailed, 0 for silent). |
| validation\_data | (X\_test, y\_test) | Data used to validate the model after each epoch. |

|  |
| --- |
|  |

**11. Installation Steps / Performance Steps and Results –**

import tensorflow as tf

import keras

from keras.models import Sequential

from keras.layers import Dense , Dropout , Flatten

from keras.layers import Conv2D , MaxPooling2D

from keras import backend as k

from keras.models import load\_model

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.metrics import confusion\_matrix

from keras.datasets import mnist

(X\_train , y\_train) , (X\_test , y\_test)=mnist.load\_data()

# Shape of the dataset

print(X\_train.shape , y\_train.shape)

# Reshape of the dataset

X\_train=X\_train.reshape(X\_train.shape[0],28,28,1)

X\_test=X\_test.reshape(X\_test.shape[0],28,28,1)

input\_shape=(28,28,1)

#Convert class vectors to binary class met

num\_classes=10

y\_train=keras.utils.to\_categorical(y\_train, num\_classes)

y\_test=keras.utils.to\_categorical(y\_test,num\_classes)

X\_train = X\_train.astype('float32')

X\_test = X\_test.astype('float32')

# Normalizing the data

X\_train=X\_train/255

X\_test=X\_test/255

print('X\_train.shape', X\_test.shape)

print(X\_train.shape[0],'Train Sample')

print(X\_test.shape[0],'Test Sample')

# Create The Model

model=Sequential()

model.add(Conv2D(32, kernel\_size=(3,3), activation='relu', input\_shape=input\_shape))

model.add(Conv2D(64,(3,3),activation='relu'))

model.add(MaxPooling2D(pool\_size=(2,2)))

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(256, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(num\_classes, activation='softmax'))

# Compile model

model.compile(loss=keras.losses.categorical\_crossentropy,optimizer=keras.optimizers.Adadelta(),metrics=['accuracy'])

# Model training

model.fit(X\_train,y\_train , batch\_size=128,epochs=10,verbose=1,validation\_data=(X\_test,y\_test))

print("The Model has Successfully Trained")

# Model evaluating

score=model.evaluate(X\_test,y\_test ,verbose=2)

print('test loss:',score[0])

print('test accuracy:',score[1])

# Model Saving

model.save('mnist.h5')

print('Saving the model as mnist.h5')

# Prediction

new\_model=load\_model('mnist.h5')

prediction=new\_model.predict(X\_test)

print(prediction)

#predict on the first five images

pred=model.predict(X\_test[:5])

# print our model prediction

print(np.argmax(pred ,axis=1))

print(y\_test[:5])

for i in range (0,5):

  first\_img=X\_test[i]

  first\_img=np.array(first\_img,dtype='float')

  pixels=first\_img.reshape((28,28))

  plt.imshow(pixels, cmap='gray')

  plt.show()

# Generate Predictions

y\_pred = model.predict(X\_test)

y\_pred\_classes = np.argmax(y\_pred, axis=1)  # Convert softmax outputs to class labels

y\_true = np.argmax(y\_test, axis=1)  # Convert one-hot encoded labels back to class labels

# Compute Confusion Matrix

cm = confusion\_matrix(y\_true, y\_pred\_classes)

# Plot Confusion Matrix with Colors

plt.figure(figsize=(8, 6))

sns.heatmap(cm, annot=True, fmt='d', cmap="coolwarm", linewidths=0.5, linecolor='black', square=True)

# Add Labels and Title

plt.xlabel('Predicted Label', fontsize=12, fontweight='bold', color='darkred')

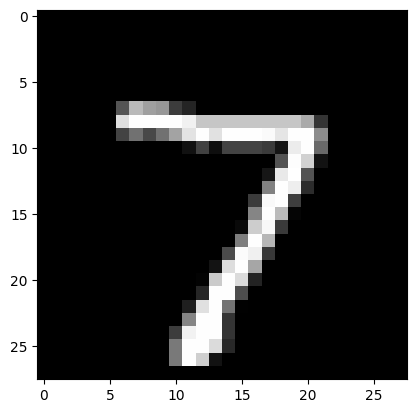
plt.ylabel('True Label', fontsize=12, fontweight='bold', color='darkblue')

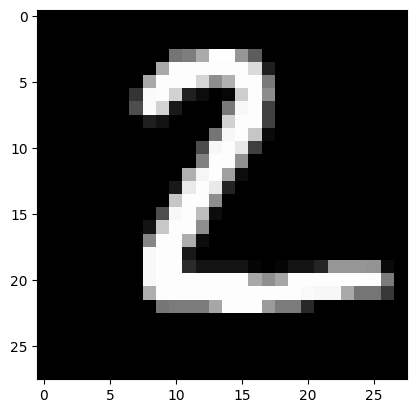
plt.title('Confusion Matrix - MNIST', fontsize=14, fontweight='bold', color='purple')

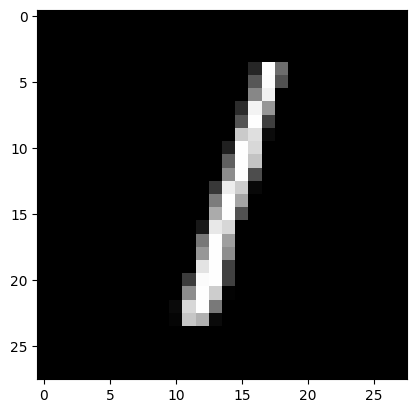
# Show the plot

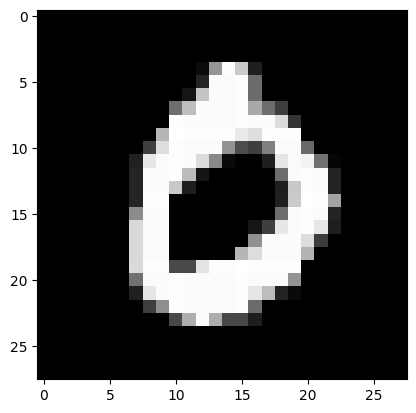
plt.show()

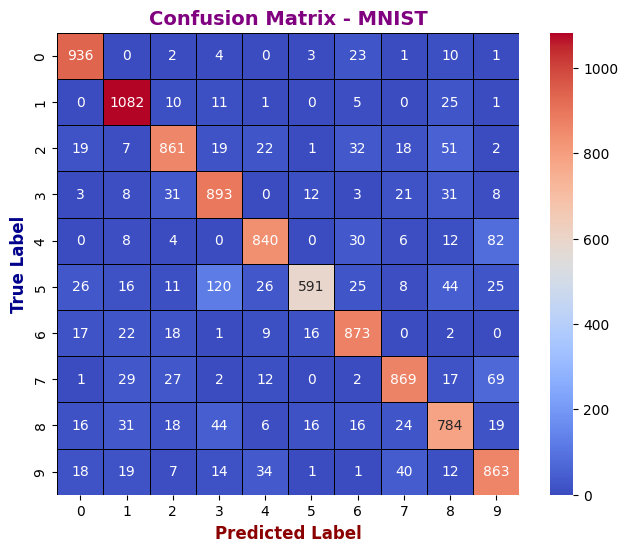
**Output:**











**12. Learning Outcomes Achieved**

1. Students are able to perform implement McCulloch Pitts model.

**13. Conclusion:**

**1. Applications of the Studied Technique in Industry**

The MNIST dataset serves as a benchmark for handwritten digit recognition, which has various real-world applications. Industries use similar models in **automated check processing, postal mail sorting, and signature verification**. Deep learning models trained on such datasets are also applied in **banking, security, and digital document processing**. The experiment provides insights into how AI can enhance accuracy and efficiency in automated recognition tasks.

**2. Engineering Relevance**

Handwritten digit recognition is a fundamental problem in **image processing and pattern recognition**, making it highly relevant to engineering applications. The experiment demonstrates how **convolutional neural networks (CNNs)** and machine learning models can extract meaningful patterns from images. Understanding these techniques is crucial for developing **advanced AI systems in robotics, automation, and human-computer interaction**. Engineers can apply similar methods to various fields, including **medical imaging and industrial automation**.

**3. Skills Developed**

Through this experiment, valuable skills in **deep learning model development, data preprocessing, and performance evaluation** were gained. Hands-on experience with **TensorFlow, Keras, and NumPy** improved proficiency in implementing and fine-tuning machine learning models. The analysis of metrics such as **accuracy, precision, recall, and F1-score** strengthened the ability to assess model performance. Additionally, debugging and optimizing neural networks provided practical problem-solving skills essential for AI development.

**14. References**:

1. Nathalie Japkowicz & Mohak Shah, ―Evaluating Learning Algorithms: A Classification Perspective‖, Cambridge.
2. Marc Peter Deisenroth, Aldo Faisal, Cheng Soon Ong, ―Mathematics for machine learning‖
3. Samir Roy and Chakraborty, ―Introduction to soft computing‖, Pearson Edition.
4. Ethem Alpaydın, ―Introduction to Machine Learning‖, MIT Press McGraw-Hill Higher

Education

1. Peter Flach, ―Machine Learning‖, Cambridge University Press
2. Tom M. Mitchell, ―Machine Learning‖, McGraw Hill
3. Kevin P. Murphy, ―Machine Learning ― A Probabilistic Perspective‖, MIT Press