Functional API 기초

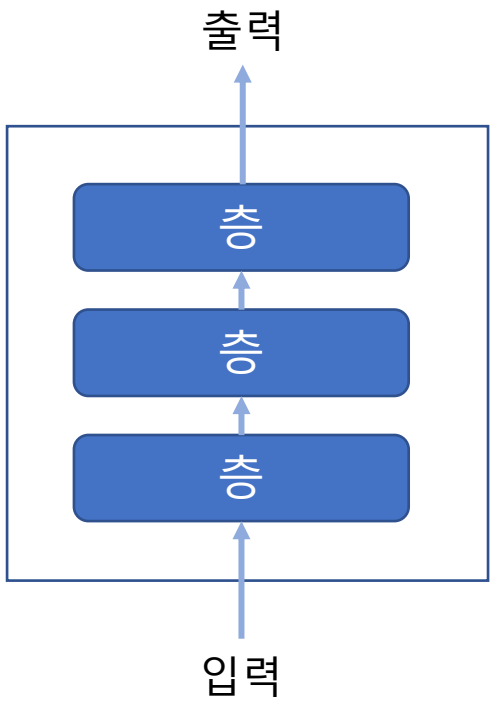
최석재 *lingua@naver.com*

함수형 API

Functional API

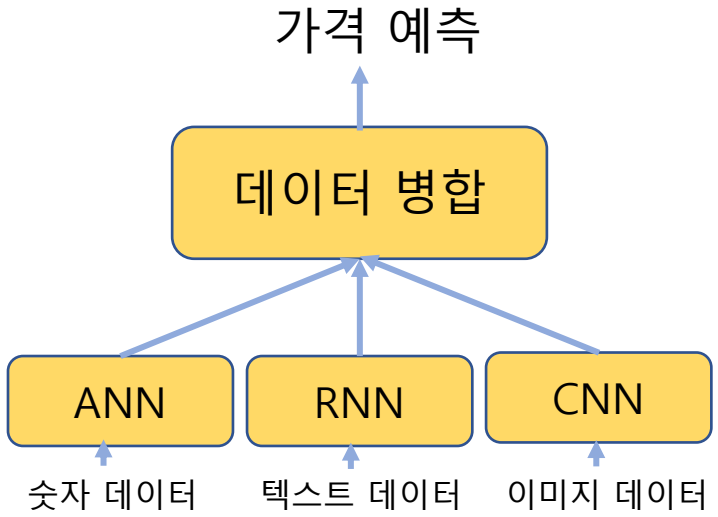
Sequential 모델

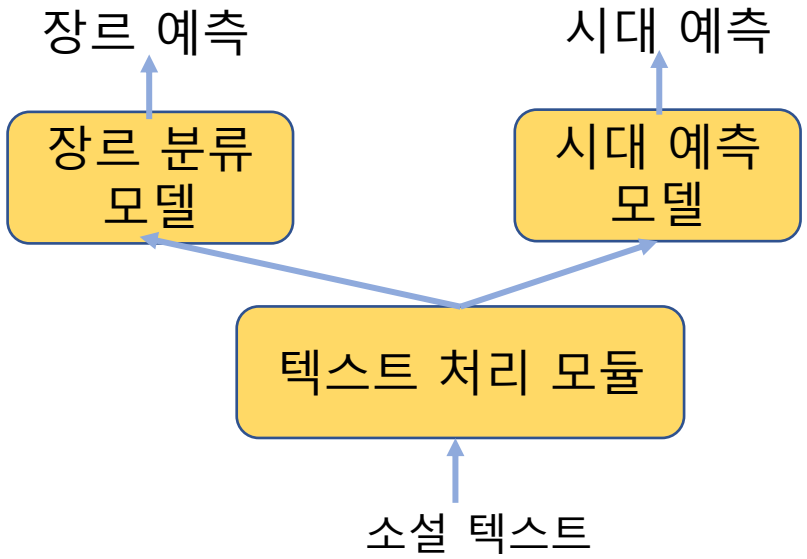
- Sequential 모델은 네트워크의 입력과 출력이 하나라고 가정한다

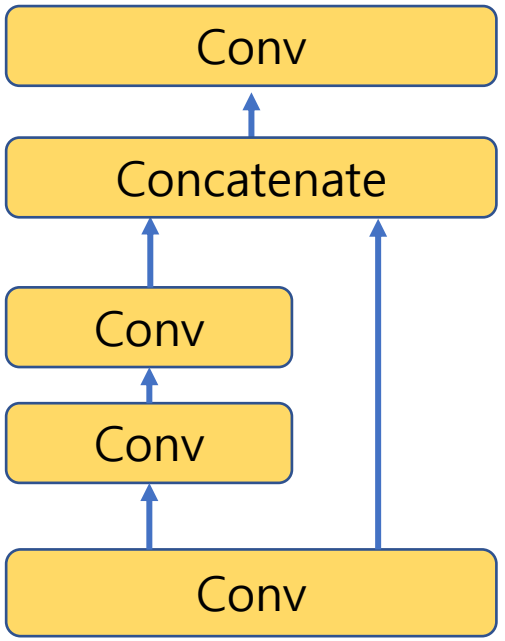


함수형 API *Functional API*

- 함수형 API는 다중입력, 다중출력, 비선형 연결 등 다양한 구조를 지원한다







Sequential과 함수형 API 비교

Sequential에서도 입력 부분을 별도로 분리할 수 있다

```
model = Sequential()
model.add(Input(shape=(64,)))
model.add(layers.Dense(32, activation='relu')
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(8, activation='softmax'))
```

- 단순한 모델로 Sequential과 Functional 을 비교한다
- Sequential 모델은 층을 하나하나 쌓아가면서 모델을 만드는 구조이다
- Functional API는 input에서 output으로 가는 과정을 설정하고,
  이를 Model() 함수의 inputs와 outputs에 각각 넣어 모델을 만드는 구조이다

Sequential

```
model = Sequential()
model.add(layers.Dense(32, activation='relu', input_shape=(64,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(8, activation='softmax'))
```

Functional API

```
input_tensor = Input(shape=(64,))
x = layers.Dense(32, activation='relu')(input_tensor)
x = layers.Dense(16, activation='relu')(x)
output_tensor = layers.Dense(8, activation='softmax')(x)
model = Model(inputs=input_tensor, outputs=output_tensor)
```

input에서 output까지의 과정을 연결하고, 이를 Model 클래스에 전달한다 Model(인풋, 아웃풋)

Keras Model의 종류

- Keras를 이용해 Model을 만드는 방법으로는 세 가지가 있다

1. Sequential Model
2. Functional API
3. Model subclassing

Models API

There are three ways to create Keras models:

- The Sequential model, which is very straightforward (a simple list of layers), but is limited to single-input, single-output stacks of layers (as the name gives away).
- The Functional API, which is an easy-to-use, fully-featured API that supports arbitrary model architectures. For most people and most use cases, this is what you should be using. This is the Keras "industry strength" model.
- Model subclassing, where you implement everything from scratch on your own. Use this if you have complex, out-of-the-box research use cases.

https://keras.io/api/models/

기본 사용법

MNIST 데이터로 CNN 구현하기

- MNIST 데이터로 CNN 모델을 만드는 과정을 Functional API로 구현해본다

- from keras.datasets import mnist
- (train_images, train_labels), (test_images, test_labels) = mnist.load_data()

- print(train_images.shape) # (60000, 28, 28)
- print(test_images.shape) # (10000, 28, 28)

- print(train_labels) # [5 0 4 ... 5 6 8]
- print(test_labels)

데이터 전처리

- train_images = train_images.reshape((60000, 28, 28, 1))
- train_images = train_images.astype('float32')/255

- test_images = test_images.reshape((10000, 28, 28, 1))
- test_images = test_images.astype('float32')/255

- from keras.utils import to_categorical
- train_labels = to_categorical(train_labels)
- test_labels = to_categorical(test_labels)

※ from keras.utils import to_categorical 로 잘 안되면
from tensorflow.keras.utils import to_categorical 로 변경

모델 설계

- from keras import layers

- inputs = layers.Input(shape=(28, 28, 1))

- conv1 = layers.Conv2D(32, kernel_size=(3, 3), activation='relu')(inputs)
- conv2 = layers.Conv2D(32, kernel_size=(3, 3), activation='relu')(conv1)
- pool = layers.MaxPooling2D(pool_size=2)(conv2)
- drop1 = layers.Dropout(0.25)(pool)
- flat = layers.Flatten()(drop1)
- dense1 = layers.Dense(128, activation='relu')(flat)
- drop2 = layers.Dropout(0.25)(dense1)

- outputs = layers.Dense(10, activation='softmax')(drop2)

inputs와 outputs를 제외한 중간층은 재사용할 것이 아니므로 모두 x라고 하여도 무방하다

모델 컴파일

- from keras.models import Model

- model = Model(inputs=inputs, outputs=outputs)
- model.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=['acc'])
- model.summary()

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 28, 28, 1)]       0
conv2d (Conv2D)              (None, 26, 26, 32)        320
conv2d_1 (Conv2D)            (None, 24, 24, 32)        9248
max_pooling2d (MaxPooling2D  (None, 12, 12, 32)        0
)
dropout (Dropout)            (None, 12, 12, 32)        0
flatten (Flatten)            (None, 4608)              0
dense (Dense)                (None, 128)               589952
dropout_1 (Dropout)          (None, 128)               0
dense_1 (Dense)              (None, 10)                1290
=================================================================
Total params: 600,810
Trainable params: 600,810
Non-trainable params: 0
_________________________________________________________________
```

모델 훈련

- history = model.fit(train_images, train_labels, epochs=5, batch_size=128, validation_data=(test_images, test_labels))

```
Epoch 1/5
469/469 [==============================] - 73s 154ms/step - loss: 0.2187 - acc: 0.9320 - val_loss: 0.0487 - val_acc: 0.9849
Epoch 2/5
469/469 [==============================] - 71s 151ms/step - loss: 0.0670 - acc: 0.9791 - val_loss: 0.0372 - val_acc: 0.9881
Epoch 3/5
469/469 [==============================] - 71s 151ms/step - loss: 0.0474 - acc: 0.9856 - val_loss: 0.0339 - val_acc: 0.9888
Epoch 4/5
469/469 [==============================] - 71s 151ms/step - loss: 0.0377 - acc: 0.9887 - val_loss: 0.0361 - val_acc: 0.9878
Epoch 5/5
469/469 [==============================] - 71s 151ms/step - loss: 0.0327 - acc: 0.9897 - val_loss: 0.0305 - val_acc: 0.9909
```

예측 및 평가

- test_loss, test_acc = model.evaluate(test_images, test_labels)
- print('test_acc:', test_acc)

```
313/313 [==============================] - 4s 11ms/step - loss: 0.0305 - acc: 0.9909
test_acc: 0.9908999800682068
```

※ 성능은 Sequential CNN 모델과 동일하다

Accuracy & Loss 확인

- acc = history.history['acc']
- val_acc = history.history['val_acc']
- loss = history.history['loss']
- val_loss = history.history['val_loss’]

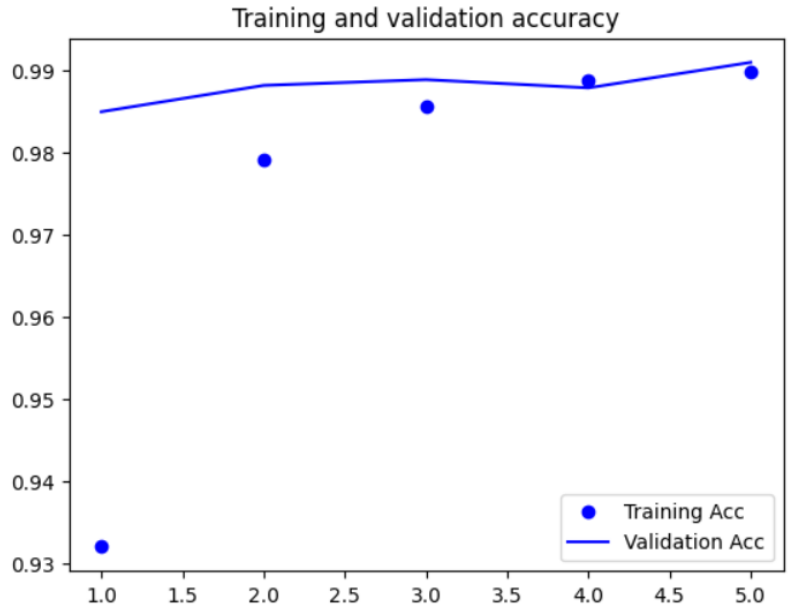
- print('Accuracy of each epoch:', acc)
- epochs = range(1, len(acc) +1)

```
Accuracy of each epoch: [0.932033360004425, 0.9791333079338074, 0.9855833053588867, 0.9886999726295471, 0.9897333383560181]
```

정확도 그래프

- import matplotlib.pyplot as plt

- plt.plot(epochs, acc, 'bo', label='Training Acc')
- plt.plot(epochs, val_acc, 'b', label='Validation Acc')
- plt.title('Training and validation accuracy')
- plt.legend()



손실값 그래프

- plt.figure() # 새로운 그림을 그린다

- plt.plot(epochs, loss, 'bo', label='Training Loss')
- plt.plot(epochs, val_loss, 'b', label='Validation Loss')
- plt.title('Training and validation loss')
- plt.legend()