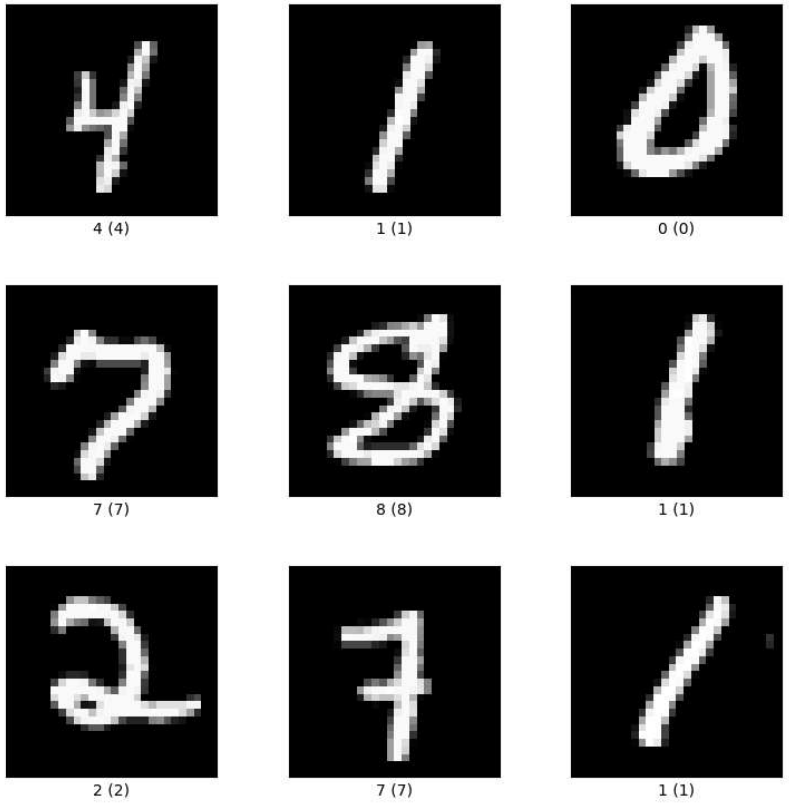
Classifcation on hand writen numbers

In this activity we will be using MNIST dataset: https://www.tensorflow.org/datasets/catalog/mnist (https://www.tensorflow.org/datasets/catalog/mnist)

To develop a classifer of hand writen numbers as shown in the following figure



This activity is modiffied from https://keras.io/examples/vision/mnist_convnet/ (https://keras.io/examples/vision/mnist_convnet/)

Title: Simple MNIST convnet

Description: A simple convnet that achieves ~99% test accuracy on MNIST.

Setup

In [1]:
```
import numpy as np
from tensorflow import keras
from tensorflow.keras import layers
```

Prepare the data

Model / data parameters

In [2]:
```
num_classes = 10
input_shape = (28, 28, 1)
```

the data, split between train and test sets

In [3]:
```
(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 (https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz)
11490434/11490434 [==============================] - 1s 0us/step
```

Scale images to the [0, 1] range

In [4]:
```
x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255
```

Make sure images have shape (28, 28, 1)

In [5]:
```
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")
```

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

convert class vectors to binary class matrices

In [6]:
```
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

Build the model

In [7]:

```
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              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 flatten (Flatten)           (None, 1600)              0

 dropout (Dropout)           (None, 1600)              0

 dense (Dense)               (None, 10)                16010

=================================================================
Total params: 34,826
Trainable params: 34,826
Non-trainable params: 0
_________________________________________________________________
```

Train the model

In [8]:

```
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 [==============================] - 38s 84ms/step - loss: 0.3719 - accuracy: 0.8880 - val_loss: 0.0853 - val_acc
uracy: 0.9770
Epoch 2/15
422/422 [==============================] - 33s 78ms/step - loss: 0.1141 - accuracy: 0.9654 - val_loss: 0.0591 - val_acc
uracy: 0.9825
Epoch 3/15
422/422 [==============================] - 35s 82ms/step - loss: 0.0885 - accuracy: 0.9720 - val_loss: 0.0513 - val_acc
uracy: 0.9877
Epoch 4/15
422/422 [==============================] - 34s 81ms/step - loss: 0.0730 - accuracy: 0.9778 - val_loss: 0.0507 - val_acc
uracy: 0.9863
Epoch 5/15
422/422 [==============================] - 33s 79ms/step - loss: 0.0643 - accuracy: 0.9801 - val_loss: 0.0383 - val_acc
uracy: 0.9895
Epoch 6/15
422/422 [==============================] - 33s 79ms/step - loss: 0.0577 - accuracy: 0.9816 - val_loss: 0.0367 - val_acc
uracy: 0.9905
Epoch 7/15
422/422 [==============================] - 33s 79ms/step - loss: 0.0524 - accuracy: 0.9836 - val_loss: 0.0350 - val_acc
uracy: 0.9913
Epoch 8/15
422/422 [==============================] - 32s 76ms/step - loss: 0.0480 - accuracy: 0.9843 - val_loss: 0.0350 - val_acc
uracy: 0.9905
Epoch 9/15
422/422 [==============================] - 40s 95ms/step - loss: 0.0459 - accuracy: 0.9847 - val_loss: 0.0349 - val_acc
uracy: 0.9907
Epoch 10/15
422/422 [==============================] - 34s 79ms/step - loss: 0.0438 - accuracy: 0.9855 - val_loss: 0.0313 - val_acc
uracy: 0.9918
Epoch 11/15
422/422 [==============================] - 32s 76ms/step - loss: 0.0413 - accuracy: 0.9864 - val_loss: 0.0330 - val_acc
uracy: 0.9910
Epoch 12/15
422/422 [==============================] - 32s 75ms/step - loss: 0.0368 - accuracy: 0.9882 - val_loss: 0.0290 - val_acc
uracy: 0.9917
Epoch 13/15
422/422 [==============================] - 32s 76ms/step - loss: 0.0374 - accuracy: 0.9879 - val_loss: 0.0295 - val_acc
uracy: 0.9927
Epoch 14/15
422/422 [==============================] - 34s 80ms/step - loss: 0.0363 - accuracy: 0.9877 - val_loss: 0.0311 - val_acc
```

```
uracy: 0.9915
Epoch 15/15
422/422 [==============================] - 31s 73ms/step - loss: 0.0325 - accuracy: 0.9891 - val_loss: 0.0301 - val_acc
uracy: 0.9922
```

Out[8]:
```
<keras.callbacks.History at 0x2127a67b890>
```

Evaluate the trained model

In [9]:
```
1 score = model.evaluate(x_test, y_test, verbose=0)
2 print("Test loss:", score[0])
3 print("Test accuracy:", score[1])
```

```
Test loss: 0.02624954842031002
Test accuracy: 0.9909999966621399
```