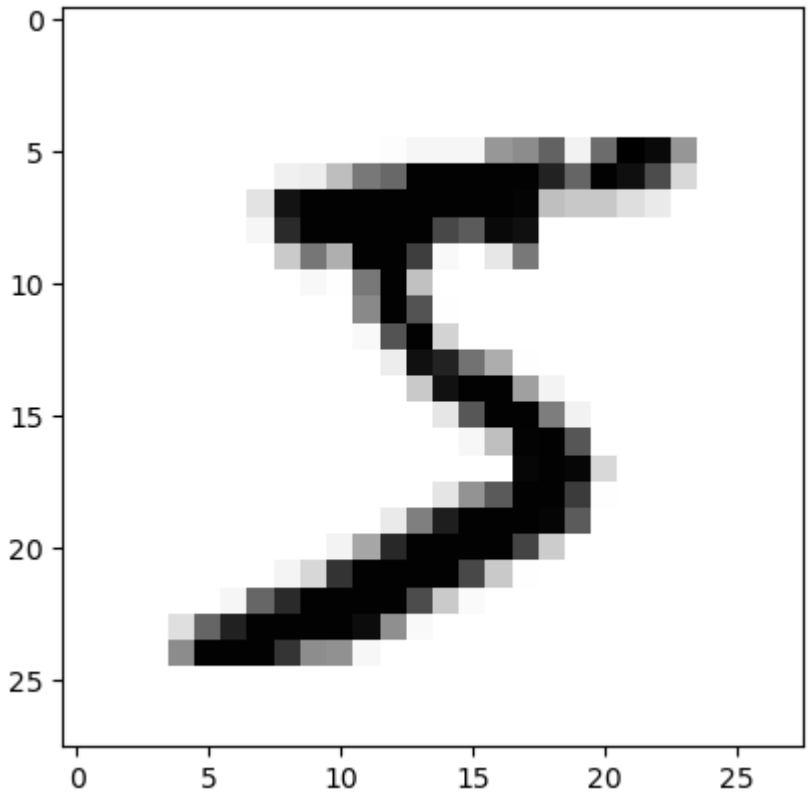
이미지 인식의 꽃, 컨볼루션 신경망(CNN)

1. 이미지를 인식하는 원리

In [3]:
```
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical

import matplotlib.pyplot as plt
import sys

# MNIST 데이터셋을 불러와 학습셋과 테스트셋으로 저장합니다.
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# 학습셋과 테스트셋이 각각 몇 개의 이미지로 되어 있는지 확인합니다.
print("학습셋 이미지 수 : %d 개" % (X_train.shape[0]))
print("테스트셋 이미지 수 : %d 개" % (X_test.shape[0]))
```

```
학습셋 이미지 수 : 60000 개
테스트셋 이미지 수 : 10000 개
```

In [4]:
```
# 첫 번째 이미지를 확인해 봅시다.
plt.imshow(X_train[0], cmap='Greys')
plt.show()
```



In [5]:
```
# 이미지가 인식되는 원리를 알아봅시다.
for x in X_train[0]:
    for i in x:
        sys.stdout.write("%-3s" % i)
    sys.stdout.write('\n')
```

```
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 12613617526 1662552471270  0  0  0
0  0  0  0  0  0  0  0  30 36 94 154170253253253253253225172253242195640  0  0  0
0  0  0  0  0  0  0  49 238253253253253253253253253193 82 82 56 39 0  0  0  0  0
0  0  0  0  0  0  0  18 219253253253253253198182247241 0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  80 156107253253205110 0  43 154 0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  14 1  154253900 0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  13925319020 0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  11 190253700 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 2412251601081  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  81 240253253119250 0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  45 186253253150270 0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  16 93 25225318700  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  249253249640 0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  46 13018325325320720  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  39 1482292532532532501820  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  24 114221253253253253201780 0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  23 66 213253253253253198812 0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  18 171219253253253253195800 9  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  55 172226253253253253244133110 0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  136253253253212135132160 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
```

In [6]:

```
# 차원 변환 과정을 실습해 봅니다.
X_train = X_train.reshape(X_train.shape[0], 784)
X_train = X_train.astype('float64')
X_train = X_train / 255

X_test = X_test.reshape(X_test.shape[0], 784).astype('float64') / 255

# 클래스 값을 확인해 봅니다.
print("class : %d " % (y_train[0]))

# 바이너리화 과정을 실습해 봅니다.
y_train = to_categorical(y_train, 10)
```

```
y_test = to_categorical(y_test, 10)

print(y_train[0])
```

```
class : 5
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

2. 딥러닝 기본 프레임 만들기

In [8]:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.callbacks import ModelCheckpoint,EarlyStopping
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical

import matplotlib.pyplot as plt
import numpy as np
import os

# MNIST 데이터를 불러옵니다.
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# 차원 변환 후, 테스트셋과 학습셋으로 나누어 줍니다.
X_train = X_train.reshape(X_train.shape[0], 784).astype('float32') / 255
X_test = X_test.reshape(X_test.shape[0], 784).astype('float32') / 255

y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# 모델 구조를 설정합니다.
model = Sequential()
model.add(Dense(512, input_dim=784, activation='relu'))
model.add(Dense(10, activation='softmax'))
model.summary()
```

```
C:\Users\user\AppData\Roaming\Python\Python312\site-packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an `in
put_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first la
yer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 512)	401,920
dense_1 (Dense)	(None, 10)	5,130

Total params: 407,050 (1.55 MB)

Trainable params: 407,050 (1.55 MB)

Non-trainable params: 0 (0.00 B)

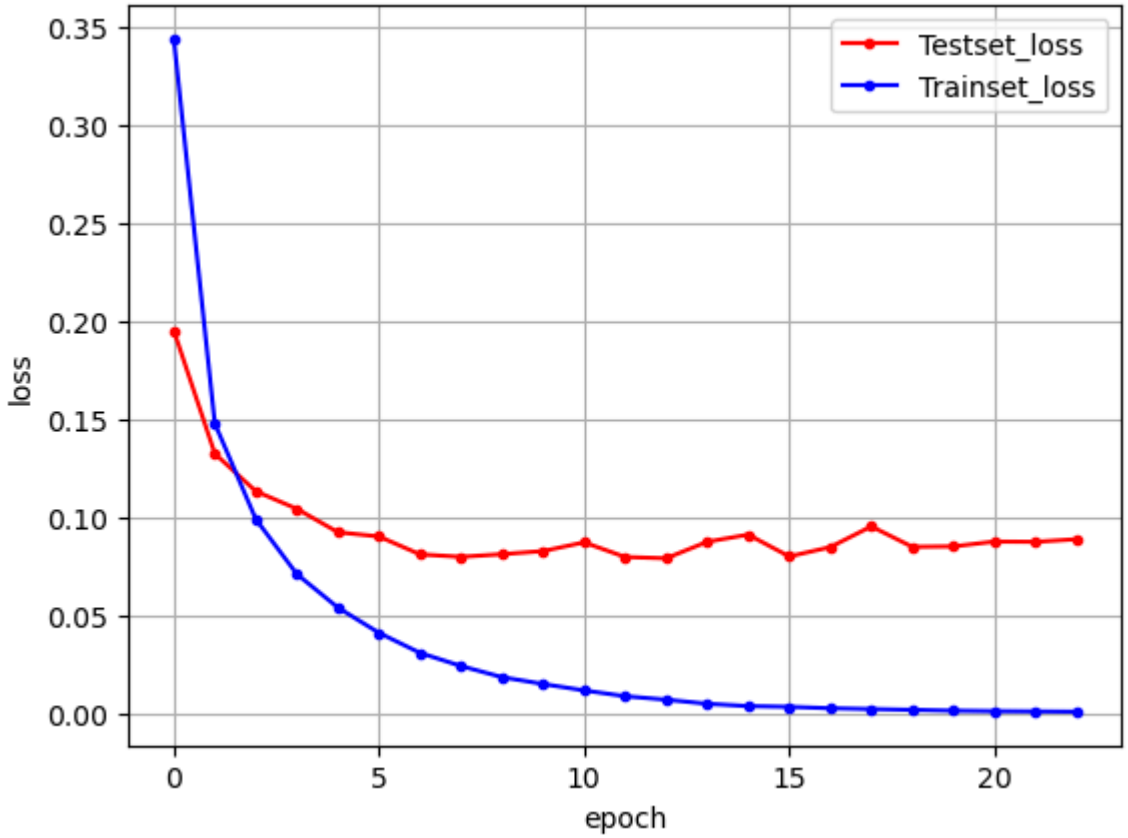
In [9]:
```
# 모델 실행 환경을 설정합니다.
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# 모델 최적화를 위한 설정 구간입니다.
modelpath="./MNIST_MLP.keras"
checkpointer = ModelCheckpoint(filepath=modelpath, monitor='val_loss',
                               verbose=1, save_best_only=True)
early_stopping_callback = EarlyStopping(monitor='val_loss', patience=10)

# 모델을 실행합니다.
history = model.fit(X_train, y_train, validation_split=0.25, epochs=30,
                    batch_size=200, verbose=0,
                    callbacks=[early_stopping_callback,checkpointer])

# 테스트 정확도를 출력합니다.
print("\n Test Accuracy: %.4f" % (model.evaluate(X_test, y_test)[1]))
```

```
Epoch 1: val_loss improved from inf to 0.19521, saving model to ./MNIST_MLP.keras

Epoch 2: val_loss improved from 0.19521 to 0.13239, saving model to ./MNIST_MLP.keras

Epoch 3: val_loss improved from 0.13239 to 0.11319, saving model to ./MNIST_MLP.keras

Epoch 4: val_loss improved from 0.11319 to 0.10424, saving model to ./MNIST_MLP.keras

Epoch 5: val_loss improved from 0.10424 to 0.09230, saving model to ./MNIST_MLP.keras

Epoch 6: val_loss improved from 0.09230 to 0.09014, saving model to ./MNIST_MLP.keras

Epoch 7: val_loss improved from 0.09014 to 0.08101, saving model to ./MNIST_MLP.keras

Epoch 8: val_loss improved from 0.08101 to 0.07975, saving model to ./MNIST_MLP.keras

Epoch 9: val_loss did not improve from 0.07975

Epoch 10: val_loss did not improve from 0.07975

Epoch 11: val_loss did not improve from 0.07975

Epoch 12: val_loss improved from 0.07975 to 0.07967, saving model to ./MNIST_MLP.keras

Epoch 13: val_loss improved from 0.07967 to 0.07899, saving model to ./MNIST_MLP.keras

Epoch 14: val_loss did not improve from 0.07899

Epoch 15: val_loss did not improve from 0.07899

Epoch 16: val_loss did not improve from 0.07899

Epoch 17: val_loss did not improve from 0.07899

Epoch 18: val_loss did not improve from 0.07899

Epoch 19: val_loss did not improve from 0.07899

Epoch 20: val_loss did not improve from 0.07899

Epoch 21: val_loss did not improve from 0.07899
```

```
Epoch 22: val_loss did not improve from 0.07899

Epoch 23: val_loss did not improve from 0.07899
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9777 - loss: 0.0882

 Test Accuracy: 0.9806
```

In [10]:

```python
# 검증셋과 학습셋의 오차를 저장합니다.
y_vloss = history.history['val_loss']
y_loss = history.history['loss']

# 그래프로 표현해 봅니다.
x_len = np.arange(len(y_loss))
plt.plot(x_len, y_vloss, marker='.', c="red", label='Testset_loss')
plt.plot(x_len, y_loss, marker='.', c="blue", label='Trainset_loss')

# 그래프에 그리드를 주고 레이블을 표시해 보겠습니다.
plt.legend(loc='upper right')
plt.grid()
plt.xlabel('epoch')
plt.ylabel('loss')
plt.show()
```



5. 컨볼루션 신경망 실행하기

In [12]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D
from tensorflow.keras.callbacks import ModelCheckpoint,EarlyStopping
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical

import matplotlib.pyplot as plt
import numpy as np

# 데이터를 불러옵니다.
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

```
X_train = X_train.reshape(X_train.shape[0], 28, 28, 1).astype('float32') / 255
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1).astype('float32') / 255
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

# 컨볼루션 신경망의 설정
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), input_shape=(28, 28, 1), activation='relu'))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128,  activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))

# 모델의 실행 옵션을 설정합니다.
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

# 모델 최적화를 위한 설정 구간입니다.
modelpath="./MNIST_CNN.keras"
checkpointer = ModelCheckpoint(filepath=modelpath, monitor='val_loss', verbose=1,
                               save_best_only=True)
early_stopping_callback = EarlyStopping(monitor='val_loss', patience=10)

# 모델을 실행합니다.
history = model.fit(X_train, y_train, validation_split=0.25, epochs=30,
                    batch_size=200, verbose=0, callbacks=[early_stopping_callback,checkpointer])

# 테스트 정확도를 출력합니다.
print("\n Test Accuracy: %.4f" % (model.evaluate(X_test, y_test)[1]))
```

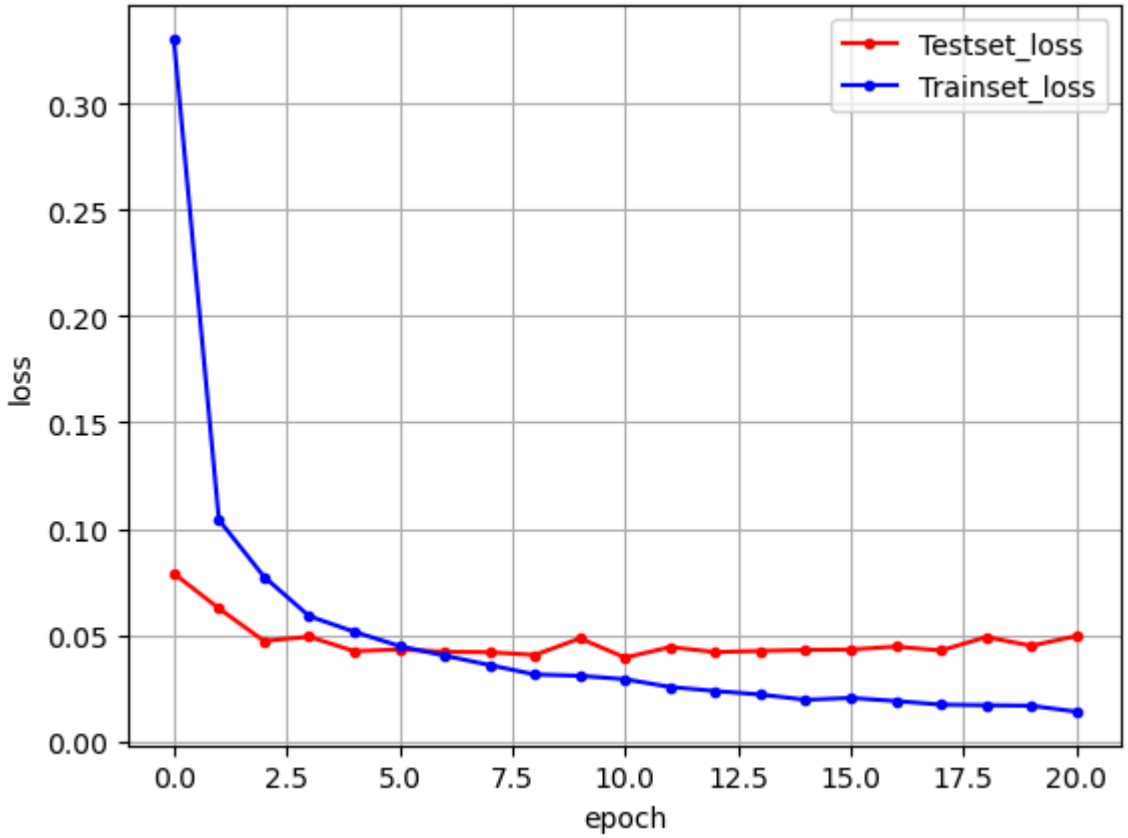
```
C:\Users\user\AppData\Roaming\Python\Python312\site-packages\keras\src\layers\convolutional\base_conv.py:107: UserWarning: Do n
ot pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object a
s the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
Epoch 1: val_loss improved from inf to 0.07898, saving model to ./MNIST_CNN.keras

Epoch 2: val_loss improved from 0.07898 to 0.06250, saving model to ./MNIST_CNN.keras

Epoch 3: val_loss improved from 0.06250 to 0.04719, saving model to ./MNIST_CNN.keras

Epoch 4: val_loss did not improve from 0.04719

Epoch 5: val_loss improved from 0.04719 to 0.04241, saving model to ./MNIST_CNN.keras

Epoch 6: val_loss did not improve from 0.04241

Epoch 7: val_loss improved from 0.04241 to 0.04210, saving model to ./MNIST_CNN.keras

Epoch 8: val_loss improved from 0.04210 to 0.04192, saving model to ./MNIST_CNN.keras

Epoch 9: val_loss improved from 0.04192 to 0.04062, saving model to ./MNIST_CNN.keras

Epoch 10: val_loss did not improve from 0.04062

Epoch 11: val_loss improved from 0.04062 to 0.03938, saving model to ./MNIST_CNN.keras

Epoch 12: val_loss did not improve from 0.03938

Epoch 13: val_loss did not improve from 0.03938

Epoch 14: val_loss did not improve from 0.03938

Epoch 15: val_loss did not improve from 0.03938

Epoch 16: val_loss did not improve from 0.03938

Epoch 17: val_loss did not improve from 0.03938

Epoch 18: val_loss did not improve from 0.03938

Epoch 19: val_loss did not improve from 0.03938

Epoch 20: val_loss did not improve from 0.03938

Epoch 21: val_loss did not improve from 0.03938
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9887 - loss: 0.0432

Test Accuracy: 0.9911
```

In [13]:

```python
# 검증셋과 학습셋의 오차를 저장합니다.
y_vloss = history.history['val_loss']
y_loss = history.history['loss']

# 그래프로 표현해 봅니다.
x_len = np.arange(len(y_loss))
plt.plot(x_len, y_vloss, marker='.', c="red", label='Testset_loss')
plt.plot(x_len, y_loss, marker='.', c="blue", label='Trainset_loss')

# 그래프에 그리드를 주고 레이블을 표시하겠습니다.
plt.legend(loc='upper right')
plt.grid()
plt.xlabel('epoch')
plt.ylabel('loss')
plt.show()
```


Testset_loss
Trainset_loss
loss
epoch
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

In [ ]: