Import Libraries and Dataset:

Import all the needed modules for training the model. Dataset can be imported easily because the Keras library already contains many datasets and MNIST is one of them.

> **Keras** is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an innterface for the TensorFlow library.
>
> **TensorFlow** is a foundation library that can be used to create Deep Learning models directly or by using wrapper libraries that simplify the process built on top of TensorFlow. Is a fast numeral computing created and released by Google.
>
> We call **mnist.load_data()** function to get training data with its labels and also the testing with its labels.
>
> **Sequential** groups a linear stack of layers into a tf.keras.Model. Sequential provides training and inference features on this model.

Flatten is the function that converts the pooled feature map to a single column that is passed to the fully connected layer. **Dense** adds the fully connected layer to the neural network.

Keras **Conv2D** is a 2D Convolution Layer, this layer creates a convolution kernel that is wind with l layers input which helps produce a tensor of outputs.

Max pooling 2D spatial data. Downsamples the input along its spatial dimensions (height and width) by taking the maximum value over an input window (of size defined by pool_size) for each channel of the input. The window is shifted by strides along each dimension.

Keras utilities model plotting utilities.

SGD optimizers schedules LearningRateSchedule, or a callable that takes no arguments and returns the actual value to use. The learning rate. Defaults. To 0.01.

Load the MNIST dataset using the Keras helper function. Split the data training and testing sets.

I did a visualization of four images in the training dataset to make sure it was downloaded correctly.

The Data Preprocessing:

Model cannot take the image data directly, so we need to perform some basic operations and process the data to make it ready for our neural network. The training dataset is structured as a 3 dimensional array of instance, image width, and image height. For a multi-layer perception model, we must reduce the images down into a vector of pixel. The dimension of training dat is (60000*28*28). One more dimension is needed for the CNN model, so we reshape the matrix to shape (60000*28*28*1)

You can do this transformation easily using the reshape() function on the NumPy array. You can also reduce your memory requirements by forcing the precision of the pixel values to be 32-bit, the default precision used by Keras.

The pixel values are grayscale between 0 and 255. It is almost always a good idea to perform some scaling of input values when using neural network models. Because the scale is well known and well behaved, you can very quickly normalize the pixel values to the range 0 to 1 by dividing each value by the maximum of 255.

Finally, the output variable is an integer from 0 to 9. This is a multi-class classification problem. It is a good practice to use a one-en-coding of the class values, transforming the vector of class integers into a binary matrix.

You can easily do this by using the built-in tf.keras.utils.to_categorical() helper function in Keras.

Create the model:

We are now ready to create your simple neural network model. We will define the model in a function. This is handy if you want to extend the example later and try and get a better score.

We will create the CNN model for this Python-based data science project. A convolutional layer and pooling layers are the two wheels of a CNN model. The reason behind the success of CNN for image classification problems is it feasibility with grid structured data. We will use the Adadelta optimizer for the model compilation.

Train the model:

To start the training of the model we can simply call the model.fit() function of Keras. It takes the training data, validation data, epochs, and batch size as a parameter,

The model is a simple neural network with one hidden layer with the same number of neurons as there are inputs. A rectifier activation function is used for the neurons in the hidden layer. A SoftMax activation function is used on the output layer to turn the outputs into probability-like values and allow one class of the ten to be selected as the model's output prediction. Logarithmic loss is used as the loss function (called categorical_corossentropy in Keras), and the efficient ADAM gradient descent algorithm is used to learn the weights.

We can now fit and evaluate the model. The model is fit over ten epochs with updates every 128 images. The test data is used as the validation dataset, allowing us to see the skill of the model as it trains. A verbose value of 2 is used to reduce the output to one line for each training epoch.

After tying this all together, the complete code listing is provide below.

The training of model takes some time. After successful model training, we can save the weights and model definition in the ‘mnist.h5’ file.

Note: Your results may vary given the stochastic nature of the algorithm or evaluation procedure, or differences in numerical precision. Consider running the example a few times and compare the average outcome.

Evaluate the model:

Finally, the test dataset is used to evaluate the model, and a classification error rate is printed.

To evaluate how accurately our model works, we have around 10,000 images in our dataset. In the training of the data model, we do not include the testing data, which is why it is new data for our model. Around 99% accuracy is achieved with this well-balanced MNIST dataset.

Output:

```
The model has successfully trained.
Saving the bot as mnist.hf
```

We should see then see the output. This very simple network, defined in very few lines of code achieves a respectable baseline error of 17.33% and a loss of 29% and accuracy of 92%

Source:
https://machinelearningmastery.com/handwritten-digit-recognition-using-convolutional-neural-networks-python-keras/