Handwritten digits recognition (using Convolutional Neural Network)

```
# Selecting Tensorflow version v2 (the command is relevant for Colab only).
%tensorflow_version 2.x
```

```
Colab only includes TensorFlow 2.x; %tensorflow_version has no effect.
```

```
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sn
import numpy as np
import pandas as pd
import math
import datetime
import platform

print('Python version:', platform.python_version())
print('Tensorflow version:', tf.__version__)
print('Keras version:', tf.keras.__version__)
```

```
Python version: 3.11.12
Tensorflow version: 2.18.0
Keras version: 3.8.0
```

Configuring Tensorboard

We will use [Tensorboard] to debug the model later.

```
# Load the TensorBoard notebook extension.
# %reload_ext tensorboard
%load_ext tensorboard
```

```
# Clear any logs from previous runs.
!rm -rf ./.logs/
```

Load the data

The **training** dataset consists of 60000 28x28px images of hand-written digits from `0` to `9`.

The **test** dataset consists of 10000 28x28px images.

```
mnist_dataset = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist_dataset.load_data()
```

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

```
print('x_train:', x_train.shape)
print('y_train:', y_train.shape)
print('x_test:', x_test.shape)
print('y_test:', y_test.shape)
```

```
x_train: (60000, 28, 28)
y_train: (60000,)
x_test: (10000, 28, 28)
y_test: (10000,)
```

```
# Save image parameters to the constants that we will use later for data re-shaping and for model traning.
(_, IMAGE_WIDTH, IMAGE_HEIGHT) = x_train.shape
IMAGE_CHANNELS = 1

print('IMAGE_WIDTH:', IMAGE_WIDTH);
print('IMAGE_HEIGHT:', IMAGE_HEIGHT);
print('IMAGE_CHANNELS:', IMAGE_CHANNELS);
```

```
IMAGE_WIDTH: 28
IMAGE_HEIGHT: 28
```

```
IMAGE_CHANNELS: 1
```

Explore the data

Here is how each image in the dataset looks like. It is a 28x28 matrix of integers (from `0` to `255`). Each integer represents a color of a pixel.

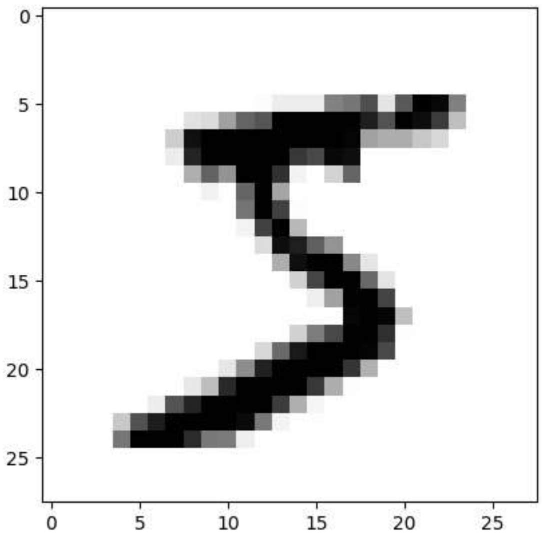
```
pd.DataFrame(x_train[0])
```

	0	1	2	3	4	5	6	7	8	9	...	18	19	20	21	22	23	24	25	26	27
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	...	175	26	166	255	247	127	0	0	0	0
6	0	0	0	0	0	0	0	0	30	36	...	225	172	253	242	195	64	0	0	0	0
7	0	0	0	0	0	0	0	49	238	253	...	93	82	82	56	39	0	0	0	0	0
8	0	0	0	0	0	0	0	18	219	253	...	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	80	156	...	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	14	...	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	...	25	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	...	150	27	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	...	253	187	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	...	253	249	64	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	...	253	207	2	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	...	250	182	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	...	78	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	23	66	...	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	18	171	219	253	...	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	55	172	226	253	253	253	...	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	136	253	253	253	212	135	...	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

28 rows × 28 columns

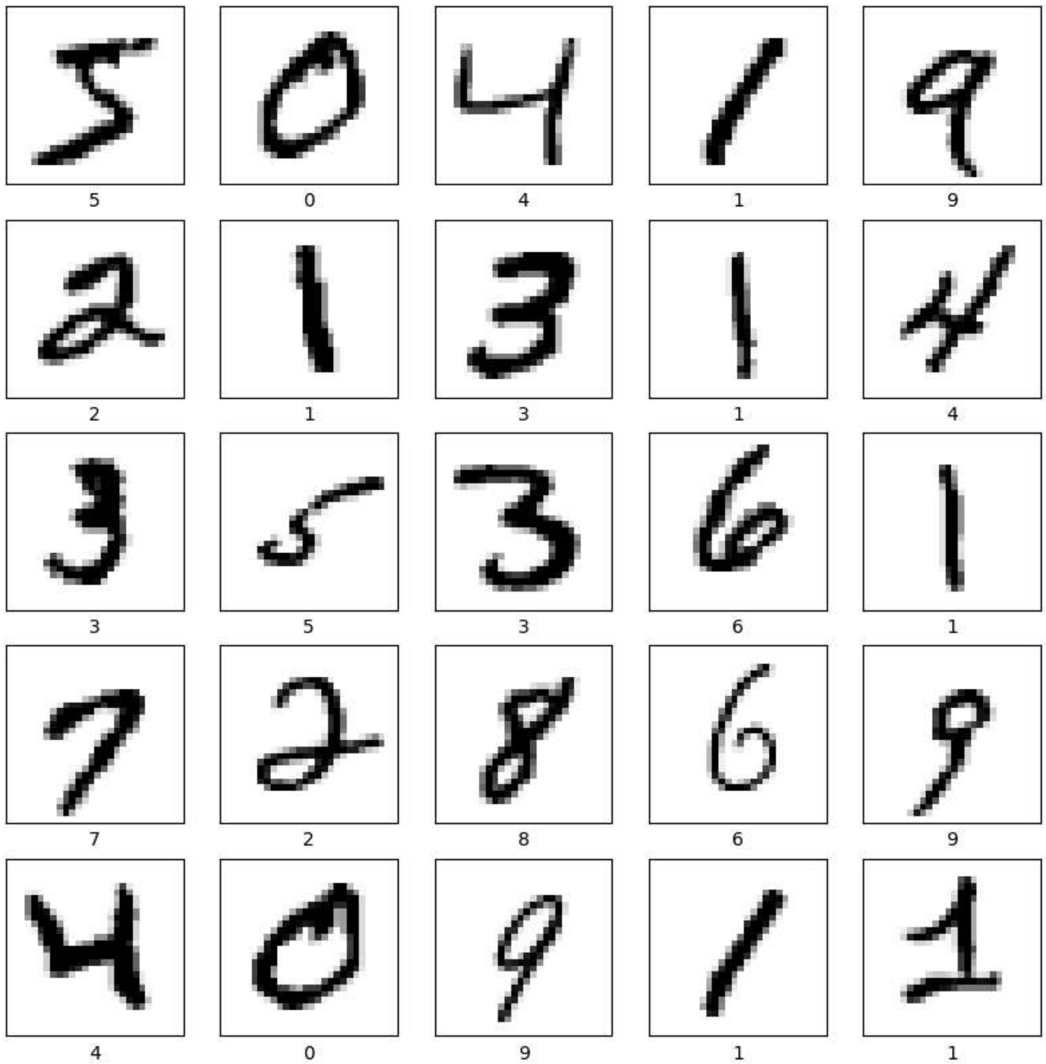
This matrix of numbers may be drawn as follows:

```
plt.imshow(x_train[0], cmap=plt.cm.binary)
plt.show()
```



Let's print some more training examples to get the feeling of how the digits were written.

```
numbers_to_display = 25
num_cells = math.ceil(math.sqrt(numbers_to_display))
plt.figure(figsize=(10,10))
for i in range(numbers_to_display):
    plt.subplot(num_cells, num_cells, i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(x_train[i], cmap=plt.cm.binary)
    plt.xlabel(y_train[i])
plt.show()
```



Reshaping the data

In order to use convolution layers we need to reshape our data and add a color channel to it. As you've noticed currently every digit has a shape of `(28, 28)` which means that it is a 28x28 matrix of color values form `0` to `255`. We need to reshape it to `(28, 28, 1)` shape so that each pixel potentially may have multiple channels (like Red, Green and Blue).

```
x_train_with_chanels = x_train.reshape(
    x_train.shape[0],
    IMAGE_WIDTH,
    IMAGE_HEIGHT,
    IMAGE_CHANNELS
)

x_test_with_chanels = x_test.reshape(
    x_test.shape[0],
    IMAGE_WIDTH,
    IMAGE_HEIGHT,
    IMAGE_CHANNELS
)


print('x_train_with_chanels:', x_train_with_chanels.shape)
print('x_test_with_chanels:', x_test_with_chanels.shape)
```

```
x_train_with_chanels: (60000, 28, 28, 1)
x_test_with_chanels: (10000, 28, 28, 1)
```

Normalize the data

Here we're just trying to move from values range of `[0...255]` to `[0...1]`.

```
x_train_normalized = x_train_with_chanels / 255
x_test_normalized = x_test_with_chanels / 255
```

```
# Let's check just one row from the 0th image to see color chanel values after normalization.
x_train_normalized[0][18]
```

```
array([[0.        ],
       [0.        ],
       [0.        ],
       [0.        ],
       [0.        ],
       [0.        ],
       [0.        ],
       [0.        ],
       [0.        ],
       [0.        ],
       [0.        ],
       [0.        ],
       [0.        ],
       [0.        ],
       [0.18039216],
       [0.50980392],
       [0.71764706],
       [0.99215686],
       [0.99215686],
       [0.81176471],
       [0.00784314],
       [0.        ],
       [0.        ],
       [0.        ],
       [0.        ],
       [0.        ],
       [0.        ],
       [0.        ]])
```

Build the model

We will use Sequential Keras model.

Then we will have two pairs of Convolution2D and MaxPooling2D layers. The MaxPooling layer acts as a sort of downsampling using max values in a region instead of averaging.

After that we will use Flatten layer to convert multidimensional parameters to vector.

The las layer will be a Dense layer with `10` Softmax outputs. The output represents the network guess. The 0-th output represents a probability that the input digit is `0`, the 1-st output represents a probability that the input digit is `1` and so on...

```
model = tf.keras.models.Sequential()

model.add(tf.keras.layers.Convolution2D(
    input_shape=(IMAGE_WIDTH, IMAGE_HEIGHT, IMAGE_CHANNELS),
    kernel_size=5,
    filters=8,
    strides=1,
    activation=tf.keras.activations.relu,
    kernel_initializer=tf.keras.initializers.VarianceScaling()
))

model.add(tf.keras.layers.MaxPooling2D(
    pool_size=(2, 2),
    strides=(2, 2)
))

model.add(tf.keras.layers.Convolution2D(
    kernel_size=5,
    filters=16,
    strides=1,
    activation=tf.keras.activations.relu,
    kernel_initializer=tf.keras.initializers.VarianceScaling()
))

model.add(tf.keras.layers.MaxPooling2D(
    pool_size=(2, 2),
```

```
    strides=(2, 2)
))

model.add(tf.keras.layers.Flatten())

model.add(tf.keras.layers.Dense(
    units=128,
    activation=tf.keras.activations.relu
));

model.add(tf.keras.layers.Dropout(0.2))

model.add(tf.keras.layers.Dense(
    units=10,
    activation=tf.keras.activations.softmax,
    kernel_initializer=tf.keras.initializers.VarianceScaling()
))
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`inpu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Here is our model summary so far.

```
model.summary()
```

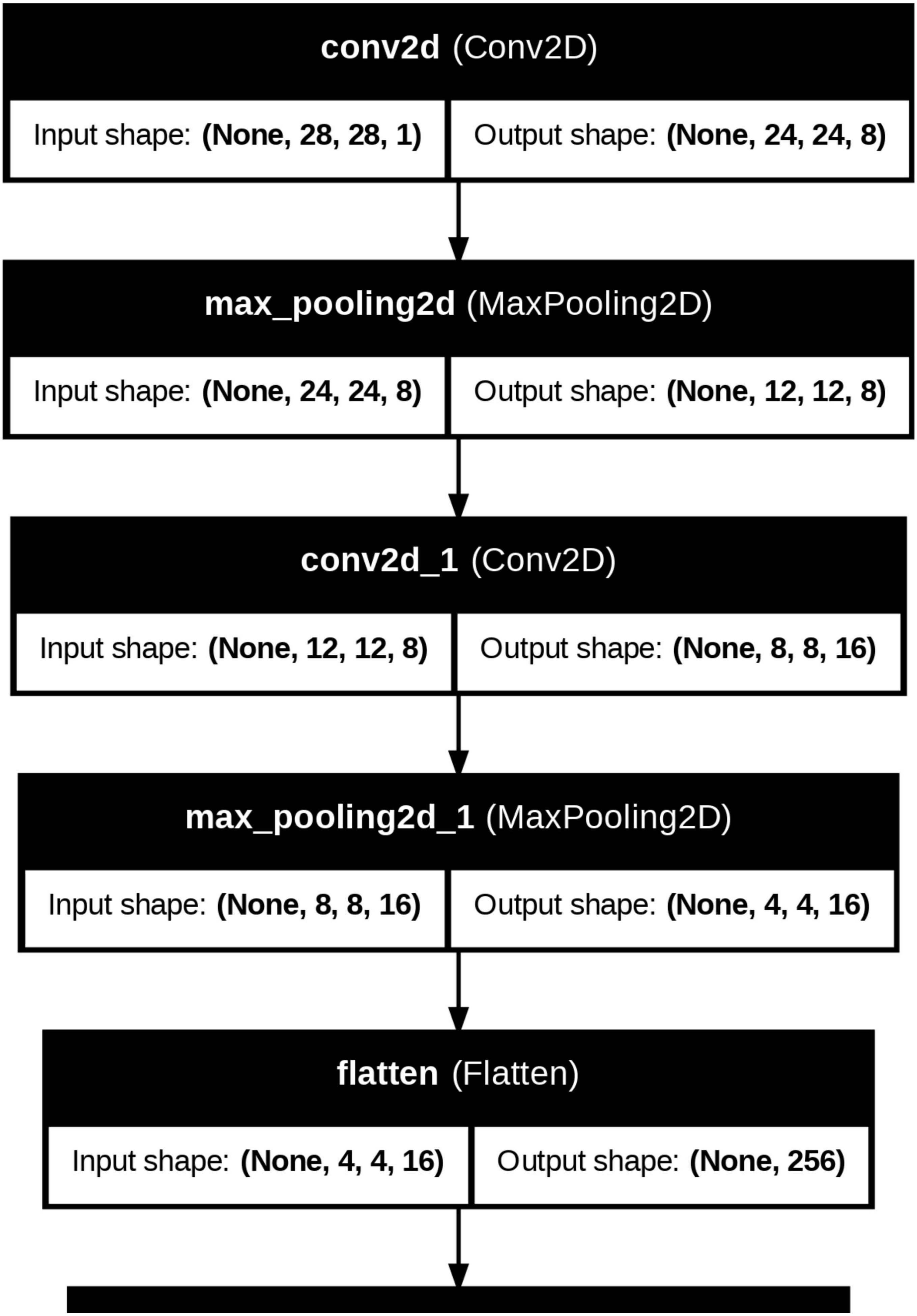
Model: "sequential"

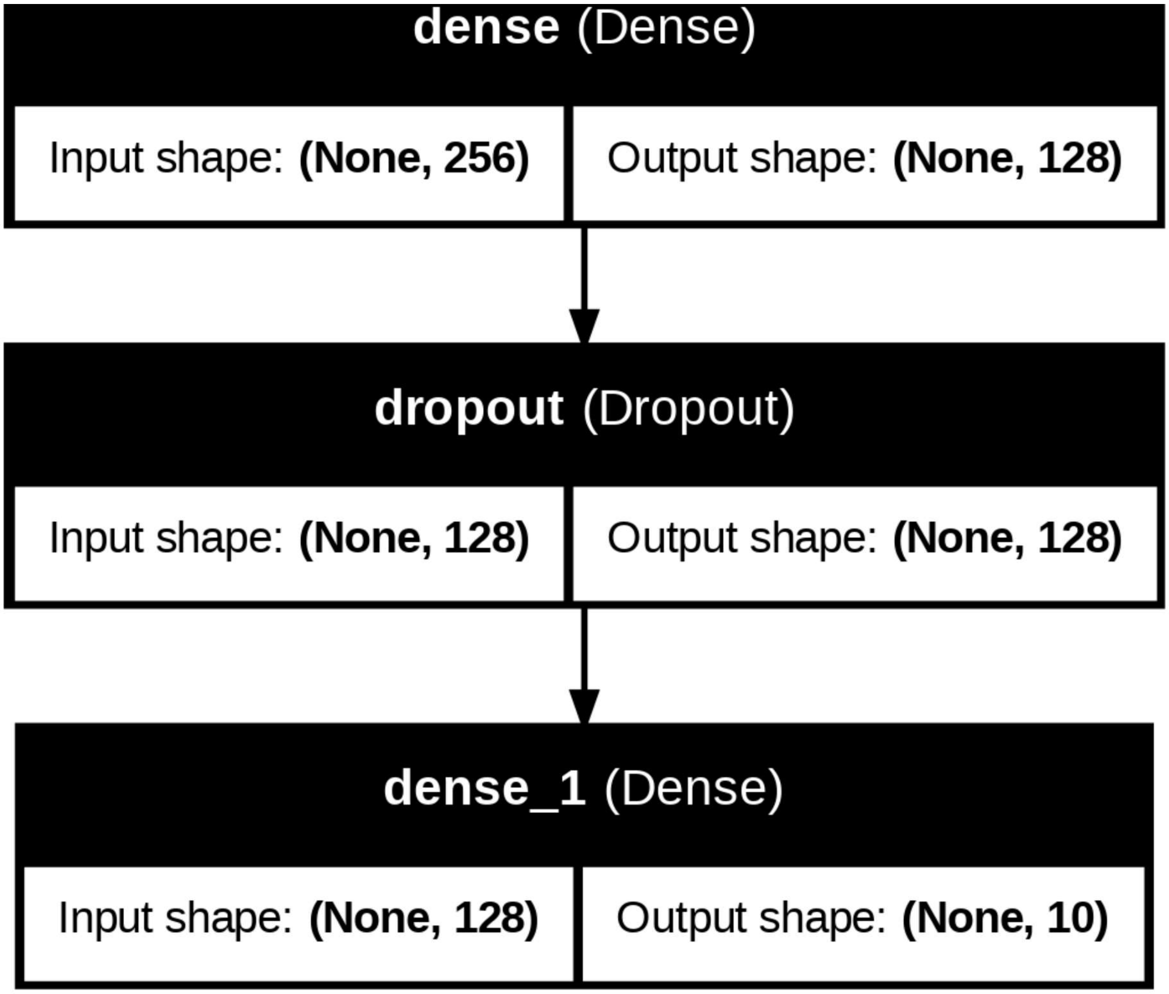
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 24, 24, 8)	208
max_pooling2d (MaxPooling2D)	(None, 12, 12, 8)	0
conv2d_1 (Conv2D)	(None, 8, 8, 16)	3,216
max_pooling2d_1 (MaxPooling2D)	(None, 4, 4, 16)	0
flatten (Flatten)	(None, 256)	0
dense (Dense)	(None, 128)	32,896
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 10)	1,290

Total params: 37,610 (146.91 KB)
Trainable params: 37,610 (146.91 KB)
Non-trainable params: 0 (0.00 B)

In order to plot the model the `graphviz` should be installed. For Mac OS it may be installed using `brew` like `brew install graphviz`.

```
tf.keras.utils.plot_model(
    model,
    show_shapes=True,
    show_layer_names=True,
)
```


conv2d (Conv2D)
Input shape: (None, 28, 28, 1)
Output shape: (None, 24, 24, 8)
max_pooling2d (MaxPooling2D)
Input shape: (None, 24, 24, 8)
Output shape: (None, 12, 12, 8)
conv2d_1 (Conv2D)
Input shape: (None, 12, 12, 8)
Output shape: (None, 8, 8, 16)
max_pooling2d_1 (MaxPooling2D)
Input shape: (None, 8, 8, 16)
Output shape: (None, 4, 4, 16)
flatten (Flatten)
Input shape: (None, 4, 4, 16)
Output shape: (None, 256)


dense (Dense)
Input shape: (None, 256)
Output shape: (None, 128)
dropout (Dropout)
Input shape: (None, 128)
Output shape: (None, 128)
dense_1 (Dense)
Input shape: (None, 128)
Output shape: (None, 10)

Compile the model

```
adam_optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)

model.compile(
    optimizer=adam_optimizer,
    loss=tf.keras.losses.sparse_categorical_crossentropy,
    metrics=['accuracy']
)
```

Train the model

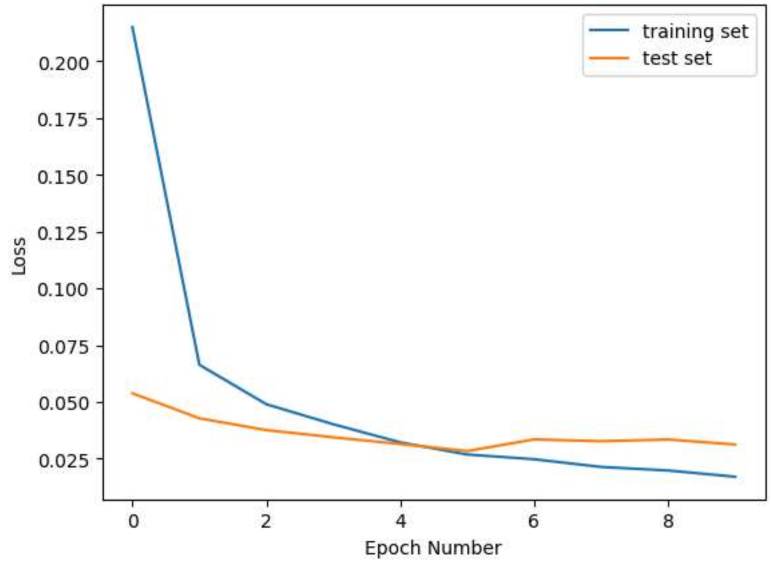
```
log_dir=".logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)

training_history = model.fit(
    x_train_normalized,
    y_train,
    epochs=10,
    validation_data=(x_test_normalized, y_test),
    callbacks=[tensorboard_callback]
)
```

```
Epoch 1/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 39s 19ms/step - accuracy: 0.8495 - loss: 0.4769 - val_accuracy: 0.9834 - val_loss: 0.0538
Epoch 2/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 36s 17ms/step - accuracy: 0.9787 - loss: 0.0701 - val_accuracy: 0.9866 - val_loss: 0.0428
Epoch 3/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 39s 16ms/step - accuracy: 0.9855 - loss: 0.0481 - val_accuracy: 0.9875 - val_loss: 0.0376
Epoch 4/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 30s 16ms/step - accuracy: 0.9861 - loss: 0.0413 - val_accuracy: 0.9900 - val_loss: 0.0344
Epoch 5/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 41s 16ms/step - accuracy: 0.9897 - loss: 0.0319 - val_accuracy: 0.9902 - val_loss: 0.0315
Epoch 6/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 41s 16ms/step - accuracy: 0.9923 - loss: 0.0250 - val_accuracy: 0.9906 - val_loss: 0.0284
Epoch 7/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 30s 16ms/step - accuracy: 0.9919 - loss: 0.0235 - val_accuracy: 0.9899 - val_loss: 0.0335
Epoch 8/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 30s 16ms/step - accuracy: 0.9934 - loss: 0.0203 - val_accuracy: 0.9900 - val_loss: 0.0327
Epoch 9/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 43s 17ms/step - accuracy: 0.9935 - loss: 0.0206 - val_accuracy: 0.9914 - val_loss: 0.0335
Epoch 10/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 39s 16ms/step - accuracy: 0.9940 - loss: 0.0165 - val_accuracy: 0.9918 - val_loss: 0.0312
```

Let's see how the loss function was changing during the training. We expect it to get smaller and smaller on every next epoch.

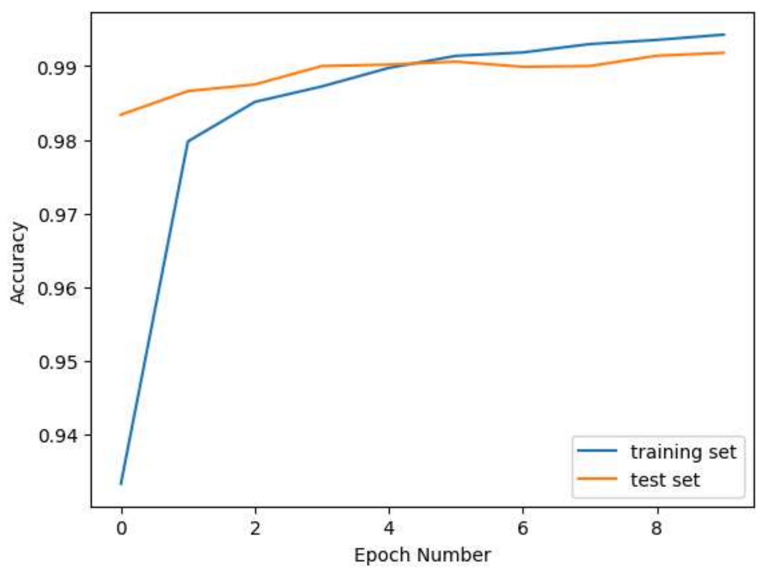
```
plt.xlabel('Epoch Number')
plt.ylabel('Loss')
plt.plot(training_history.history['loss'], label='training set')
plt.plot(training_history.history['val_loss'], label='test set')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7b0010769b90>
```




```
plt.xlabel('Epoch Number')
plt.ylabel('Accuracy')
plt.plot(training_history.history['accuracy'], label='training set')
plt.plot(training_history.history['val_accuracy'], label='test set')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7b001071f5d0>
```



Evaluate model accuracy

We need to compare the accuracy of our model on **training** set and on **test** set. We expect our model to perform similarly on both sets. If the performance on a test set will be poor comparing to a training set it would be an indicator for us that the model is overfitted and we have a "high variance" issue.

Test set accuracy

```
%%capture
validation_loss, validation_accuracy = model.evaluate(x_test_normalized, y_test)
```

```
print('Validation loss: ', validation_loss)
print('Validation accuracy: ', validation_accuracy)
```

```
Validation loss:  0.031244924291968346
Validation accuracy:  0.9918000102043152
```

Save the model

We will save the entire model to a `HDF5` file. The `.h5` extension of the file indicates that the model shuold be saved in Keras format as HDF5 file. To use this model on the front-end we will convert it (later in this notebook) to Javascript understandable format (`tfjs_layers_model` with .json and .bin files) using [tensorflowjs_converter] as it is specified in the [main README].

```
model_name = 'digits_recognition_cnn.h5'
model.save(model_name, save_format='h5')
```

```
WARNING:absl:The `save_format` argument is deprecated in Keras 3. We recommend removing this argument as it can be inferred from the fil
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is consi
```

```
loaded_model = tf.keras.models.load_model(model_name)
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you t
```

Use the model (do predictions)

To use the model that we've just trained for digits recognition we need to call `predict()` method.

```
predictions_one_hot = loaded_model.predict([x_test_normalized])
```

```
 24/313 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step/usr/local/lib/python3.11/dist-packages/keras/src/models/functional.py:237: UserWarning: The str
Expected: input_layer
Received: inputs=('Tensor(shape=(32, 28, 28, 1))',)
  warnings.warn(msg)
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step
/usr/local/lib/python3.11/dist-packages/keras/src/models/functional.py:237: UserWarning: The structure of `inputs` doesn't match the exp
Expected: input_layer
Received: inputs=('Tensor(shape=(None, 28, 28, 1))',)
  warnings.warn(msg)
```

```
print('predictions_one_hot:', predictions_one_hot.shape)
```

```
predictions_one_hot: (10000, 10)
```

Each prediction consists of 10 probabilities (one for each number from `0` to `9`). We need to pick the digit with the highest probability since this would be a digit that our model most confident with.

```
# Predictions in form of one-hot vectors (arrays of probabilities).
pd.DataFrame(predictions_one_hot)
```

	0	1	2	3	4	5	6	7	8	9
0	1.450584e-17	3.130298e-08	4.027326e-11	4.884618e-09	1.290563e-09	1.003922e-11	1.826908e-18	9.999999e-01	1.311677e-12	6.221824e-09
1	6.829007e-10	2.472261e-13	9.999999e-01	3.427839e-14	7.999394e-13	8.036788e-16	1.103626e-09	7.768365e-16	1.245810e-11	9.745129e-15
2	2.393551e-09	9.999991e-01	1.885732e-08	6.755610e-14	2.235855e-07	8.130527e-08	2.342753e-08	5.267313e-07	1.333733e-08	5.171940e-09
3	9.999993e-01	2.343331e-15	5.171399e-09	9.078210e-14	1.620711e-12	1.905912e-10	6.191001e-07	8.285109e-14	9.177316e-12	6.561930e-10
4	2.110806e-12	2.917405e-10	7.259939e-09	1.199959e-11	9.999999e-01	7.011323e-12	2.941575e-11	1.120925e-09	1.281875e-10	1.134998e-08
...	...	...	...	...	...	...	...	...	...	...
9995	1.711537e-15	1.953314e-10	9.999999e-01	1.655074e-14	6.577903e-16	1.416574e-20	1.931610e-18	1.489753e-09	2.514288e-14	6.157669e-17
9996	1.213293e-12	2.269070e-08	7.643114e-12	9.999998e-01	1.589459e-16	6.720178e-08	8.611951e-14	1.397494e-11	7.523598e-13	1.443314e-10

```
# Let's extract predictions with highest probabilites and detect what digits have been actually recognized.
predictions = np.argmax(predictions_one_hot, axis=1)
pd.DataFrame(predictions)
```

	0
0	7
1	2
2	1
3	0
4	4
...	...
9995	2
9996	3
9997	4
9998	5
9999	6

10000 rows × 1 columns

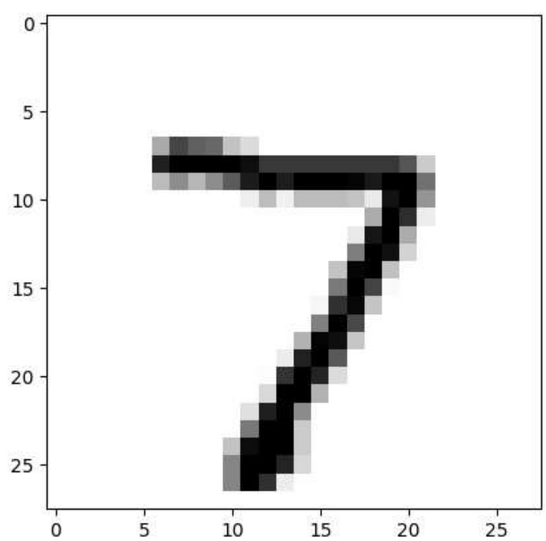
So our model is predicting that the first example from the test set is `7`.

```
print(predictions[0])
```

```
7
```

Let's print the first image from a test set to see if model's prediction is correct.

```
plt.imshow(x_test_normalized[0].reshape((IMAGE_WIDTH, IMAGE_HEIGHT)), cmap=plt.cm.binary)
plt.show()
```



We see that our model made a correct prediction and it successfully recognized digit `7`. Let's print some more test examples and correspondent predictions to see how model performs and where it does mistakes.

```
numbers_to_display = 196
num_cells = math.ceil(math.sqrt(numbers_to_display))
plt.figure(figsize=(15, 15))

for plot_index in range(numbers_to_display):
    predicted_label = predictions[plot_index]
    plt.xticks([])
    plt.yticks([])
```