```
import numpy as np
import keras
from keras import layers
```

Prepare the data

```
# Model / data parameters
num_classes = 10
input_shape = (28, 28, 1)

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


```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
```

```
# 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)
```

```
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
```

Build the model

```
model = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation="softmax"),
    ]
)

model.summary()
```

```
Model: "sequential"
```

Layer (type)	Output Shape	
conv2d (Conv2D)	(None, 26, 26, 32)	
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	
conv2d_1 (Conv2D)	(None, 11, 11, 64)	
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	
flatten (Flatten)	(None, 1600)	
dropout (Dropout)	(None, 1600)	
dense (Dense)	(None, 10)	

```
Total params: 34,826 (136.04 KB)
Trainable params: 34,826 (136.04 KB)
Non-trainable params: 0 (0.00 B)
```

Train the model

```
batch_size = 128
epochs = 15

model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])

model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_split=0.1)

```

```
Epoch 1/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 57s 127ms/step - accuracy: 0.7690 - loss: 0.7500 - val_accuracy: 0.9755 - val_loss
Epoch 2/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 73s 107ms/step - accuracy: 0.9595 - loss: 0.1318 - val_accuracy: 0.9828 - val_loss
Epoch 3/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 43s 101ms/step - accuracy: 0.9711 - loss: 0.0928 - val_accuracy: 0.9860 - val_loss
Epoch 4/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 82s 101ms/step - accuracy: 0.9772 - loss: 0.0757 - val_accuracy: 0.9885 - val_
Epoch 5/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 42s 101ms/step - accuracy: 0.9804 - loss: 0.0641 - val_accuracy: 0.9895 - val_loss
Epoch 6/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 82s 101ms/step - accuracy: 0.9811 - loss: 0.0610 - val_accuracy: 0.9898 - val_loss
Epoch 7/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 44s 105ms/step - accuracy: 0.9824 - loss: 0.0536 - val_accuracy: 0.9893 - val_loss
Epoch 8/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 44s 103ms/step - accuracy: 0.9836 - loss: 0.0511 - val_accuracy: 0.9902 - val_loss
Epoch 9/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 82s 103ms/step - accuracy: 0.9851 - loss: 0.0489 - val_accuracy: 0.9917 - val_loss
Epoch 10/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 82s 102ms/step - accuracy: 0.9850 - loss: 0.0462 - val_accuracy: 0.9908 - val_loss
Epoch 11/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 85s 110ms/step - accuracy: 0.9861 - loss: 0.0440 - val_accuracy: 0.9913 - val_loss
Epoch 12/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 48s 113ms/step - accuracy: 0.9873 - loss: 0.0410 - val_accuracy: 0.9918 - val_loss
Epoch 13/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 77s 100ms/step - accuracy: 0.9879 - loss: 0.0362 - val_accuracy: 0.9923 - val_loss
Epoch 14/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 43s 101ms/step - accuracy: 0.9878 - loss: 0.0370 - val_accuracy: 0.9920 - val_loss
Epoch 15/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 82s 101ms/step - accuracy: 0.9888 - loss: 0.0341 - val_accuracy: 0.9915 - val_loss
<keras.src.callbacks.history.History at 0x7f753a3936d0>
```

Evaluate the trained model

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

```
Test loss: 0.02719920687377453
Test accuracy: 0.9908999800682068
```

```
Start coding or generate with AI.
```