```
# -*- coding: utf-8 -*-
'''
In this assignment:
·        You learn how build a Convolution Neural Network for image classification a compute
.        The image classification task is the handwritten digit classification using the MNI

Dataset

The MNIST dataset is an acronym that stands for the Modified National Institute of Standards

It is a dataset of 60,000 small square 28×28 pixel grayscale images of handwritten single di

The task is to classify a given image of a handwritten digit into one of 10 classes represen

'''



import numpy as np
from tensorflow import keras
from tensorflow.keras import layers
#Prepare the data
# Model / data parameters
num_classes = 10
input_shape = (28, 28, 1)

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

```
#  Scale images to the [0, 1] range
x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255
# Make sure images have shape (28, 28, 1)
x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -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)
# Set the model to be a simple feed-forward (layered) architecture
# See https://keras.io/api/models/ and https://keras.io/api/models/sequential/
# not to be confused with a sequence-based alg/model to process sequential data
model = keras.Sequential()
# Add an input layer with shape=(28, 28, 1)
# See https://keras.io/api/layers/core_layers/input/
model.add(keras.Input(shape=(28, 28, 1)))
# Add a Conv2D hidden layer with 32 relu activation units and kernel_size of 3*3
# See https://keras.io/api/layers/convolution_layers/convolution2d/
model.add(layers.Conv2D(32, kernel_size=(3, 3), activation='relu'))
# Add a MaxPooling hidden layer and set pool_size of 2*2
# See https://keras.io/api/layers/pooling_layers/max_pooling2d/
model.add(layers.MaxPooling2D(pool_size=(2, 2)))
# Add a Conv2D hidden layer with 64 relu activation units and kernel_size of 3*3
# See https://keras.io/api/layers/convolution_layers/convolution2d/
model.add(layers.Conv2D(64, kernel_size=(3, 3), activation='relu'))
# Add a MaxPooling hidden layer and set pool_size of 2*2
# See https://keras.io/api/layers/pooling_layers/max_pooling2d/
model.add(layers.MaxPooling2D(pool_size=(2, 2)))
# Add a Flatten hidden layer
# See https://keras.io/api/layers/reshaping_layers/flatten/
model.add(layers.Flatten())
# Add a Dropout hidden layer and set rate=0.5
# See https://keras.io/api/layers/regularization_layers/dropout/
model.add(layers.Dropout(0.5))
# Add a dense output layer with units=10 and softmax activation unit
# https://keras.io/api/layers/core_layers/dense/
model.add(layers.Dense(units=10, activation='softmax'))
model.summary()
```

```
# Compile the model, specifying (1) the Adam optimizer,
# (2) the 'categorical_rossentropy' loss function, and (3) metrics=['accuracy']
# See https://keras.io/api/models/model_training_apis/
model.compile(loss='categorical_crossentropy', optimizer='adam',metrics=['accuracy'])
# fit the keras model on the _TRAIN_ dataset set epochs to 15, batch_size to 128, validatior
model.fit(x_train, y_train, batch_size=128, epochs=15, validation_split=0.1)
```

```
60000 train samples
10000 test samples
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320       
                                                                 
 max_pooling2d (MaxPooling2  (None, 13, 13, 32)        0         
 D)                                                              
                                                                 
 conv2d_1 (Conv2D)           (None, 11, 11, 64)        18496     
                                                                 
 max_pooling2d_1 (MaxPoolin  (None, 5, 5, 64)          0         
 g2D)                                                            
                                                                 
 flatten (Flatten)           (None, 1600)              0         
                                                                 
 dropout (Dropout)           (None, 1600)              0         
                                                                 
 dense (Dense)               (None, 10)                16010     
                                                                 
=================================================================
Total params: 34826 (136.04 KB)
Trainable params: 34826 (136.04 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/15
422/422 [==============================] - 51s 118ms/step - loss: 0.3752 - accuracy: 
Epoch 2/15
422/422 [==============================] - 45s 107ms/step - loss: 0.1098 - accuracy: 
Epoch 3/15
422/422 [==============================] - 46s 110ms/step - loss: 0.0830 - accuracy: 
Epoch 4/15
422/422 [==============================] - 44s 105ms/step - loss: 0.0702 - accuracy: 
Epoch 5/15
422/422 [==============================] - 44s 105ms/step - loss: 0.0616 - accuracy: 
Epoch 6/15
422/422 [==============================] - 54s 127ms/step - loss: 0.0562 - accuracy: 
Epoch 7/15
422/422 [==============================] - 44s 104ms/step - loss: 0.0493 - accuracy: 
Epoch 8/15
422/422 [==============================] - 43s 101ms/step - loss: 0.0474 - accuracy: 
Epoch 9/15
422/422 [==============================] - 44s 105ms/step - loss: 0.0438 - accuracy: 
Epoch 10/15
422/422 [==============================] - 43s 102ms/step - loss: 0.0422 - accuracy: 
Epoch 11/15
422/422 [==============================] - 58s 138ms/step - loss: 0.0393 - accuracy: 
Epoch 12/15
422/422 [==============================] - 43s 103ms/step - loss: 0.0362 - accuracy: 
     422/422 [==============================] - 43s 102ms/step - loss: 0.0354 - accuracy: 0
```

```
score = model.evaluate(x_test, y_test, verbose=0)
print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

```
     Test loss: 0.023518381640315056
     Test accuracy: 0.9922000169754028
```

QUESTIONS

Why are Convolutional Neural Networks more popular than Feed Forward Neural Networks in Computer Vision?

CNNs are widely preferred over Feed Forward Neural Networks in computer vision because they are specifically designed for image data. They use convolutional layers to identify patterns in images, allowing them to understand shapes and objects regardless of their position. This makes them effective for tasks like recognizing objects, dividing images into parts, and deciding what an image contains. Compared to traditional neural networks, CNNs perform better on large image datasets because they can learn from shared features and ignore irrelevant details.