3. Applying the Convolution Neural Network on computer vision problems.

Step 1: Import Required Libraries

Start by importing TensorFlow and necessary modules.

Program:

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

Step 2: Load and Preprocess the Data

Load the MNIST dataset and normalize the pixel values to be between 0 and 1.

Program:

```
# Load the dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Reshape the data to add a single channel (since images are grayscale)
x_train = x_train.reshape((x_train.shape[0], 28, 28, 1))
x_test = x_test.reshape((x_test.shape[0], 28, 28, 1))

# Normalize the pixel values to be between 0 and 1
x_train, x_test = x_train / 255.0, x_test / 255.0
```

Step 3: Define a Simple CNN Model

Create a simple CNN with one convolutional layer followed by max pooling, and a dense layer for classification.

Program:

```
# Define a basic CNN model
model = models.Sequential([
```

```
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),  # 32 filters, 3x3 kernel
    layers.MaxPooling2D((2, 2)),  # Pooling layer with 2x2 pool size
    layers.Flatten(),  # Flatten the output from the convolutional layer
    layers.Dense(64, activation='relu'),  # Fully connected layer with 64 neurons
    layers.Dense(10, activation='softmax')  # Output layer for 10 classes
])
```

Step 4: Compile the Model

Compile the model using the Adam optimizer, a suitable loss function, and accuracy as the metric.

Program:

```
# Compile the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

Step 5: Train the Model

Train the CNN model on the training dataset for 5 epochs.

Program:

```
# Train the model
model.fit(x_train, y_train, epochs=5, batch_size=64)
```

Step 6: Evaluate the Model

After training, evaluate the model's performance on the test dataset.

Program:

```
# Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_acc}')
```

4. Image classification on MNIST dataset (CNN model with a fully connected layer).

Step 1: Import Required Libraries

Start by importing TensorFlow and necessary modules.

Program:

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

Step 2: Load and Preprocess the Data

Load the MNIST dataset and normalize the pixel values to be between 0 and 1.

Program:

```
# Load the dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
# Reshape the data to add a single channel (since images are grayscale)
x_train = x_train.reshape((x_train.shape[0], 28, 28, 1))
x_test = x_test.reshape((x_test.shape[0], 28, 28, 1))
# Normalize the pixel values to be between 0 and 1
x_train, x_test = x_train / 255.0, x_test / 255.0
```

Step 3: Define a Simple CNN Model

Create a simple CNN with one convolutional layer followed by max pooling, and a dense layer for classification.

Program:

```
# Define a basic CNN model
model = models.Sequential([
# First convolutional layer layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
layers.MaxPooling2D((2, 2)), # Second convolutional layer
layers.Conv2D(64, (3, 3), activation='relu'),
```

```
layers.MaxPooling2D((2, 2)), # Third convolutional layer
 layers.Conv2D(64, (3, 3), activation='relu'), layers.Flatten(),
layers.Dense(64, activation='relu'),
layers.Dense(10, activation='softmax')
 ])
```

Step 4: Compile the Model

Compile the model using the Adam optimizer, a suitable loss function, and accuracy as the metric.

Program:

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

Step 5: Train the Model

Train the CNN model on the training dataset for 5 epochs.

Program:

```
model.fit(x_train, y_train, epochs=5, batch_size=64)
```

Step 6: Evaluate the Model

After training, evaluate the model's performance on the test dataset.

Program:

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

Step 7: Make Predictions

You can also use the model to predict the digit class of a test image.

Program:

```
predictions = model.predict(x_test)
predicted_label = tf.argmax(predictions[0])
print(f'Predicted label: {predicted_label.numpy()}')
```

5. Applying the Deep Learning Models in the field of Natural Language Processing.

Step 1: Import Required Libraries

We will use TensorFlow/Keras for building the model, and the IMDb dataset provided by Keras for the text data.

Program:

```
import tensorflow as tf

from tensorflow.keras import layers, models

from tensorflow.keras.datasets import imdb

from tensorflow.keras.preprocessing import sequence
```

Step 2: Load and Preprocess the IMDb Dataset

Program:

```
# Load the IMDb dataset (only top 10,000 most frequent words)

num_words = 10000

(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=num_words)

# Set the maximum length of each review to 500 words (truncating or padding shorter/longer reviews)

max_len = 500

x_train = sequence.pad_sequences(x_train, maxlen=max_len)

x_test = sequence.pad_sequences(x_test, maxlen=max_len)
```

Step 3: Build a Deep Learning Model for Text Classification

Program:

```
# Define the model

model = models.Sequential([

    # Embedding layer: Converts words into dense vectors

    layers.Embedding(input_dim=num_words, output_dim=64, input_length=max_len),


    # LSTM layer: Captures sequential patterns
```

```
    layers.LSTM(64),
    # Dense layer with sigmoid activation for binary classification
    layers.Dense(1, activation='sigmoid')
])
# Compile the model
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
# Display the model architecture
model.summary()
```

Step 4: Train the Model

Program:

```
# Train the model
model.fit(x_train, y_train, epochs=5, batch_size=64, validation_split=0.2)
```

Step 5: Evaluate the Model

Program:

```
# Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_acc}')
```

Step 6: Make Predictions on New Text

Program:

```
# Example review (preprocessed as integer sequences)
new_review = [1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 4581, 66, 394, 2,
530, 973]
new_review = sequence.pad_sequences([new_review], maxlen=max_len)
# Predict sentiment (1: Positive, 0: Negative)
prediction = model.predict(new_review)
print(f'Predicted sentiment: {"Positive" if prediction >= 0.5 else "Negative"}')
```

6. Train a sentiment analysis model on IMDB dataset, use RNN layers with LSTM/GRU notes.

Step 1: Import Required Libraries

Start by importing TensorFlow, Keras, and necessary modules**.**

Program:

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing import sequence
```

Step 2: Load and Preprocess the Data

Load the IMDb dataset and preprocess the reviews by limiting the vocabulary and padding the sequences.

Program:

```
# Load the IMDb dataset with a limit of 10,000 words
num_words = 10000
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=num_words)
# Set the maximum length for each review
max_len = 500
x_train = sequence.pad_sequences(x_train, maxlen=max_len)
x_test = sequence.pad_sequences(x_test, maxlen=max_len)
```

Step 3: Define the Model Architecture Using LSTM

Program:

```
# Define the model using LSTM
model = models.Sequential([
    layers.Embedding(input_dim=num_words, output_dim=64, input_length=max_len),
    layers.LSTM(64),  # LSTM layer with 64 units
    layers.Dense(1, activation='sigmoid')  # Output layer for binary classification
])
```

```
# Compile the model
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
# Display the model architecture
model.summary()
```

Step 4: Train the Model

Program:

```
# Train the model
model.fit(x_train, y_train, epochs=5, batch_size=64, validation_split=0.2)
```

Step 5: Evaluate the Model on the Test Set

Program:

```
# Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_acc}')
```

Step 6: Example Prediction

Program:

```
# Example review (preprocessed as integer sequences)
new_review = [1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 4581, 66, 394, 2,
530, 973]
new_review = sequence.pad_sequences([new_review], maxlen=max_len)
# Predict sentiment (1: Positive, 0: Negative)
prediction = model.predict(new_review)
print(f'Predicted sentiment: {"Positive" if prediction >= 0.5 else "Negative"}')
generate_images(generator, 10)
```