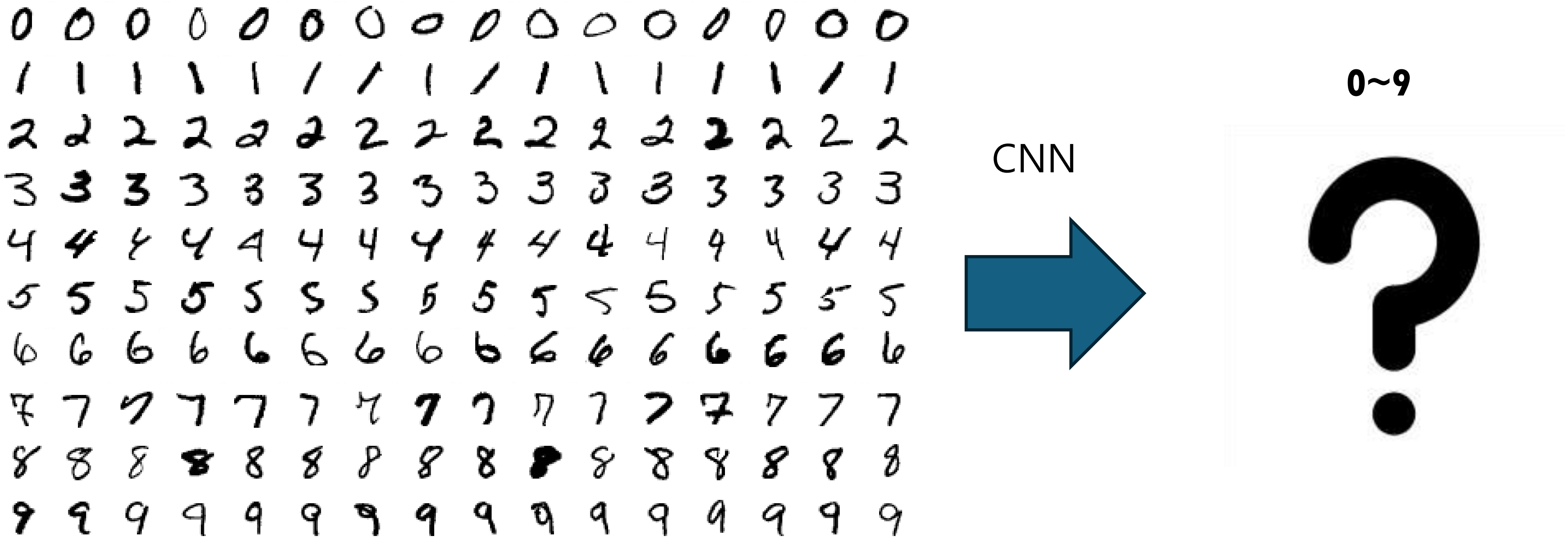
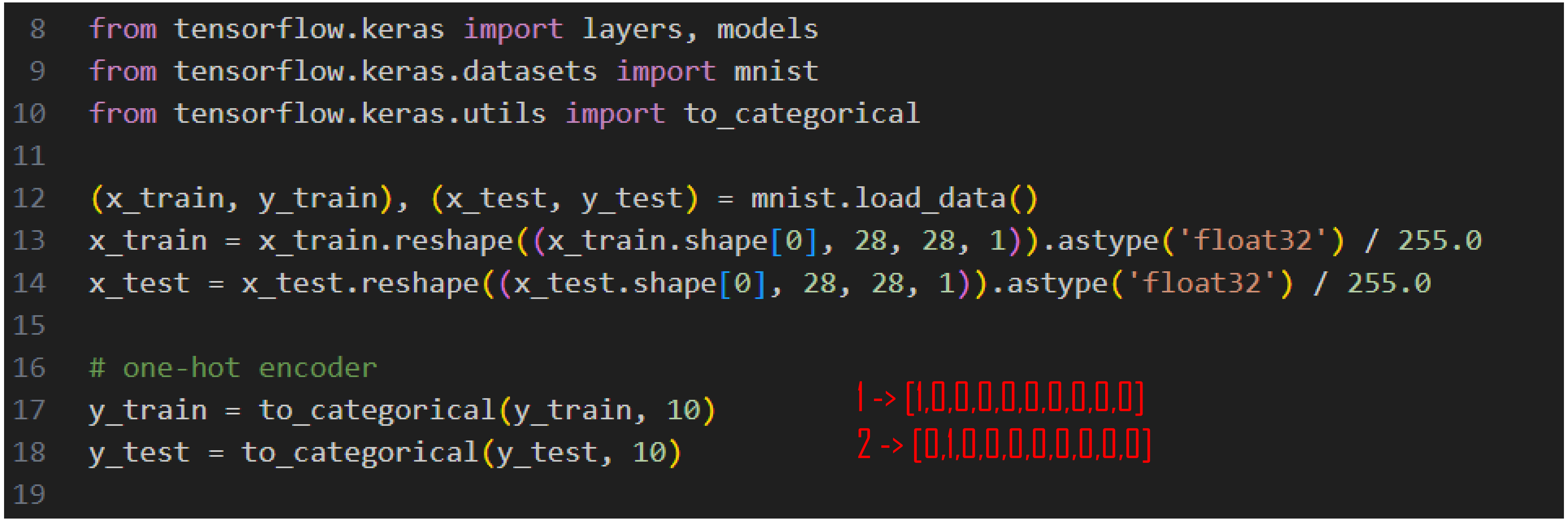
MNIST CNN 실습



CNN

0~9

MNIST CNN 실습

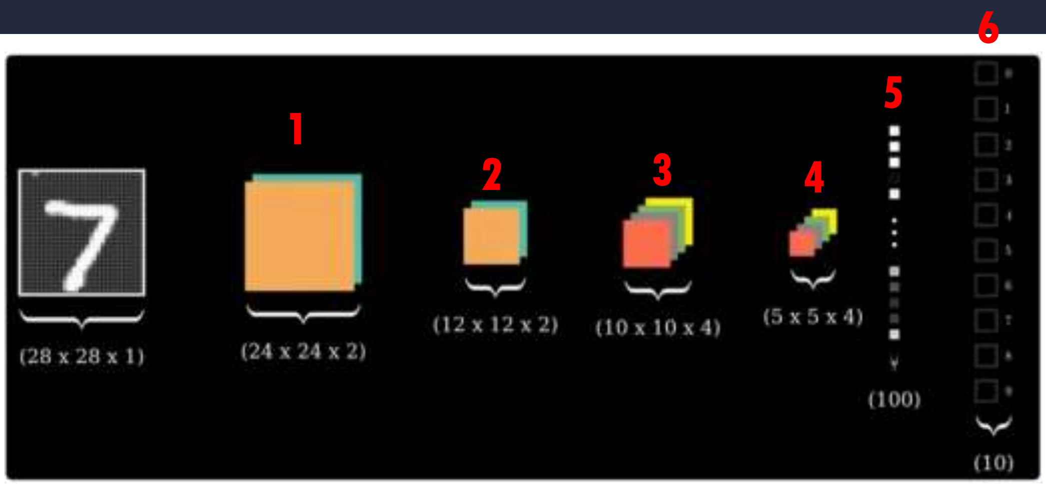


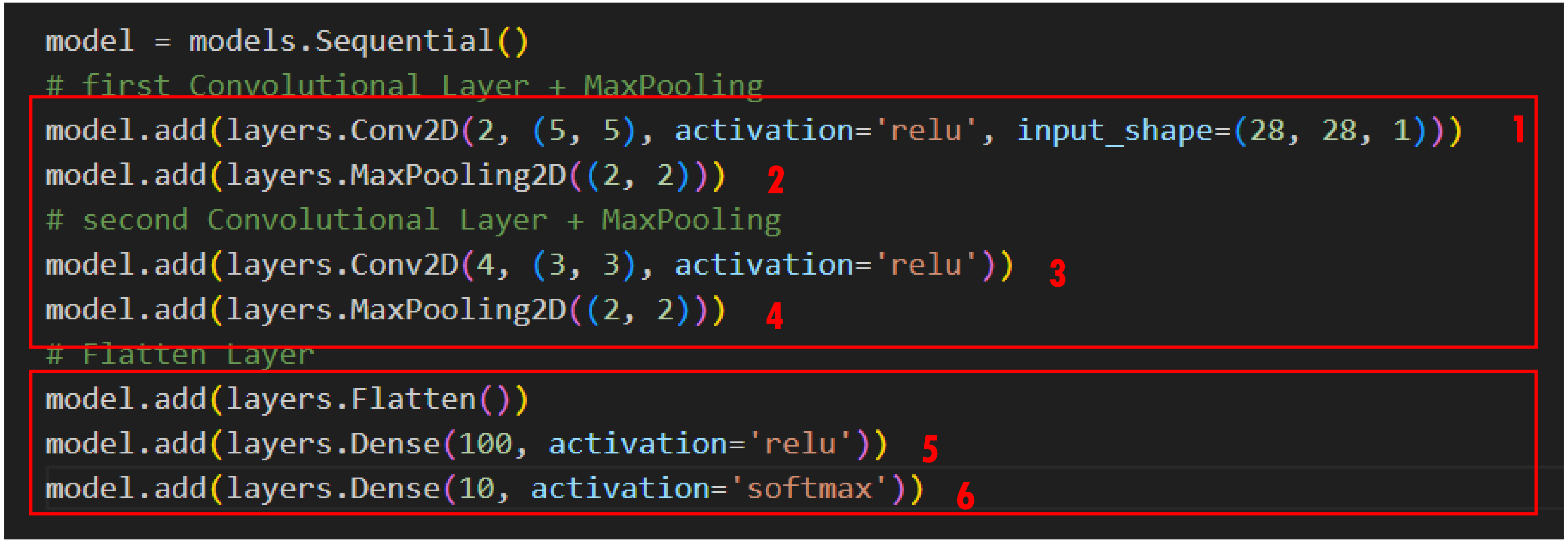
```python
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical

(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') / 255.0
x_test = x_test.reshape((x_test.shape[0], 28, 28, 1)).astype('float32') / 255.0

# one-hot encoder
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

```

1 -> [1,0,0,0,0,0,0,0,0,0]

2 -> [0,1,0,0,0,0,0,0,0,0]

MNIST CNN 실습



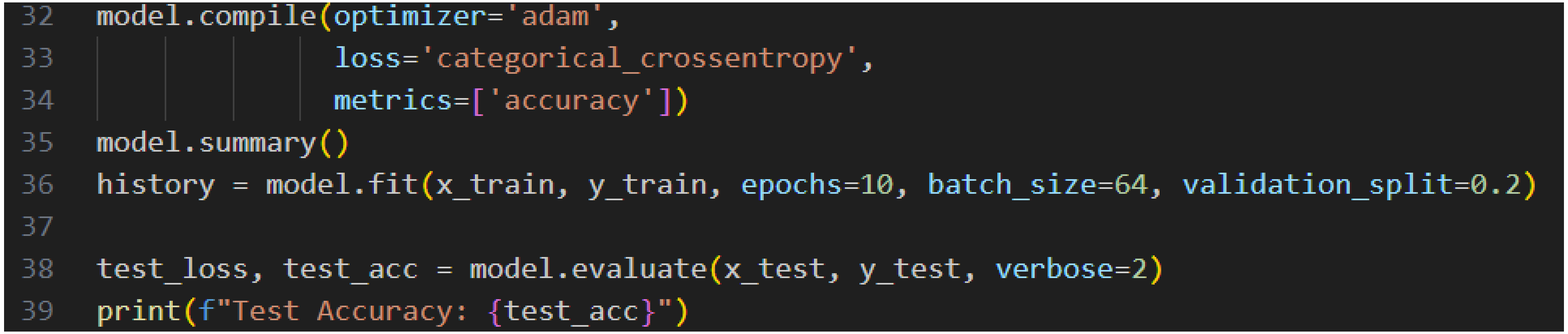


```
model = models.Sequential()
# first Convolutional Layer + MaxPooling
model.add(layers.Conv2D(2, (5, 5), activation='relu', input_shape=(28, 28, 1)))  1
model.add(layers.MaxPooling2D((2, 2)))  2
# second Convolutional Layer + MaxPooling
model.add(layers.Conv2D(4, (3, 3), activation='relu'))  3
model.add(layers.MaxPooling2D((2, 2)))  4
# Flatten Layer
model.add(layers.Flatten())
model.add(layers.Dense(100, activation='relu'))  5
model.add(layers.Dense(10, activation='softmax'))  6
```

특징 추출

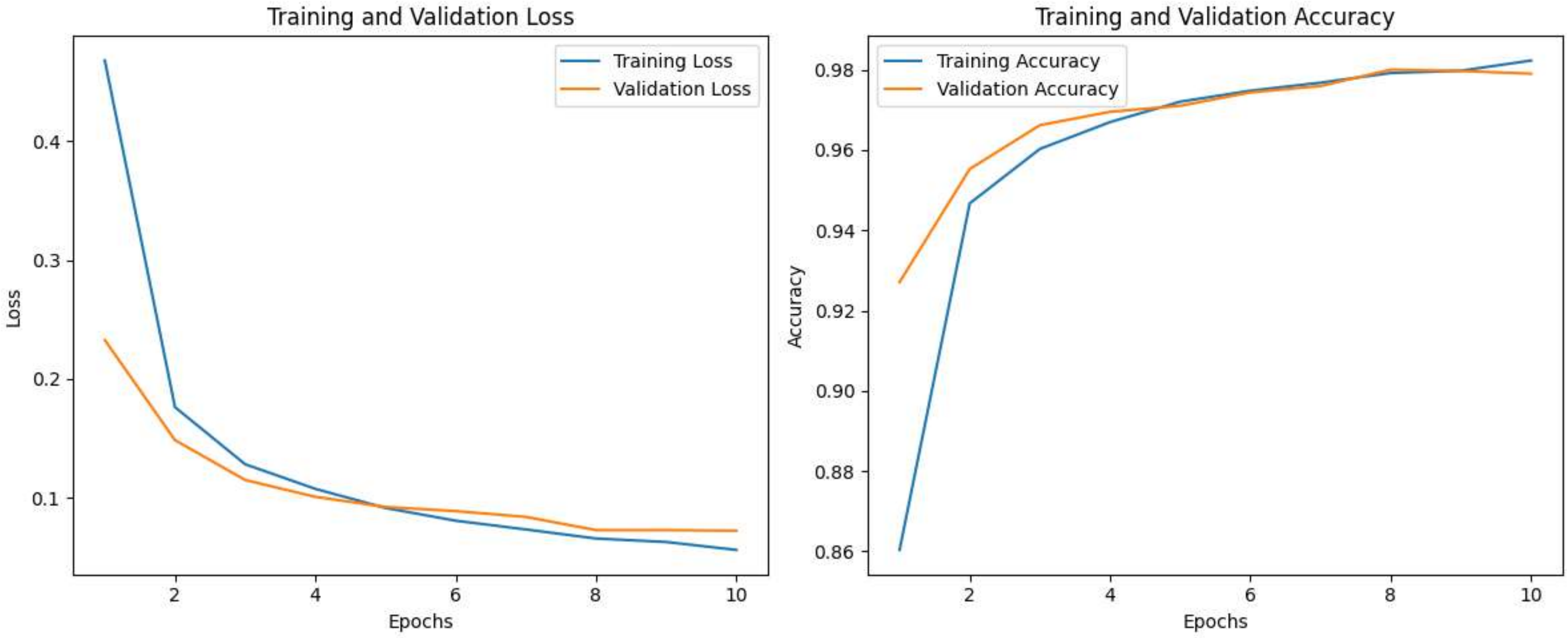
분류

MNIST CNN 실습

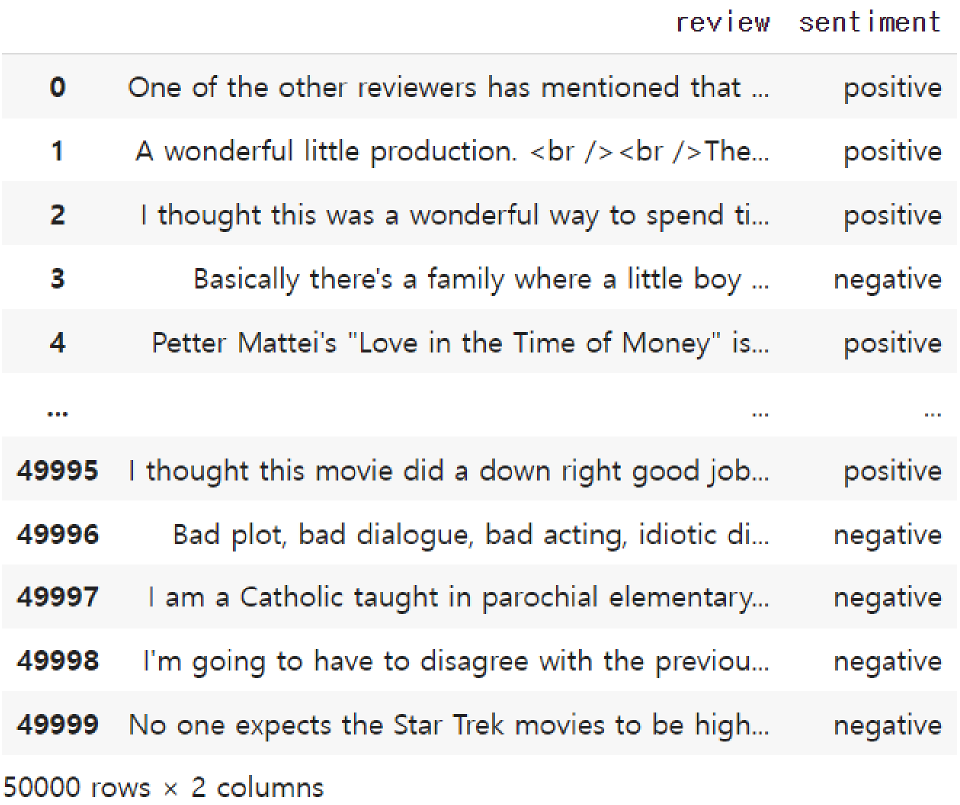


```python
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.summary()
history = model.fit(x_train, y_train, epochs=10, batch_size=64, validation_split=0.2)

test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print(f"Test Accuracy: {test_acc}")
```

결과


Training and Validation Loss
Training Loss
Validation Loss
Loss
0.4
0.3
0.2
0.1
2
4
6
8
10
Epochs
Training and Validation Accuracy
Training Accuracy
Validation Accuracy
Accuracy
0.98
0.96
0.94
0.92
0.90
0.88
0.86
2
4
6
8
10
Epochs

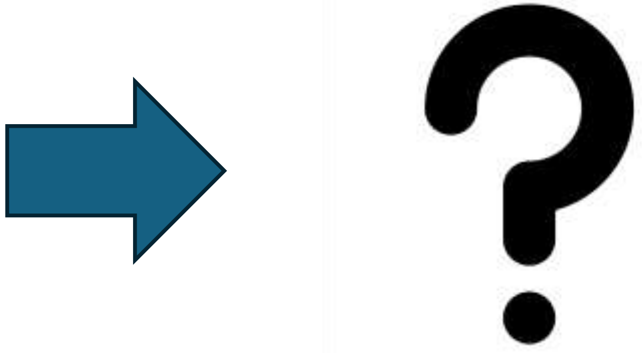
IMDB Conv 1D 실습



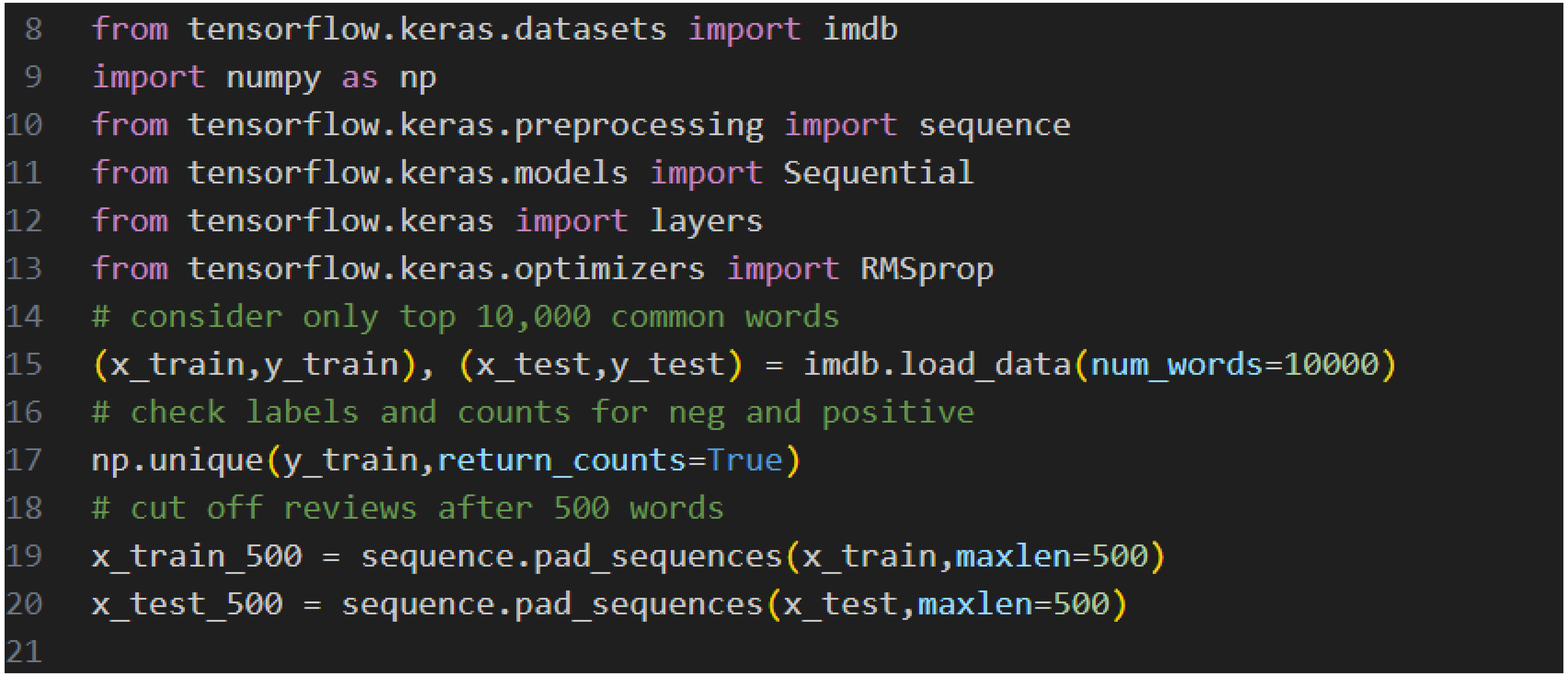
	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. <br /><br />The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive
...	...	...
49995	I thought this movie did a down right good job...	positive
49996	Bad plot, bad dialogue, bad acting, idiotic di...	negative
49997	I am a Catholic taught in parochial elementary...	negative
49998	I'm going to have to disagree with the previou...	negative
49999	No one expects the Star Trek movies to be high...	negative

50000 rows × 2 columns

긍정/부정

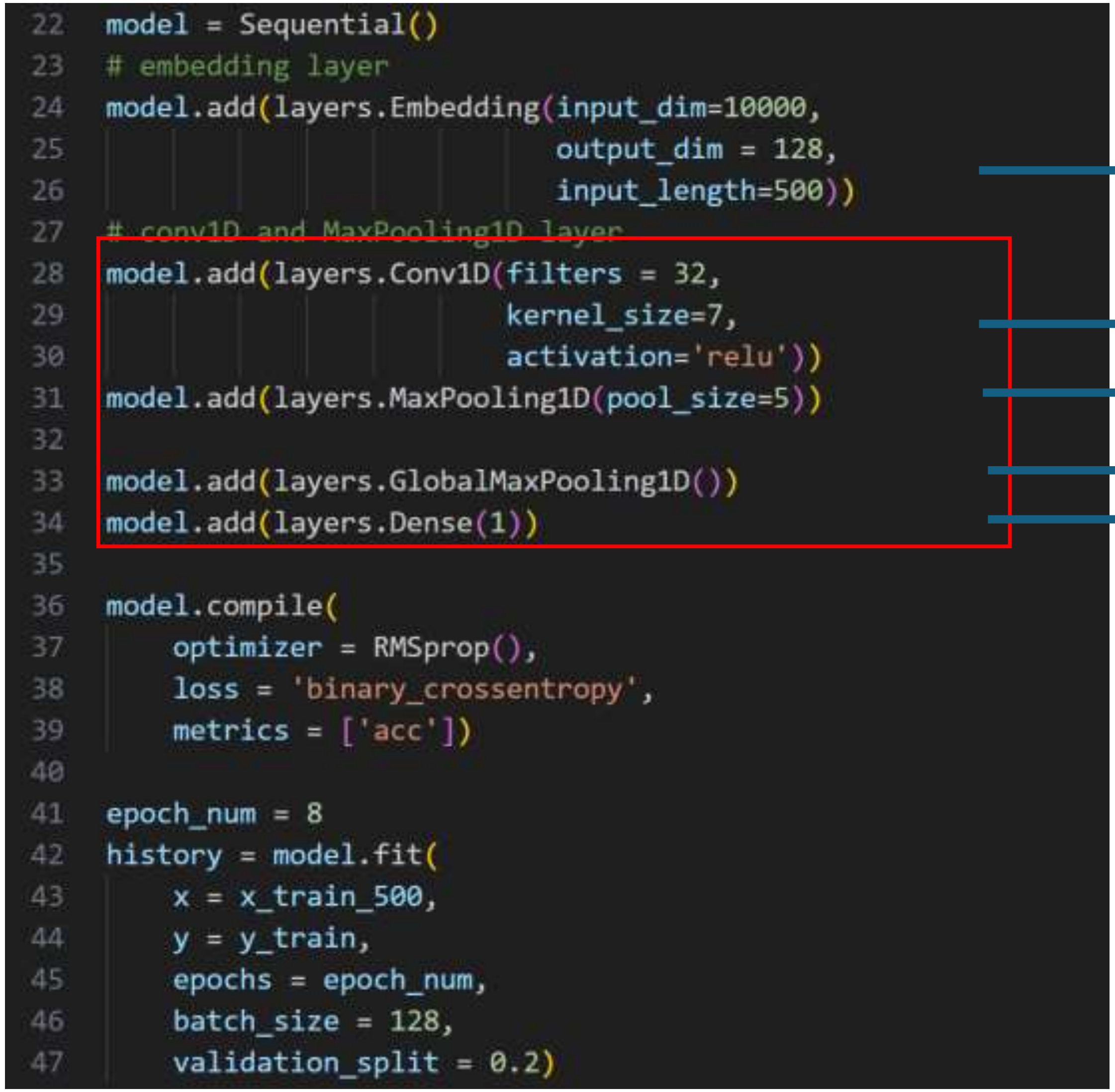


IMDB Conv 1D 실습



```python
from tensorflow.keras.datasets import imdb
import numpy as np
from tensorflow.keras.preprocessing import sequence
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers
from tensorflow.keras.optimizers import RMSprop
# consider only top 10,000 common words
(x_train,y_train), (x_test,y_test) = imdb.load_data(num_words=10000)
# check labels and counts for neg and positive
np.unique(y_train,return_counts=True)
# cut off reviews after 500 words
x_train_500 = sequence.pad_sequences(x_train,maxlen=500)
x_test_500 = sequence.pad_sequences(x_test,maxlen=500)

```

IMDB Conv 1D 실습



```
model = Sequential()
# embedding layer
model.add(layers.Embedding(input_dim=10000,
                           output_dim = 128,
                           input_length=500))
# conv1D and MaxPooling1D layer
model.add(layers.Conv1D(filters = 32,
                        kernel_size=7,
                        activation='relu'))
model.add(layers.MaxPooling1D(pool_size=5))

model.add(layers.GlobalMaxPooling1D())
model.add(layers.Dense(1))

model.compile(
    optimizer = RMSprop(),
    loss = 'binary_crossentropy',
    metrics = ['acc'])

epoch_num = 8
history = model.fit(
    x = x_train_500,
    y = y_train,
    epochs = epoch_num,
    batch_size = 128,
    validation_split = 0.2)
```

각 단어를 128차원으로 임베딩

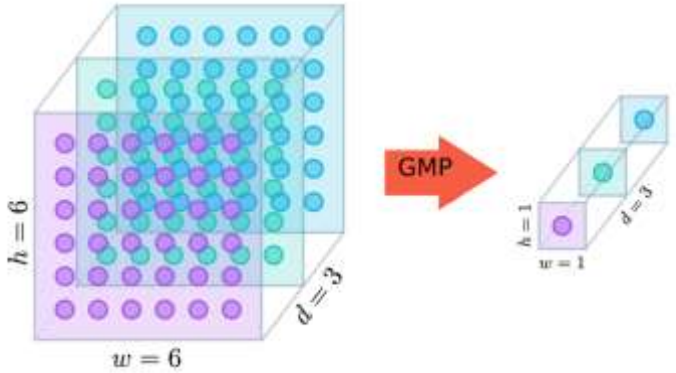
출력 크기 : (500,128)

문장에서 특징 추출 출력 크기 : (494,32)

문장에서 중요한 특징 추출 출력 크기 : (98,32)

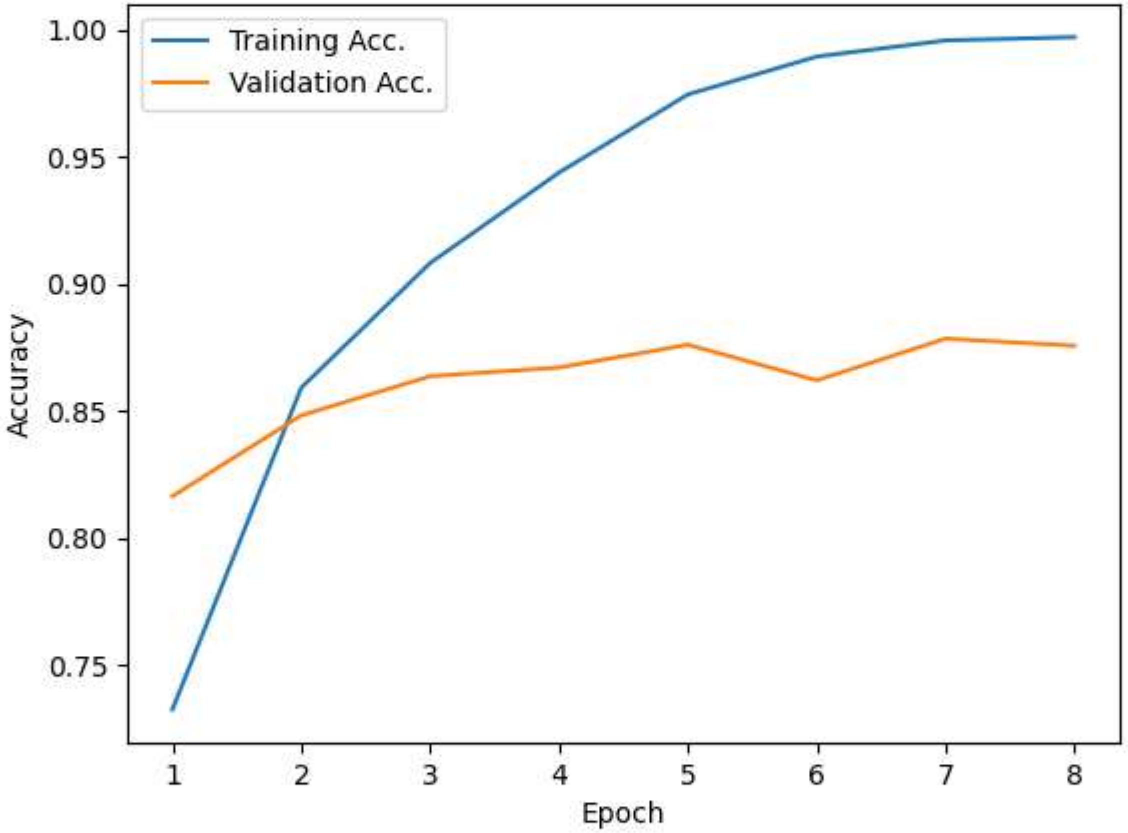
각 차원에서 가장 중요한 값(큰 값) 추출 출력 크기 : (batch_size,32)

분류 출력 크기 : (batch_size,1)

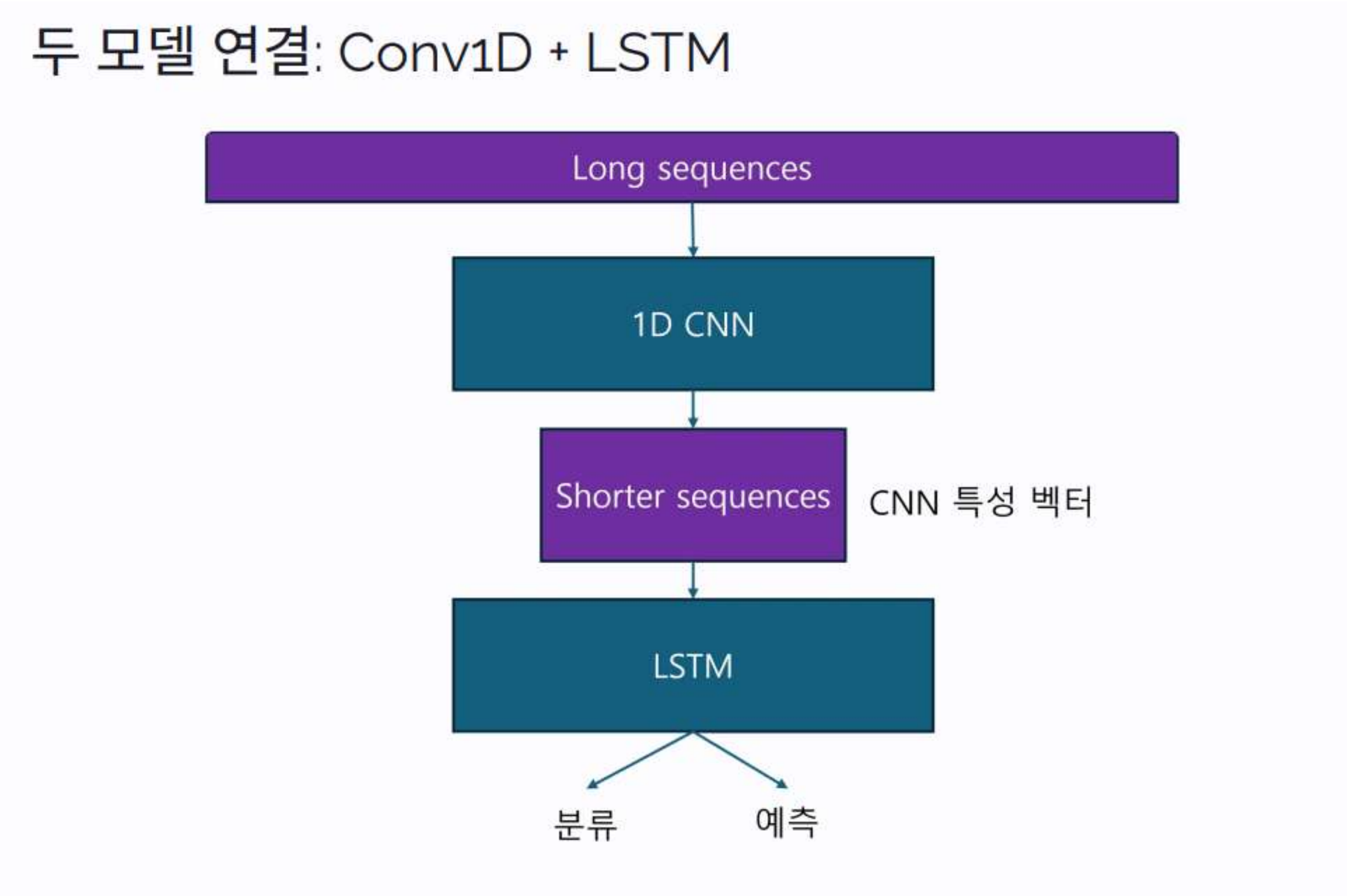


GlobalMaxPooling

결과


Training Acc.
Validation Acc.
Accuracy
1.00
0.95
0.90
0.85
0.80
0.75
Epoch
1
2
3
4
5
6
7
8

IMDB Conv 1D + LSTM 실습



두 모델 연결: Conv1D + LSTM

Long sequences

1D CNN

Shorter sequences

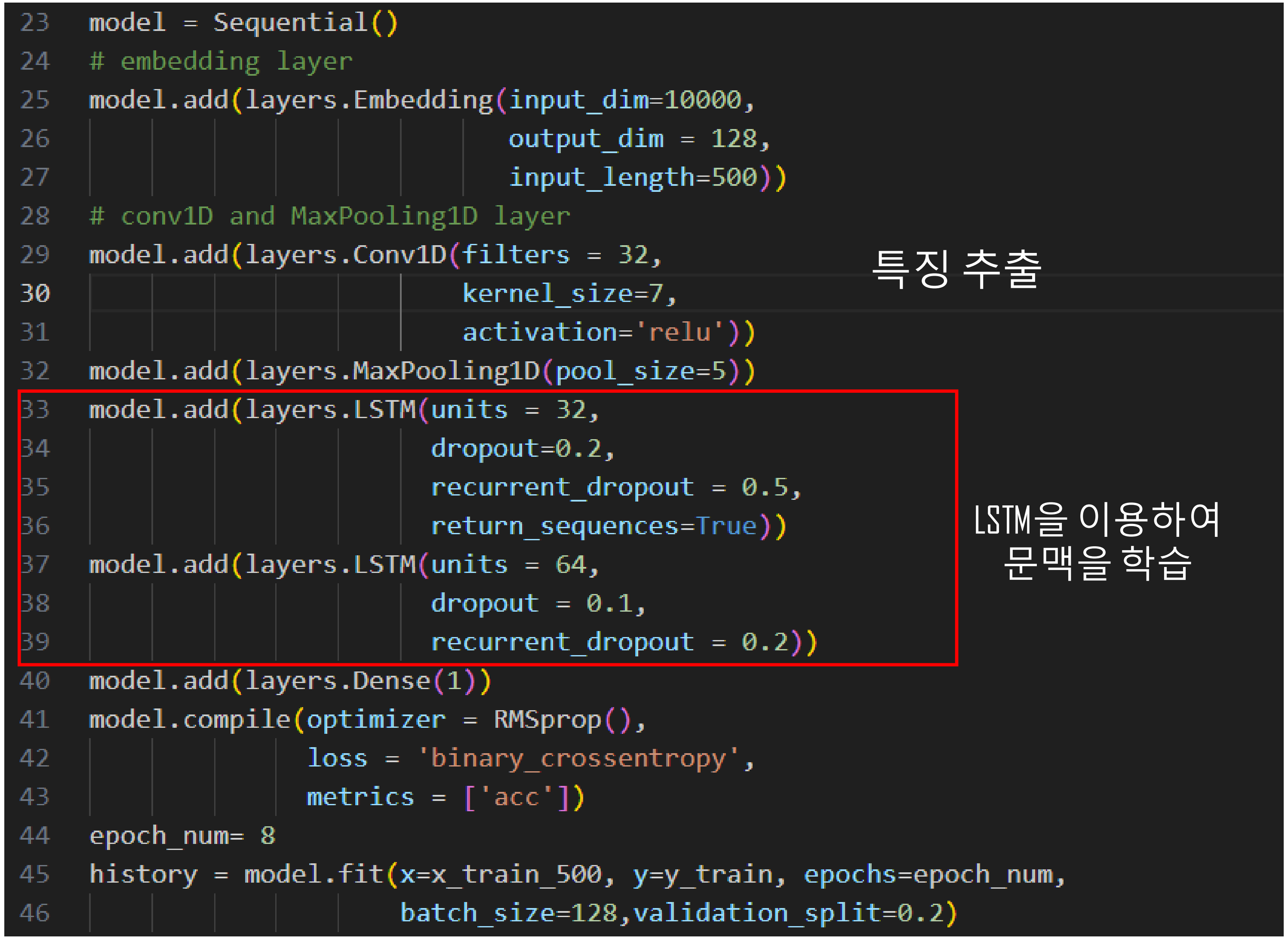
CNN 특성 벡터

LSTM

분류

예측

IMDB Conv 1D + LSTM 실습

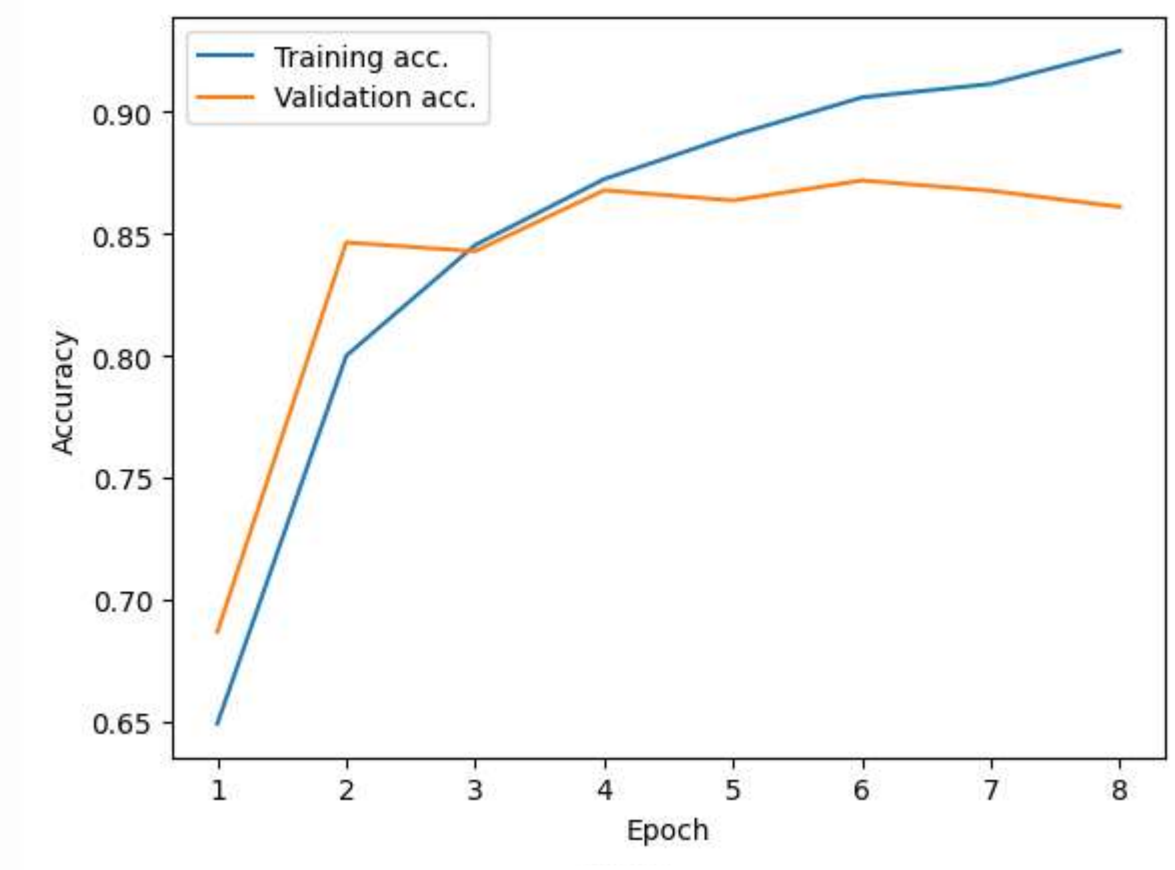


```
model = Sequential()
# embedding layer
model.add(layers.Embedding(input_dim=10000,
                           output_dim = 128,
                           input_length=500))
# conv1D and MaxPooling1D layer
model.add(layers.Conv1D(filters = 32,
                        kernel_size=7,
                        activation='relu'))
model.add(layers.MaxPooling1D(pool_size=5))
model.add(layers.LSTM(units = 32,
                      dropout=0.2,
                      recurrent_dropout = 0.5,
                      return_sequences=True))
model.add(layers.LSTM(units = 64,
                      dropout = 0.1,
                      recurrent_dropout = 0.2))
model.add(layers.Dense(1))
model.compile(optimizer = RMSprop(),
              loss = 'binary_crossentropy',
              metrics = ['acc'])
epoch_num= 8
history = model.fit(x=x_train_500, y=y_train, epochs=epoch_num,
                    batch_size=128,validation_split=0.2)
```

특징 추출

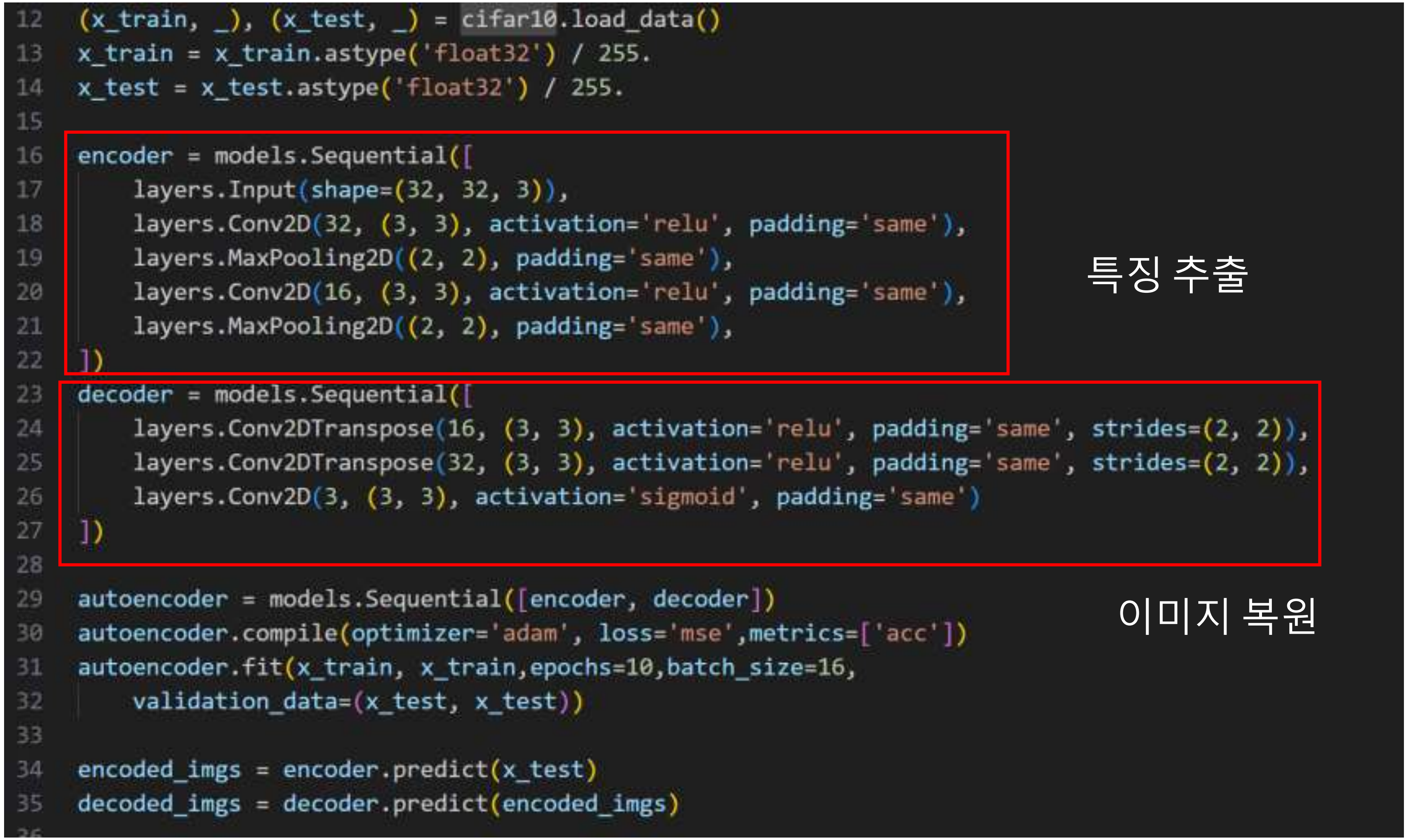
LSTM을 이용하여
문맥을 학습

결과



두 개의 모델의 차이점을 생각해보기

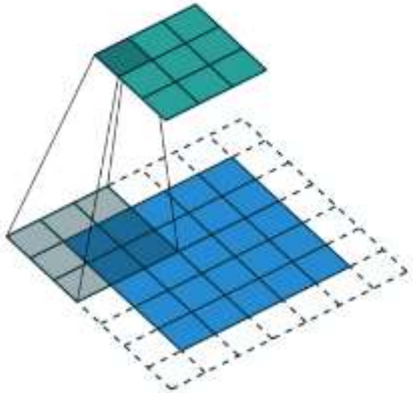
오토인코더



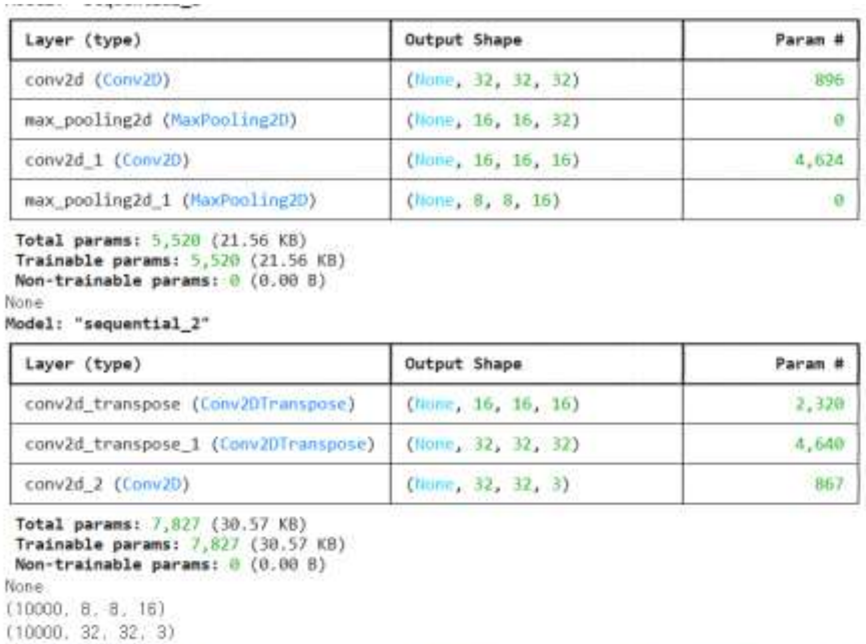
```
(x_train, _), (x_test, _) = cifar10.load_data()
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.

encoder = models.Sequential([
    layers.Input(shape=(32, 32, 3)),
    layers.Conv2D(32, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D((2, 2), padding='same'),
    layers.Conv2D(16, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D((2, 2), padding='same'),
])
decoder = models.Sequential([
    layers.Conv2DTranspose(16, (3, 3), activation='relu', padding='same', strides=(2, 2)),
    layers.Conv2DTranspose(32, (3, 3), activation='relu', padding='same', strides=(2, 2)),
    layers.Conv2D(3, (3, 3), activation='sigmoid', padding='same')
])

autoencoder = models.Sequential([encoder, decoder])
autoencoder.compile(optimizer='adam', loss='mse',metrics=['acc'])
autoencoder.fit(x_train, x_train,epochs=10,batch_size=16,
    validation_data=(x_test, x_test))

encoded_imgs = encoder.predict(x_test)
decoded_imgs = decoder.predict(encoded_imgs)
```

특징 추출

이미지 복원



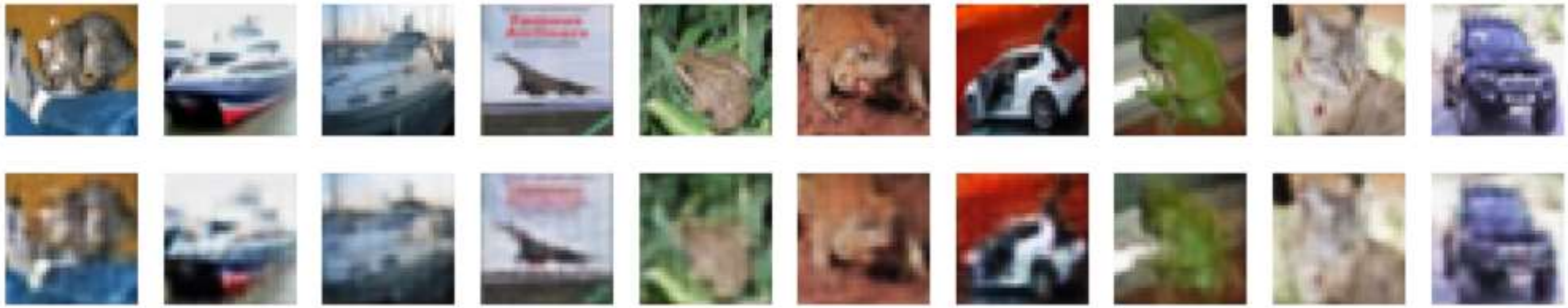
결과



```
┌──────────────────────────────┬──────────────────────┬──────────────┐
│ Layer (type)                 │ Output Shape         │      Param # │
├──────────────────────────────┼──────────────────────┼──────────────┤
│ conv2d (Conv2D)              │ (None, 32, 32, 32)   │          896 │
├──────────────────────────────┼──────────────────────┼──────────────┤
│ max_pooling2d (MaxPooling2D) │ (None, 16, 16, 32)   │            0 │
├──────────────────────────────┼──────────────────────┼──────────────┤
│ conv2d_1 (Conv2D)            │ (None, 16, 16, 16)   │        4,624 │
├──────────────────────────────┼──────────────────────┼──────────────┤
│ max_pooling2d_1 (MaxPooling2D)│ (None, 8, 8, 16)    │            0 │
└──────────────────────────────┴──────────────────────┴──────────────┘
 Total params: 5,520 (21.56 KB)
 Trainable params: 5,520 (21.56 KB)
 Non-trainable params: 0 (0.00 B)
None
Model: "sequential_2"
┌──────────────────────────────┬──────────────────────┬──────────────┐
│ Layer (type)                 │ Output Shape         │      Param # │
├──────────────────────────────┼──────────────────────┼──────────────┤
│ conv2d_transpose (Conv2DTranspose) │ (None, 16, 16, 16) │      2,320 │
├──────────────────────────────┼──────────────────────┼──────────────┤
│ conv2d_transpose_1 (Conv2DTranspose)│ (None, 32, 32, 32) │     4,640 │
├──────────────────────────────┼──────────────────────┼──────────────┤
│ conv2d_2 (Conv2D)            │ (None, 32, 32, 3)    │          867 │
└──────────────────────────────┴──────────────────────┴──────────────┘
 Total params: 7,827 (30.57 KB)
 Trainable params: 7,827 (30.57 KB)
 Non-trainable params: 0 (0.00 B)
None
(10000, 8, 8, 16)
(10000, 32, 32, 3)
```

→ 인코더

→ 디코더



→ 원본

→ 복원 결과