ASSIGMENT 2

COURSE CODE: CEN458

COURSE NAME: Introduction to Deep Learning

NAME: YAHIA MOUNIB AMAMRA

ID: *********

Objective:

The project's main goal is to create a Convolutional Neural Network (CNN) that can classify handwritten digits with high accuracy. It uses the MNIST dataset, which contains many labeled images of handwritten digits. CNN will have multiple layers, including convolutional layers for extracting features and dense layers for classification. The model will be trained and validated on a part of the MNIST dataset to make sure it works well on new data it hasn't seen before. The aim is to have a model that can correctly identify the digit in each handwritten image, showing how well CNNs can recognize images.

Dataset Used: MNIST

The MNIST dataset is used to train a Convolutional Neural Network (CNN) via the Keras library. This dataset includes 60,000 training and 10,000 testing images, each being a 28x28 pixel grayscale image of a handwritten digit. The images are reshaped to fit the CNN's input requirements, and their pixel values are normalized between 0 and 1 by dividing by 255. The digit labels (0-9) are one-hot encoded to work with the CNN SoftMax output layer. The MNIST dataset's simplicity makes it perfect for benchmarking the CNN performance, allowing focus on the model's architecture and hyperparameters without complex data preprocessing.

Code:

```
from __future__ import print_function

import tensorflow as tf

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

from keras import backend as K


batch_size = 128

num_classes = 10

epochs = 10
```

```
# input image dimensions
img_rows, img_cols = 28, 28

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

if K.image_data_format() == 'channels_first':
   x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
   x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
   input_shape = (1, img_rows, img_cols)
else:
   x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
   x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
   input_shape = (img_rows, img_cols, 1)

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
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)

#construct the model
model = Sequential()
```

```
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

#set paprmeters to the model
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=tf.keras.optimizers.Adam(),
              metrics=['accuracy'])

#train the model
model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test))

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

Explanations of the code:

Importing Libraries:

- tensorflow and keras libraries are imported to build and train the neural network.
- mnist dataset is imported from keras.datasets to access the MNIST dataset.

Defining Constants:

- batch_size, num_classes, and epochs are defined to control the training process.
- img_rows and img_cols represent the dimensions of the input images (28x28 pixels).

Loading and Preprocessing Data:

- MNIST dataset is loaded and divided into training and testing sets (x_train, y_train) and (x_test, y_test) respectively.
- If the image data format is set to 'channels_first', the input data is reshaped accordingly; otherwise, it reshaped differently.
- The pixel values of the input images are normalized to the range [0, 1] by dividing by 255.

Converting Labels to Categorical:

- Labels (y_train and y_test) are converted to one-hot encoded binary class matrices using keras.utils.to_categorical.

Model Construction:

- A Sequential model is initialized.
- Convolutional layers (Conv2D) with ReLU activation are added to extract features from input images.
- MaxPooling layers (MaxPooling2D) are added to downsample the feature maps.
- The feature maps are flattened into a one dimensional vector (Flatten) before passing through fully connected layers (Dense) with ReLU activation.
- The output layer consists of num_classes neurons with softmax activation to output class probabilities.

Model Compilation:

- The model is compiled with categorical cross entropy loss, Adam optimizer, and accuracy metric using model.compile.

Model Training:

- The model is trained on the training data (x_train, y_train) using model.fit.

- Training is conducted for a specified number of epochs with a defined batch size.
- Validation data (x_test, y_test) is provided to monitor model performance during training.

Model Evaluation:

- The trained model is evaluated on the test data to assess its performance using model.evaluate.
- Test loss and accuracy metrics are printed to the console.

Comparison between CNN & FNN:

FNN output (Assignment 1):

```
Test loss: 0.18834075331687927
Test accuracy: 0.9660999774932861
```

CNN output (Assignment 2):

```
Test loss: 0.04495071619749069
Test accuracy: 0.9894000291824341
```

The accuracy of the Convolutional Neural Network (CNN) Assignment 2 is higher than the Feedforward Neural Network (FNN) Assignment 1. CNN achieved an accuracy of approximately 98.94% while FNN achieved an accuracy of around 96.61%

The reason for the difference in accuracy lies in how each network architecture processes the input data.

FNN:

- FNN treats the input data as a flattened array of pixel values. In the case of the MNIST dataset, each image is 28x28 pixels, so the input is flattened to a vector of length 784.
- It uses fully connected layers (Dense layers) to process this flattened input. Each neuron in one layer is connected to every neuron in the next layer.
- While FNNs are powerful, they may struggle to capture spatial dependencies in the data because they ignore the spatial structure of the input images.

CNN:

- CNN, on the other hand, preserves the spatial structure of the input images.
- It uses convolutional layers to apply filters to small patches of the input images, capturing local patterns.
- Pooling layers are used to reduce the spatial dimensions of the feature maps while retaining important information.
- By using convolutional and pooling layers, CNNs can learn hierarchical representations of the input images, capturing both local and global patterns.

In the case of image classification tasks like MNIST, where spatial relationships between pixels are crucial for accurate classification, CNNs generally outperform FNNs. This is because CNNs are designed to handle spatial data efficiently, capturing features at different spatial levels, whereas FNNs treat all input features equally, which may not be optimal for tasks where spatial relationships matter.