Course-B. Tech	**Type-** General Elective
Course Code- CSET-335	**Course Name-** Deep Leaning
Year- 2024	**Semester-** Even
Date- 19/02/2024	**Batch-** 2023-2024

CO-Mapping

	CO1	**CO2**	**CO3**
Q1-Q3	√		

Objectives

CO1: To explain the fundamentals of deep learning, Convolution neural network.
CO2: To articulate different problem of classification, detection, segmentation, generation and understand existing solutions/ deep learning architectures.
CO3: To implement a solution for the given problem and improve it using various methods transfer learning, hyperparameter optimization.

Assignment-4

Goal: Build and train a Convolutional Neural Network (CNN) to classify handwritten digits from the MNIST dataset.

1. **Set Up Your Environment:**
   Create a Python environment with the required libraries, including TensorFlow and Matplotlib. Set up a Jupyter Notebook or script for your implementation.

```
# Install required libraries
!pip install tensorflow matplotlib
```

2. **Load and Preprocess the MNIST Dataset:**
   Load the MNIST dataset and preprocess the data.

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical

# Load MNIST data
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

# Preprocess the data
train_images = train_images.reshape((60000, 28, 28, 1)).astype('float32') / 255
test_images = test_images.reshape((10000, 28, 28, 1)).astype('float32') / 255

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

3. **Define and Implement the CNN Model:**
   Design a CNN architecture for MNIST classification.

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))

# Add fully connected layers
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

# Display the model summary
model.summary()
```

4. **Compile and Train the Model:**
   Compile the model with an appropriate loss function, optimizer, and metrics.
   Train the CNN model on the training set.

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

history = model.fit(train_images, train_labels, epochs=10, batch_size=64, validation_split=0.2)
```

5. **Evaluate the Model:**
   Evaluate the trained model on the test set and report the classification accuracy.

```
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f'Test accuracy: {test_acc}')
```

6. **Visualizations:**
   Visualize the training and validation loss and accuracy.

```
import matplotlib.pyplot as plt

# Plot training and validation accuracy
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

Q1. Replicate the above code and observe the accuracy.

Q2. Experiment with different hyperparameters such as the number of filters, kernel size, or activation functions and observe the impact on the performance of the model.

Q3. Print the model summary for each of the combinations used in Q2.