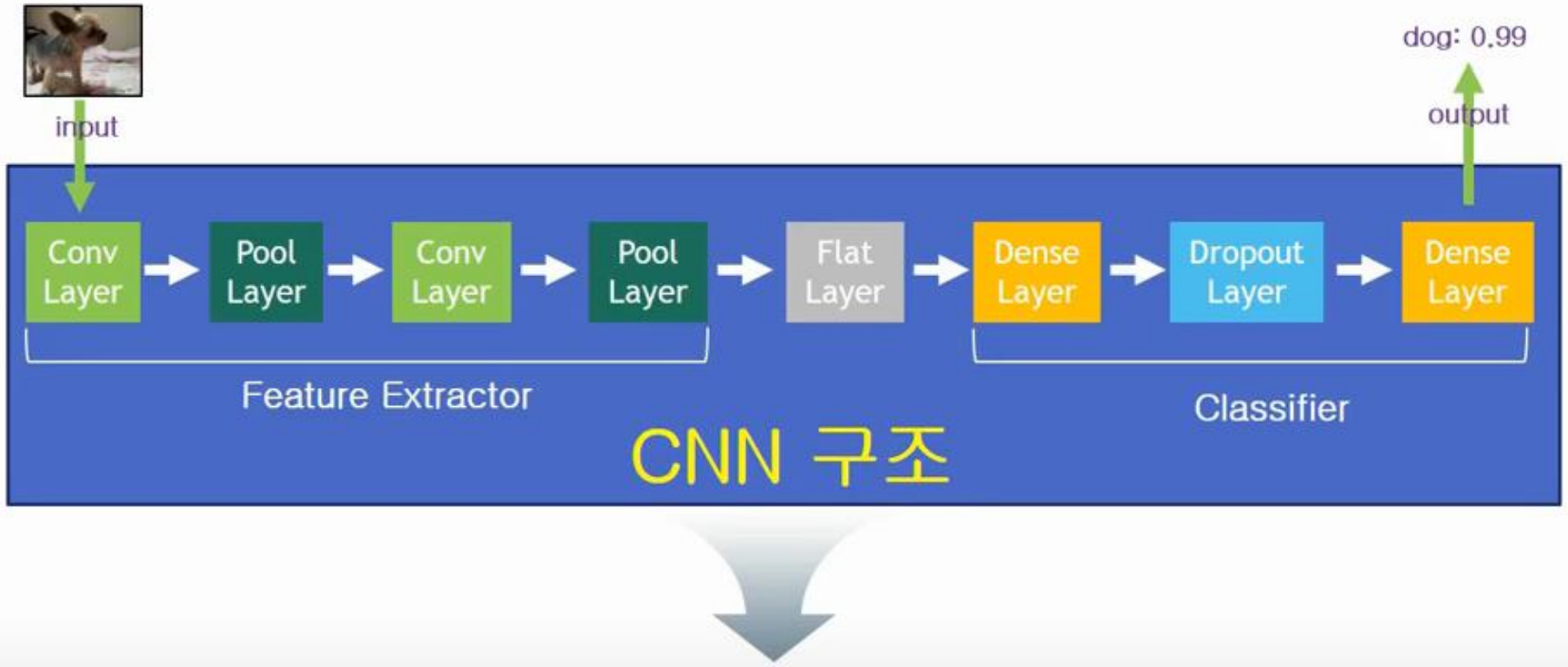
CNN Example

(MNIST, Fashion MNIST, CIFAR-10)

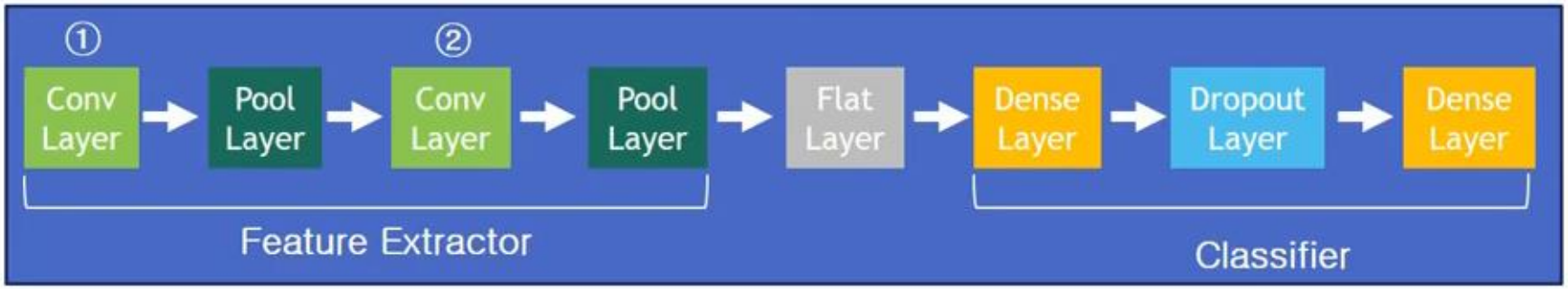
박경미

CNN 구조 => 특징 추출기(Feature Extractor)+분류기(Classifier)



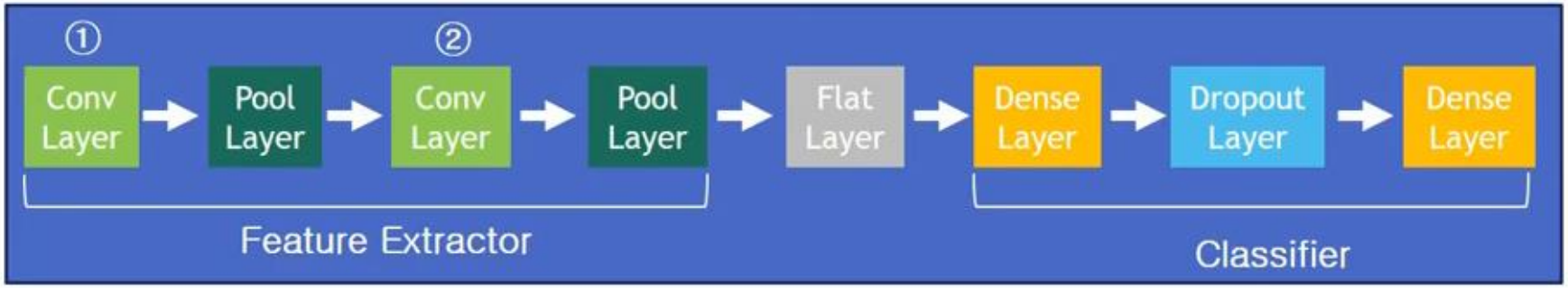
- 이미지를 분류하기 위해서 사용되는 일반적인 컨볼루션 신경망(CNN) 구조는 특징추출기(Feature Extractor)와 분류기(Classifier)가 합쳐져 있는 형태임
  - 특징 추출기(Feature Extractor)는 컨볼루션 레이어(Conv Layer)와 풀링 레이어(Pool Layer) 다양한 조합으로 구성되며, 이미지 데이터의 특징을 추출하고 압축하는 역할 수행
  - 분류기(Classifier)부분은 완전연결층인 Dense Layer와 과적합(overfitting)을 방지하기 위한 Dropout Layer등의 다양한 조합으로 구성되며, 정답을 분류하는 역할 수행

TensorFlow CNN API – Conv Layer . Pool Layer



Layer	TensorFlow 2.x API	Description
Conv	Conv2D(input_shape=(28, 28, 1), kernel_size=(3, 3) filters=32, strides=(1, 1), activation='relu', use_bias=True, padding='valid')	CNN 에서 1st Conv Layer (①)를 나타내기 위해 (높이, 너비, 채널) 형태의 텐서로 주어지는 input_shape 사용하며, 필터 크기(kernel_size). 필터 개수(filters)는 반드시 기술되어야 함. 필터 크기는 (높이, 너비) 형태이며 숫자를 하나만 쓸 경우 높이와 너비가 동일 값 이라는 것을 의미함
	Conv2D(kernel_size=(3, 3) filters=32, strides=(1, 1), activation='relu', use_bias=True, padding='valid')	2nd Conv Layer (②) 부터 input_shape 사용되지 않음. 필터 크기(kernel_size)와 필터 개수(filters) 값을 지정하고 나머지 파라미터는 기본 값만을 이용하여도 컨벌루션 층 구현 가능함
Pool	MaxPool2D(pool_size=(2, 2), padding='valid'))	한번에 Max Pooling 수행할 범위를 pool_size에 나타냄. pool_size=(2, 2) 라면 높이 2, 너비 2 사각형 안에서 최대값만 남기는 연산을 수행함

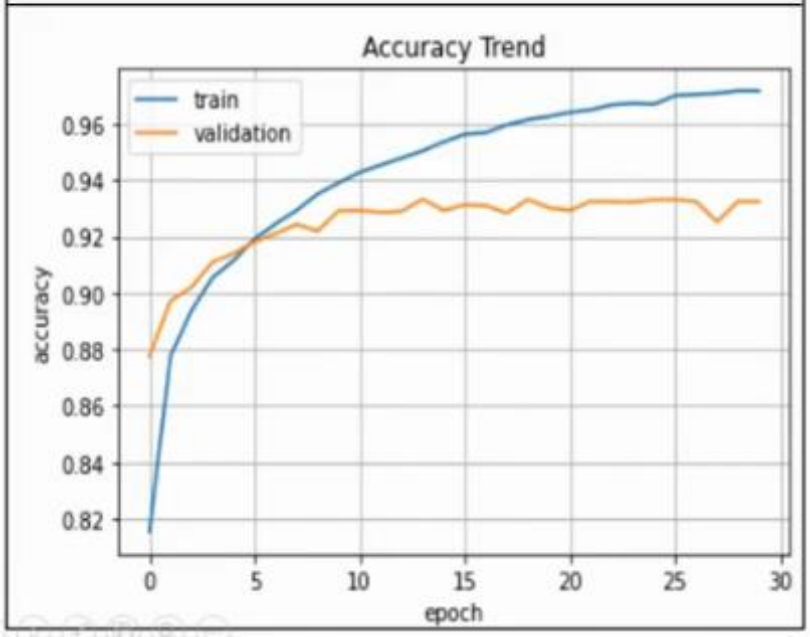
TensorFlow CNN API – Flat Layer . Dense Layout . Dropout Layer



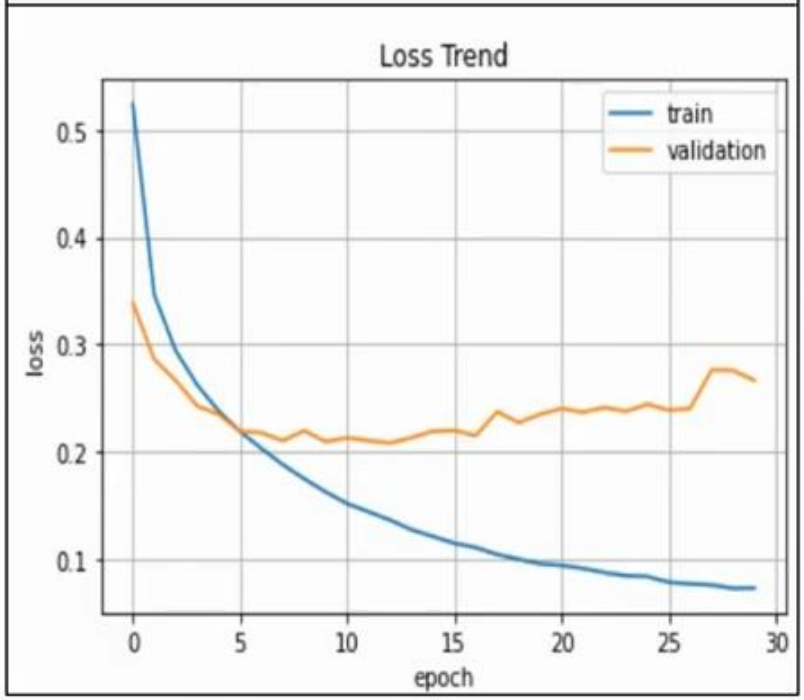
Layer	TensorFlow 2.x API	Description
Flat	Flatten()	Feature Extractor 출력은 (높이, 너비, 채널) 로 나타나는 3차원 텐서이므로, 완전연결층(Dense) 과의 연결을 위해서 3차원 텐서를 1차원 vector 로 만들어주는 역할을 수행함
Dense	Dense(100, activation='relu') Dense(100, activation='sigmoid') Dense(10, activation='softmax')	Dense 함수는 신경망에서 은닉층과 출력층을 의미하는 완전연결층을 나타내며, 1st 파라미터는 완전연결층의 출력 노드 수 이며, 활성화 함수는 activation='...' 나타냄
Dropout	Dropout(rate=0.2)	학습과정중에 rate 에 지정된 비율만큼 랜덤하게 층(layer)과 층 사이의 연결을 끊어서 네트워크의 과적합(overfitting)을 막는 역할을 수행함

TensorFlow CNN API- Accuracy . Loss

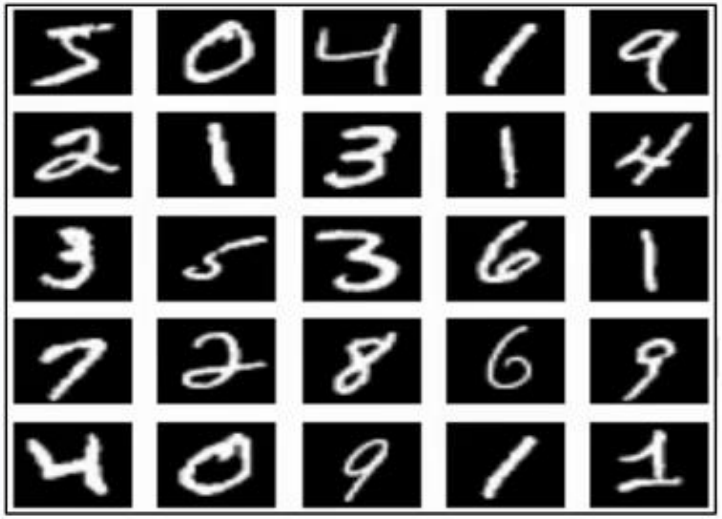
```
import matplotlib.pyplot as plt

plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title('Accuracy Trend')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train','validation'], loc='best')
plt.grid()
plt.show()
```




```
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('Loss Trend')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train','validation'], loc='best')
plt.grid()
plt.show()
```



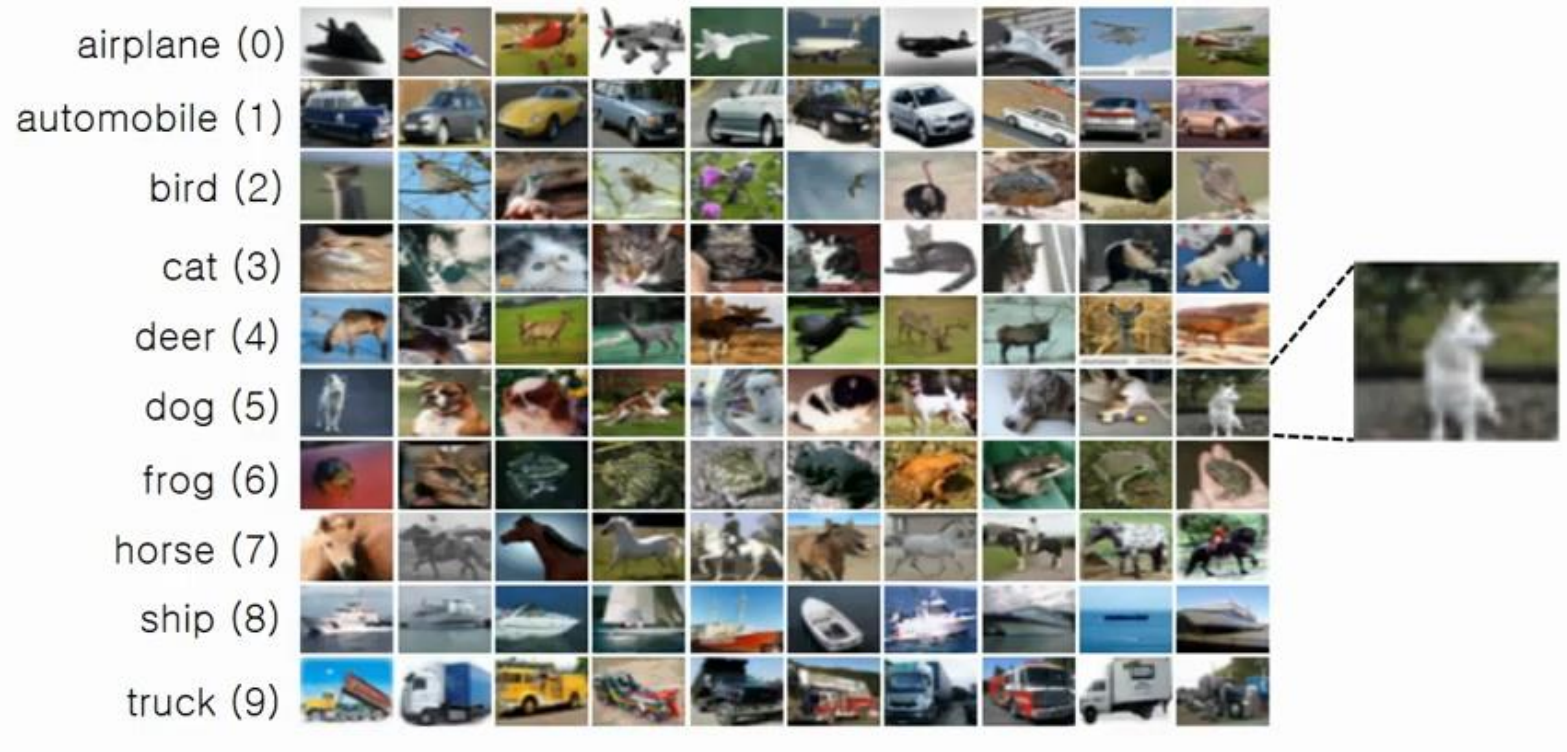
Accuracy Comparison – ANN vs CNN

	MNIST	Fashion MNIST
Training Data		
ANN Accuracy	97.50 %	88.10 %
CNN Accuracy	99.39 %	93.25 %

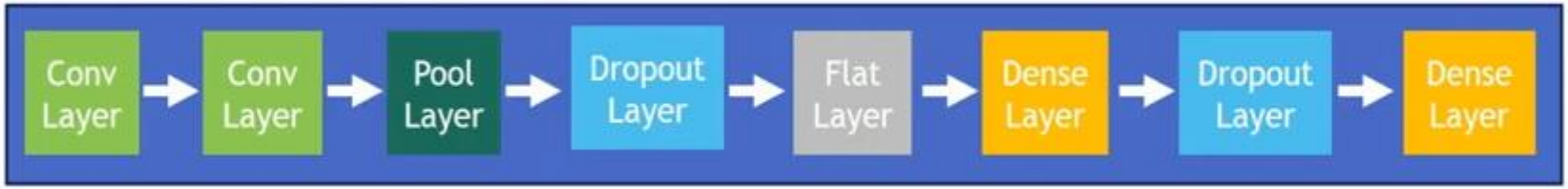
Training-Data CIFAR 10

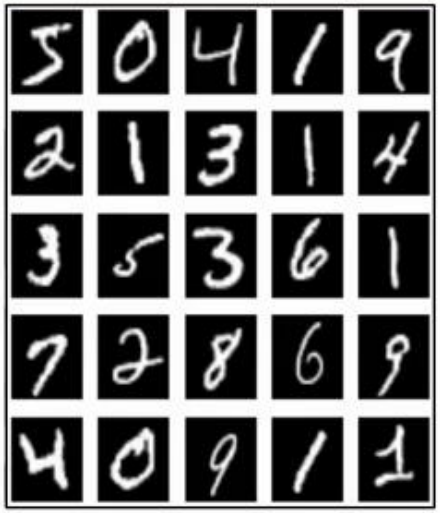
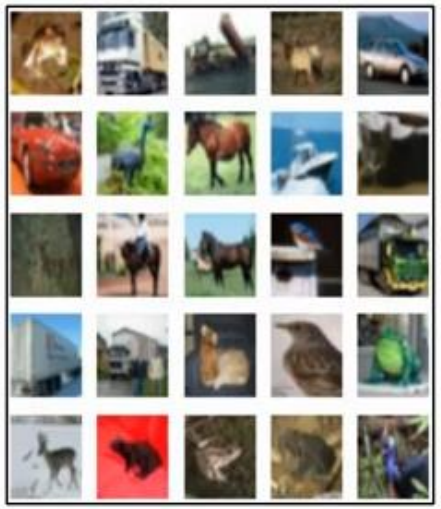
❖ CIFAR 10 개요

- airplane, automobile, bird 등 10개의 정답으로 분류된 이미지이며, 딥러닝 학습을 위해 총 50000개의 학습데이터와 10000개의 테스트 데이터로 구성
- 이미지는 32 X 32 크기의 작은 컬러 이미지, **즉 30 X 32 X 3 형상(shape)을 가지는 아주 작은 컬러 데이터들로 구성**



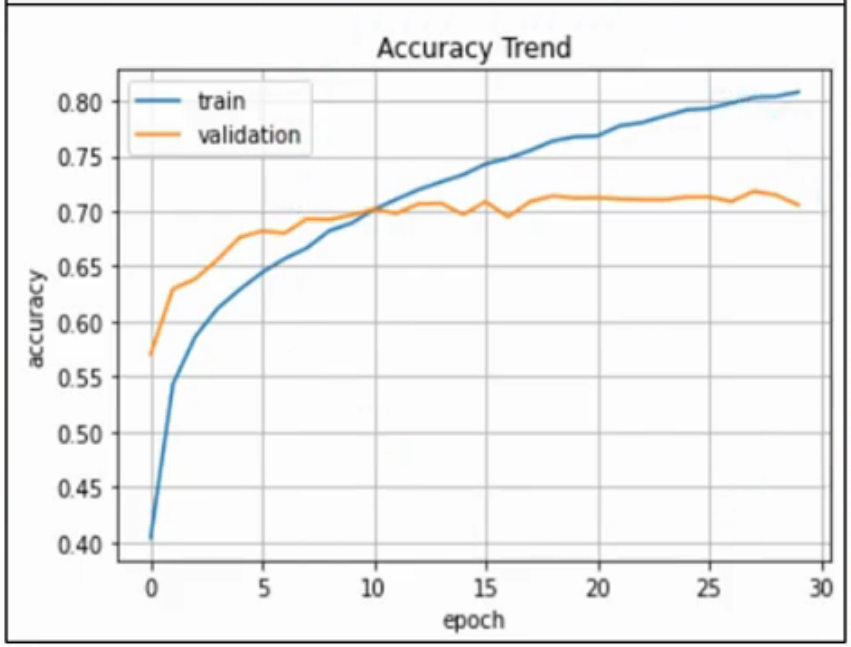
정확도 비교 – MNIST. Fashiom MNIS . CIFAR 10



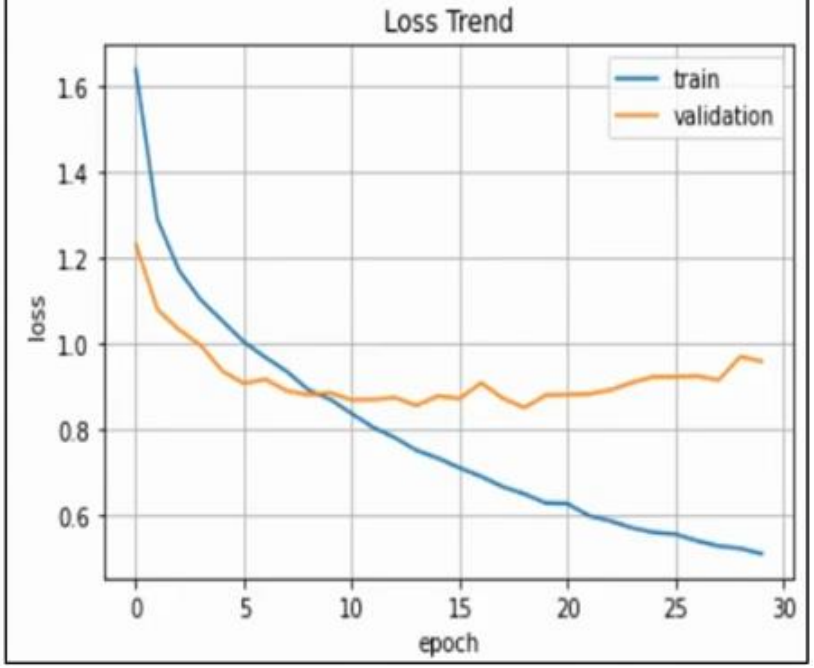
	MNIST	Fashion MNIST	CIFAR 10
Training Data			
CNN Accuracy	99.39 %	93.25 %	???

CIFAR 10 정확도 및 손실

```
import matplotlib.pyplot as plt

plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title('Accuracy Trend')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train','validation'], loc='best')
plt.grid()
plt.show()
```




```
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('Loss Trend')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train','validation'], loc='best')
plt.grid()
plt.show()
```



CNN 성능향상(오버피팅 ↓ . 정확도 ↑)

- 더 많은 레이어(layer) 쌓기
  - 컨볼루션 레이어가 중첩된 더 깊은 구조가 될수록 성능은 크게 개선됨
  - AlexNet(2012, 8 layers), VGGNet(2014, 19 layers), GoogleLeNet(2014, 22 layers), ResNet(2015, 152 layers)....

- 이미지 데이터 보강 (Image Data Augmentation)
  - 답러닝에서는 많은 학습 데이터를 사용하면 성능을 개선 시킬 수 있음
  - 기존의 (이미지) 데이터가 있을 때, 해당 데이터를 원본으로 해서 다양한 변형 (rotate, shear, zoom, shift, horizontal flip..)을 주고, 이렇게 생성된 데이터를 원본 학습 데이터에 포함시켜 수 많은 학습 데이터를 확보할 수 있음

- 높은 해상도(High Resolution) 학습 데이터 확보
  - 동일한 CNN구조라면, 상대적으로 높은 해상도의 학습데이터를 통해서 성능을 개선 시킬 수 있음 [CIFAR 10] 32 x 32 ⇔ [ImageNet image] 469 x 387

- L1 Norm · L2 Norm 등의 가중치 규제(Regularization), Dropout, 배치 정규화(Batch Normalization) 등을 통해 성능을 개선 시킬 수 있음

CNN 성능 향상 – Image Data Augmentation

❖ Image Data Augmentation

- 원본 이미지에 적절한 변형을 가해서 새로운 데이터를 만들어 내는 방식
- 즉 원본 이미지를 상하좌우 방향으로 조금 이동하거나, 약간 회전 또는 기울이거나 또는 확대 등 여러가지 변환을 조합해 이미지 데이터 개수를 증가시킴

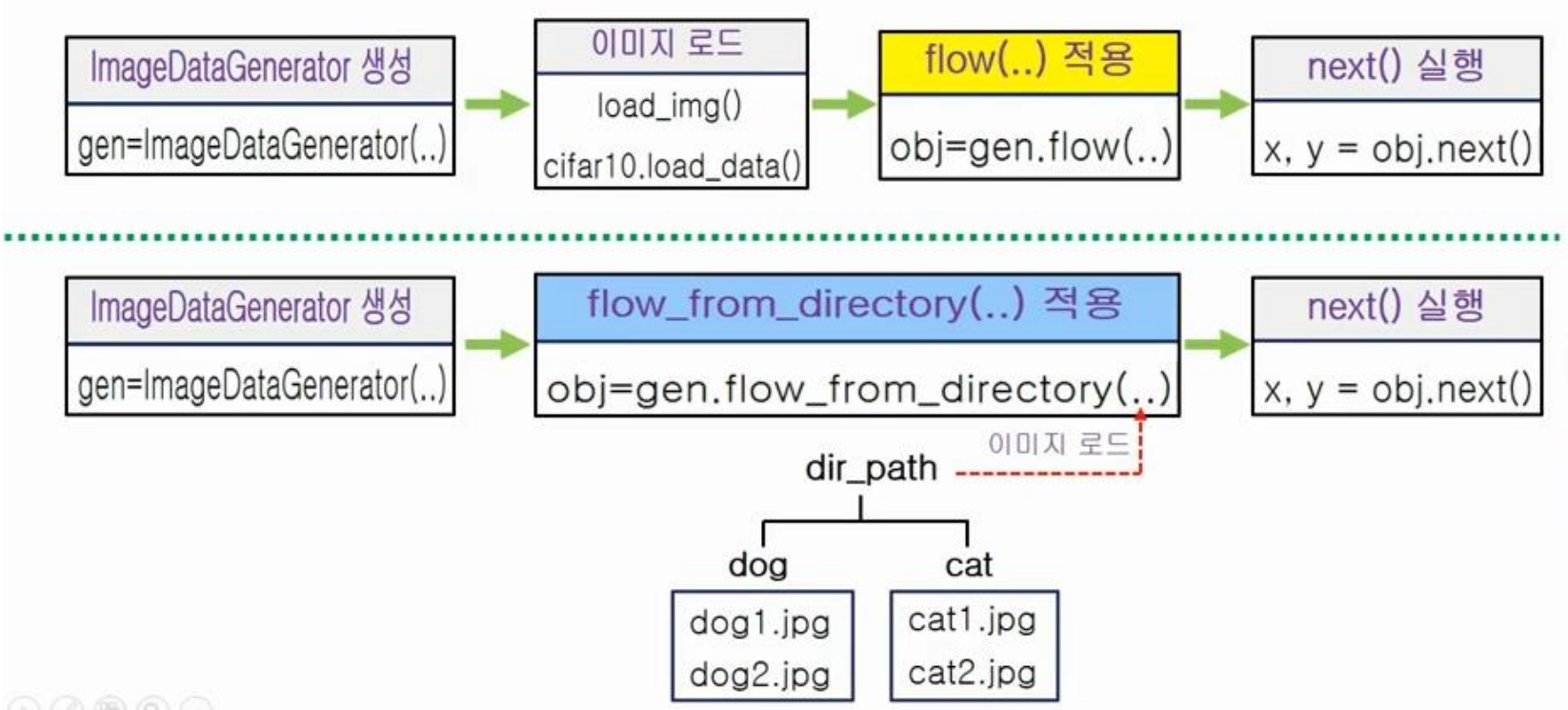
❖ Image Data Augmentation 장점

- 다양한 데이터를 입력시킴으로써 모델을 더욱 견고하게 만들어 주기때문에 실제 데이터를 가지고 테스트 시에 더 높은 성능을 기대할 수 있음
- CNN 모델을 학습시키기에 수집된 데이터가 적은 경우에 강력한 힘을 발휘함

CNN 성능 향상 – Image Data Augmentation

❖ ImageDataGenrator(1)

- 케라스에서는 이미지 데이터 보강을 위한 ImagedataGenrator 제공
- 주요 인수 : rotation_range, width_shift_range, height_shift_range, rescale, shear_range, fhorizontal_flip 등
- flow(), flow directory() 함수를 통해 이미지 데이터를 보강할 수 있음



CNN 성능 향상 – Image Data Augmentation

❖ ImageDataGenrator(2)

[1] ImageDataGenerator 생성 ✓ gen = ImageDataGenerator(rotation_range=30, horizontal_flip=True, rescale=1./255)	ImageDataGenerator() 함수를 이용해 데이터 변형 방식을 설정함. 예를 보면 30도 이내에서 회전, 좌우 반전시도, 색상의 최대값 255 로 나누어 줌으로서 0~1 값으로 정규화 시행하라는 의미임
[2] 이미지 로드 및 정규화 ✓ load_img() 또는 cifar10.load_data() 이용하여 로드 ✓ 데이터를 1./255 이용하여 0~1 값으로 정규화	load_img() 함수는 리턴 타입이 JpegImageFile 이므로 이미지를 img_to_array() 이용하여 numpy 타입으로 변환이 필요함
[3] flow() 또는 flow_from_directory() 적용 ✓ data_gen = gen.flow(xdata, ydata, batch_size=...) ✓ data_gen = gen.flow_from_directory(dir_path, batch_size=.. , class_mode='...')	flow() 함수는 주어진 데이터에서 batch_size 에 지정된 개수만큼 무작위로 뽑아 변형을 가하라는 의미이며, flow_from_directory() 함수는 주어진 dir_path 에서 batch_size 만큼 읽어서 변형을 가하고, 정답(label)은 dir_path 의 하위 디렉토리 이름으로 인식되며 class_mode로 지정된, binary, sparse, categorical 같은 형태로 표현됨
[4] next() 실행 ✓ img, label = data_gen.next() [참고] img.shape = (batch_size, height, width, channel)	next() 함수 실행할 때마다 flow() 함수 또는 flow_from_directory() 함수에서 지정한 batch_size 만큼의 변형된 데이터(img)와 정답(label)이 리턴됨

flow()함수 예제

[1] ImageDataGenerator 생성

```
import tensorflow as tf

import numpy as np
import matplotlib.pyplot as plt

from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
gen = ImageDataGenerator(rotation_range = 30,
                         width_shift_range=0.3,
                         shear_range=0.4,
                         horizontal_flip=True)
```

ImageDataGenerator() 함수를 사용해서 원본이미지에 대해서 30도 이내에서 회전, 가로 방향 30% 범위에서 이동, 이미지 기울임은 40% 범위에서 기울임, 좌우반전 가능한 변화를 줌

[2] 이미지 로드 및 정규화

```
img_array_list = []
img_names = [ './dog1.jpg', './dog2.jpg',
              './cat1.jpg', './cat2.jpg' ]

for i in range(len(img_names)):
    loaded_img = load_img(img_names[i], target_size=(100,100))

    loaded_img_array = img_to_array(loaded_img) / 255.0

    img_array_list.append(loaded_img_array)

plt.figure(figsize=(6,6))
for i in range(len(img_array_list)):
    plt.subplot(1, len(img_array_list), i+1)
    plt.xticks([]); plt.yticks([])
    plt.title(img_names[i])
    plt.imshow(img_array_list[i])
```

JpegImageFile 타입 리턴

flow()함수 예제

[3] flow() 함수 적용

```
batch_siz = 2

data_gen = gen.flow(np.array(img_array_list),
                    batch_size=batch_siz)
```

flow() 함수에 입력으로 주어지는 원본 데이터 형상 (shape) = (원본 데이터 전체 개수, 높이, 너비, 채널) 형상을 가지는 4차원 텐서로 주어져야 함.

즉 (100, 100, 3) 형상을 가지는 원본데이터가 총 4개 있다면 flow() 함수를 사용하기 위해서는 다음과 같이 3차원 텐서를 원본 데이터의 총 개수가 포함된 4차원 텐서로 변환해주어야 함

(100, 100, 3) ⇒ (4, 100, 100, 3)

이처럼 전체 데이터 개수를 포함한 4차원 텐서로 만들기 위한 일반적인 방법은 넘파이의 np.array(..), np.array([..]), np.expand_dims(.., axis=0), np.vstack(([..], [..], [..],..)) 등이 있으며, 원본 데이터 형상에 따라서 적절하게 사용할 수 있음

[4] next() 실행 및 변형 이미지 출력

```
img = data_gen.next()

plt.figure(figsize=(6,6))
for i in range(len(img)):
    plt.subplot(1, len(img), i+1)
    plt.xticks([]); plt.yticks([])
    plt.imshow(img[i])
```

batch_siz=2 지정된 개수만큼 이미지 생성

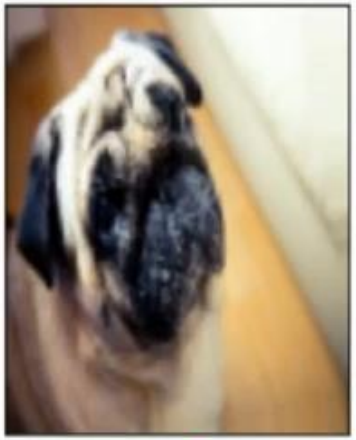
img[0] img[1]

4] next() 실행 및 변형 이미지 출력(계속)

```
img = data_gen.next()
```
→ batch_siz=2 지정된 개수만큼 이미지 생성

```
plt.figure(figsize=(6,6))
for i in range(len(img)):
    plt.subplot(1, len(img), i+1)
    plt.xticks([]); plt.yticks([])
    plt.imshow(img[i])
```

img[0]



img[1]

```
img = data_gen.next()
```
→ batch_siz=2 지정된 개수만큼 이미지 생성

```
plt.figure(figsize=(6,6))
for i in range(len(img)):
    plt.subplot(1, len(img), i+1)
    plt.xticks([]); plt.yticks([])
    plt.imshow(img[i])
```

img[0]

img[1]

flow_from_directory() 함수 예제

[1] ImageDataGenerator 생성

```
import tensorflow as tf

import numpy as np
import matplotlib.pyplot as plt

from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
gen = ImageDataGenerator(rotation_range = 30,
                         width_shift_range = 0.3,
                         shear_range=0.3,
                         rescale=1./255)
```

ImageDataGenerator() 함수를 사용해서 30도 이내에서 회전, 가로 방향 20% 범위에서 이동, 이미지 기울임은 30% 범위에서 기울임, 최대값 255로 나누어서 0~1 값을 가지는 정규화 시행

[2] flow_from_directory() 적용

```
data_path = './test_dir/'
batch_siz = 3

data_gen = gen.flow_from_directory(directory=data_path,
                                   batch_size=batch_siz,
                                   shuffle=True,
                                   target_size=(100, 100),
                                   class_mode='categorical')
```

flow_from_directory() 함수는 이미지를 불러 올때, 주어진 디렉토리의 하위 디렉토리 이름에 맞춰 자동으로 labelling 해줌

test_dir
- cat : cat1.jpg / cat2.jpg
- deer : deer1.jpg / deer2.jpg
- dog : dog1.jpg / dog2.jpg

또한 class_mode는 정답을 나타내는 방식을 나타내며 다음과 같이 지정할 수 있음

'binary' ⇒ 정답은 0 또는 1

'categorical' ⇒ 정답은 one-hot encoding 형태

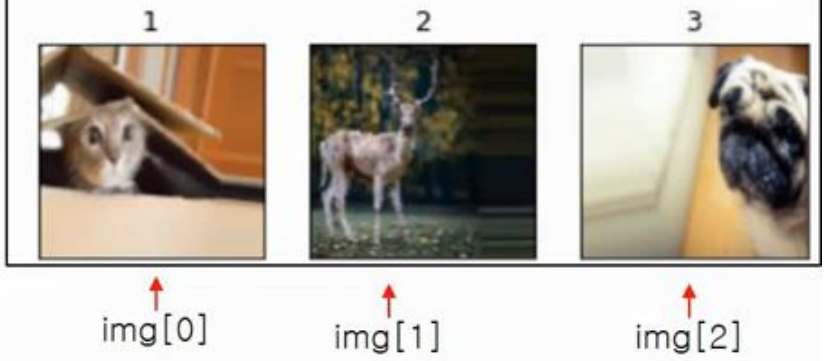
'sparse' ⇒ 정답은 십진수 형태

flow_from_directory 함수 예제

[3] next() 실행 및 변형 이미지 출력

```
img, label = data_gen.next()

plt.figure(figsize=(6,6))
for i in range(len(img)):
    plt.subplot(1, len(img), i+1)
    plt.xticks([]); plt.yticks([])

    plt.title(str(np.argmax(label[i])))

    plt.imshow(img[i])
```

batch_siz=3
지정된 개수만
큼 이미지 생성



mode='categorical' 방식으로 설정하였기 때문에 next() 실행으로 리턴되는 정답(label)은 one-hot encoding 방식으로 나타남. 따라서 정답을 십진수로 표시하기 위해 np.argmax() 사용

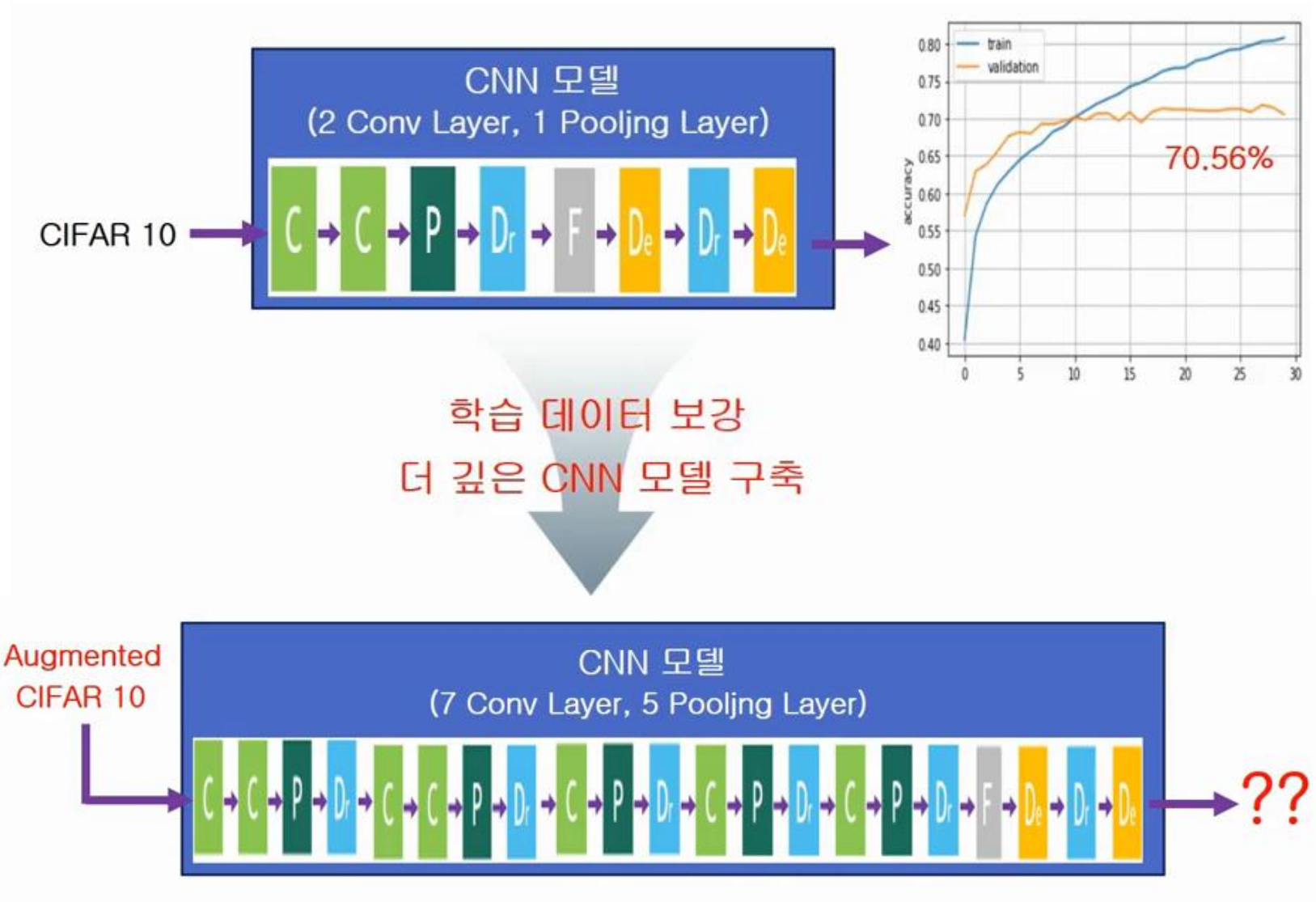
[3] next() 실행 및 변형 이미지 출력 (계속)

```
img, label = data_gen.next()

plt.figure(figsize=(6,6))
for i in range(len(img)):
    plt.subplot(1, len(img), i+1)
    plt.xticks([]); plt.yticks([])

    plt.title(str(np.argmax(label[i])))

    plt.imshow(img[i])
```

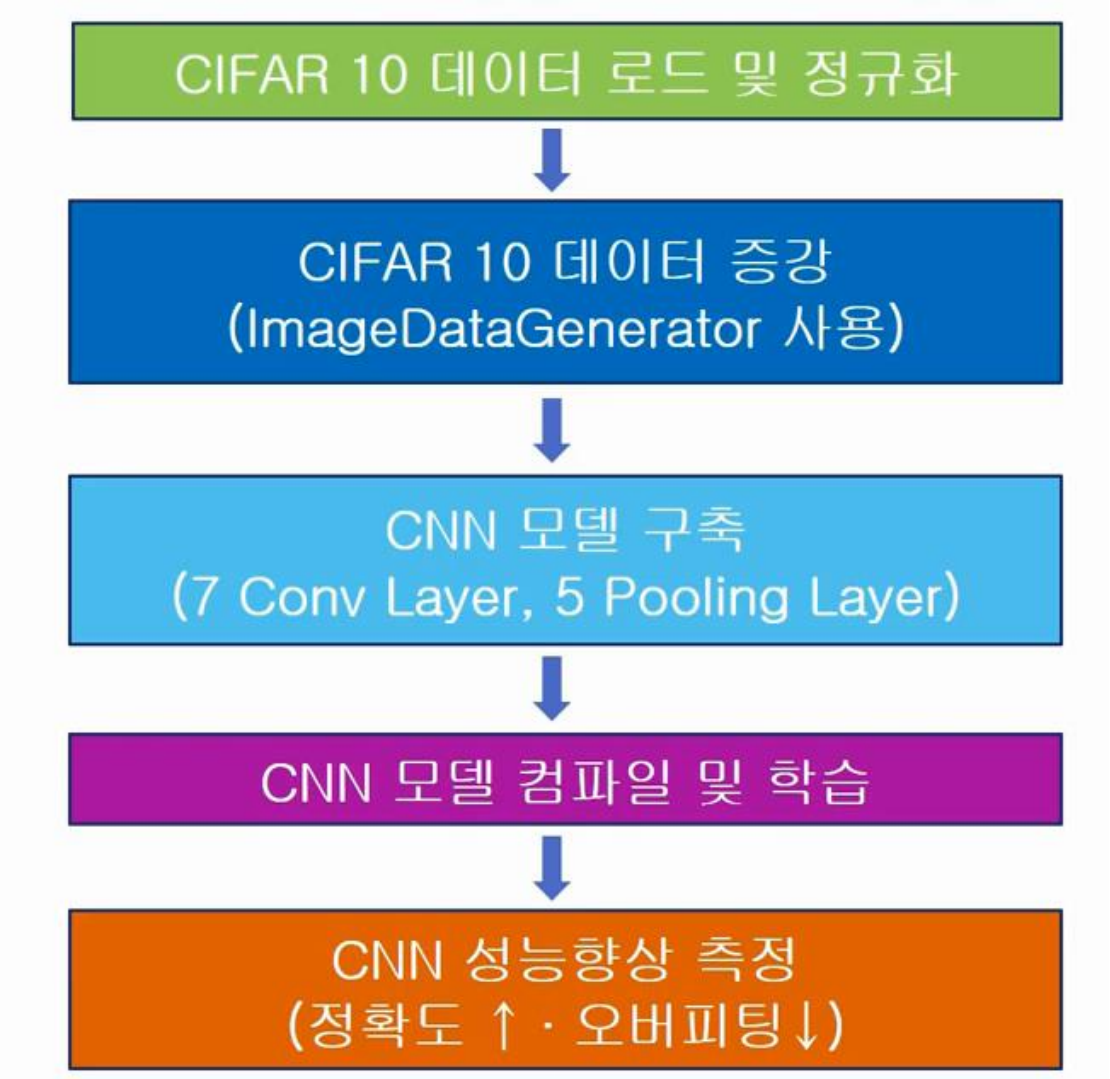
batch_siz=3
지정된 개수만
큼 이미지 생성

mode='categorical' 방식으로 설정하였기 때문에 next() 실행으로 리턴되는 정답(label)은 one-hot encoding 방식으로 나타남. 따라서 정답을 십진수로 표시하기 위해 np.argmax() 사용

성능 향상을 위한 CNN 구조 및 데이터(정확성↓.오버피팅↑)



CNN 성능 향상을 위한 개발과정



CIFAR 10 Example2

[1] CIFAR-10 데이터 불러오기 및 정규화

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D,MaxPooling2D
from tensorflow.keras.layers import Flatten,Dense,Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

x_train = x_train.astype(np.float32) / 255.0
x_test = x_test.astype(np.float32) / 255.0

print(x_train.shape, x_test.shape)
print(y_train.shape, y_test.shape)
```

데이터 불러오기

정규화

```
(50000, 32, 32, 3) (10000, 32, 32, 3)
(50000, 1) (10000, 1)
```

[2] CIFAR 10 데이터 보강 (150% 증대)

```
gen = ImageDataGenerator(rotation_range=20,
                         shear_range=0.2,
                         width_shift_range=0.2,
                         height_shift_range=0.2,
                         horizontal_flip=True)

augment_ratio = 1.5   # 전체 데이터의 150%
augment_size = int(augment_ratio * x_train.shape[0])

randidx = np.random.randint(x_train.shape[0], size=augment_size)

x_augmented = x_train[randidx].copy()
y_augmented = y_train[randidx].copy()

x_augmented, y_augmented = gen.flow(x_augmented, y_augmented,
                                    batch_size=augment_size,
                                    shuffle=False).next()

x_train = np.concatenate( (x_train, x_augmented) )
y_train = np.concatenate( (y_train, y_augmented) )

s = np.arange(x_train.shape[0])
np.random.shuffle(s)

x_train = x_train[s]
y_train = y_train[s]
```

보강할 데이터 변형방식 설정

copy() 사용해 원본 데이터 복사본 만듬

보강할 이미지 데이터 생성

보강된 학습데이터, 정답 데이터를 랜덤하게 섞음

CIFAR 10 Example2

[3] CNN 모델 구축

```
cnn=Sequential()

cnn.add(Conv2D(32,(3,3),activation='relu', padding='same',input_shape=(32,32,3)))
cnn.add(Conv2D(32,(3,3),activation='relu', padding='same'))
cnn.add(MaxPooling2D(pool_size=(2,2)))
cnn.add(Dropout(0.25))

cnn.add(Conv2D(64,(3,3),activation='relu', padding='same'))
cnn.add(Conv2D(64,(3,3),activation='relu', padding='same'))
cnn.add(MaxPooling2D(pool_size=(2,2)))
cnn.add(Dropout(0.25))

cnn.add(Conv2D(128,(3,3),activation='relu', padding='same'))
cnn.add(MaxPooling2D(pool_size=(2,2)))
cnn.add(Dropout(0.25))
cnn.add(Conv2D(128,(3,3),activation='relu', padding='same'))
cnn.add(MaxPooling2D(pool_size=(2,2)))
cnn.add(Dropout(0.25))
cnn.add(Conv2D(256,(3,3),activation='relu', padding='same'))
cnn.add(MaxPooling2D(pool_size=(2,2)))
cnn.add(Dropout(0.25))

cnn.add(Flatten())

cnn.add(Dense(128,activation='relu'))
cnn.add(Dropout(0.5))
cnn.add(Dense(10,activation='softmax'))
```

cifar 10 텐서 (높이,너비,채널)

3차원 텐서를 1차원 벡터로 변환

은닉층 개념

출력층

CIFAR 10 Example2

[4] CNN 모델 컴파일 및 학습

```
cnn.compile(loss='sparse_categorical_crossentropy', optimizer=Adam(), metrics=['accuracy'])

hist = cnn.fit(x_train, y_train, batch_size=256, epochs=250, validation_data=(x_test, y_test))
```

```
Epoch 1/250
489/489 [==============================] - 215s 439ms/step - loss: 1.7921 - accuracy: 0.3293 - val_loss: 1.3308 - val_accuracy: 0.5097
Epoch 2/250
489/489 [==============================] - 17s 35ms/step - loss: 1.3544 - accuracy: 0.5139 - val_loss: 1.0673 - val_accuracy: 0.6184
Epoch 3/250
489/489 [==============================] - 12s 24ms/step - loss: 1.1650 - accuracy: 0.5897 - val_loss: 0.9829 - val_accuracy: 0.6492
```

```
Epoch 248/250
489/489 [==============================] - 7s 14ms/step - loss: 0.4188 - accuracy: 0.8647 - val_loss: 0.4579 - val_accuracy: 0.8652
Epoch 249/250
489/489 [==============================] - 7s 14ms/step - loss: 0.4159 - accuracy: 0.8643 - val_loss: 0.4338 - val_accuracy: 0.8674
Epoch 250/250
489/489 [==============================] - 7s 14ms/step - loss: 0.4162 - accuracy: 0.8635 - val_loss: 0.4229 - val_accuracy: 0.8721
```

[4] 모델 (정확도) 평가

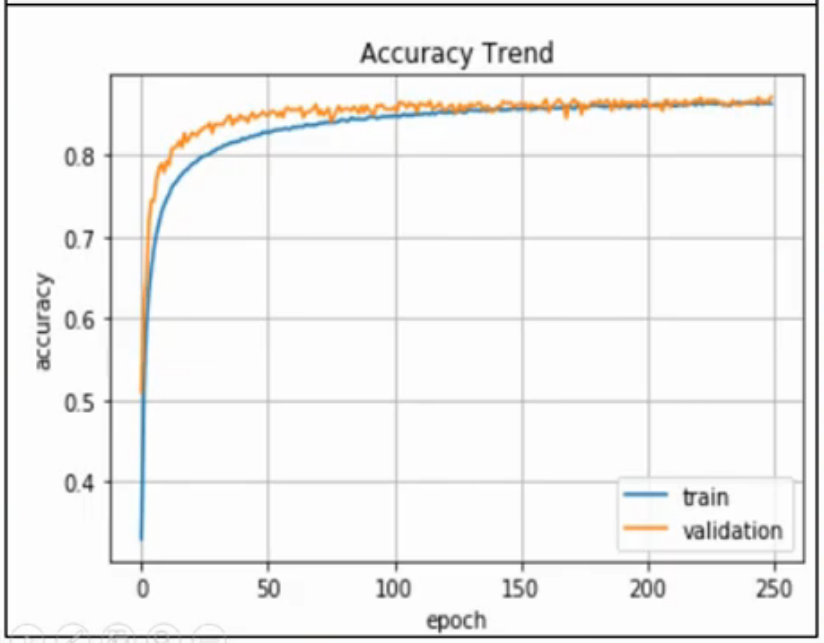
```
cnn.evaluate(x_test, y_test)
```

```
313/313 [==============================] - 1s 4ms/step - loss: 0.4229 - accuracy: 0.8721
[0.4228985011577606, 0.8720999956130981]
```

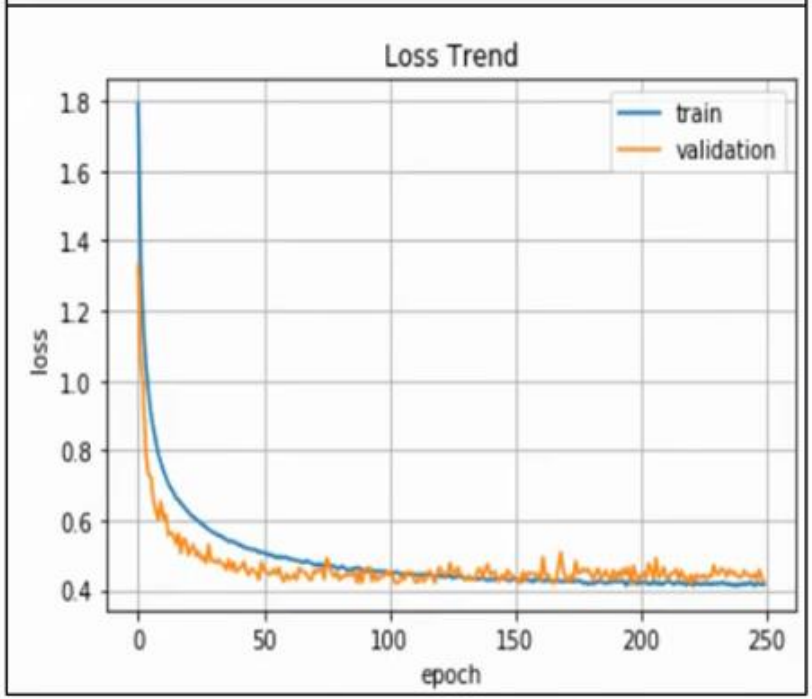
CIFAR 10 Example2

[5] 정확도 및 손실

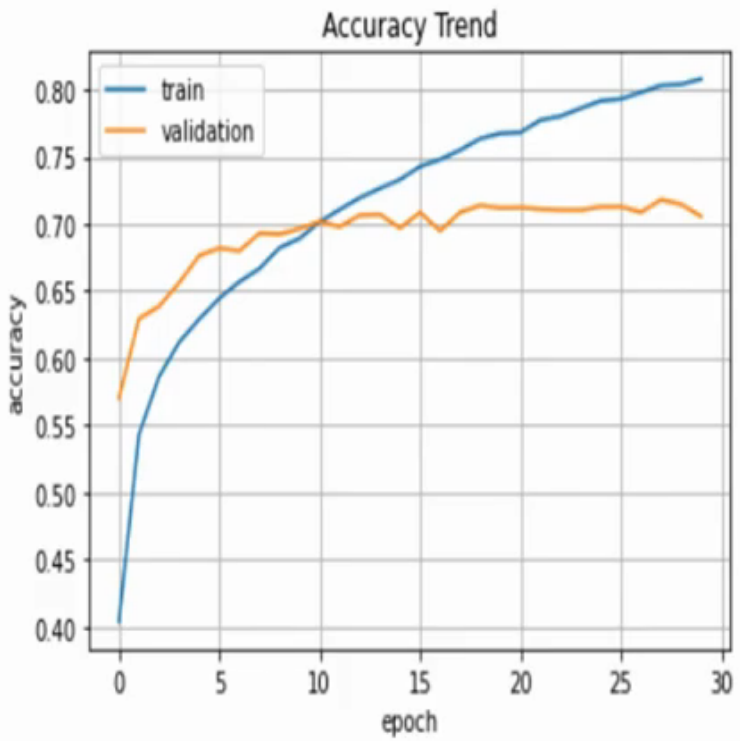
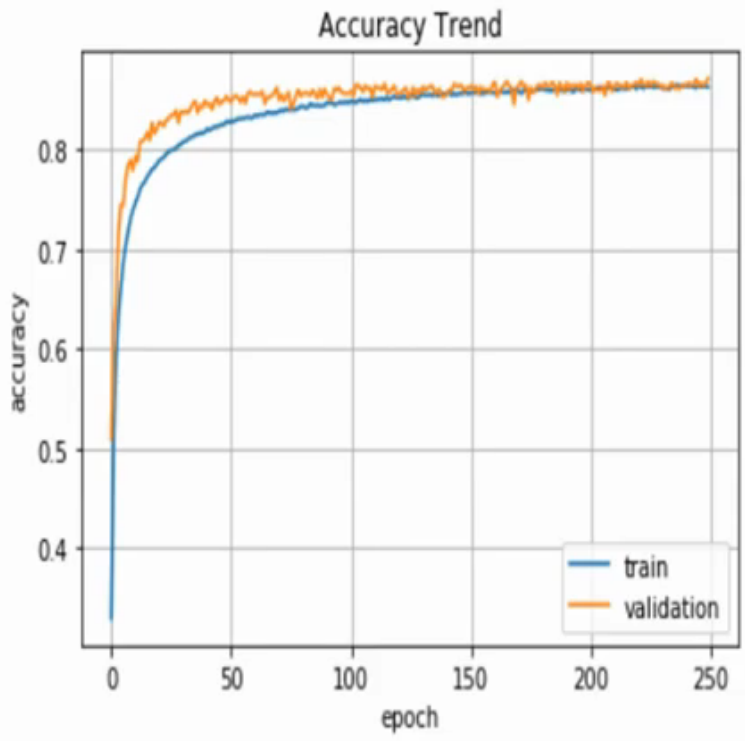
```
import matplotlib.pyplot as plt

plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title('Accuracy Trend')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train','validation'], loc='best')
plt.grid()
plt.show()
```




```
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('Loss Trend')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train','validation'], loc='best')
plt.grid()
plt.show()
```



CNN 성능향상(정확도 ↑ . 오버피팅 ↓)

CIFAR 10 (2 Conv Layer, 1 Pooling Layer)	→	Augmented CIFAR 10 (7 Conv Layer, 5 Pooling Layer)
		
70.56 %	→	87.21 %