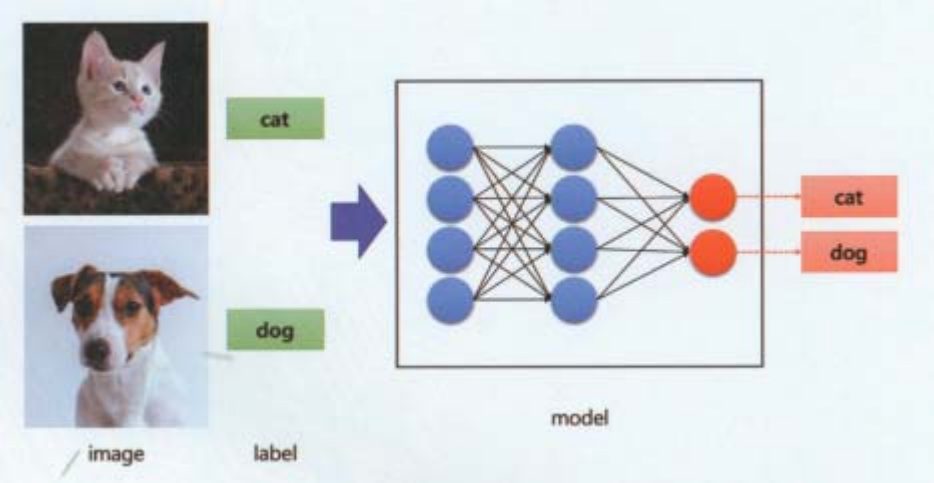
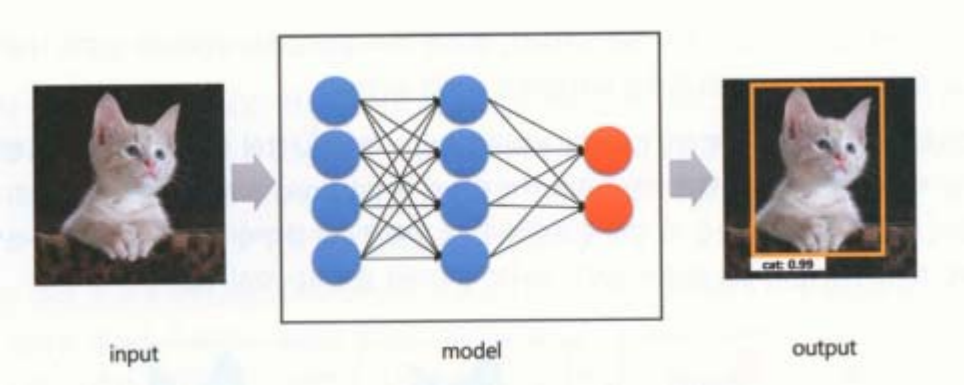
텐서플로우 딥러닝: CNN, RNN/LSTM

1. Convolutional Neural Network: CNN

   A. 이미지 분류



   B. 객체 탐지



   C. 특징 추출(Feature Extraction)

      i. 이미지에서 외곽선 특징 추출방법(object detection): SIFT(Scale-Invariant Feature Transform)

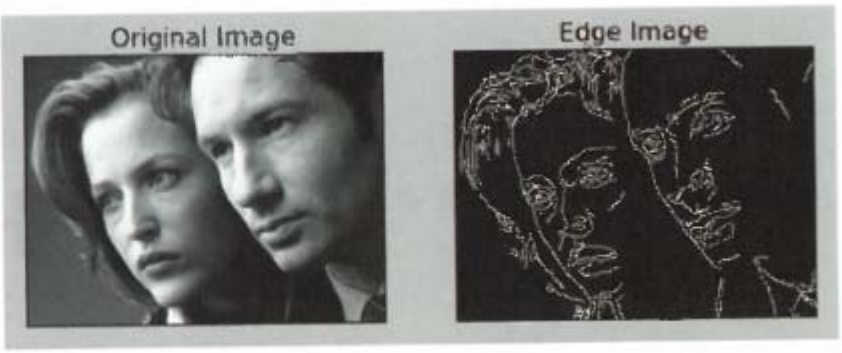


그림 6.1 외곽선 검출 알고리즘 중 하나인 Canny Edge Detection을 이용한 결과 이미지[1]

ii. convolution 연산: 각 pixcel과 그 주변 pixcel의 조합의 어떤 계산식으로 대체하는 변환 이미지 생성

1. filter 또는 kernel: pixcel들의 어떤 계산식에 사용되는 행렬

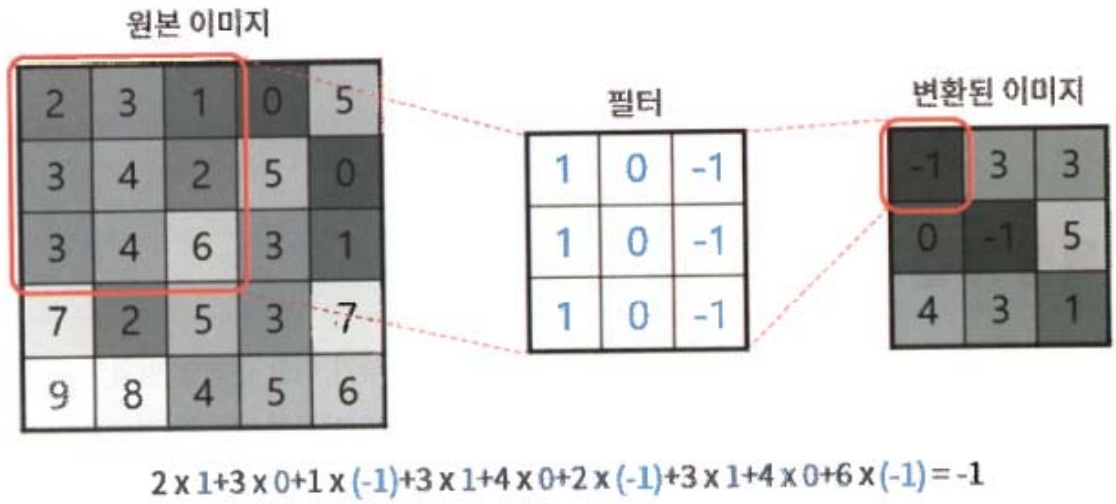
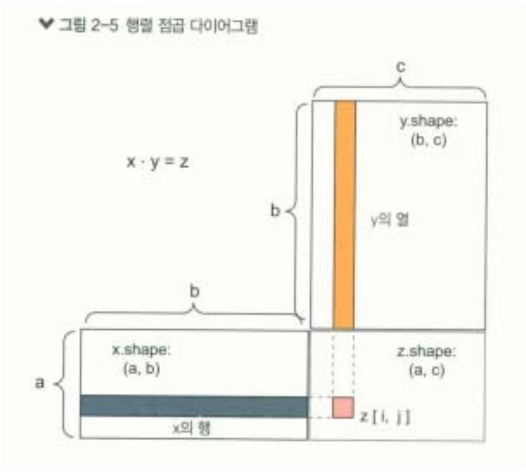
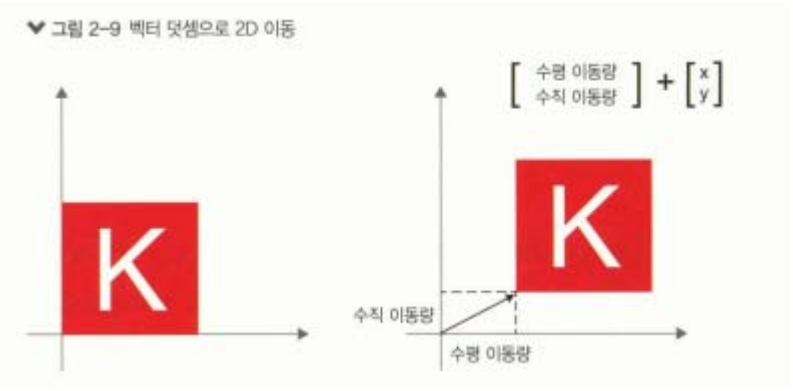


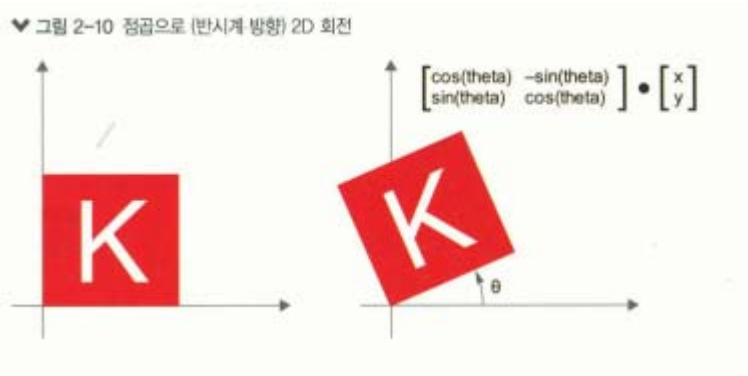
그림 6.2 컨볼루션 연산은 원본 이미지와 필터 행렬의 합성곱입니다.

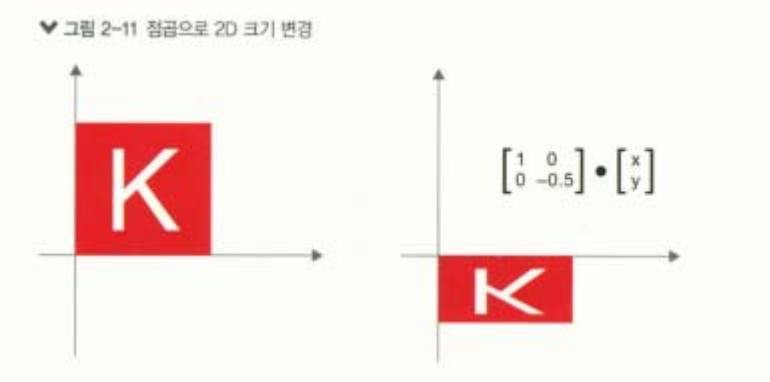
-> matrix product 연산



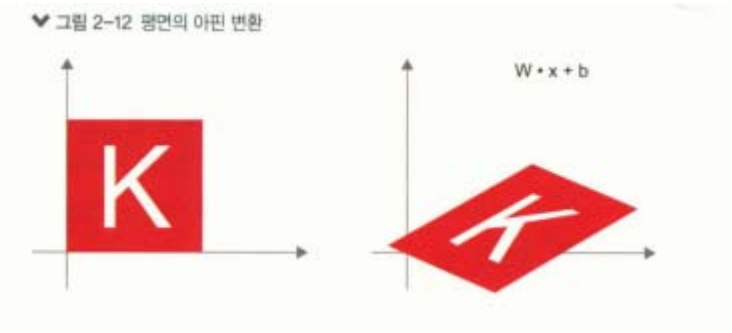
-> tensor reshaping



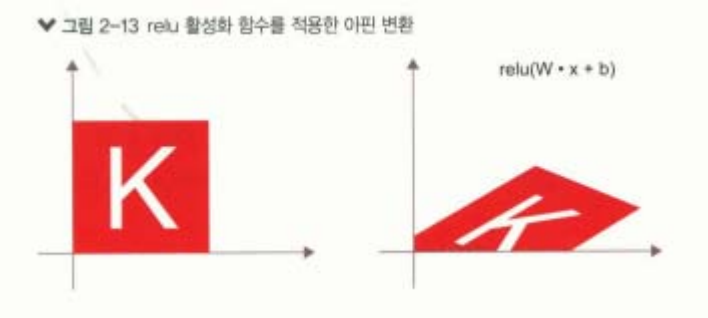




-> affine transform:



-> relu activation 함수를 사용하는 dense layer



iii. convolution 연산:

1. convolute: twist, coil
2. convolution: 나선형의 > 합성곱이라고 번역
   A. 각 pixcel을 포함한 주변 pixcel과 filter 간의 행렬 곱의 합 값을 변환된 이미지 pixcel 값으로 사용

D. convolution filter의 종류

그림 6.3 다양한 필터를 적용했을 때의 컨볼루션 연산의 결과[3]

i. hand-crafted feature: 수작업으로 입력한 filter 값

1. 외곽선 검출 알고리즘, SIFT, 그림 6.3의 filter: 수작업으로 설계한 feature
2. hand-crafted feature의 문제점:
   A. 이미지를 사용한 application에 대한 전문적 knowledge 필요
   B. 수작업으로 filter 값을 정하는 것은 time-consuming 작업
   C. 주어진 application을 위한 filter가 다른 application에 적합하지 않음

E. deep learning 기반의 convoluton 연산

i. deep learning network가 filter를 자동 생성

ii. learning으로 network neuron은 입력 데이터에 대한 특정 패턴을 잘 extract 할 수 있도록 적응

iii. Question: convolution neural network 는 어떻게 feature를 자동으로 extract할 수 있나?

2. convolution neural network의 구조

A. convolution neural net은 feature extractor와 classifier로 구성

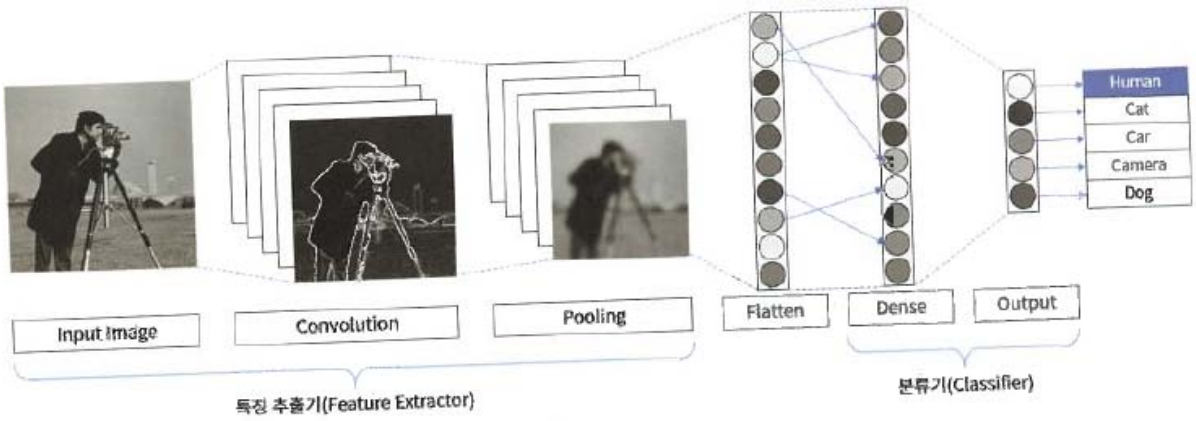


그림 6.5 이미지 분류에 사용되는 컨볼루션 신경망의 구조[6]

i. Feature extractor: convolution layer와 pooling layer

ii. Classifier: Dense layer + Dropout layer + Dense layer

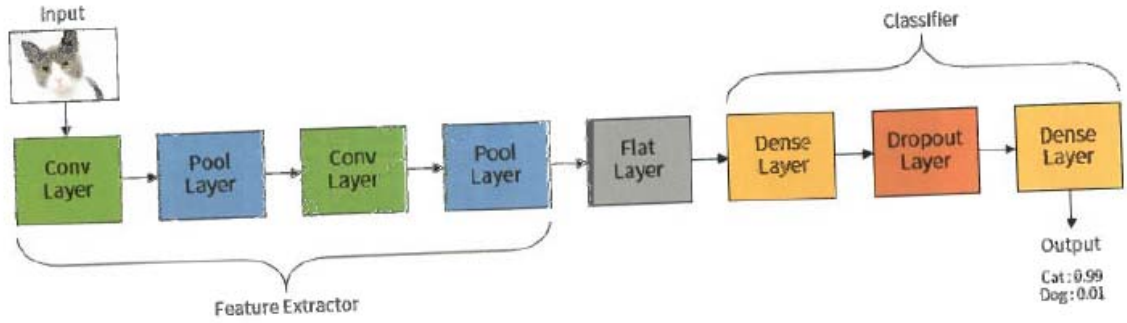
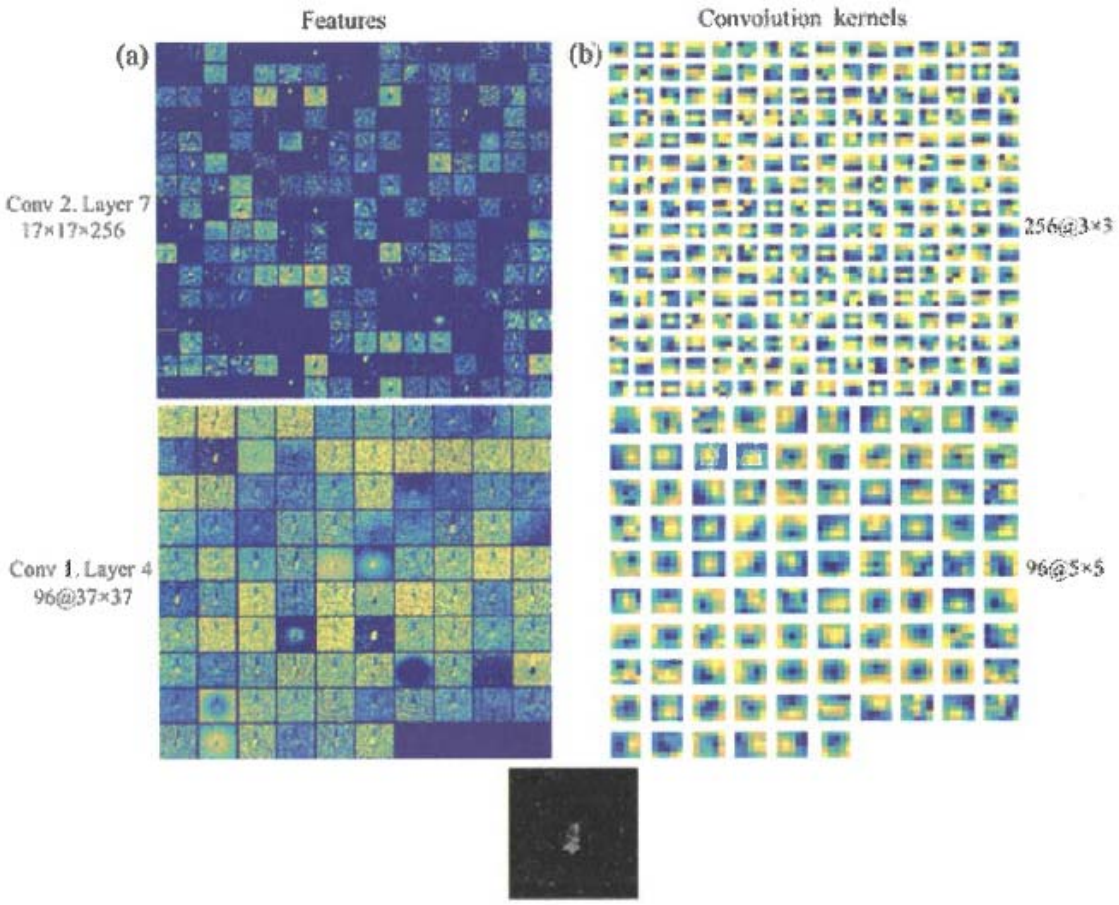


그림 6.6 이미지 분류에 사용되는 컨볼루션 신경망의 구조

B. Convolution layer에서 사용하는 filter: 네트워크 학습으로 자동 extract

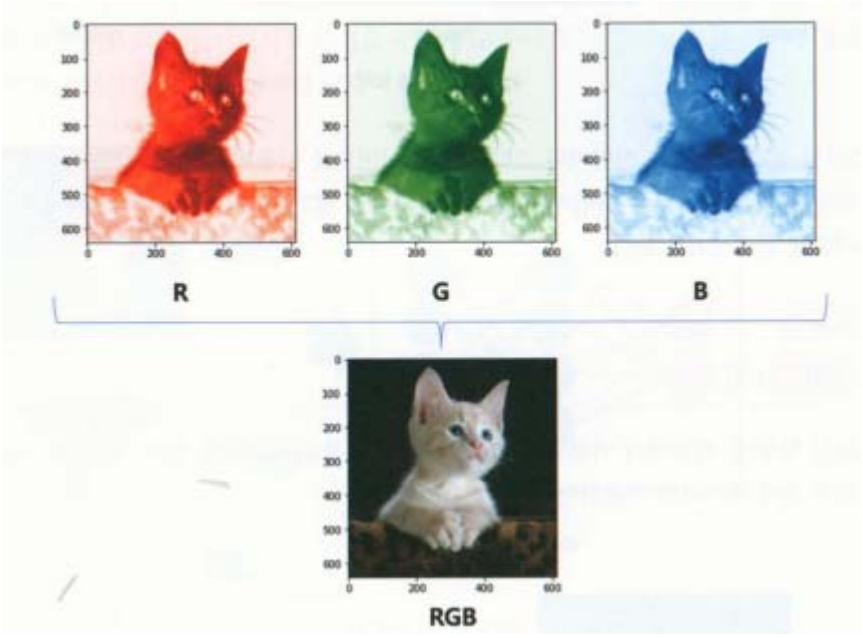
i. Filter의 개수를 hyperparameter로 지정하는 것이 딥러닝 코딩



ii. Channel: samples 수

1. Color image: RGB 채널

```
# 그림 6.8 출력 코드. 참고 링크 : https://stackoverflow.com/a/37435090/2689257
import matplotlib.pyplot as plt

image_path = tf.keras.utils.get_file('cat.jpg', 'http://bit.ly/33U6mH9')
image = plt.imread(image_path)

titles = ['RGB Image', 'Red channel', 'Green channel', 'Blue channel']
cmaps = [None, plt.cm.Reds_r, plt.cm.Greens_r, plt.cm.Blues_r]

from numpy import array, zeros_like
def channel(image, color):
    if color not in (0, 1, 2): return image
    c = image[..., color]
    z = zeros_like(c)
    return array([(c, z, z), (z, c, z), (z, z, c)][color]).transpose(1,2,0)

colors = range(-1, 3)
fig, axes = plt.subplots(1, 4, figsize=(13,3))
objs = zip(axes, titles, colors)
for ax, title, color in objs:
    ax.imshow(channel(image, color))
    ax.set_title(title)
    ax.set_xticks(())
    ax.set_yticks(())

plt.show()
```



iii. Convolution 연산의 적용 결과

1. Filter 개수가 convolution 연산 결과 숫자

2. Filter 개수를 늘리면 변환된 이미지의 차원 수가 증가(dimension의 증가)

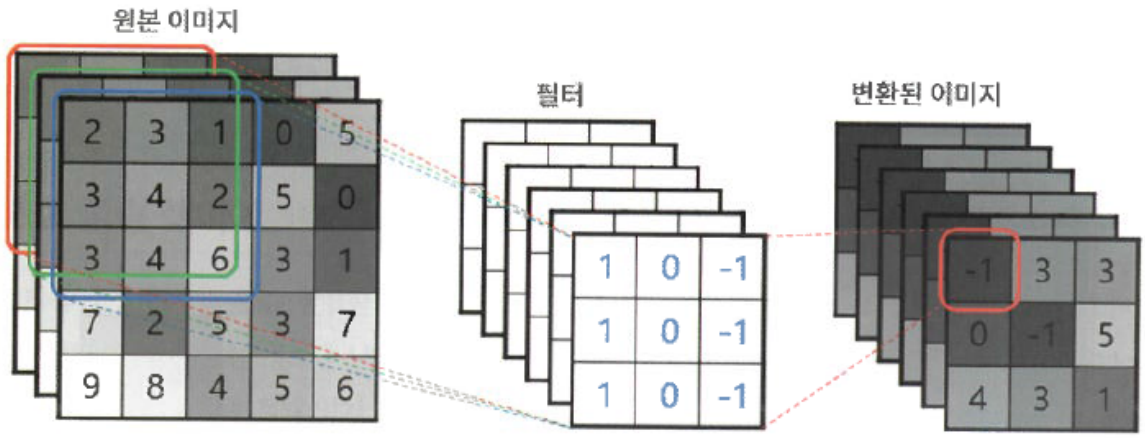
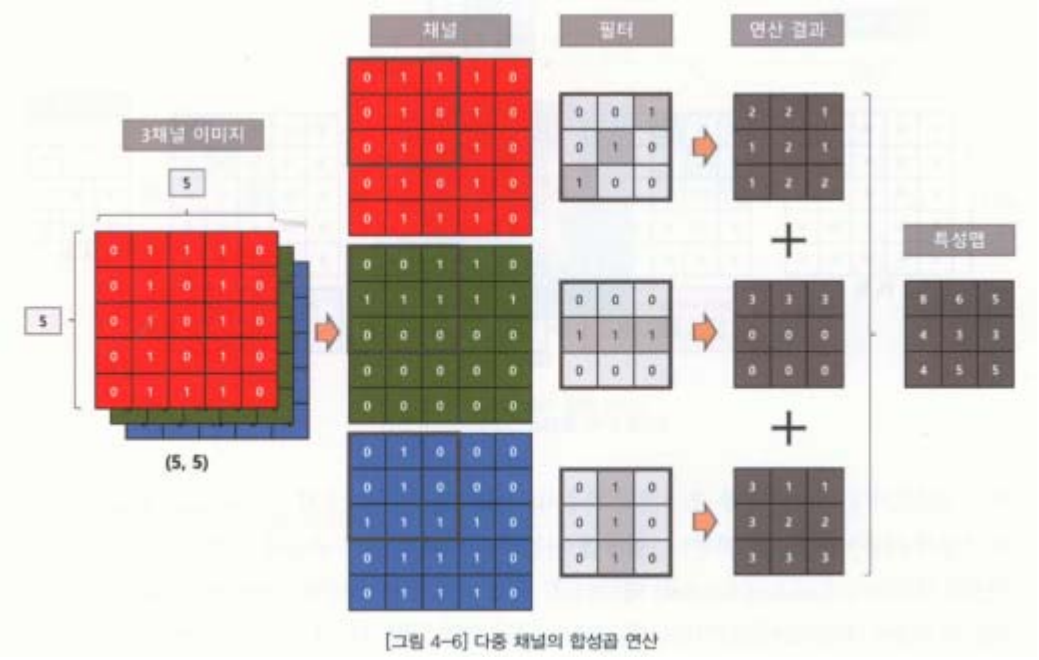


그림 6.9 RGB 채널을 가진 이미지에 컨볼루션 연산을 적용한 결과



[그림 4-6] 다중 채널의 합성곱 연산

C. Convolution layer를 생성하는 코드

```
# 6.1 Conv2D 레이어 생성 코드
conv1 = tf.keras.layers.Conv2D(kernel_size=(3,3),strides=(2,2),padding='valid' ,filters=16)
```

i. Kernel size: filter matrix의 크기(rows, cols)

ii. Strides: filter를 적용하여 다음 step으로 이동하는 크기

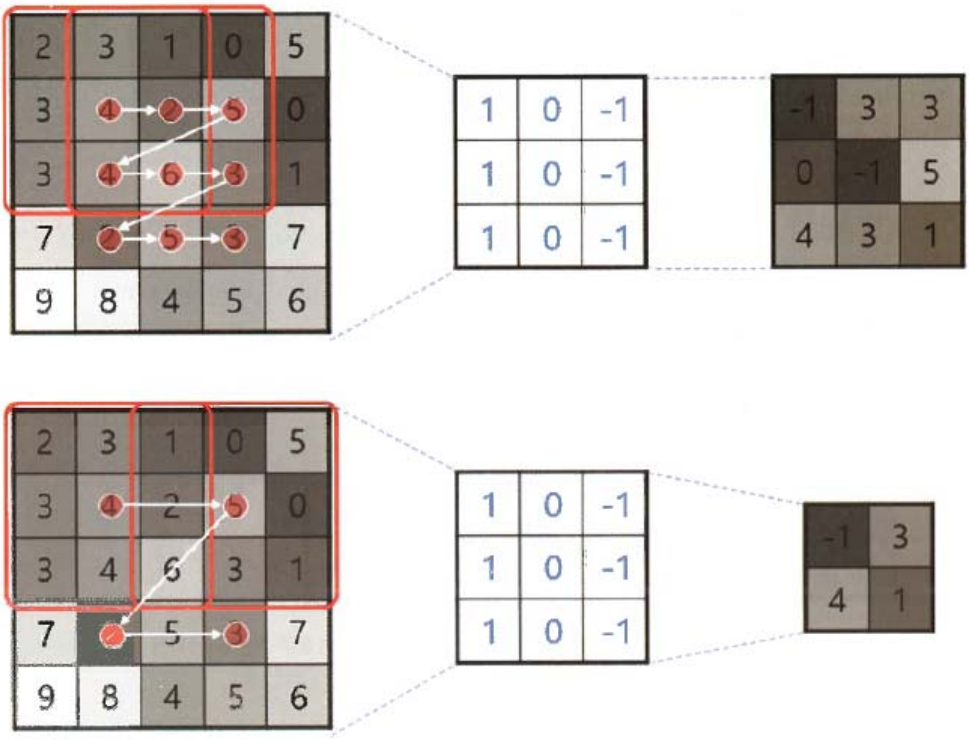


그림 6.10 strides=1일 때와 strides=2일 때의 결과 이미지 비교. 붉은색 원은 필터의 중심

iii. Padding: 입력 이미지 주변 pixel 값을 넣을지 지정하는 option

1. Valid: 빈값을 사용하지 않음
2. Same: 빈값을 상하, 좌우에 추가하여 출력 이미지 크기를 입력과 같도록 만듬 > zero padding

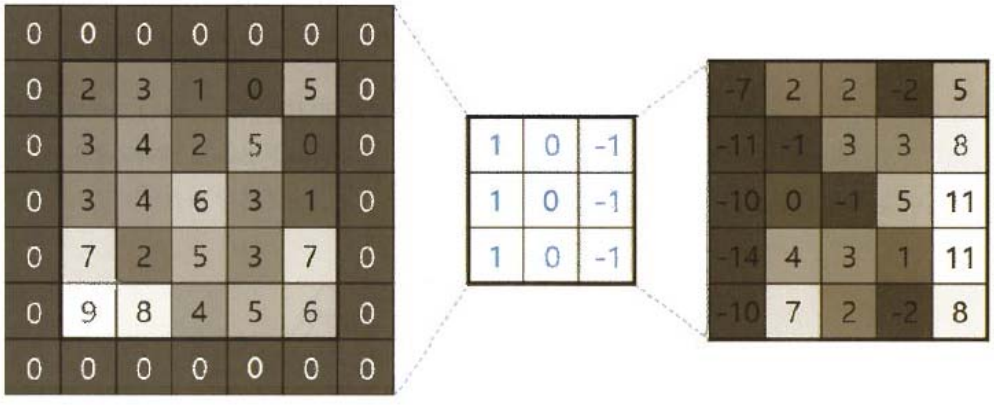


그림 6.11 padding='same'일 때 입력 이미지와 출력 이미지의 크기가 같아집니다.

iv. Filters: filter의 개수

1. VGG 사례: 64 > 128 > 256 > 512 등으로 증가

D. Pooling layer

i. Subsampling: 이미지의 크기를 줄이면서 중요한 정보만 남기는 방법

ii. Pooling layer 구성 방법:

1. Max pooling layer: convolution layer에서는 max pooling layer를 더 많이 사용

2. Average pooling layer

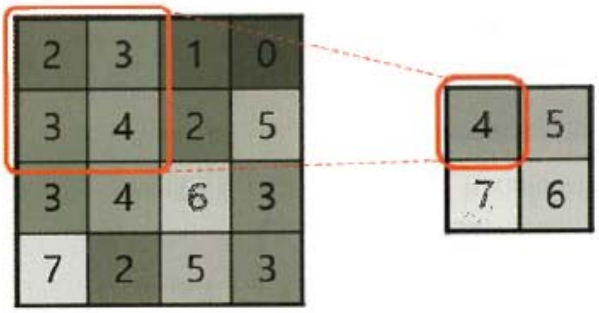
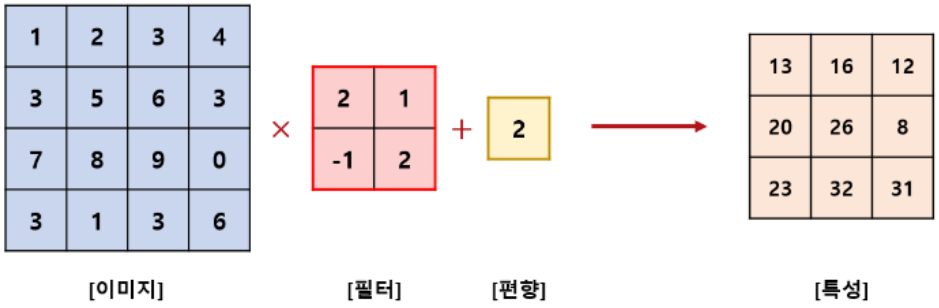


그림 6.12 pool_size=(2,2), strides=(2,2)일 때의 MaxPool2D 레이어 적용 결과

3. Pool_size=(2,2), strides=(2,2) option으로 이미지 크기가 half가 됨

4. Pooling layer는 가중치가 존재하지 않으므로 학습되지 않음

5. Convolution layer에서 가중치는 filter 값임



iii. Max pooling layer 생성 코드

```
# 6.2 MaxPool2D 레이어 생성 코드
pool1 = tf.keras.layers.MaxPool2D(pool_size=(2,2), strides=(2,2))
```

E. Dropout layer

i. Neural network의 overfitting을 막기 위한 방법

1. 학습 과정에서 무작위로 neuron의 부분 집합을 제거 > neuron들이 전체 학습 과정을 특정 방향으로 몰아가는 것을 막기 때문에 overfitting을 해결

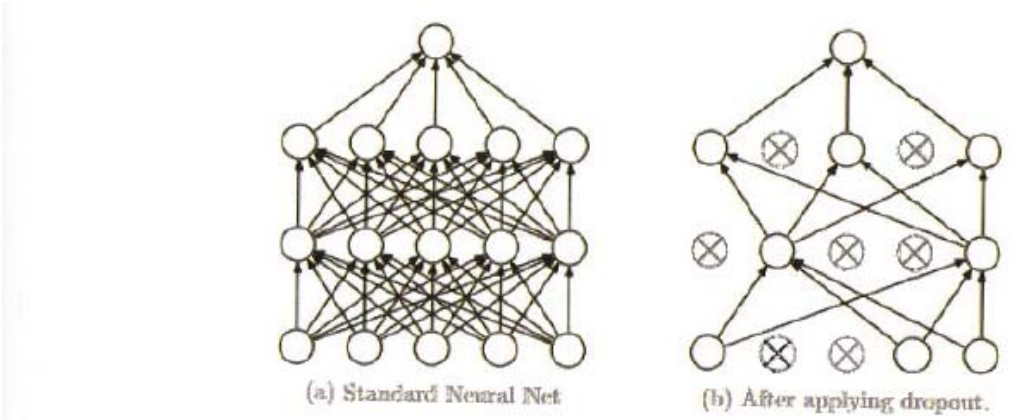


그림 6.13 드롭아웃 레이어의 동작 구조[11]

ii. Dropout layer 생성 코드

```
# 6.3 Dropout 레이어 생성 코드
pool1 = tf.keras.layers.Dropout(rate=0.3)
```

1. 가중치 부여 없고 학습되지 않음

3. Fashion MNIST dataset를 사용한 CNN 학습

A. Dense layer를 사용한 Fashion MNIST 분류 문제 해결

B. Convolution layer와 Pooling layer를 사용한 Fashion MNIST 분류 문제 해결

i. Fashion dataset의 불러오기와 normalization

```
# 6.4 Fashion MNIST 데이터셋 불러오기 및 정규화
import tensorflow as tf

fashion_mnist = tf.keras.datasets.fashion_mnist
(train_X, train_Y), (test_X, test_Y) = fashion_mnist.load_data()

train_X = train_X / 255.0
test_X = test_X / 255.0
```

ii. Conv2D layer 적용을 위한 이미지 tensor(3차원)로 변환

1. Color image: 3channel

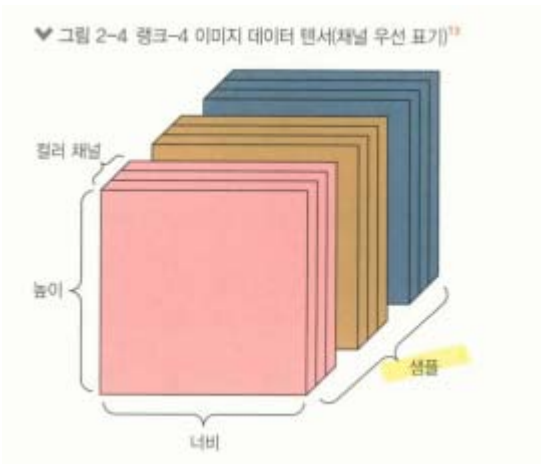
2. Black/white image: 1 channel

```
# 6.5 데이터를 채널을 가진 이미지 형태(3차원)으로 바꾸기
# reshape 이전
print(train_X.shape, test_X.shape)

train_X = train_X.reshape(-1, 28, 28, 1)
test_X = test_X.reshape(-1, 28, 28, 1)

# reshape 이후
print(train_X.shape, test_X.shape)
```

```
(60000, 28, 28) (10000, 28, 28)
(60000, 28, 28, 1) (10000, 28, 28, 1)
```

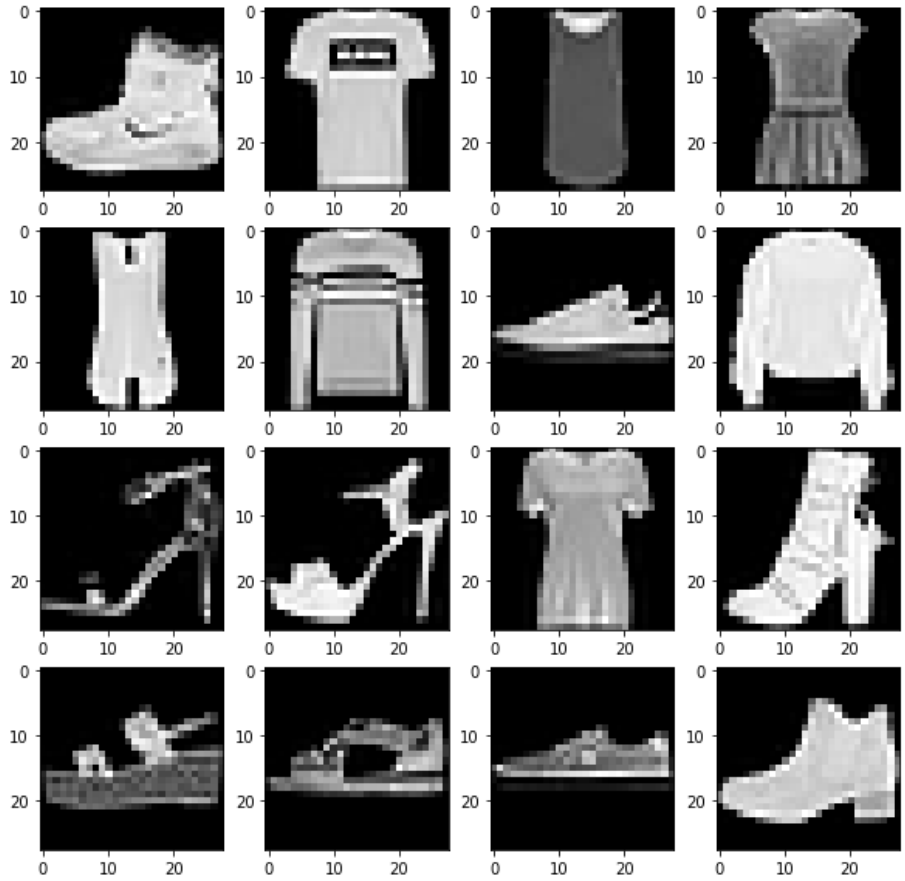
-> 이미지 데이터의 tensor: 256 x 256 흑백 이미지가 128개 batch > (128, 256, 256, 1)

-> color 이미지 128개 batch: (128, 256, 256, 3)



iii. 데이터 확인

```
# 6.6 데이터 확인
import matplotlib.pyplot as plt
# 전체 그래프의 사이즈를 width=10, height=10 으로 지정합니다.
plt.figure(figsize=(10, 10))
for c in range(16):
    # 4행 4열로 지정한 grid 에서 c+1 번째의 칸에 그래프를 그립니다. 1~16 번째 칸을 채우게 됩니다.
    plt.subplot(4,4,c+1)
    plt.imshow(train_X[c].reshape(28,28), cmap='gray')

plt.show()

# train 데이터의 첫번째 ~ 16번째 까지의 라벨을 프린트합니다.
print(train_Y[:16])
```



```
[9 0 0 3 0 2 7 2 5 5 0 9 5 5 7 9]
```

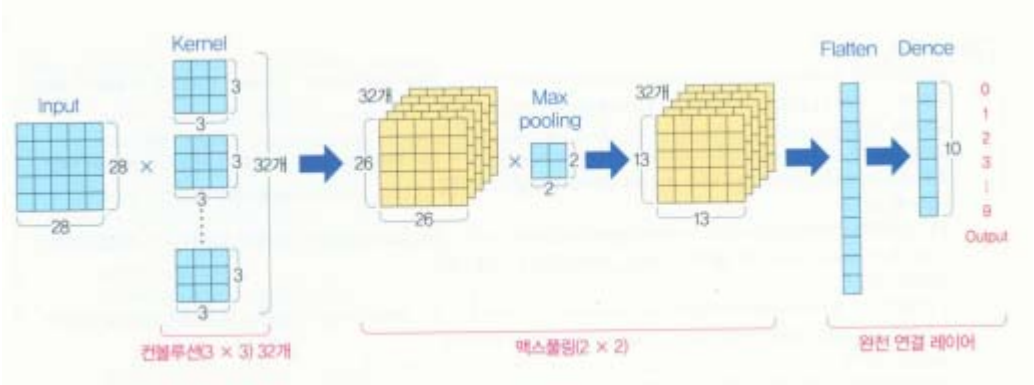
iv. Data category 분류

표 6.1 Fashion MNIST의 범주

라벨	범주
0	티셔츠/상의
1	바지
2	스웨터
3	드레스
4	코트
5	샌들
6	셔츠
7	운동화
8	가방
9	부츠

v. Fashion MNIST 분류를 위한 convolution neural network model

```
[7] # Sequential API를 사용해 샘플 모델 생성
    model = tf.keras.Sequential([
      # Convolution 적용(32 filters)
      tf.keras.layers.Conv2D(32,(3, 3), activation='relu',
                             input_shape=(28, 28, 1), name='conv'),
      # Max Pooling 적용
      tf.keras.layers.MaxPooling2D((2, 2), name='pool'),
      # Classifier 출력층
      tf.keras.layers.Flatten(),
      tf.keras.layers.Dense(10, activation='softmax'),
    ])
```



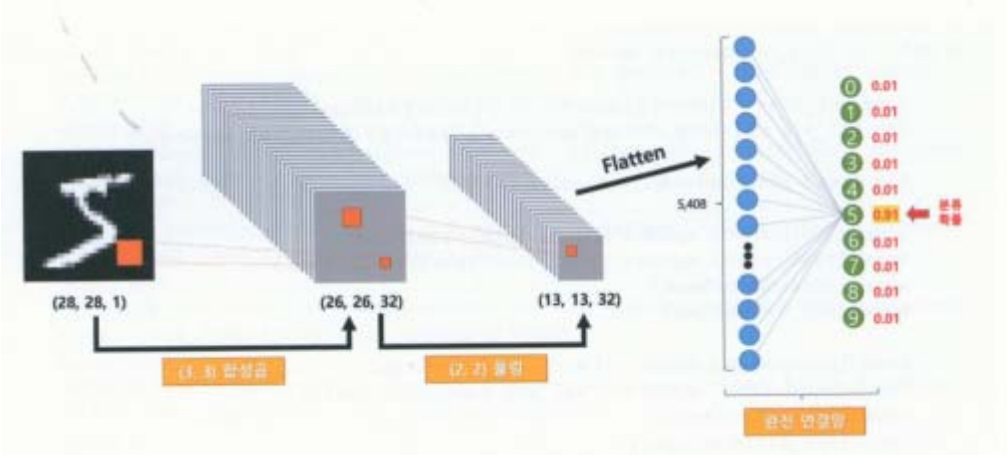
-> 입력 tensor (28,28,1)

-> Conv2D: (26,26, 32) > filter =(3,3)이므로 합성곱은 (26,26) feature map이 32개

-> Pooling: (2,2)이므로 이미지가 1/2로 줄어들어 (13,13,32) tensor로 바뀜

-> flatten layer: 13 * 13 * 32 = 5408 원소를 갖는 1차원 vector

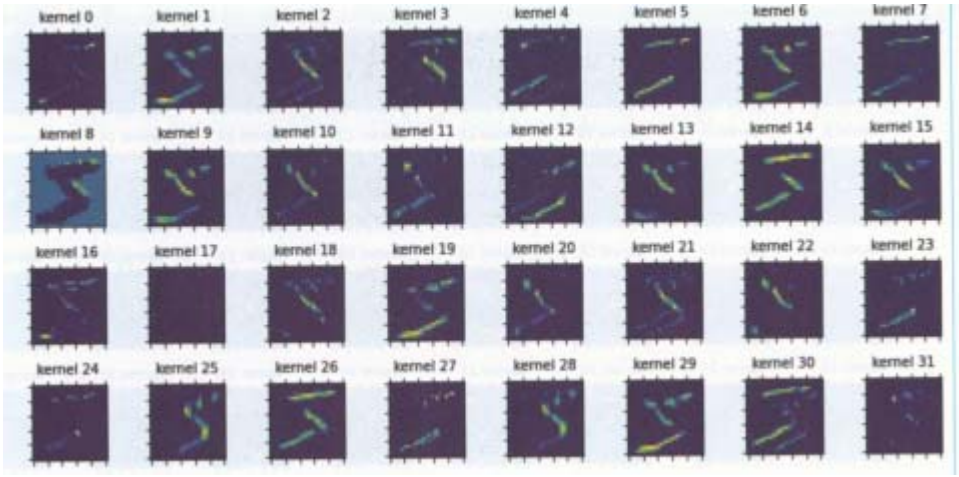
-> Dense layer: 출력 노드 10개




```
[11] # 모델 구조
     model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv (Conv2D)	(None, 26, 26, 32)	320
pool (MaxPooling2D)	(None, 13, 13, 32)	0
flatten (Flatten)	(None, 5408)	0
dense (Dense)	(None, 10)	54090



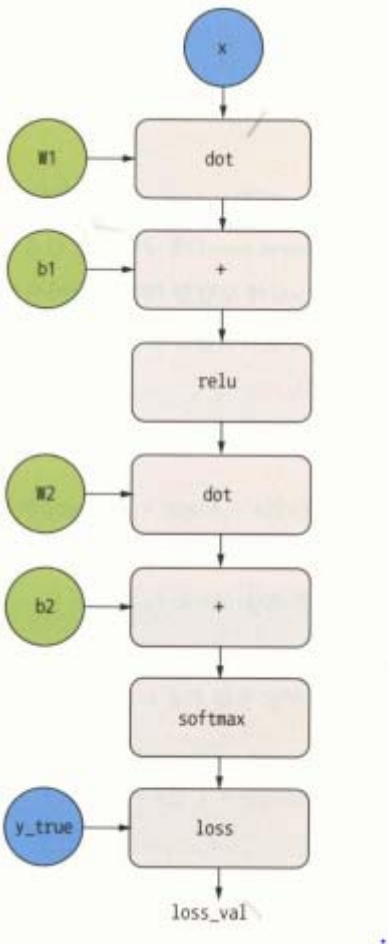
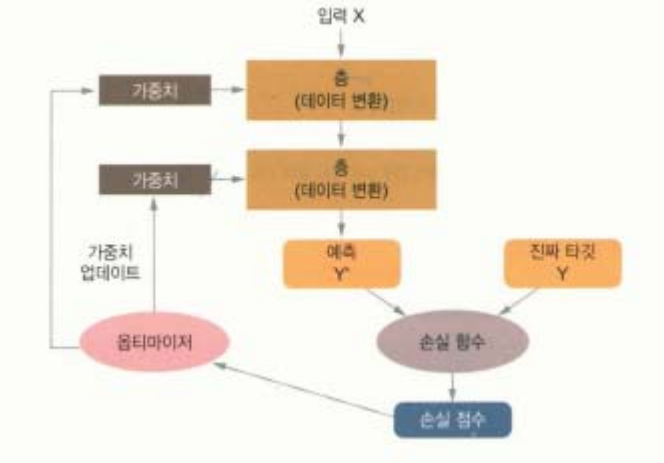


그림 2-26 모델, 층, 손실 함수, 옵티마이저 사이의 관계



```
# 6.7 Fashion MNIST 분류 컨볼루션 신경망 모델 정의
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(input_shape=(28,28,1), kernel_size=(3,3), filters=16),
    tf.keras.layers.Conv2D(kernel_size=(3,3), filters=32),
    tf.keras.layers.Conv2D(kernel_size=(3,3), filters=64),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(units=128, activation='relu'),
    tf.keras.layers.Dense(units=10, activation='softmax')
])

model.compile(optimizer=tf.keras.optimizers.Adam(),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.summary()
```

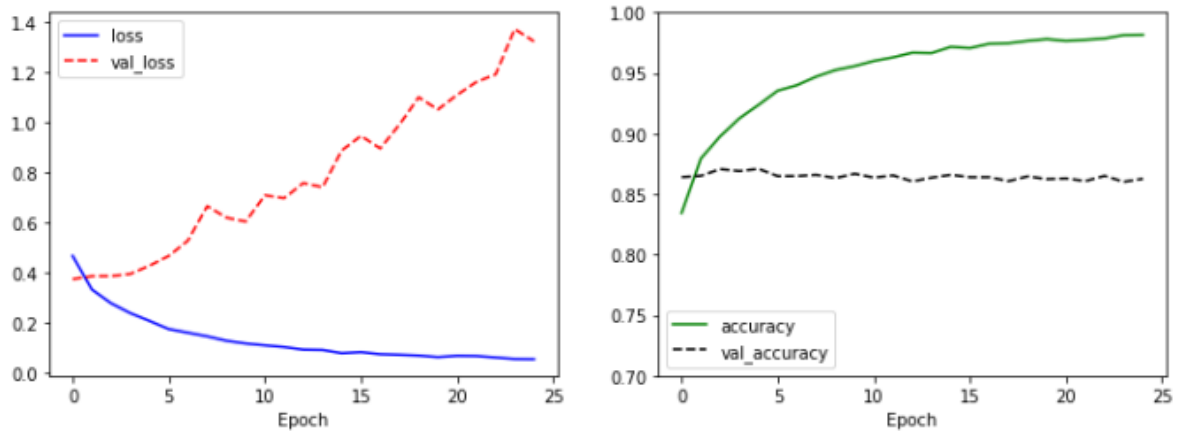
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_1 (Conv2D)           (None, 26, 26, 16)        160

 conv2d_2 (Conv2D)           (None, 24, 24, 32)        4640

 conv2d_3 (Conv2D)           (None, 22, 22, 64)        18496

 flatten (Flatten)           (None, 30976)             0

 dense (Dense)               (None, 128)               3965056

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 3,989,642
Trainable params: 3,989,642
Non-trainable params: 0
_________________________________________________________________
```

1. Input_shape: (28,28, 1)로 (height, width, channel 수)
2. Filters: 16 > 32 > 64로 증가
3. Flattern layer: 다차원 데이터 > 1차원 데이터로 정렬
4. Dense layer: 분류기

vi. Fashion MNIST 분류를 위한 convolution neural network model의 학습

```python
# 6.9 Fashion MNIST 분류 컨볼루션 신경망 모델 학습
history = model.fit(train_X, train_Y, epochs=25, validation_split=0.25)

import matplotlib.pyplot as plt
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], 'b-', label='loss')
plt.plot(history.history['val_loss'], 'r--', label='val_loss')
plt.xlabel('Epoch')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], 'g-', label='accuracy')
plt.plot(history.history['val_accuracy'], 'k--', label='val_accuracy')
plt.xlabel('Epoch')
plt.ylim(0.7, 1)
plt.legend()

plt.show()

model.evaluate(test_X, test_Y, verbose=0)
```

1. Model.fit()의 결과는 history 변수가 loss, accuracy 정보를 포함




```
[1.3586030006408691, 0.8579999804496765]
```

   A. Loss graph: loss는 감소, val_loss는 증가하는 overfitting 문제 발생

   B. Accuracy graph: training data에 대한 model의 정확도인 accuracy는 빠르게 증가(green line), validation에 대한 정확도인 val_accuracy는 학습이 진행될수록 오히려 감소

2. Model.evaluate():

   A. Loss: first output

   B. Accuracy: test data의 정확도, 85.79%

i. Dense layer가 달성한 정확도 88.5%보다 나쁘다

ii. 해결 방안: pooling layer, dropout layer 사용 시도

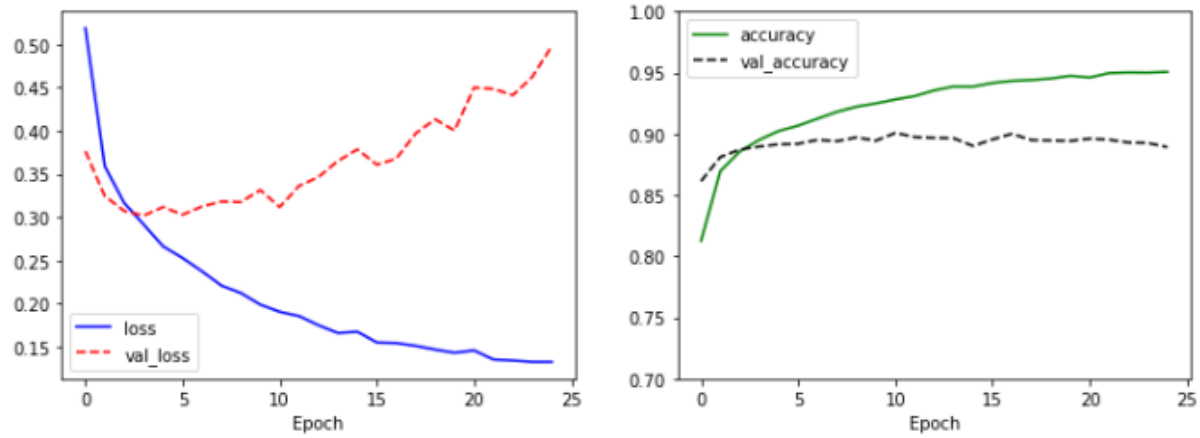
vii. Fashion MNIST 분류를 위한 pooling, dropout layer의 추가 모델

```
# 6.10 Fashion MNIST 분류 컨볼루션 신경망 모델 정의 - 풀링 레이어, 드랍아웃 레이어 추가
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(input_shape=(28,28,1), kernel_size=(3,3), filters=32),
    tf.keras.layers.MaxPool2D(strides=(2,2)),
    tf.keras.layers.Conv2D(kernel_size=(3,3), filters=64),
    tf.keras.layers.MaxPool2D(strides=(2,2)),
    tf.keras.layers.Conv2D(kernel_size=(3,3), filters=128),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(units=128, activation='relu'),
    tf.keras.layers.Dropout(rate=0.3),
    tf.keras.layers.Dense(units=10, activation='softmax')
])

model.compile(optimizer=tf.keras.optimizers.Adam(),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)            (None, 26, 26, 32)        320

 max_pooling2d_1 (MaxPooling  (None, 13, 13, 32)       0
 2D)

 conv2d_5 (Conv2D)            (None, 11, 11, 64)        18496

 max_pooling2d_2 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 conv2d_6 (Conv2D)            (None, 3, 3, 128)         73856

 flatten_1 (Flatten)          (None, 1152)              0

 dense_2 (Dense)              (None, 128)               147584

 dropout_1 (Dropout)          (None, 128)               0

 dense_3 (Dense)              (None, 10)                1290

=================================================================
Total params: 241,546
Trainable params: 241,546
Non-trainable params: 0
_________________________________________________________________
```

viii. Fashion MNIST 분류를 위한 pooling, dropout layer를 포함한 모델의 학습

```
# 6.11 Fashion MNIST 분류 컨볼루션 신경망 모델 학습 - 풀링 레이어, 드랍아웃 레이어 추가
history = model.fit(train_X, train_Y, epochs=25, validation_split=0.25)

import matplotlib.pyplot as plt
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], 'b-', label='loss')
plt.plot(history.history['val_loss'], 'r--', label='val_loss')
plt.xlabel('Epoch')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], 'g-', label='accuracy')
plt.plot(history.history['val_accuracy'], 'k--', label='val_accuracy')
plt.xlabel('Epoch')
plt.ylim(0.7, 1)
plt.legend()

plt.show()

model.evaluate(test_X, test_Y, verbose=0)
```



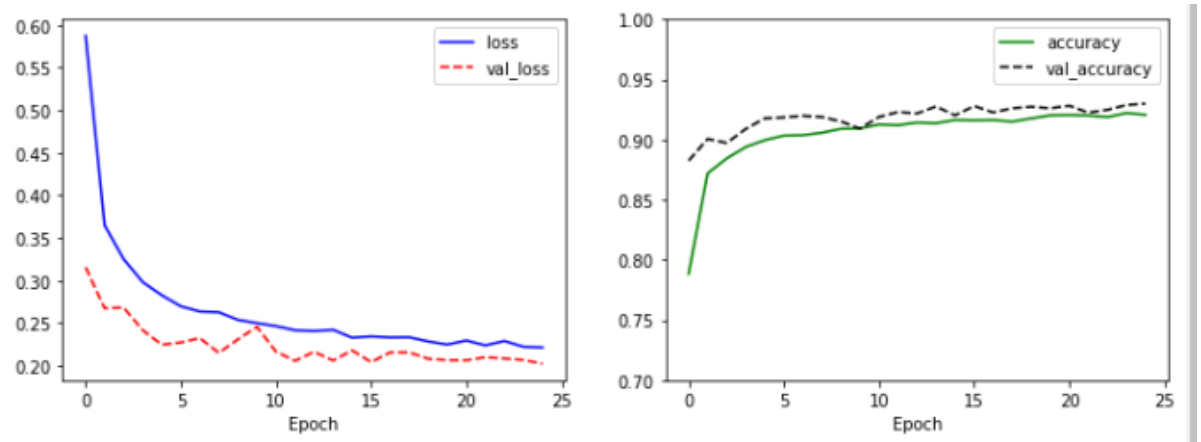

```
[0.5214602351188866, 0.8881000280380249]
```

1. Val_loss는 빠르게 증가하나 val_accuracy는 일정 수준으로 향상
2. Test data에 대한 분류 효율은 88.81%로 개선됨
3. 분류 성능을 향상하기 위한 방법
   A. Layer를 추가하기

+ 코드 + 텍스트

```
# 6.12 VGGNet 스타일의 Fashion MNIST 분류 컨볼루션 신경망 모델 정의
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(input_shape=(28,28,1), kernel_size=(3,3), filters=32, padding='same',
    tf.keras.layers.Conv2D(kernel_size=(3,3), filters=64, padding='same', activation='relu'),
    tf.keras.layers.MaxPool2D(pool_size=(2,2)),
    tf.keras.layers.Dropout(rate=0.5),
    tf.keras.layers.Conv2D(kernel_size=(3,3), filters=128, padding='same', activation='relu'),
    tf.keras.layers.Conv2D(kernel_size=(3,3), filters=256, padding='valid', activation='relu'),
    tf.keras.layers.MaxPool2D(pool_size=(2,2)),
    tf.keras.layers.Dropout(rate=0.5),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(units=512, activation='relu'),
    tf.keras.layers.Dropout(rate=0.5),
    tf.keras.layers.Dense(units=256, activation='relu'),
    tf.keras.layers.Dropout(rate=0.5),
    tf.keras.layers.Dense(units=10, activation='softmax')
])

model.compile(optimizer=tf.keras.optimizers.Adam(),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.summary()
```

```
# 6.13 VGGNet 스타일의 Fashion MNIST 분류 컨볼루션 신경망 모델 학습
history = model.fit(train_X, train_Y, epochs=25, validation_split=0.25)

import matplotlib.pyplot as plt
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], 'b-', label='loss')
plt.plot(history.history['val_loss'], 'r--', label='val_loss')
plt.xlabel('Epoch')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], 'g-', label='accuracy')
plt.plot(history.history['val_accuracy'], 'k--', label='val_accuracy')
plt.xlabel('Epoch')
plt.ylim(0.7, 1)
plt.legend()

plt.show()

model.evaluate(test_X, test_Y, verbose=0)
```



[0.2165478765964508, 0.9276999831199646]

i. Test data에 대한 분류 성능: 92.76%

ii. Val_accuracy 더 좋아지고 있어 overfitting 문제 해결

B. Image augmentation

i. Training data에 없는 새로운 이미지를 만들어 훈련 데이터를 보강

1. Convolution neural network의 학습 능력 향상을 위한 다양한 훈련데이터를 보강

A. Horizontal flip(수평 뒤집기), rotate(회전), shear(기울이기), zoom(확대),shift(평행 이동)

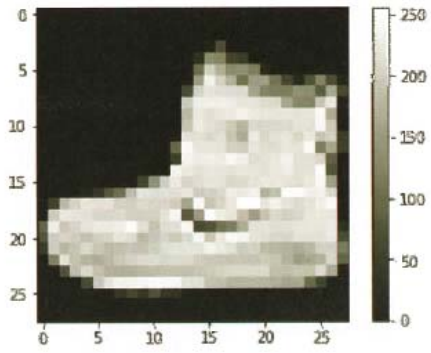


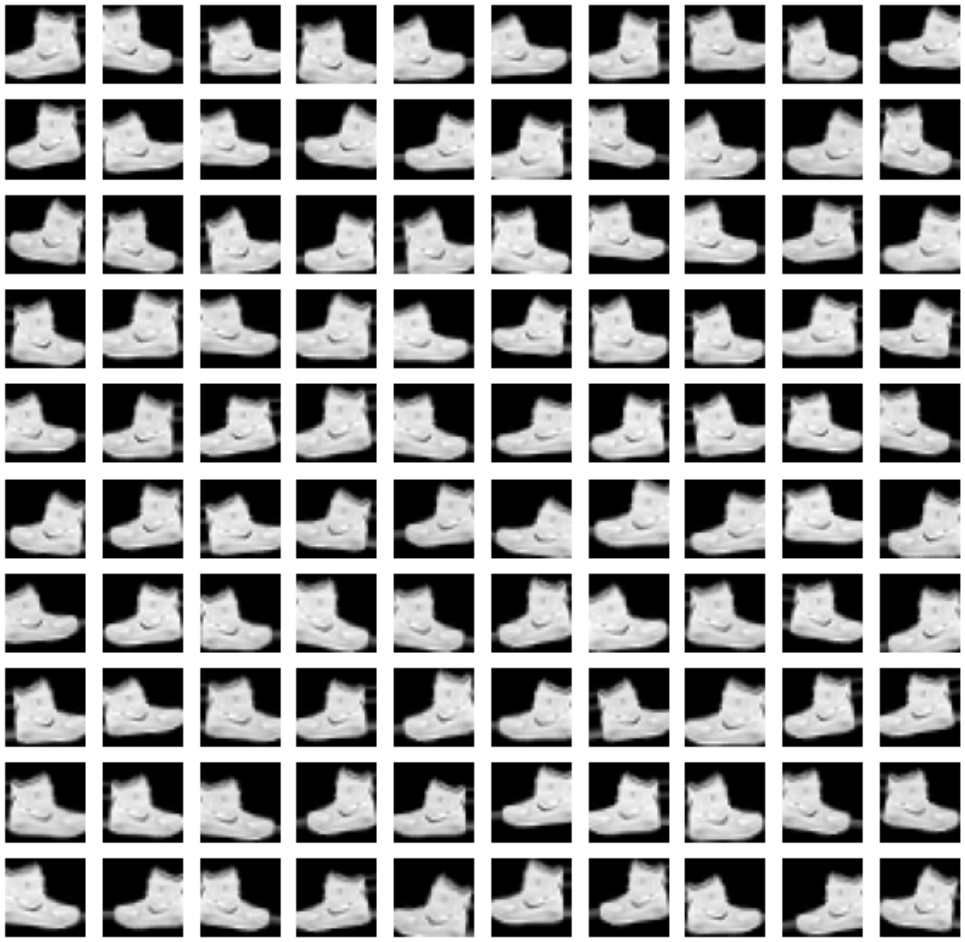
그림 6.16 Fashion MNIST 훈련 데이터의 첫 번째 이미지

ii. Tf.keras의 ImageDataGenerator 사용

```
# 6.14 Image Augmentation 데이터 표시
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import numpy as np

# 이 ImageDataGenerator 코드 부분은 다음 링크에서 참조했습니다.
# https://github.com/franneck94/MNIST-Data-Augmentation/blob/master/mnist.py
# rotation, zoom, shift, flip 등을 지정합니다.
image_generator = ImageDataGenerator(
            rotation_range=10,
            zoom_range=0.10,
            shear_range=0.5,
            width_shift_range=0.10,
            height_shift_range=0.10,
            horizontal_flip=True,
            vertical_flip=False)

augment_size = 100

x_augmented = image_generator.flow(np.tile(train_X[0].reshape(28*28),100).reshape(-1,28,28,1),
                np.zeros(augment_size), batch_size=augment_size, shuffle=False).next()[0]

# 새롭게 생성된 이미지 표시
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 10))
for c in range(100):
    plt.subplot(10,10,c+1)
    plt.axis('off')
    plt.imshow(x_augmented[c].reshape(28,28), cmap='gray')
plt.show()
```

A. Image_generator.flow(): 보강된 이미지를 생성

B. Image_generator.flow().next(): 보강된 이미지를 꺼내서 x_augmented에 반환



iii. Training data에 보강된 이미지 추가

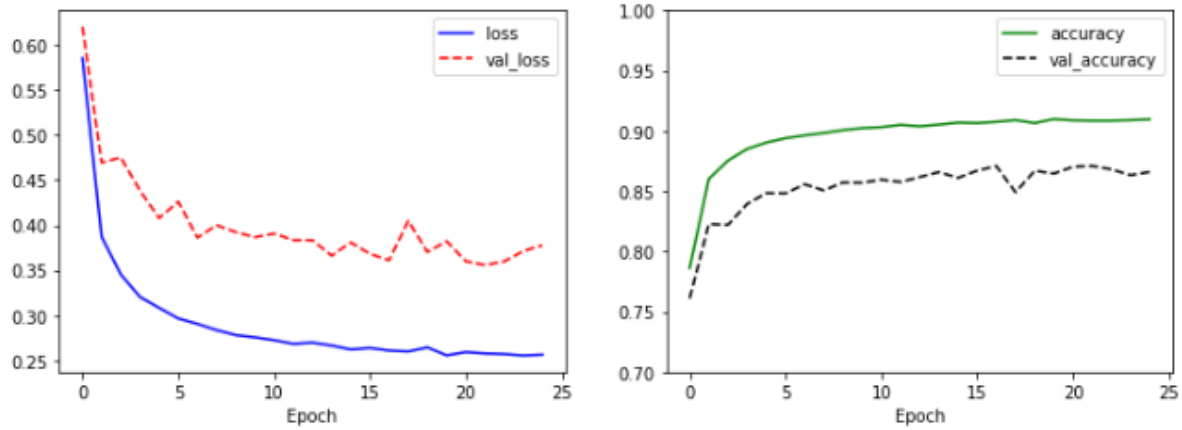
```
# 6.15 Image Augmentation
image_generator = ImageDataGenerator(
            rotation_range=10,
            zoom_range=0.10,
            shear_range=0.5,
            width_shift_range=0.10,
            height_shift_range=0.10,
            horizontal_flip=True,
            vertical_flip=False)

augment_size = 30000

randidx = np.random.randint(train_X.shape[0], size=augment_size)
x_augmented = train_X[randidx].copy()
y_augmented = train_Y[randidx].copy()
x_augmented = image_generator.flow(x_augmented, np.zeros(augment_size),
                      batch_size=augment_size, shuffle=False).next()[0]

# 원래 데이터인 x_train 에 Image Augmentation 된 x_augmented 를 추가합니다.
train_X = np.concatenate((train_X, x_augmented))
train_Y = np.concatenate((train_Y, y_augmented))

print(train_X.shape)
```

```
(90000, 28, 28, 1)
```

1. 보강된 이미지 숫자: 3000개

iv. 보강된 이미지를 사용한 학습

```python
# 6.16 VGGNet style 네트워크 + Image Augmentation 학습
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(input_shape=(28,28,1), kernel_size=(3,3), filters=32, padding='same',
                           activation='relu'),
    tf.keras.layers.Conv2D(kernel_size=(3,3), filters=64, padding='same', activation='relu'),
    tf.keras.layers.MaxPool2D(pool_size=(2,2)),
    tf.keras.layers.Dropout(rate=0.5),
    tf.keras.layers.Conv2D(kernel_size=(3,3), filters=128, padding='same', activation='relu'),
    tf.keras.layers.Conv2D(kernel_size=(3,3), filters=256, padding='valid', activation='relu'),
    tf.keras.layers.MaxPool2D(pool_size=(2,2)),
    tf.keras.layers.Dropout(rate=0.5),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(units=512, activation='relu'),
    tf.keras.layers.Dropout(rate=0.5),
    tf.keras.layers.Dense(units=256, activation='relu'),
    tf.keras.layers.Dropout(rate=0.5),
    tf.keras.layers.Dense(units=10, activation='softmax')
])

model.compile(optimizer=tf.keras.optimizers.Adam(),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(train_X, train_Y, epochs=25, validation_split=0.25)

import matplotlib.pyplot as plt
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], 'b-', label='loss')
plt.plot(history.history['val_loss'], 'r--', label='val_loss')
plt.xlabel('Epoch')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], 'g-', label='accuracy')
plt.plot(history.history['val_accuracy'], 'k--', label='val_accuracy')
plt.xlabel('Epoch')
plt.ylim(0.7, 1)
plt.legend()

plt.show()

model.evaluate(test_X, test_Y, verbose=0)
```




```
[0.22083032131195068, 0.9199000000953674]
```

A. Test data에 대한 분류 성능: 91.99%로 향상

B. Val_accuracy도 증가 추세를 보이지 않아 아직 overfitting 아님

2. 더 많은 layer를 사용하거나 이미지 보강 기법으로 분류 성능을 향상할 수 있다

4. RNN: Recurrent Neural Network

A. 순서가 있는 데이터를 입력으로 받고, 변화하는 입력에 대한 출력

i. 순서 있는 데이터: 악보, 자연어, 날씨, 주가 등 time series data

B. Recurrent: RNN의 출력이 다시 입력으로 사용됨

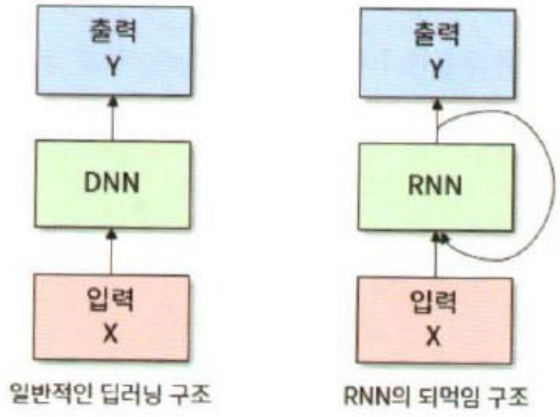


그림 7.1 일반적인 딥러닝 네트워크(DNN)와 순환 신경망(RNN)의 구조 차이

C. Recurrent 함수: $Y_i = f(X_i, Y_{i-1})$

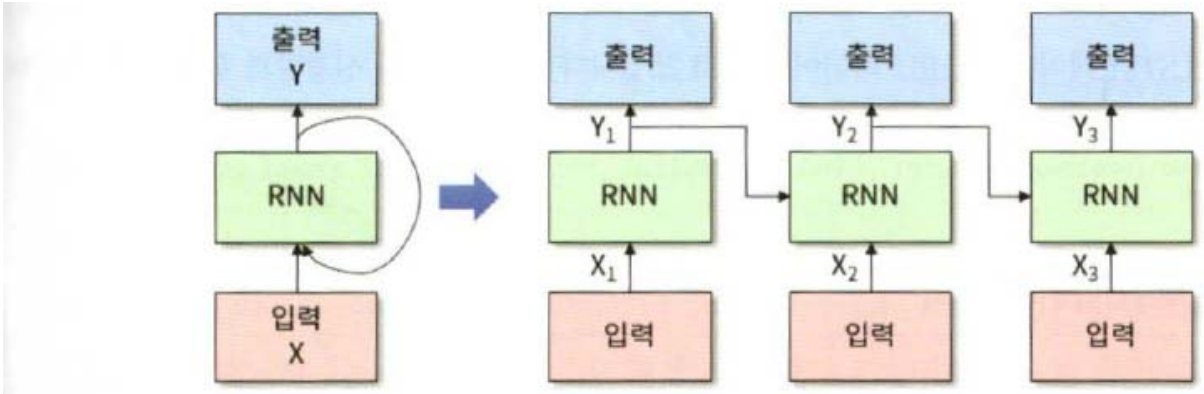


그림 7.2 순환 신경망 구조 풀이

i. 입력과 출력 길이에 제한 없음

D. RNN 용도

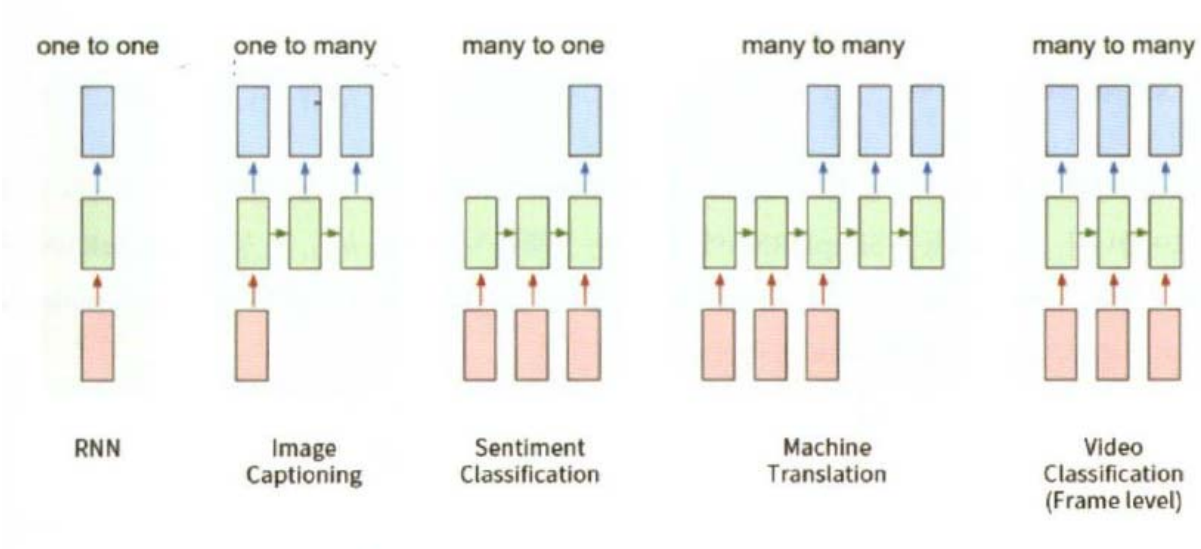


그림 7.3 다양한 형태의 순환 신경망[1]

E. Simple RNN layer

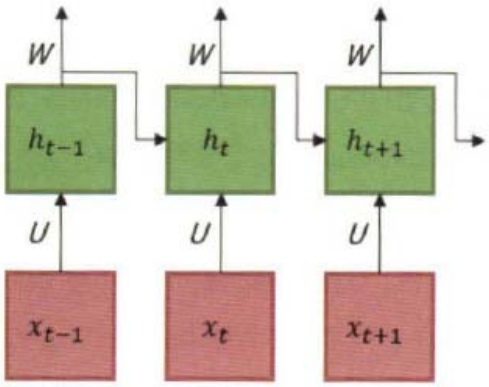


그림 7.4 SimpleRNN 레이어의 구조[2]

i. x_{t-1}, x_t, x_{t+1}: time series 입력

ii. h_{t-1},h_t, h_{t+1}: 출력

iii. U, W:가중치

iv. h_t = tanh(U * x_{t-1} +W * h_{t-1})

F. SimpleRNN layer 생성 코드

```
# 7.1 SimpleRNN 레이어 생성 코드
rnn1 = tf.keras.layers.SimpleRNN(units=1, activation='tanh', return_sequences=True)
```

i. Return_sequences=true: 출력으로 sequence 전체를 사용

ii. SimpleRNN 사용예: [0.0, 0.1, 0.2, 0.3] 입력에 대하여 0.4를 예측하는 것이 목표

```
# 7.2 시퀀스 예측 데이터 생성
X = []
Y = []
for i in range(6):
    # [0,1,2,3], [1,2,3,4] 같은 정수의 시퀀스를 만듭니다.
    lst = list(range(i,i+4))

    # 위에서 구한 시퀀스의 숫자들을 각각 10으로 나눈 다음 저장합니다.
    # SimpleRNN 에 각 타임스텝에 하나씩 숫자가 들어가기 때문에 여기서도 하나씩 분리해서 배열에 저장합니
    X.append(list(map(lambda c: [c/10], lst)))

    # 정답에 해당하는 4, 5 등의 정수를 역시 위처럼 10으로 나눠서 저장합니다.
    Y.append((i+4)/10)

X = np.array(X)
Y = np.array(Y)
for i in range(len(X)):
    print(X[i], Y[i])
```

```
[[0. ]
 [0.1]
 [0.2]
 [0.3]] 0.4
[[0.1]
 [0.2]
 [0.3]
 [0.4]] 0.5
[[0.2]
 [0.3]
 [0.4]
 [0.5]] 0.6
[[0.3]
 [0.4]
 [0.5]
 [0.6]] 0.7
[[0.4]
 [0.5]
 [0.6]
 [0.7]] 0.8
[[0.5]
 [0.6]
 [0.7]
 [0.8]] 0.9
```

1. X는 [6,4,1]: 6은 batch 차원(samples), 4는 timesteps, 1은 input_dim

iii. Sequence 예측 모델 정의

```python
# 7.3 시퀀스 예측 모델 정의
model = tf.keras.Sequential([
    tf.keras.layers.SimpleRNN(units=10, return_sequences=False, input_shape=[4,1]),
    tf.keras.layers.Dense(1)
])

model.compile(optimizer='adam', loss='mse')
model.summary()
```

1. Input_shape = [4,1]

   A. 4는 timesteps: RNN이 입력에 대하여 반복하는 횟수

   B. 1은 input_dim: 입력 벡터의 크기

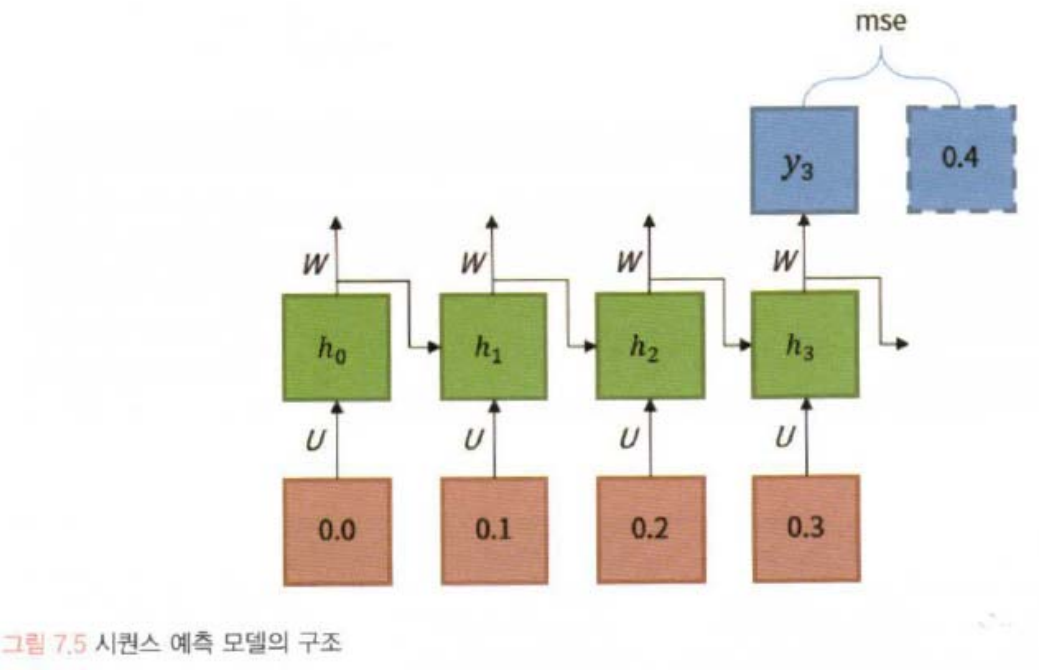


그림 7.5 시퀀스 예측 모델의 구조

iv. SimpleRNN 모델의 학습

```python
# 7.4 네트워크 훈련 및 결과 확인
model.fit(X, Y, epochs=100, verbose=0)
print(model.predict(X))
```

```
1/1 [==============================] - 0s 140ms/step
[[0.3810786 ]
 [0.5046522 ]
 [0.61686313]
 [0.7161935 ]
 [0.80267614]
 [0.877167  ]]
```

v. 학습에 사용되지 않은 sequence에 대한 예측 결과

```
# 7.5 학습되지 않은 시퀀스에 대한 예측 결과
print(model.predict(np.array([[[0.6],[0.7],[0.8],[0.9]]])))
print(model.predict(np.array([[[-0.1],[0.0],[0.1],[0.2]]])))
```

```
1/1 [==============================] - 0s 15ms/step
[[0.9952342]]
1/1 [==============================] - 0s 13ms/step
[[0.294714]]
```

5. LSTM layer

   A. SimpleRNN layer의 단점: 입력 데이터가 길어질수록, 즉 timesteps이 증가할수록 학습능력이 떨어짐

      i. Long-Term Dependency 문제라고 함

         1. 입력 데이터와 출력 사이의 길이가 멀어질수록 연관 관계가 적어 학습능력이 떨어짐

         2. RNN은 현재의 답을 찾기 위해 과거 데이터에 의존, 그러나 과거 시점이 현재와 멀어지면 현재 학습 결과에 나쁜 영향을 준다

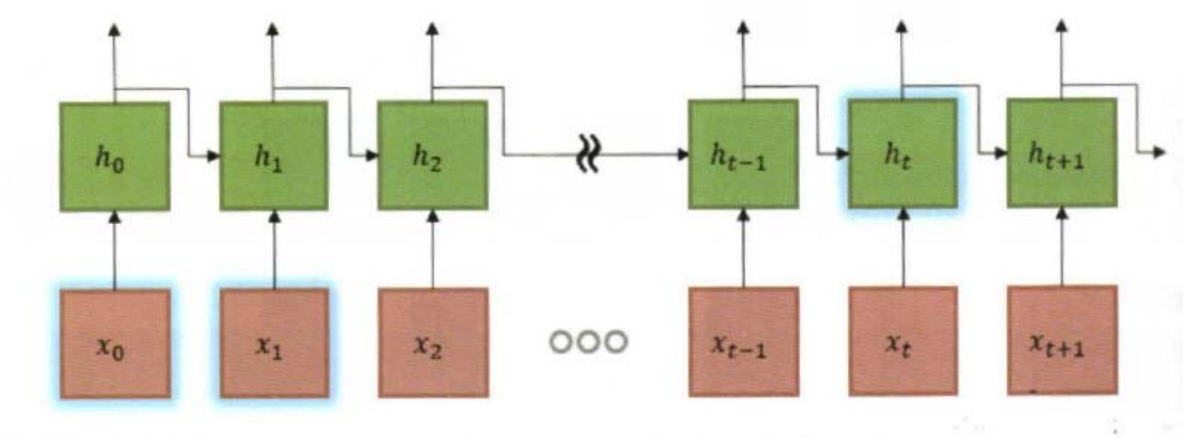


그림 7.6 RNN의 장기의존성 문제. 입력의 길이가 길어지면 먼 거리의 정보를 잘 처리하지 못합니다.

   B. Long Term Dependency 문제의 해결

      i. LSTM(long Short Term Memory), 97년 Sepp Hochreiter & Jurgen Schmidhuber 제안

         1. 출력 이외에 LSTM cell 사이에만 공유되는 cell state을 가지고 있음

      ii. RNN을 cell로 표현한 흐름

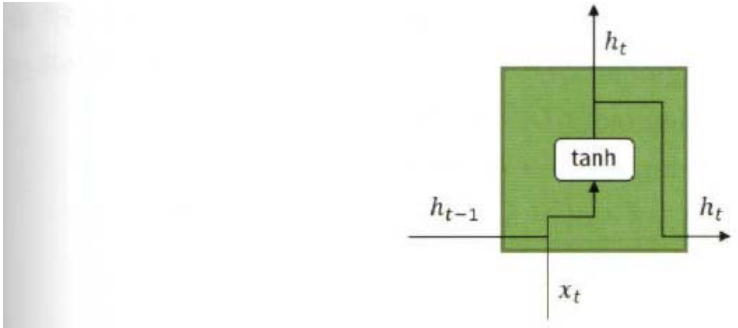


그림 7.7 셀로 나타낸 SimpleRNN 레이어의 계산 흐름

1. $h_t = \tanh(h_{t-1} * x_t)$

iii. LSTM을 cell로 표현

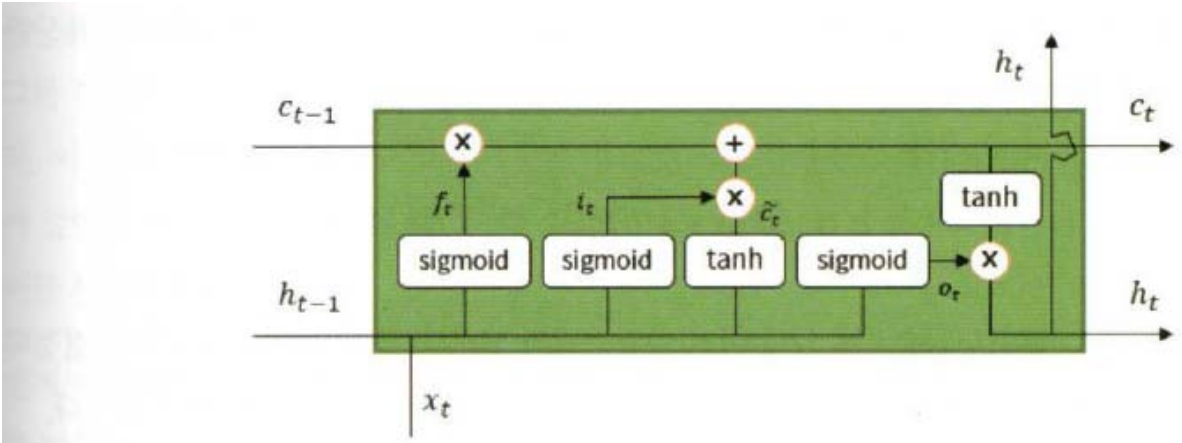


그림 7.8 셀로 나타낸 LSTM 레이어의 계산 흐름

1. LSTM cell state: c_{t-1}

   A. LSTM cell은 cell state인 c_t를 다음 cell에 전달

   B. Timestep을 지나면서 cell state가 보존 전달되므로 long term dependency가 해결된다: LSTM key idea

   C. Sigmoid(): 0 ~ 1

      i. 0이면 입력 정보가 통과 못함

      ii. 1이면 100% 통과

$$i_t = \text{sigmoid}(x_t U^i + h_{t-1} W^i)$$
$$f_t = \text{sigmoid}(x_t U^f + h_{t-1} W^f)$$
$$o_t = \text{sigmoid}(x_t U^o + h_{t-1} W^o)$$
$$\tilde{c}_t = \tanh(x_t U^c + h_{t-1} W^c)$$
$$c_t = f_t \times c_{t-1} + i_t \times \tilde{c}_t$$
$$h_t = \tanh(c_t) \times o_t$$

2. U,W는 가중치
3. I_t, f_t, o_t: timestep t에서의 input, forget, output

$\tilde{c}_t$는 SimpleRNN에도 존재하던 x_t와 h_{t-1}을 각각 U와 W에 곱한 뒤에 tanh 활성화함수를 취한 값으로, 셀 상태인 c^t가 되기 전의 출력값입니다.

4. Cell state와 LSTM 출력 계산

$$c_t = f_t \times c_{t-1} + i_t \times \tilde{c}_t$$

A. Cell state은 forget gate출력에 의해 이전 timestep의 cell state을 남길지를 결정

B. 입력 게이트 출력과 $\tilde{c}_t$ 곱한 값에 더하여 다음 timestep의 cell 상태를 결정

C. LSTM 학습능력 테스트 예제 문제

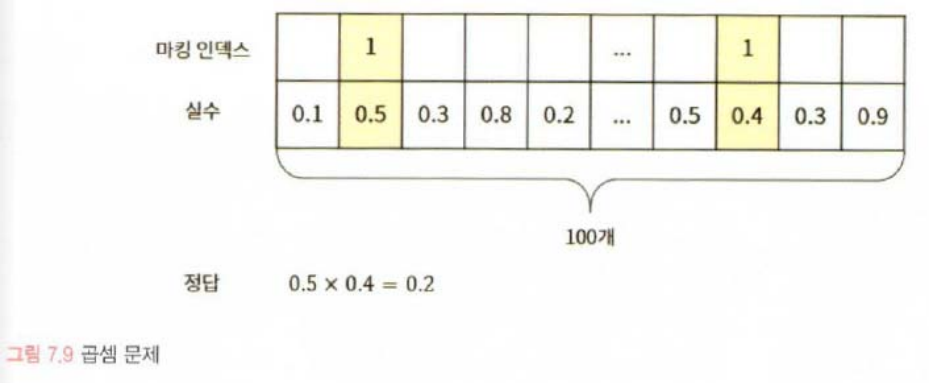


그림 7.9 곱셈 문제

1. 100개 실수 중에서 marking index는 0.5, 0.4에 표시됨
2. 정답: 0.5 * 0.4 = 0.2

ii. SimpleRNN으로 문제 해결

```
# 7.6 곱셈 문제 데이터 생성
X = []
Y = []
for i in range(3000):
    # 0~1 사이의 랜덤한 숫자 100 개를 만듭니다.
    lst = np.random.rand(100)
    # 마킹할 숫자 2개의 인덱스를 뽑습니다.
    idx = np.random.choice(100, 2, replace=False)
    # 마킹 인덱스가 저장된 원-핫 인코딩 벡터를 만듭니다.
    zeros = np.zeros(100)
    zeros[idx] = 1
    # 마킹 인덱스와 랜덤한 숫자를 합쳐서 X 에 저장합니다.
    X.append(np.array(list(zip(zeros, lst))))
    # 마킹 인덱스가 1인 값들만 서로 곱해서 Y 에 저장합니다.
    Y.append(np.prod(lst[idx]))

print(X[0], Y[0])
```

```
[0.         0.63933116]
[1.         0.55637236]
[0.         0.11837606]
[0.         0.1040563 ]
[0.         0.95372269]
```

1. 3000개 실수 데이터 생성 2569개를 training data + validation data로 사용, 440개는 test data로 사용

iii. SimpleRNN layer를 사용한 곱셈 문제 모델 정의

```
# 7.7 SimpleRNN 레이어를 사용한 곱셈 문제 모델 정의
model = tf.keras.Sequential([
    tf.keras.layers.SimpleRNN(units=30, return_sequences=True, input_shape=[100,2]),
    tf.keras.layers.SimpleRNN(units=30),
    tf.keras.layers.Dense(1)
])

model.compile(optimizer='adam', loss='mse')
model.summary()
```

1. Return_sequence=true > RNN layer를 중첩함

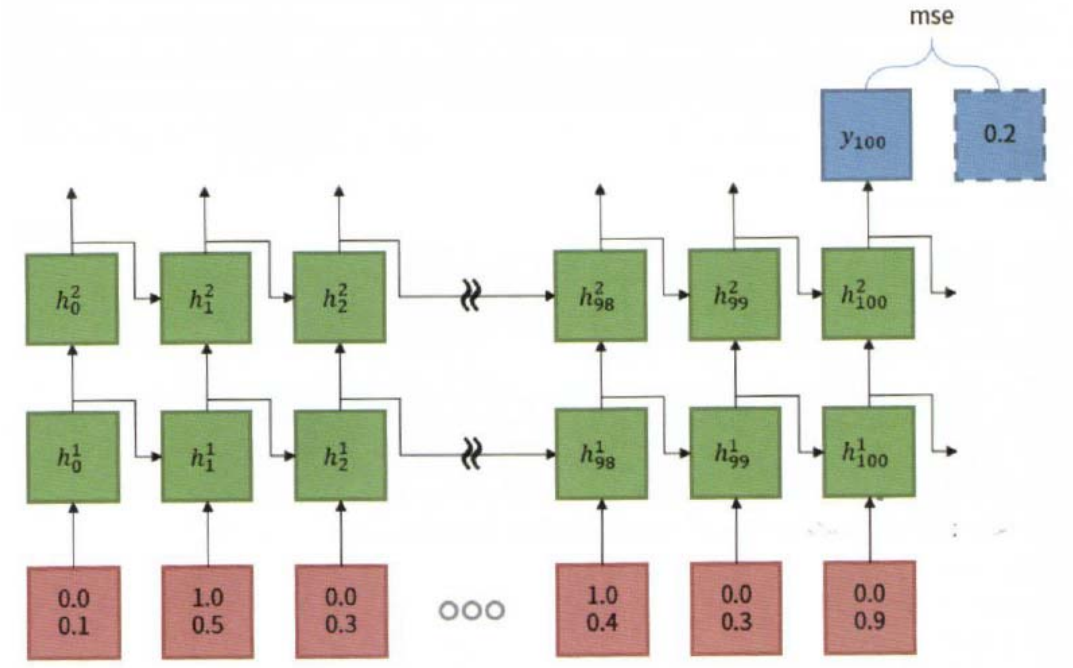


그림 7.10 겹쳐진 SimpleRNN 레이어 구조

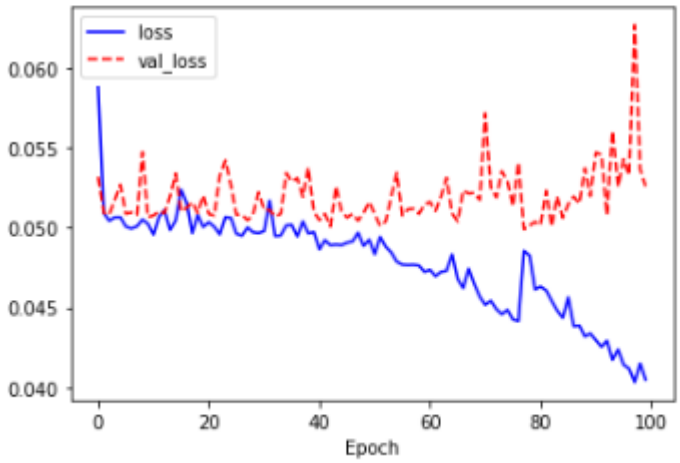
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 simple_rnn_3 (SimpleRNN)     (None, 100, 30)           990

 simple_rnn_4 (SimpleRNN)     (None, 30)                1830

 dense_1 (Dense)              (None, 1)                 31

=================================================================
Total params: 2,851
Trainable params: 2,851
Non-trainable params: 0
_________________________________________________________________
```

iv. SimpleRNN 학습

```
# 7.8 SimpleRNN 네트워크 학습
X = np.array(X)
Y = np.array(Y)
# 2560개의 데이터만 학습시킵니다. validation 데이터는 20% 로 지정합니다.
history = model.fit(X[:2560], Y[:2560], epochs=100, validation_split=0.2)
```

v. simpleRNN 학습 결과

```python
# 7.9 SimpleRNN 네트워크 학습 결과 확인
import matplotlib.pyplot as plt
plt.plot(history.history['loss'], 'b-', label='loss')
plt.plot(history.history['val_loss'], 'r--', label='val_loss')
plt.xlabel('Epoch')
plt.legend()
plt.show()
```



1. val_loss가 증가하는 전형적인 overfitting 사례

vi. test data에 대한 예측 정확도

```python
# 7.10 Test 데이터에 대한 예측 정확도 확인
model.evaluate(X[2560:], Y[2560:])
prediction = model.predict(X[2560:2560+5])
# 5개 테스트 데이터에 대한 예측을 표시합니다.
for i in range(5):
    print(Y[2560+i], '₩t', prediction[i][0], '₩tdiff:', abs(prediction[i][0] - Y[2560+i]))

prediction = model.predict(X[2560:])
fail = 0
for i in range(len(prediction)):
    # 오차가 0.04 이상이면 오답입니다.
    if abs(prediction[i][0] - Y[2560+i]) > 0.04:
        fail += 1
print('correctness:', (440 - fail) / 440 * 100, '%')
```

```
14/14 [==============================] - 0s 12ms/step - loss: 0.0497
1/1 [==============================] - 0s 259ms/step
0.1162097578656399      0.27171695      diff: 0.15550719445827366
0.19848398653168728     0.30508453      diff: 0.1066005400071616
0.05628569993520576     0.21438465      diff: 0.15809894528841179
0.03480691558825324     0.28667453      diff: 0.2518676137257879
0.022800598289235353    0.31104067      diff: 0.28824007139040636
14/14 [==============================] - 0s 12ms/step
correctness: 9.772727272727273 %
```

1. test data에 대한 정확도: loss가 0.0497
2. 학습과정에서 100번째 epoch의 loss는 0.0405보다 높다
3. Test data에 대한 정확도는 9.77%

D. LSTM layer로 곱셈 문제 풀기

```
# 7.11 LSTM 레이어를 사용한 곱셈 문제 모델 정의
model = tf.keras.Sequential([
    tf.keras.layers.LSTM(units=30, return_sequences=True, input_shape=[100,2]),
    tf.keras.layers.LSTM(units=30),
    tf.keras.layers.Dense(1)
])

model.compile(optimizer='adam', loss='mse')
model.summary()
```

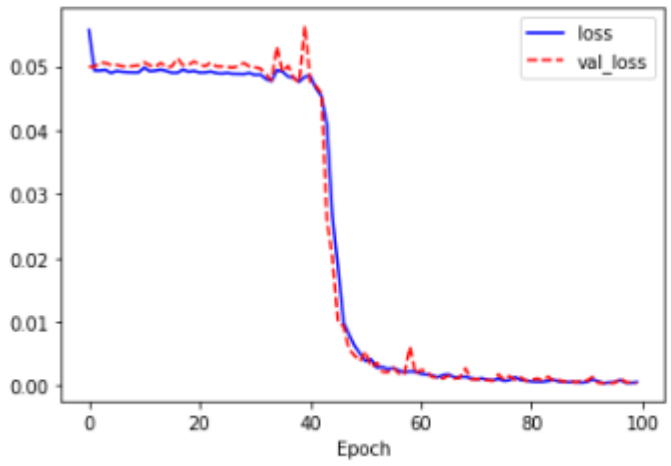
```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 100, 30)           3960

 lstm_1 (LSTM)               (None, 30)                7320

 dense_2 (Dense)             (None, 1)                 31

=================================================================
Total params: 11,311
Trainable params: 11,311
Non-trainable params: 0
_________________________________________________________________
```

i. LSTM layer로 학습

```
# 7.12 LSTM 네트워크 학습
X = np.array(X)
Y = np.array(Y)
history = model.fit(X[:2560], Y[:2560], epochs=100, validation_split=0.2)
```

ii. LSTM 네트워크 학습 결과

```python
# 7.13 LSTM 네트워크 학습 결과 확인
import matplotlib.pyplot as plt
plt.plot(history.history['loss'], 'b-', label='loss')
plt.plot(history.history['val_loss'], 'r--', label='val_loss')
plt.xlabel('Epoch')
plt.legend()
plt.show()
```



iii. Test data에 대한 예측 정확도 확인

```python
# 7.14 Test 데이터에 대한 예측 정확도 확인
model.evaluate(X[2560:], Y[2560:])
prediction = model.predict(X[2560:2560+5])
for i in range(5):
    print(Y[2560+i], '₩t', prediction[i][0], '₩tdiff:', abs(prediction[i][0] - Y[2560+i]))

prediction = model.predict(X[2560:])
cnt = 0
for i in range(len(prediction)):
    if abs(prediction[i][0] - Y[2560+i]) > 0.04:
        cnt += 1
print('correctness:', (440 - cnt) / 440 * 100, '%')
```

```
14/14 [==============================] - 1s 21ms/step - loss: 3.6632e-04
1/1 [==============================] - 1s 789ms/step
0.11620975786563399     0.14598909     diff: 0.029779332338598988
0.19848398653168728     0.19867322     diff: 0.00018923195700595863
0.05628569993520576     0.058790404    diff: 0.002504704414359744 4
0.03480691558825324     0.06369543     diff: 0.028888515167361996
0.022800598289235353    0.029333517    diff: 0.006532918666140319
14/14 [==============================] - 0s 20ms/step
correctness: 95.68181818181817 %
```

1. Loss와 val_loss가 40 epoch를 지나면서 0에 가까워짐
2. Test data에 대한 정확도: 95.681%

6. GRU 레이어
7. Embedding layer – 자연언어의 단어를 숫자로 바꾸는 것
8. 긍정,부정 감성 분석
9. 자연언어 생성