DL 7 LETTER RECOGNITION/MULTICLASS

```
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt
from sklearn import metrics

# Load the OCR dataset

# The MNIST dataset is a built-in dataset provided by Keras.
# It consists of 70,000 28x28 grayscale images, each of which displays a single handwritten digit from 0 to 9.
# The training set consists of 60,000 images, while the test set has 10,000 images.

(x_train, y_train), (x_test, y_test) = mnist.load_data()

# X_train and X_test are our array of images while y_train and y_test are our array of labels for each image.
# The first tuple contains the training set features (X_train) and the training set labels (y_train).
# The second tuple contains the testing set features (X_test) and the testing set labels (y_test).
# For example, if the image shows a handwritten 7, then the label will be the intger 7.

plt.imshow(x_train[0], cmap='gray')  # imshow() function which simply displays an image.
plt.show() # cmap is responsible for mapping a specific colormap to the values found in the array that you passed as the first argument.

# image appears black and white and that each axis of the plot ranges from 0 to 28.

# This is because of the format that all the images in the dataset have:

# 1. All the images are grayscale, meaning they only contain black, white and grey.
# 2. The images are 28 pixels by 25 pixels in size (28x28).

print(x_train[0])

# image data is just an array of digits. You can almost make out a 5 from the pattern of the digits in the array.
# Array of 28 values
# a grayscale pixel is stored as a digit between 0 and 255 where 0 is black, 255 is white and values in between are different shades of gray.
# Therefore, each value in the [28][28] array tells the computer which color to put in that position when we display the actual image.

# reformat our X_train array and our X_test array because they do not have the correct shape.
```

```
# Reshape the data to fit the model
print("X_train shape", x_train.shape)
print("y_train shape", y_train.shape)
print("X_test shape", x_test.shape)
print("y_test shape", y_test.shape)


# Here you can see that for the training sets we have 60,000 elements and the testing sets have 10,000 elements.

# y_train and y_test only have 1 dimensional shapes because they are just the labels of each element.

# x_train and x_test have 3 dimensional shapes because they have a width and height (28x28 pixels) for each element.

# (60000, 28, 28) 1st parameter in the tuple shows us how much image we have 2nd and 3rd parameters are the pixel values from x to y (28x28)
# The pixel value varies between 0 to 255.
# (60000,) Training labels with integers from 0-9 with dtype of uint8. It has the shape (60000,).
# (10000, 28, 28) Testing data that consists of grayscale images. It has the shape (10000, 28, 28) and the dtype of uint8. The pixel value varies between 0 to 255.
# (10000,) Testing labels that consist of integers from 0-9 with dtype uint8. It has the shape (10000,).


# X: Training data of shape (n_samples, n_features)
# y: Training label values of shape (n_samples, n_labels)
# 2D array of height and width, 28 pixels by 28 pixels will just become 784 pixels (28 squared).
# Remember that X_train has 60,000 elemenets, each with 784 total pixels so will become shape (60000, 784).
# Whereas X_test has 10,000 elements, each with each with 784 total pixels so will become shape (10000, 784).

x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
x_train = x_train.astype('float32') # use 32-bit precision when training a neural network, so at one point the training data will have to be converted to 32 bit floats. Since the dataset fits easily in RAM, we might as well convert to float immediately.
x_test = x_test.astype('float32')
x_train /= 255  # Each image has Intensity from 0 to 255
x_test /= 255

# Regarding the division by 255, this is the maximum value of a byte (the input feature's type before the conversion to float32),
# so this will ensure that the input features are scaled between 0.0 and 1.0.
# USING svm-https://mgta.gmu.edu/courses/ml-with-python/handwrittenDigitRecognition.php#:~:text=Remember%20that%20X_train%20has%2060%2C000,keras.
```

```
# Convert class vectors to binary class matrices
num_classes = 10
y_train = np.eye(num_classes)[y_train]  # Return a 2-D array with ones on the diagonal and zeros elsewhere.
y_test = np.eye(num_classes)[y_test] # f your particular categories is present then it mark as 1 else 0 in remain row



# Define the model architecture
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))  # The input_shape argument is passed to the foremost layer. It comprises of a tuple shape,
model.add(Dropout(0.2)) # DROP OUT RATIO 20%
model.add(Dense(512, activation='relu')) #returns a sequence of vectors of dimension 512
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))



# Compile the model
model.compile(loss='categorical_crossentropy',  # for a multi-class classification problem
              optimizer=RMSprop(),
              metrics=['accuracy'])



# Train the model
batch_size = 128 # batch_size argument is passed to the layer to define a batch size for the inputs.
epochs = 20
history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1, # verbose=1 will show you an animated progress bar eg. [==========]
                    validation_data=(x_test, y_test)) # Using validation_data means you are providing the training set and validation set yourself,
                                      # validation_split means you only provide a training set and keras splits it into a training set and a validation set



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

EXPLANATION

Let's go through the code step by step:

This code demonstrates the training and evaluation of a neural network model for digit recognition using the MNIST dataset. Here's a breakdown of the code:

1. Import libraries: The necessary libraries are imported, including `numpy` for numerical computations, `Sequential` and `Dense` from Keras for defining the

model architecture, `Dropout` for regularization, `RMSprop` optimizer, `mnist` from Keras for loading the MNIST dataset, `matplotlib` for visualization, and `metrics` from scikit-learn for evaluation.
2. Load the MNIST dataset: The MNIST dataset is loaded using the `load_data()` function from `mnist`. It provides 60,000 training images and 10,000 testing images of handwritten digits.
3. Visualize an image: The first image from the training set (`x_train[0]`) is displayed using `imshow()` from `matplotlib`.
4. Reshape and preprocess the data: The training and testing data are reshaped to have a single dimension of 784 pixels. The data is then converted to float32 and divided by 255 to scale the pixel values between 0 and 1.
5. Convert labels to categorical: The labels (`y_train` and `y_test`) are converted to categorical format using `np.eye(num_classes)` to create one-hot encoded vectors.
6. Define the model architecture: A sequential model is created using `Sequential()`. It consists of two hidden layers with 512 units and ReLU activation function. Dropout regularization with a rate of 0.2 is applied after each hidden layer. The output layer has `num_classes` units with softmax activation.
7. Compile the model: The model is compiled with categorical cross-entropy loss, RMSprop optimizer, and accuracy metric.
8. Train the model: The model is trained on the training data (`x_train` and `y_train`) using `fit()`. The training is performed for a specified number of epochs and with a batch size of 128. The training progress is displayed (`verbose=1`), and the validation data (`x_test` and `y_test`) is used for monitoring the performance during training.
9. Evaluate the model: The model is evaluated on the testing data using `evaluate()`, and the test loss and accuracy are printed.

The code demonstrates the process of building and training a neural network model for digit recognition using the MNIST dataset. The model achieves a certain accuracy on the test set, which indicates its performance in classifying handwritten digits.

ORAL QUESTIONS

1. Multi Classification, also known as multiclass classification or multiclass classification problem, is a type of classification problem where the goal is to assign input data to one of three or more classes or categories. In other words, instead of binary classification, where the goal is to assign input data to one of two classes (e.g., positive or negative), multiclass classification involves assigning input data to one of several possible classes or categories (e.g., animal species, types of products, etc)
2. Example of multiclass classification- Here are a few examples of multiclass classification problems:

- Image classification: The goal is to classify images into one of several categories. For example, an image classification model might be trained to classify images of animals into categories such as cats, dogs, and birds.

- Text classification: The goal is to classify text documents into one of several categories. For example, a text classification model might be trained to classify news articles into categories such as politics, sports, and entertainment.
- Disease diagnosis: The goal is to diagnose patients with one of several diseases based on their symptoms and medical history. For example, a disease diagnosis model might be trained to classify patients into categories such as diabetes, cancer, and heart disease.
- Speech recognition: The goal is to transcribe spoken words into text. A speech recognition model might be trained to recognize spoken words in several languages or dialects.
- Credit risk analysis: The goal is to classify loan applicants into categories such as low risk, medium risk, and high risk. A credit risk analysis model might be trained to classify loan applicants based on their credit score, income, and other factors.