Practical 7: Convolutional Neural Network

Implementation:

```
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
import numpy as np

batch_size = 128
num_classes = 10
epochs = 12

# input image dimensions
img_rows, img_cols = 28, 28

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train.reshape(60000, 28, 28, 1)
x_test = x_test.reshape(10000, 28, 28, 1)

print("x_train shape:", x_train.shape)
print(x_train.shape[0], "train samples")
print(x_test.shape[0], "test samples")

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation="relu", input_shape=(28, 28, 1)))
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(128, activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation="softmax"))

model.compile(
    loss=keras.losses.categorical_crossentropy,
    optimizer=keras.optimizers.Adadelta(),
    metrics=["accuracy"],
)

model.fit(
    x_train,
    y_train,
    batch_size=batch_size,
    epochs=epochs,
    verbose=1,
    validation_data=(x_test, y_test),
)
score = model.evaluate(x_test, y_test, verbose=0)
print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

Output

```
(base) PS C:\Users\harsh> & C:/Users/harsh/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/harsh/OneDrive - pdpu.ac.in/HARSH/_PDEU/SEM 6/
Artificial Intelligence/ASSIGNMENTS/21BCP359 AI Class Assignment 5.py"
2024-03-20 09:11:28.730820: I tensorflow/core/util/port.cc:113] oneDNN custom operations are on. You may see slightly different numerical results due
to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2024-03-20 09:11:31.598119: I tensorflow/core/util/port.cc:113] oneDNN custom operations are on. You may see slightly different numerical results due
to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 2s 0us/step
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
C:\Users\harsh\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:99: UserWarning: Do not pass an `
input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model inst
ead.
  super().__init__(
2024-03-20 09:11:40.064406: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions
 in performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
Epoch 1/12
469/469 ━━━━━━━━━━━━━━━━━━━━ 47s 97ms/step - accuracy: 0.1162 - loss: 39.1241 - val_accuracy: 0.4118 - val_loss: 4.7768
Epoch 2/12
469/469 ━━━━━━━━━━━━━━━━━━━━ 52s 111ms/step - accuracy: 0.2310 - loss: 16.1297 - val_accuracy: 0.5825 - val_loss: 1.9206
Epoch 3/12
469/469 ━━━━━━━━━━━━━━━━━━━━ 79s 105ms/step - accuracy: 0.3199 - loss: 7.9788 - val_accuracy: 0.5928 - val_loss: 1.3063
Epoch 4/12
469/469 ━━━━━━━━━━━━━━━━━━━━ 53s 112ms/step - accuracy: 0.3507 - loss: 4.5408 - val_accuracy: 0.5516 - val_loss: 1.3733
```

```
Epoch 5/12
469/469 ━━━━━━━━━━━━━━━━━━━━ 52s 110ms/step - accuracy: 0.3562 - loss: 3.0447 - val_accuracy: 0.5067 - val_loss: 1.5433
Epoch 6/12
469/469 ━━━━━━━━━━━━━━━━━━━━ 47s 100ms/step - accuracy: 0.3543 - loss: 2.4200 - val_accuracy: 0.4842 - val_loss: 1.6276
Epoch 7/12
469/469 ━━━━━━━━━━━━━━━━━━━━ 47s 100ms/step - accuracy: 0.3692 - loss: 2.1728 - val_accuracy: 0.5037 - val_loss: 1.6175
Epoch 8/12
469/469 ━━━━━━━━━━━━━━━━━━━━ 46s 98ms/step - accuracy: 0.3781 - loss: 2.0311 - val_accuracy: 0.5177 - val_loss: 1.5752
Epoch 9/12
469/469 ━━━━━━━━━━━━━━━━━━━━ 47s 101ms/step - accuracy: 0.3878 - loss: 1.9486 - val_accuracy: 0.5353 - val_loss: 1.5090
Epoch 10/12
469/469 ━━━━━━━━━━━━━━━━━━━━ 59s 125ms/step - accuracy: 0.4073 - loss: 1.8606 - val_accuracy: 0.5694 - val_loss: 1.4370
Epoch 11/12
469/469 ━━━━━━━━━━━━━━━━━━━━ 63s 133ms/step - accuracy: 0.4303 - loss: 1.8076 - val_accuracy: 0.5874 - val_loss: 1.3649
Epoch 12/12
469/469 ━━━━━━━━━━━━━━━━━━━━ 52s 112ms/step - accuracy: 0.4500 - loss: 1.7371 - val_accuracy: 0.6071 - val_loss: 1.2898
Test loss: 1.2905181646347046
Test accuracy: 0.6071000099182129
```