In [ ]:
```
'''
    Author: A.Shrikant
'''
```

In [1]:
```
# Basic libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Importing the datasets and the DNN libraries
import keras
from keras import datasets
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from keras.models import Sequential
```

In [2]:
```
(x_train, y_train), (x_test, y_test) = datasets.mnist.load_data()
```

In [3]:
```
# data shape
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(60000, 28, 28)
(60000,)
(10000, 28, 28)
(10000,)
```

In [4]:
```
x_train_min = x_train.min()
x_train_max = x_train.max()

print(f'x_train_min: {x_train_min}')
print(f'x_train_max: {x_train_max}')
```

```
x_train_min: 0
x_train_max: 255
```

In [5]:
```
# Scale the data.
x_train = x_train/255.0
```

```
x_test = x_test/255.0
```

In [6]:
```
x_train_min = x_train.min()
x_train_max = x_train.max()

print(f'x_train_min: {x_train_min}')
print(f'x_train_max: {x_train_max}')
```

```
x_train_min: 0.0
x_train_max: 1.0
```

In [7]:
```
x_train[0].shape
```

Out[7]: (28, 28)

In [8]:
```
# Unique values in target variable.
print(np.unique(y_train))
```

```
[0 1 2 3 4 5 6 7 8 9]
```

In [9]:
```
from matplotlib import colormaps
```

In [10]:
```
# plt.rcParams is a dictionary-like object in Matplotlib that stores the default
# configuration parameters for the Matplotlib library. It contains various
# settings related to the appearance and behavior of plots.

# The colormap is a dictionary which maps numbers to colors.
plt.rcParams['image.cmap']
```

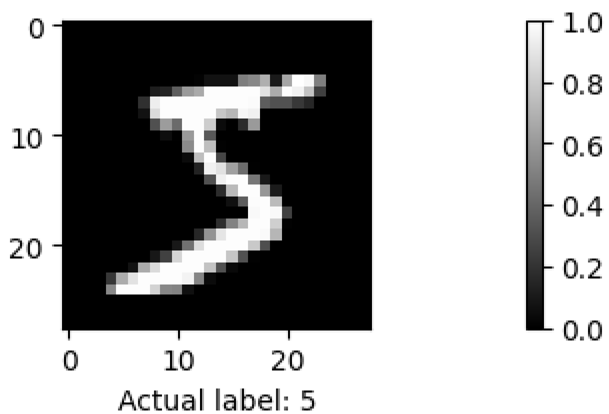
Out[10]: 'viridis'

In [11]:
```
# Get the default colormap
default_cmap = colormaps[plt.rcParams['image.cmap']]
default_cmap
```

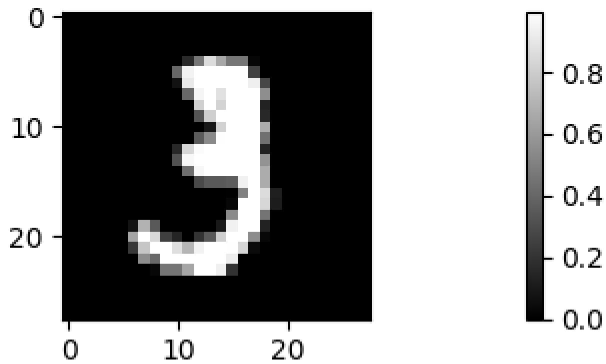
Out[11]: **viridis**

In [12]:
```
# plt.imshow(X, cmap=None)

# X is a multidimensional array representing image data.

# When image has shape (M,N,3) or (M,N,4), the values in image are interpreted
# as RGB or RGBA values. In this case the cmap is ignored.
# cmap : str or ~matplotlib.colors.Colormap, default: image.cmap

# https://stackoverflow.com/questions/25625952/matplotlib-what-is-the-function-of-cmap-in-imshow

def show_img(idx):
  plt.figure(figsize=(20, 2))
  plt.imshow(x_train[idx, :], cmap='gray')
  plt.xlabel(f'Actual label: {y_train[idx]}')
  plt.colorbar()
  plt.show()
```

In [13]:
```
show_img(0)
```

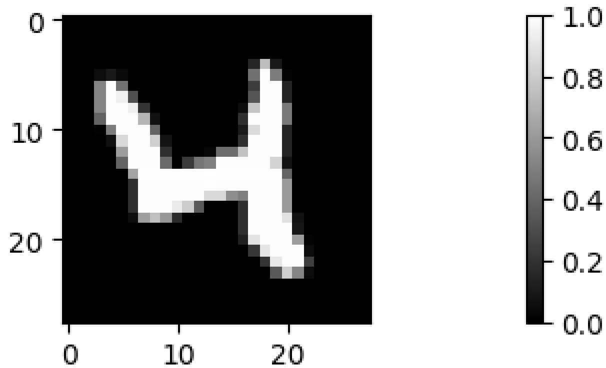


In [14]:
```
show_img(10)
```



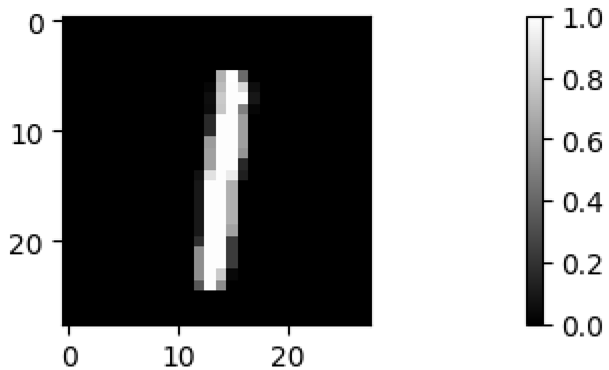
Actual label: 3

In [15]:
```
show_img(20)
```



Actual label: 4

In [16]:
```
show_img(40)
```



Actual label: 1

Building the CNN based model:

In [17]:
```
# Model Building

model = Sequential()

# For Conv2D():

# kernel_size is the filter size(k). Use odd number of filters in convolution.
# So that when padding is 'same' the value p comes out as a whole number.

# padding='valid' by default this means no padding
# padding='same' means that the padding is automatically calculated.
# When strides = 1 the output (feature map) has the same spatial dimensions as the input
# image using the formula p = (k-1)/2.
# When strides > 1 the output (feature map) has the following spatial dimensions:
# output_spatial_shape[i] = ceil(input_spatial_shape[i] / strides[i])

# strides=(1, 1) by default.

# input_shape=(batchSize, height, width, channels)
# Specifying the batchSize here in input_shape won't have any effect. The
# recommended place to specify the batchSize is the model.fit() method.
model.add(Conv2D(filters=16, kernel_size=(3,3), strides=(1, 1), padding='valid', activation='relu', input_shape=(28

# For MaxPooling2D():
```

```
# pool_size=(2, 2) by default
# strides=(2, 2) by default.
# output_shape = math.floor((input_shape - pool_size) / strides) + 1
model.add(MaxPooling2D((2,2)))

model.add(Conv2D(filters=16, kernel_size=(3,3), strides=(1, 1), padding='valid', activation='relu'))

model.add(MaxPooling2D((2,2)))

# Flatten layer in Keras converts a multi-dimensional array into a 1D array.
model.add(Flatten())

model.add(Dense(units=16, activation='relu'))

model.add(Dense(units=16, activation='relu'))

model.add(Dense(units=10, activation='softmax'))

# Model compilation

# keras.metrics.SparseCategoricalAccuracy(): Calculates how often predictions
# match integer labels.
# When loss='sparse_categorical_crossentropy', 'accuracy' metric works same
# as keras.metrics.SparseCategoricalAccuracy()
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

In [18]:

```
# What does None in the Output Shape column of model.summary() mean?

# The first dimension in Output Shape tuple is the batch size. None usually
# means that the size of that dimension is not fixed and may vary based on the
# input data.

# https://stackoverflow.com/questions/47240348/what-is-the-meaning-of-the-none-in-model-summary-of-keras

# Output shape of the convolution layer = (n-k+2p)/s+1
# Number of paramaters associated with the convolution layer = (k*k+1) * #filters

# Output shape of the max-pooling layer = (n-k)/s+1

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 16)        160

 max_pooling2d (MaxPooling2  (None, 13, 13, 16)        0
 D)

 conv2d_1 (Conv2D)           (None, 11, 11, 16)        2320

 max_pooling2d_1 (MaxPoolin  (None, 5, 5, 16)          0
 g2D)

 flatten (Flatten)           (None, 400)               0

 dense (Dense)               (None, 16)                6416

 dense_1 (Dense)             (None, 16)                272

 dense_2 (Dense)             (None, 10)                170

=================================================================
Total params: 9338 (36.48 KB)
Trainable params: 9338 (36.48 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [19]:
```
history = model.fit(x_train, y_train, epochs=10, validation_split=0.2)
```

```
Epoch 1/10
1500/1500 [==============================] - 38s 23ms/step - loss: 0.3685 - accuracy: 0.8830 - val_loss: 0.1363 - va
l_accuracy: 0.9592
Epoch 2/10
1500/1500 [==============================] - 26s 18ms/step - loss: 0.1151 - accuracy: 0.9646 - val_loss: 0.0934 - va
l_accuracy: 0.9730
Epoch 3/10
1500/1500 [==============================] - 28s 18ms/step - loss: 0.0847 - accuracy: 0.9741 - val_loss: 0.0769 - va
l_accuracy: 0.9778
Epoch 4/10
1500/1500 [==============================] - 26s 18ms/step - loss: 0.0671 - accuracy: 0.9794 - val_loss: 0.0718 - va
l_accuracy: 0.9799
Epoch 5/10
1500/1500 [==============================] - 26s 17ms/step - loss: 0.0562 - accuracy: 0.9829 - val_loss: 0.0619 - va
l_accuracy: 0.9822
Epoch 6/10
1500/1500 [==============================] - 27s 18ms/step - loss: 0.0488 - accuracy: 0.9848 - val_loss: 0.0614 - va
l_accuracy: 0.9828
Epoch 7/10
1500/1500 [==============================] - 27s 18ms/step - loss: 0.0430 - accuracy: 0.9864 - val_loss: 0.0611 - va
l_accuracy: 0.9834
Epoch 8/10
1500/1500 [==============================] - 27s 18ms/step - loss: 0.0386 - accuracy: 0.9876 - val_loss: 0.0509 - va
l_accuracy: 0.9858
Epoch 9/10
1500/1500 [==============================] - 37s 25ms/step - loss: 0.0334 - accuracy: 0.9895 - val_loss: 0.0474 - va
l_accuracy: 0.9861
Epoch 10/10
1500/1500 [==============================] - 28s 19ms/step - loss: 0.0306 - accuracy: 0.9902 - val_loss: 0.0521 - va
l_accuracy: 0.9852
```

Evaluating the model:

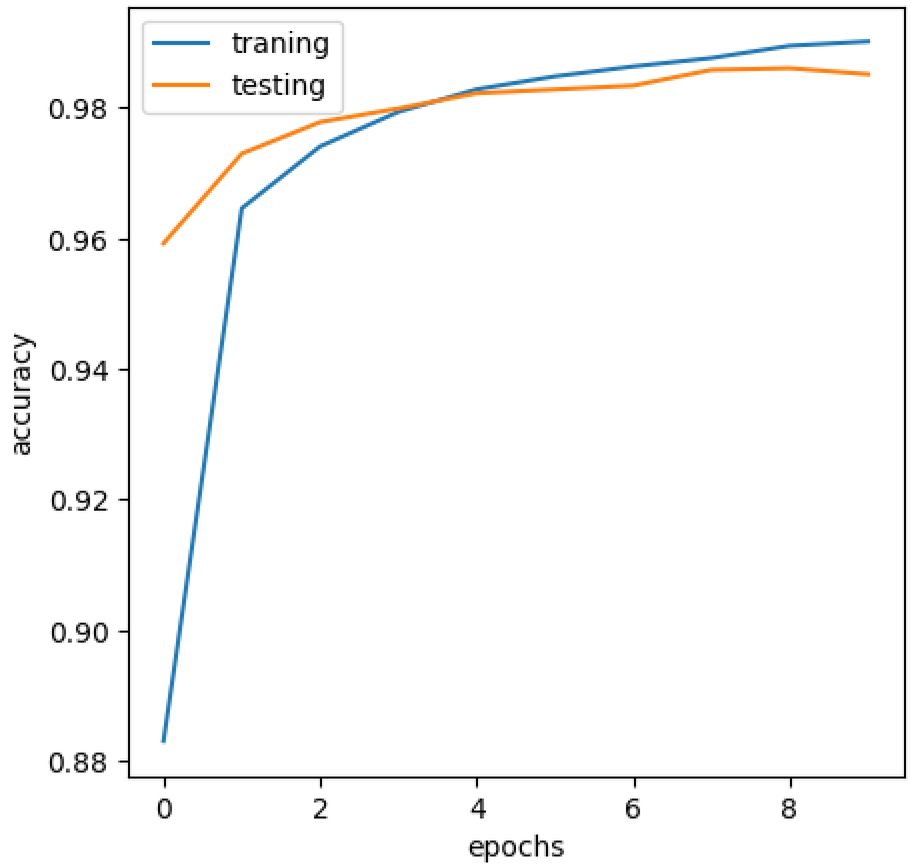
In [20]:

```python
# The model.evaluate(x_test, y_test) function uses the trained neural network to
# make predictions on the input data x_test and then compares those predictions
# with the true labels y_test to calculate the loss and any specified metrics.

model.evaluate(x_test, y_test)
```

```
313/313 [==============================] - 3s 10ms/step - loss: 0.0432 - accuracy: 0.9862
```

Out[20]: [0.04315326362848282, 0.9861999750137329]

In [21]:
```
plt.figure(figsize=(5, 5))

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.ylabel('accuracy')
plt.xlabel('epochs')
plt.legend(['traning', 'testing'])

plt.show()
```



In [22]:
```
y_pred = model.predict(x_test)
```

```
313/313 [==============================] - 2s 5ms/step
```

In [23]:
```
y_pred.shape
```

Out[23]:
```
(10000, 10)
```

In [24]:
```
# First 10 data points predictions.
y_pred_class = [np.argmax(element) for element in y_pred]
y_pred_class[:10]
```

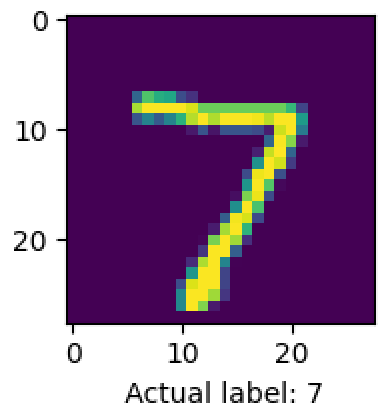
Out[24]:
```
[7, 2, 1, 0, 4, 1, 4, 9, 5, 9]
```

In [25]:
```
y_test[:10]
```

Out[25]:
```
array([7, 2, 1, 0, 4, 1, 4, 9, 5, 9], dtype=uint8)
```

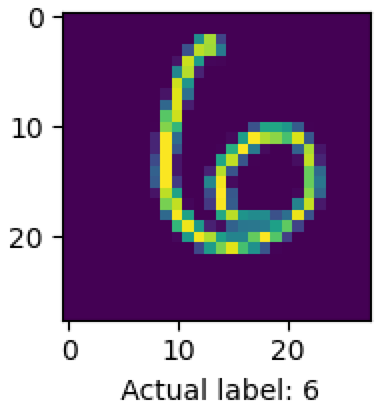
In [26]:
```
def act_to_pred(val):
  plt.figure(figsize=(20, 2))
  plt.imshow(x_test[val, :])
  plt.xlabel(f' Actual label: {y_test[val]}')
  plt.show()
  print('The predicted image label is:',y_pred_class[val])
```

In [27]:
```
act_to_pred(0)
```



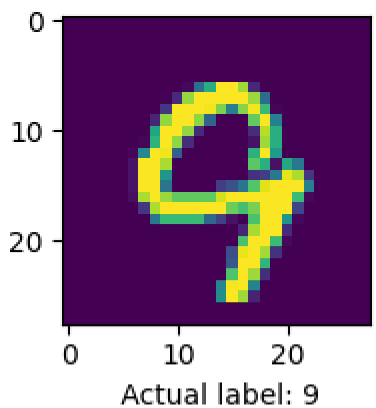

```
The predicted image label is: 7
```

In [28]:
```
act_to_pred(100)
```




```
The predicted image label is: 6
```

In [29]:
```
act_to_pred(5784)
```




```
The predicted image label is: 9
```

Conclusion:

The CNN based model to predict the hand written digit from image has a **train accuracy of 99.02%**, **validation accuracy of 98.52%** and **test accuracy of 98.62%**.