In [3]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from keras.datasets import mnist
from keras.utils import np_utils
```

In [2]:
```
#import tensorflow
#tensorflow.__version__
```

In [4]:
```
# Load the dataset
(x_train,y_train),(x_test,y_test)=mnist.load_data()
```

In [5]:
```
print(x_train.shape,y_train.shape)
```

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

In [6]:
```
print(x_test.shape,y_test.shape)
```

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

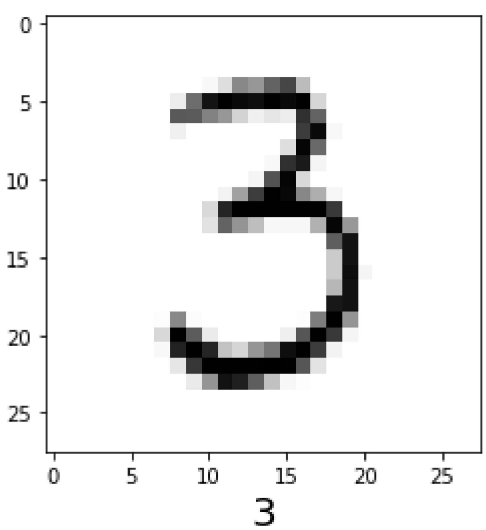
In [7]:
```
x_train[0]
```

Out[7]:
```
array([[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   3,
         18,  18,  18, 126, 136, 175,  26, 166, 255, 247, 127,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,  30,  36,  94, 154, 170,
        253, 253, 253, 253, 253, 225, 172, 253, 242, 195,  64,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,  49, 238, 253, 253, 253, 253,
        253, 253, 253, 253, 251,  93,  82,  82,  56,  39,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,  18, 219, 253, 253, 253, 253,
        253, 198, 182, 247, 241,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,  80, 156, 107, 253, 253,
        205,  11,   0,  43, 154,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,  14,   1, 154, 253,
         90,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 139, 253,
        190,   2,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  11, 190,
        253,  70,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  35,
        241, 225, 160, 108,   1,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         81, 240, 253, 253, 119,  25,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,  45, 186, 253, 253, 150,  27,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,  16,  93, 252, 253, 187,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0, 249, 253, 249,  64,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,  46, 130, 183, 253, 253, 207,   2,   0,   0,   0,   0,   0,
          0,   0],
```

```
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  39,
 148, 229, 253, 253, 253, 250, 182,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  24, 114, 221,
 253, 253, 253, 253, 201,  78,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,  23,  66, 213, 253, 253,
 253, 253, 198,  81,   2,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,  18, 171, 219, 253, 253, 253, 253,
 195,  80,   9,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,  55, 172, 226, 253, 253, 253, 253, 244, 133,
  11,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0, 136, 253, 253, 253, 212, 135, 132,  16,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0]], dtype=uint8)
```

In [8]:

```
for i in range(50,61):
    plt.imshow(x_train[i],cmap='Greys')
    plt.xlabel(y_train[i],fontsize=20)
    plt.show()
```



In [8]:
```
# We have to convert 28X28 to 28X28X1
```

In [9]:
```
x_train = x_train.reshape(x_train.shape[0],28,28,1)
```

In [ ]:
```
# reshape
"""
28 and 28 represent the desired height and width dimensions of the images.

1 represents the number of channels in the image. In this case, it's set to 1 b

By reshaping the data using x_train.reshape(x_train.shape[0], 28, 28, 1),
you are converting the original 2D images of size 28x28 into a 4D tensor.
The first dimension represents the number of samples, the second and third
dimensions represent the height and width of each image, and the fourth
dimension represents the number of channels.
"""
```

In [10]:
```
x_train.shape
```

Out[10]:
```
(60000, 28, 28, 1)
```

In [11]:
```
x_test = x_test.reshape(x_test.shape[0],28,28,1)
```

In [12]:
```
x_test.shape
```

Out[12]:
```
(10000, 28, 28, 1)
```

In [13]:
```
# Normalise the image
```

In [13]:
```
x_train = x_train/255
x_test = x_test/255
print(x_train[0])
```

```
[[[0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
  [0.        ]
```

In [14]:
```
# One hot encoding on y
y_test
```

Out[14]:
```
array([7, 2, 1, ..., 4, 5, 6], dtype=uint8)
```

In [16]:
```
y_train
```

Out[16]:
```
array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)
```

In [17]:
```
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
```

In [18]:
```
y_train[0]
```

Out[18]:
```
array([0., 0., 0., 0., 0., 1., 0., 0., 0., 0.], dtype=float32)
```

In [19]:
```
pd.set_option('display.max_columns',None)
y_train
```

Out[19]:
```
array([[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., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 1., 0.]], dtype=float32)
```

In [20]:
```
y_train[0]
```

Out[20]:
```
array([0., 0., 0., 0., 0., 1., 0., 0., 0., 0.], dtype=float32)
```

CNN model

In [21]:
```
from keras.models import Sequential
from keras.layers import Dense,Flatten,Conv2D,MaxPooling2D
```

In [22]:
```
model = Sequential()
```

In [23]:
```
model.add(Conv2D(40,(4,4),input_shape=(28,28,1),padding='same',strides=(2,2)))
model.add(MaxPooling2D(pool_size=(2,2)))          # 7x7x40
model.add(Conv2D(100,(4,4)))                      # 4x4x100
model.add(MaxPooling2D(pool_size=(2,2)))          # 2x2x100
model.add(Flatten())
model.add(Dense(200,activation='relu'))
model.add(Dense(50,activation='relu'))
model.add(Dense(10,activation='softmax'))
# (n + 2p -f)/s + 1
```

In [ ]:
```
# model.add(Conv2D(40,(4,4),input_shape=(28,28,1),padding='same',strides=(2,2))
""" # Conv2D: This is the convolutional layer in Keras.
# 40: It specifies the number of filters or output channels in the layer.
#(4, 4): It defines the size of the filters or the kernel.
#input_shape=(28, 28, 1): It specifies the shape of the input data. In this cas
#padding='same': It adds padding to the input data to ensure that the output
feature maps have the same spatial dimensions as the input.
Padding helps retain more information from the edges of the images during
convolution.
#strides=(2, 2): It specifies the stride or step size of the filter during conv
By adding this layer to the model, you are introducing a convolutional
operation that convolves the filters over the input images.
The filters extract features from the input data, capturing patterns and
spatial information.

The output shape of this layer depends on the padding, strides,
and input shape. Since padding='same' is used, the output feature maps
will have the same spatial dimensions as the input.
The number of output channels is set to 40, as specified in the layer
configuration.
"""
```

In [24]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 14, 14, 40)        680       
                                                                 
 max_pooling2d (MaxPooling2D  (None, 7, 7, 40)         0         
 )                                                               
                                                                 
 conv2d_1 (Conv2D)           (None, 4, 4, 100)         64100     
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 2, 2, 100)        0         
 2D)                                                             
                                                                 
 flatten (Flatten)           (None, 400)               0         
                                                                 
 dense (Dense)               (None, 200)               80200     
                                                                 
 dense_1 (Dense)             (None, 50)                10050     
                                                                 
 dense_2 (Dense)             (None, 10)                510       
                                                                 
=================================================================
Total params: 155,540
Trainable params: 155,540
Non-trainable params: 0
_________________________________________________________________
```

In [26]:
```
# Compile the model
```

In [25]:
```
model.compile(loss='categorical_crossentropy', metrics=['Accuracy'],optimizer='
```

In [26]:
```
model.fit(x_train,y_train,batch_size=1000, epochs=10)
```

```
Epoch 1/10
60/60 [==============================] - 96s 1s/step - loss: 0.8069 - Accurac
y: 0.7826
Epoch 2/10
60/60 [==============================] - 56s 919ms/step - loss: 0.1724 - Accu
racy: 0.9489
Epoch 3/10
60/60 [==============================] - 51s 851ms/step - loss: 0.1125 - Accu
racy: 0.9669
Epoch 4/10
60/60 [==============================] - 63s 1s/step - loss: 0.0840 - Accurac
y: 0.9758
Epoch 5/10
60/60 [==============================] - 87s 1s/step - loss: 0.0669 - Accurac
y: 0.9803
Epoch 6/10
60/60 [==============================] - 69s 1s/step - loss: 0.0602 - Accurac
y: 0.9819
Epoch 7/10
60/60 [==============================] - 56s 935ms/step - loss: 0.0506 - Accu
racy: 0.9849
Epoch 8/10
60/60 [==============================] - 70s 1s/step - loss: 0.0443 - Accurac
y: 0.9864
Epoch 9/10
60/60 [==============================] - 73s 1s/step - loss: 0.0366 - Accurac
y: 0.9890
Epoch 10/10
60/60 [==============================] - 71s 1s/step - loss: 0.0332 - Accurac
y: 0.9902
```

Out[26]:
```
<keras.callbacks.History at 0x222d2dcf130>
```

In [ ]:
```
# model.fit(x_train,y_train,batch_size=1000, epochs=10)
"""
x_train: This parameter represents the input training data. It consists of a co

y_train: This parameter represents the corresponding labels or target values fo

batch_size: This parameter determines the number of samples processed in each i

epochs: This parameter specifies the number of times the entire training datase

During the training process, the model will iterate over the training data in b
"""
```

In [2]:
```
# SAving and Loading the model
model_json = model.to_json()
```

In [ ]:
```
model_json
```

In [30]:
```
with open("model.json", "w") as json_file:
    json_file.write(model_json)
```

In [31]:
```
model.save_weights("model_mnist.h5")
```

In [32]:
```
# Loading the model
```

In [33]:
```
json_file = open('model.json', 'r')
```

In [34]:
```
loaded_model_json = json_file.read()
```

In [35]:
```
from keras.models import model_from_json
loaded_model = model_from_json(loaded_model_json)
```

In [36]:
```
loaded_model.load_weights("model_mnist.h5")
```

In [37]:
```
# Evaluate the model
```

In [39]:
```
#loaded_model.evaluate(x_test,y_test)
```

In [40]:
```
y_test.shape
```

Out[40]:
```
(10000, 10)
```

In [42]:
```
y_pred = loaded_model.predict(x_test)
```

In [2]:
```
y_pred
```

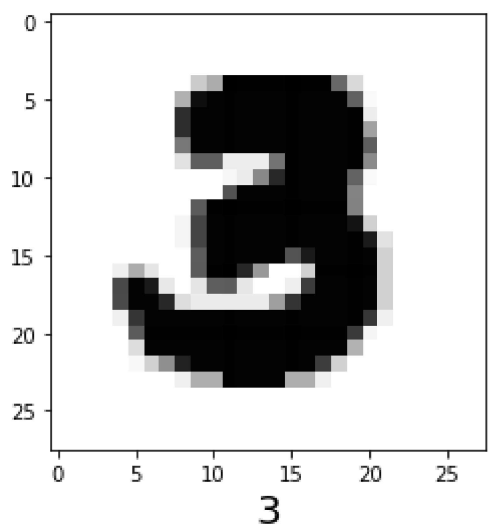
In [44]:
```
y_pred_labels = [np.argmax(i) for i in y_pred]
```

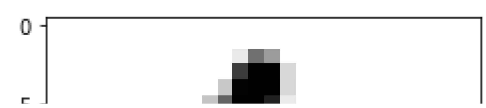
In [45]:
```
y_pred_labels
```

Out[45]:
```
[7,
 2,
 1,
 0,
 4,
 1,
 4,
 9,
 5,
 9,
 0,
 6,
 9,
 0,
 1,
 5,
 9,
 7,
 3,
```

In [46]:
```
y_test = [np.argmax(i) for i in y_test]
```

In [47]:
```
for i in range(200,250):
    plt.imshow(x_test[i], cmap='Greys')
    plt.xlabel(y_pred_labels[i],fontsize=20)
    plt.show()
```



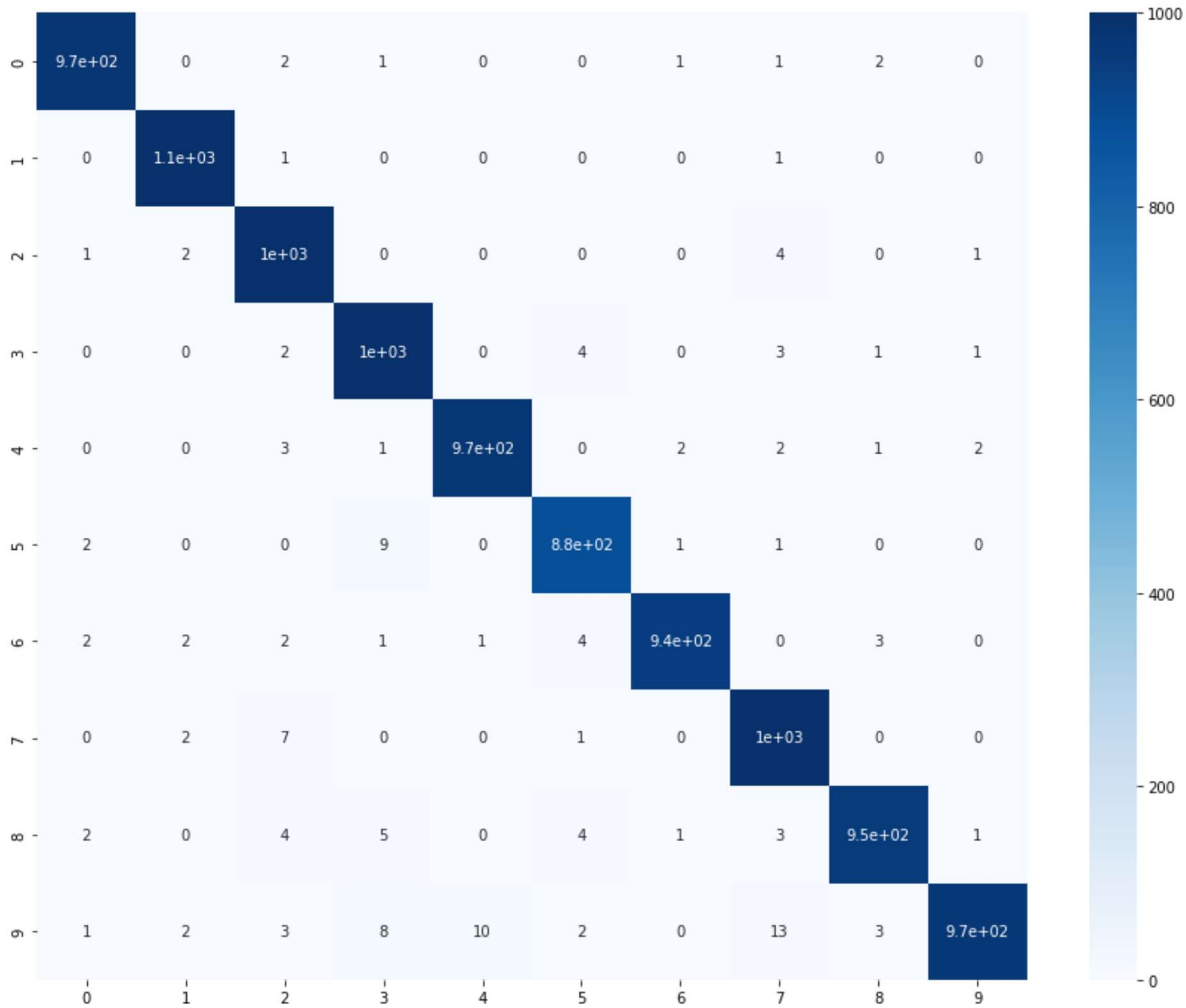


In [48]:
```
from sklearn.metrics import confusion_matrix,accuracy_score
```

In [49]:
```
cm = confusion_matrix(y_test,y_pred_labels)
```

In [52]:
```
plt.figure(figsize=(15,12))
sns.heatmap(cm,annot=True,vmax=1000,vmin=0,cmap='Blues')
```

Out[52]: <AxesSubplot:>



In [53]:
```
accuracy_score(y_test,y_pred_labels)
```

Out[53]: 0.9861

In [ ]: