Ex No: 5	Apply L2 Regularisation and Dropout Regularisation to improve accuracy of Model

AIM:

To implement L2 regularization and Dropout regularization techniques to enhance model accuracy.

PROCEDURE:

1. Import the required libraries and packages.
2. Load the MNIST digit image dataset from Keras.
3. Upgrade Keras and install np_utils if necessary.
4. Build the neural network model.
5. Flatten the images to convert them into one-dimensional arrays.
6. Split the data into training and testing sets.
7. Normalize the pixel values of the images to facilitate training.
8. Apply one-hot encoding to the class labels using Keras's utilities.
9. Build a linear stack of layers with the sequential model.
10. Implement L2 Regularization in the model to prevent overfitting.
11. Train the model on the training data and evaluate its performance on the test data.
12. Make predictions for sample images to assess the model's real-time performance.
13. Introduce Dropout Regularization to further enhance the model's generalization capability.
14. Repeat steps 10 and 11 with the Dropout Regularization applied, and compare the accuracy and loss scores between the two models.

CODE:

```
from keras.datasets import mnist

# loading the dataset

(X_train, y_train), (X_test, y_test) = mnist.load_data()

# let's print the shape of the dataset

print("X_train shape", X_train.shape)

print("y_train shape", y_train.shape)

print("X_test shape", X_test.shape)

print("y_test shape", y_test.shape)

pip install np_utils

!pip install --upgrade keras

# keras imports for the dataset and building our neural network

from keras.datasets import mnist

from keras.models import Sequential

from keras.layers import Dense, Dropout, Conv2D, MaxPool2D

from keras.utils import to_categorical

# Flattening the images from the 28x28 pixels to 1D 787 pixels

X_train = X_train.reshape(60000, 784)
```

```
X_test = X_test.reshape(10000, 784)
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
# normalizing the data to help with the training
X_train /= 255
X_test /= 255
# one-hot encoding using keras' numpy-related utilities
n_classes = 10
print("Shape before one-hot encoding: ", y_train.shape)
Y_train = to_categorical(y_train, n_classes)
Y_test = to_categorical(y_test, n_classes)
print("Shape after one-hot encoding: ", Y_train.shape)
# import keras modules
from keras import regularizers
from keras.models import Sequential
from keras.layers import Dense

# define vars
input_num_units = 784
hidden1_num_units = 500
hidden2_num_units = 500
hidden3_num_units = 500
hidden4_num_units = 500
hidden5_num_units = 500
output_num_units = 10

epochs = 10
batch_size = 128

model = Sequential([
  Dense(units=hidden1_num_units, input_dim=input_num_units, activation='relu',
      kernel_regularizer=regularizers.l2(0.0001)),
  Dense(units=hidden2_num_units, activation='relu',
      kernel_regularizer=regularizers.l2(0.0001)),
```

```
    Dense(units=hidden3_num_units, activation='relu',
          kernel_regularizer=regularizers.l2(0.0001)),
    Dense(units=hidden4_num_units, activation='relu',
          kernel_regularizer=regularizers.l2(0.0001)),
    Dense(units=hidden5_num_units, activation='relu',
          kernel_regularizer=regularizers.l2(0.0001)),
    Dense(units=output_num_units, activation='softmax'),
])

# looking at the model summary
model.summary()
# compiling the sequential model
model.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer='adam')
# training the model for 10 epochs
model.fit(X_train, Y_train, batch_size=128, epochs=10, validation_data=(X_test, Y_test))
model.save('final_model.keras')
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.models import load_model
import numpy as np

# Load and prepare the image
def load_image(sample_image):
    # Load the image
    img = load_img(sample_image, color_mode='grayscale', target_size=(28, 28))
    print("Loaded image shape:", img.size)
    # Convert to array
    img = img_to_array(img)
    print("Image array shape after conversion:", img.shape)
    # Flatten the image array
    img = img.reshape((1, 784))
    # Prepare pixel data
    img = img.astype('float32')
    img = img / 255.0
```

```
    return img

# Load an image and predict the class
def run_example(model):  # Pass the model object as an argument
    # Load the image
    img = load_image('sample_image.jpg')
    predict_value = model.predict(img)
    digit = np.argmax(predict_value)
    print("Predicted digit:", digit)

# Load the model
model = load_model('final_model.keras')
# Entry point, run the example
run_example(model)  # Pass the model object to the run_example function

from keras.layers import Dropout
model = Sequential([
    Dense(units=hidden1_num_units, input_dim=input_num_units, activation='relu',
          kernel_regularizer=regularizers.l2(0.0001)),
    Dropout(0.5), # Dropout layer with 50% dropout rate after the first hidden layer
    Dense(units=hidden2_num_units, activation='relu',
          kernel_regularizer=regularizers.l2(0.0001)),
    Dropout(0.5), # Dropout layer with 50% dropout rate after the second hidden layer
    Dense(units=hidden3_num_units, activation='relu',
          kernel_regularizer=regularizers.l2(0.0001)),
    Dropout(0.5), # Dropout layer with 50% dropout rate after the third hidden layer
    Dense(units=hidden4_num_units, activation='relu',
          kernel_regularizer=regularizers.l2(0.0001)),
    Dropout(0.5), # Dropout layer with 50% dropout rate after the fourth hidden layer
    Dense(units=hidden5_num_units, activation='relu',
          kernel_regularizer=regularizers.l2(0.0001)),
    Dropout(0.5), # Dropout layer with 50% dropout rate after the fifth hidden layer
    Dense(units=output_num_units, activation='softmax')
])
```

```
# looking at the model summary
model.summary()
# compiling the sequential model
model.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer='adam')
# training the model for 10 epochs
model.fit(X_train, Y_train, batch_size=128, epochs=10, validation_data=(X_test, Y_test))
model.save('final_model.keras')
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.models import load_model
import numpy as np
# Load and prepare the image
def load_image(sample_image):
    # Load the image
    img = load_img(sample_image, color_mode='grayscale', target_size=(28, 28))
    print("Loaded image shape:", img.size)
    # Convert to array
    img = img_to_array(img)
    print("Image array shape after conversion:", img.shape)
    # Flatten the image array
    img = img.reshape((1, 784))
    # Prepare pixel data
    img = img.astype('float32')
    img = img / 255.0
    return img
# Load an image and predict the class
def run_example(model):  # Pass the model object as an argument
    # Load the image
    img = load_image('sample_image.jpg')
    predict_value = model.predict(img)
    digit = np.argmax(predict_value)
    print("Predicted digit:", digit)
# Load the model
```

model = load_model('final_model.keras')

Entry point, run the example

run_example(model) # Pass the model object to the run_example function

OUTPUT:

Predicted Output: 7

RESULT:

L2 Regularization:

No of epochs	Training Accuracy	Testing Accuracy	Training Loss	Testing Loss
1	0.8619	0.9610	0.6473	0.2935
2	0.9721	0.9719	0.2550	0.2301
3	0.9772	0.9753	0.2110	0.2044
4	0.9834	0.9752	0.1692	0.1907
5	0.9829	0.9764	0.1587	0.1773
6	0.9865	0.9733	0.1391	0.1894
7	0.9871	0.9776	0.1273	0.1594
8	0.9892	0.9786	0.1149	0.1587
9	0.9895	0.9765	0.1089	0.1569
10	0.9906	0.9802	0.1030	0.1390

Dropout Regularization:

No of epochs	Training Accuracy	Testing Accuracy	Training Loss	Testing Loss
1	0.6039	0.9542	1.3568	0.3864
2	0.9302	0.9649	0.4797	0.3284
3	0.9468	0.9693	0.4075	0.3001
4	0.9526	0.9685	0.3617	0.2863
5	0.9584	0.9746	0.3320	0.2650
6	0.9624	0.9742	0.3065	0.2549
7	0.9616	0.9775	0.3040	0.2427
8	0.9643	0.9782	0.2920	0.2377
9	0.9649	0.9765	0.2865	0.2362
10	0.9659	0.9770	0.2819	0.2401

CONCLUSION:

L2 regularization and Dropout regularization techniques has been implemented to the dataset to enhance the model accuracy.