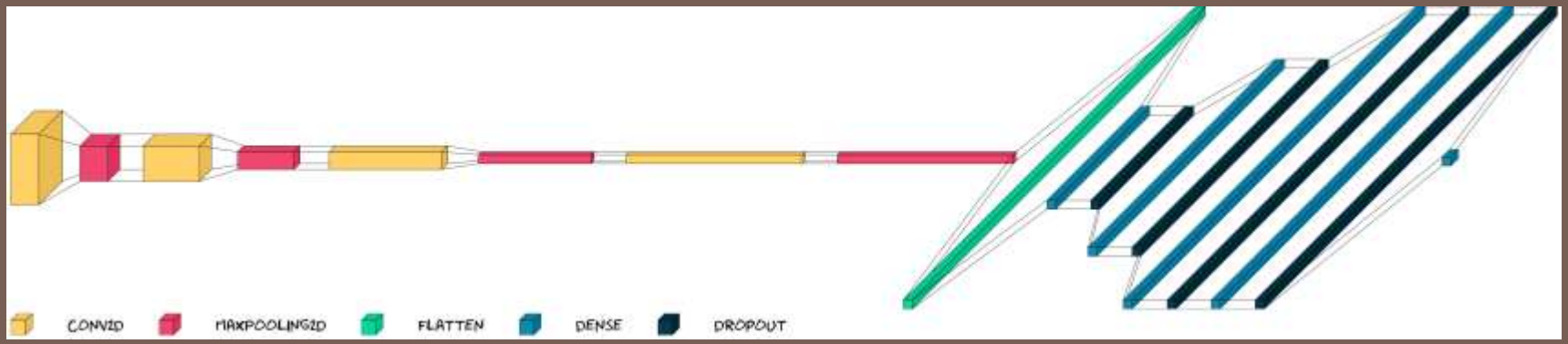
10장 영상인식



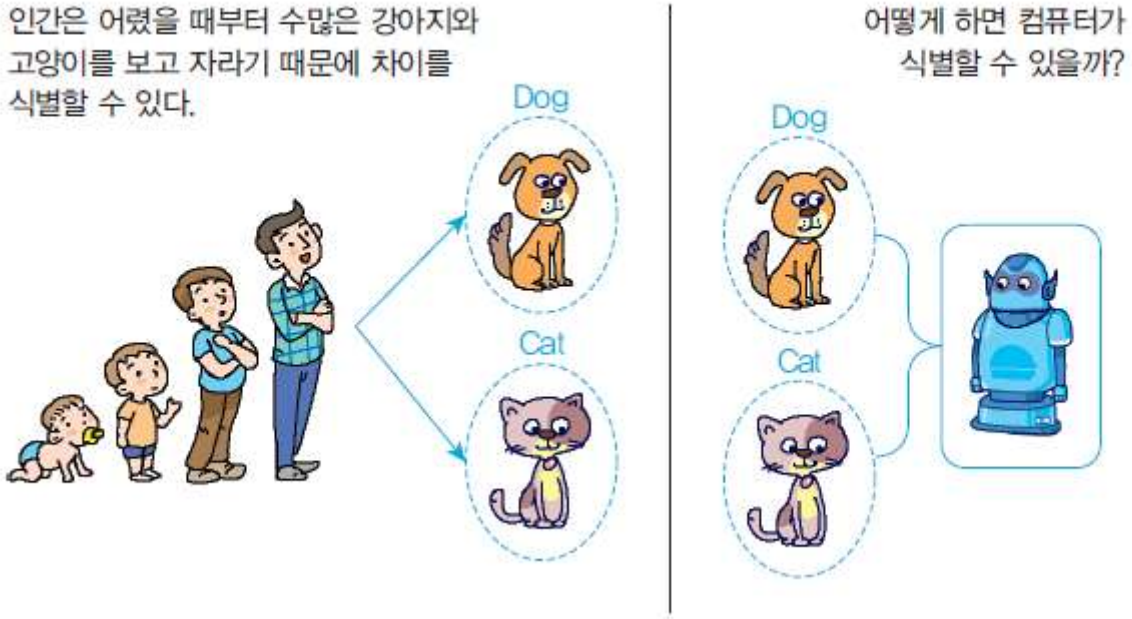
Computer Vision (CV)

학습 목표

- 컨벌루션 신경망을 이용하여 영상 인식(image recognition)을 수행
- **가중치를 저장하고 복원**하는 방법을 학습한다.
- 전이 학습을 이해한다.

영상 인식이란?

- 영상 인식(image recognition)란 영상 안의 물체를 인식하거나 분류하는 것이다.



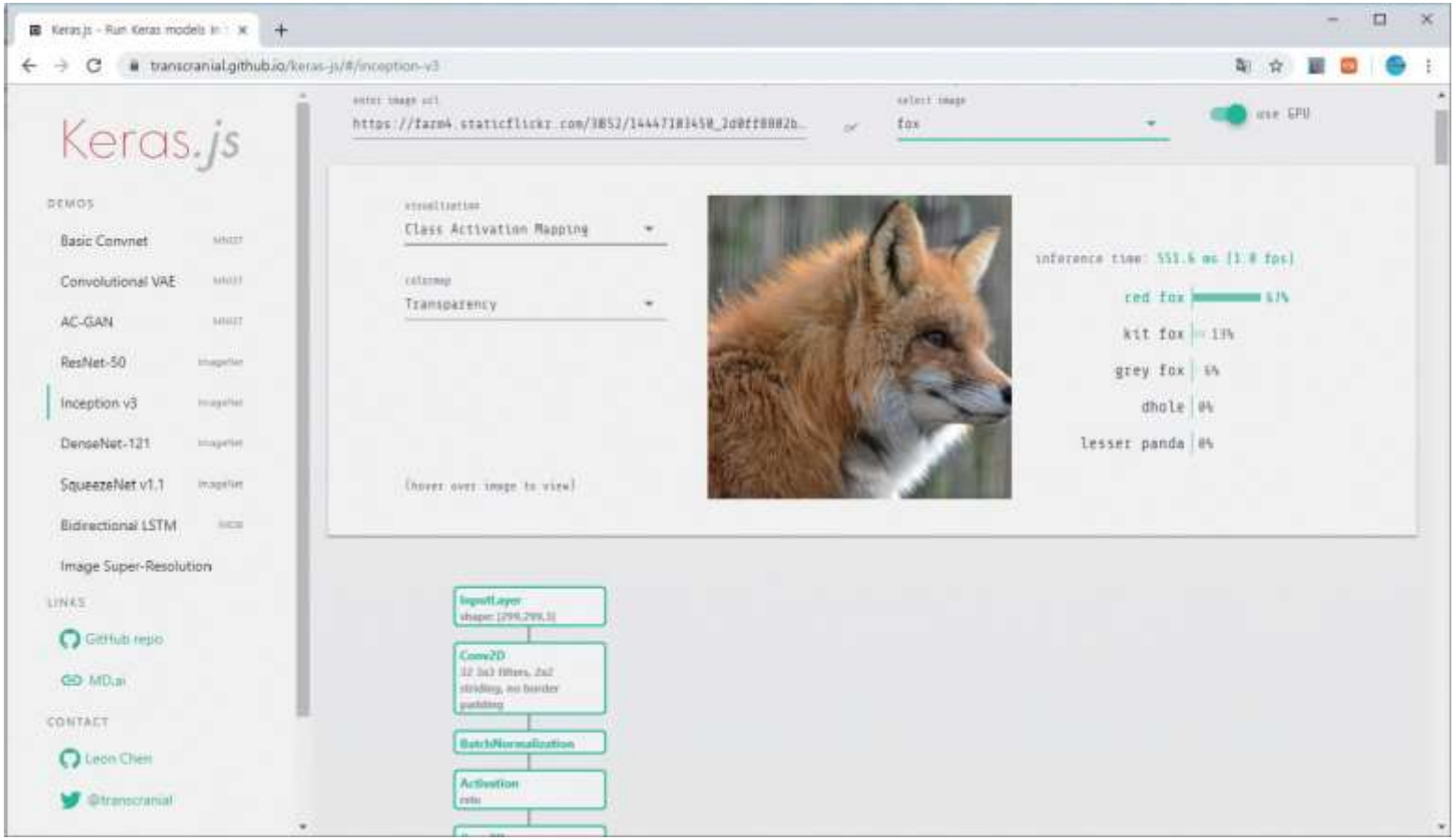
컨벌루션 신경망(CNN)

- 컨벌루션 신경망은 영상 인식에만 사용되는 것이 아니지만 생물체의 영상 처리 구조에서 힌트를 얻었다.

그림 15-2 영상 인식 시스템

Lab: 컨벌루션 네트워크 체험하기

- https://transcranial.github.io/keras-js/#/



전통적인 영상 인식 시스템의 구조

- 신경망을 이용한 방법은 전통적인 컴퓨터 시각 방법과 상당히 다르다

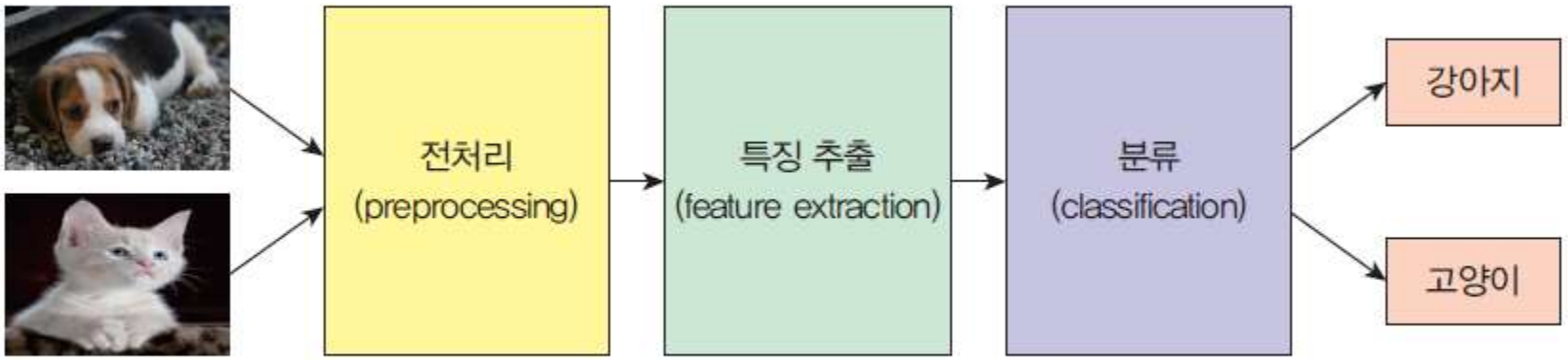


그림 10-3 전통적인 영상 인식 시스템

전통적인 영상 인식 시스템의 구조

- **특징(features based on domain knowledge)**을 사용한다.
- **특징을 수동으로 추출**
- 여러 개의 특징을 사용한다.

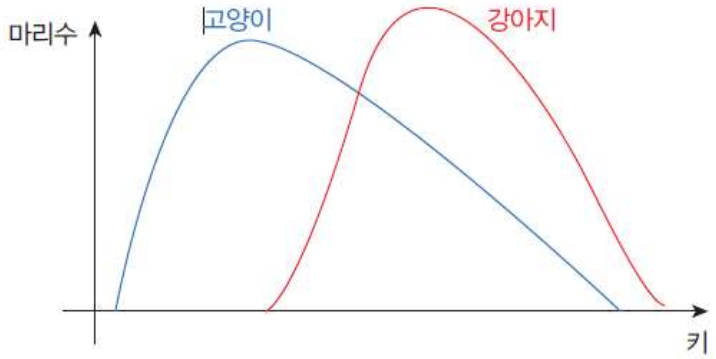


그림 10-4 특징

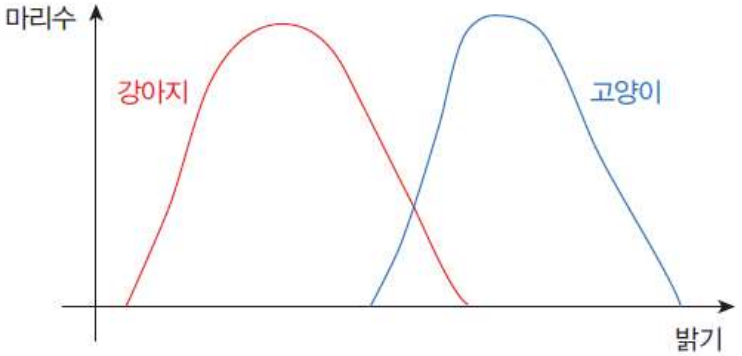


그림 10-5 밝기를 특징으로 사용한다.

전통적인 영상 인식 시스템의 구조

- 특징을 가지고 물체를 분류한다.

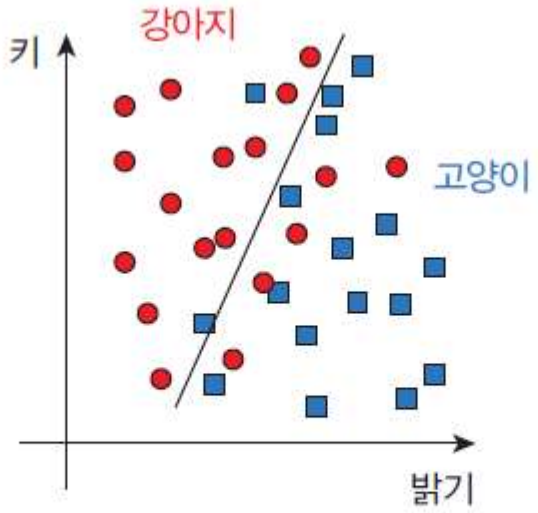


그림 10-6 키와 밝기를 동시에 특징으로 사용한다.

전통적인 영상 인식 시스템의 구조

- 과잉 적합

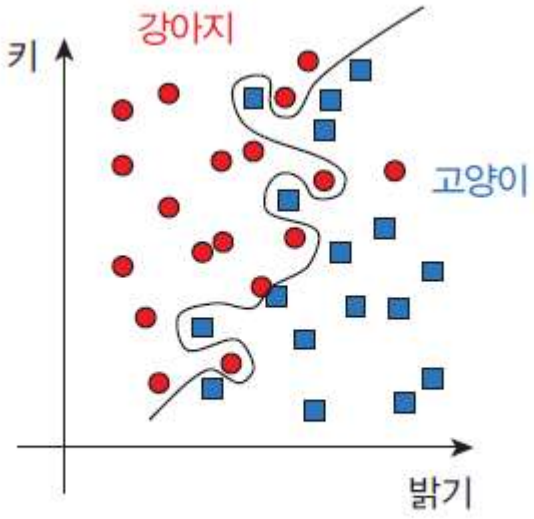


그림 10-7 과잉 적합된 분류기

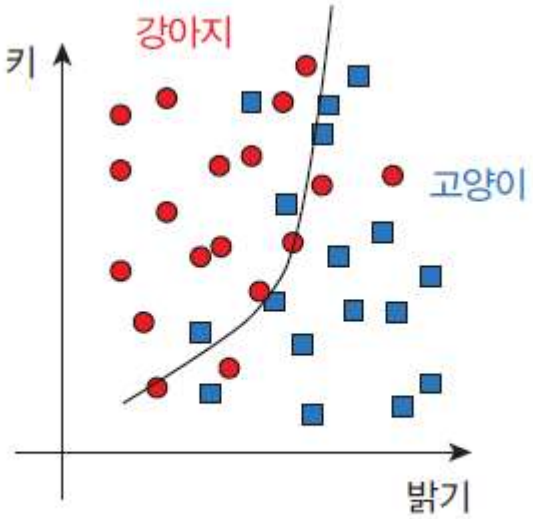


그림 10-8 적절한 판단 경계선

심층 신경망을 이용한 영상 인식

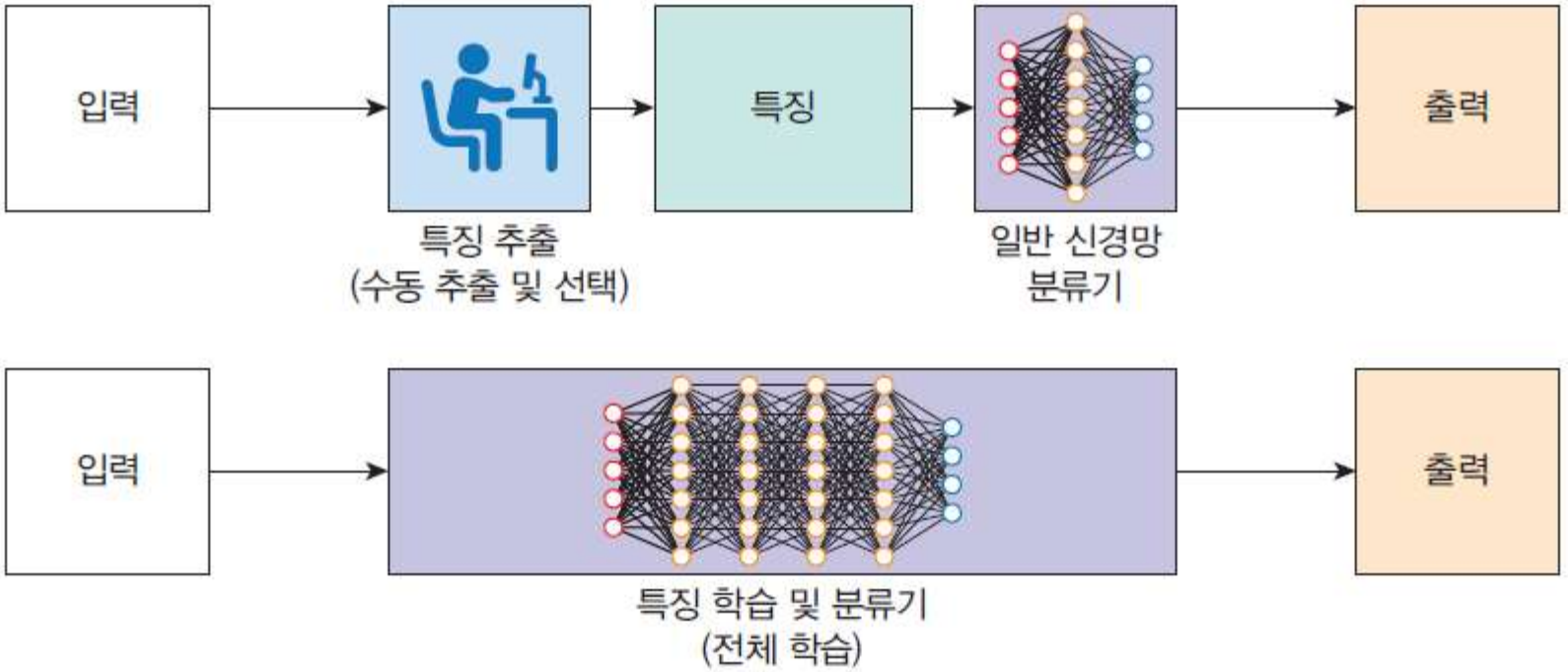


그림 10-9 전통적인 영상 인식과 심층 신경망을 이용한 영상 인식

영상 인식과 컨벌루션 신경망

- 하지만 영상을 처리할 때는 더 적합한 신경망 구조가 있다. 무엇일까? 영상 인식에 많이 사용되는 신경망은, 우리가 앞장에서 학습하였던 컨볼루션 신경망(CNN)이다.

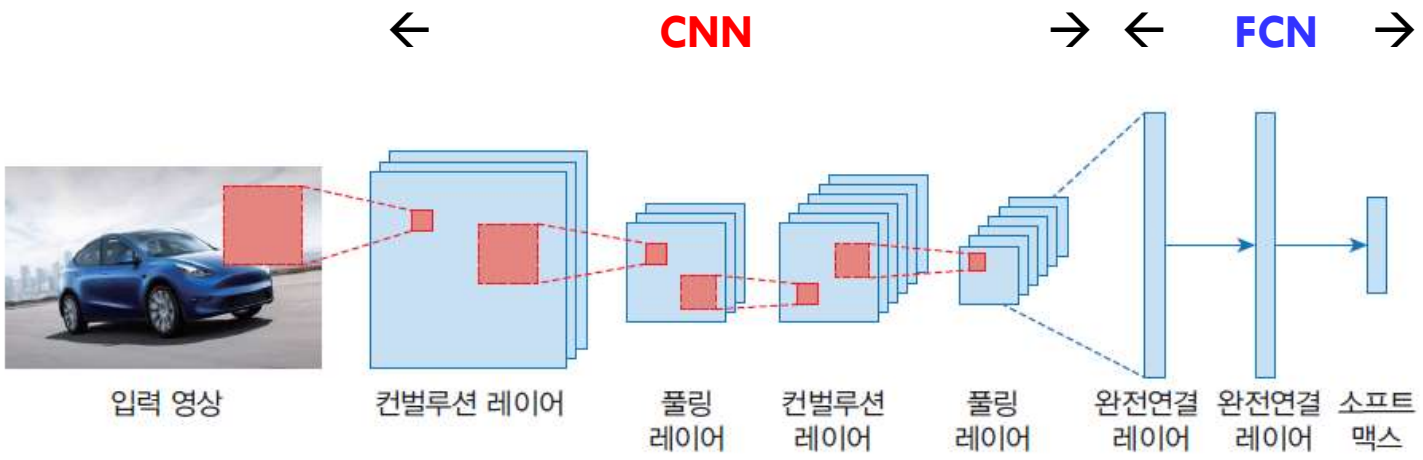
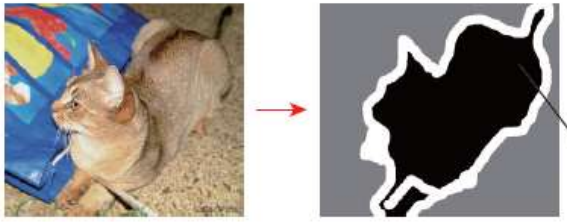


그림 10-10 컨벌루션 신경망을 이용한 영상 인식

영상 인식 신경망의 구조

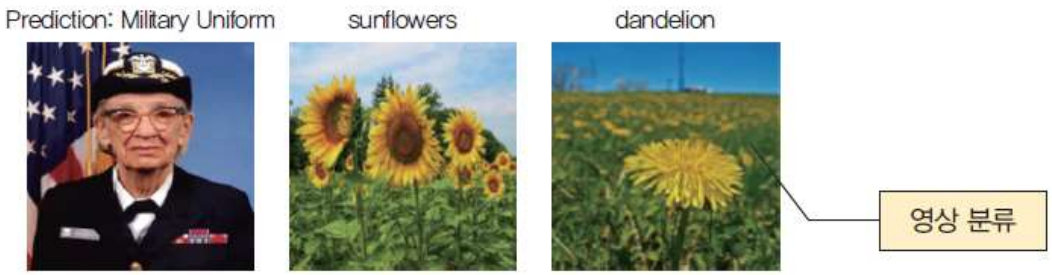
- 영상 인식 신경망은 다음과 같이 크게 2부분으로 구성되어 있다.

컴퓨터 시각 분야 응용의 예



영상 분할

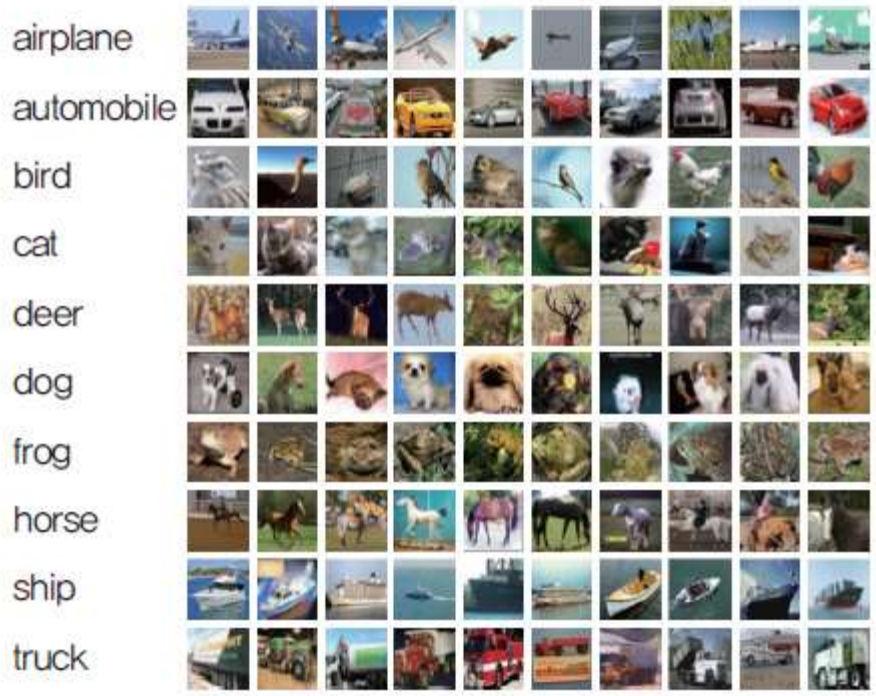
객체 감지



영상 분류

예제: *CIFAR-10* 영상 분류하기

- CIFAR는 “Canadian Institute For Advanced Research”의 약자이다. CIFAR-10 데이터 세트는 CIFAR 연구소에서 CIFAR-100 데이터 세트와 함께 개발되었다.
- 데이터 세트는 (0: airplane, 1: automobile, 2: bird, 3: cat, 4: deer, 5: dog, 6: frog, 7: horse, 8: ship, 9: truck)의 10개 부류의 컬러 영상 60,000개로 이루어진다.
- 영상의 크기는 비교적 작아서 32×32×3 이다.

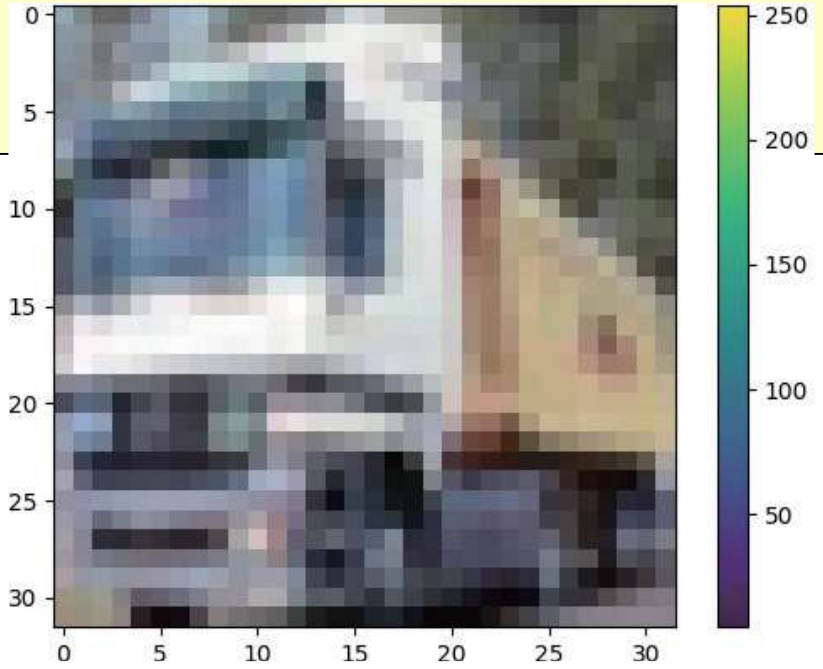


라이브러리와 데이터 세트 포함

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import Sequential
from tensorflow.keras.layers import *

# CFAR-10 데이터 세트를 적재한다.
# 훈련 세트와 테스트 세트를 반환받는다.
(X_train, y_train), (X_test, y_test) = keras.datasets.cifar10.load_data()
```

영상 화면 표시

```
# 두 번째 영상(트럭)을 화면에 표시한다.
plt.figure()
plt.imshow(X_train[1])
plt.colorbar()

# 영상의 픽셀 값을 0에서 1 사이로 변환한다.
X_train = X_train/255.0
X_test = X_test/255.0
```

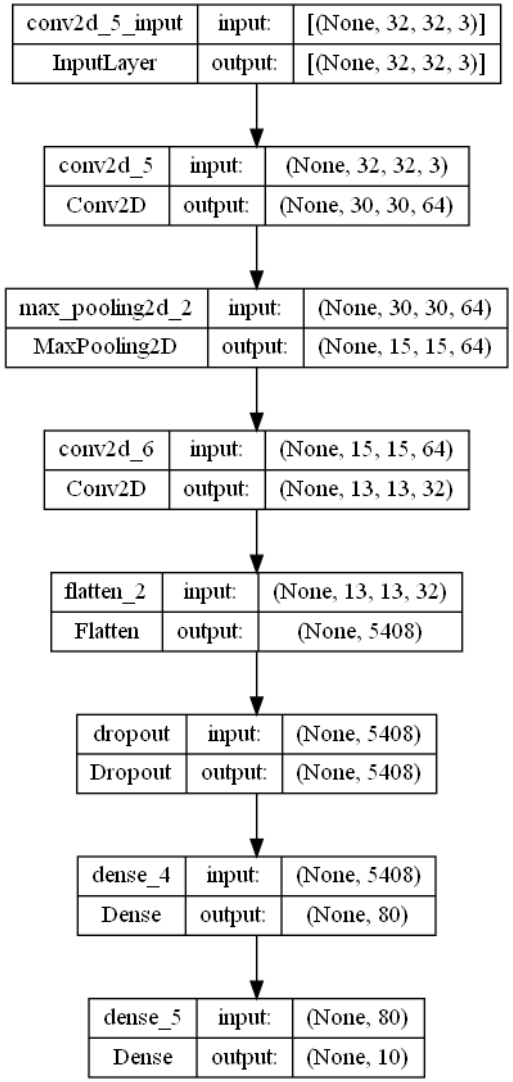


신경망 모델 구축

```
# 순차 모델을 구축한다.
model = Sequential()
# CNN
model.add(Conv2D(64, activation = 'relu', kernel_size = (3,3 ), input_shape = (32, 32, 3)))
model.add(MaxPooling2D(pool_size = (2, 2)))
model.add(Conv2D(32, activation = 'relu', kernel_size = (3,3 )))
model.add(Flatten())
# model.add(Dropout(0.5))
# FCN
model.add(Dense(80, activation = 'relu'))
model.add(Dense(10, activation = 'softmax'))
```

신경망 모델 구축

```
In [26]: model.summary()
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_5 (Conv2D)           (None, 30, 30, 64)        1792

 max_pooling2d_2 (MaxPooling  (None, 15, 15, 64)       0
 2D)

 conv2d_6 (Conv2D)           (None, 13, 13, 32)        18464

 flatten_2 (Flatten)         (None, 5408)              0

 dropout (Dropout)           (None, 5408)              0

 dense_4 (Dense)             (None, 80)                432720

 dense_5 (Dense)             (None, 10)                810

=================================================================
Total params: 453,786
Trainable params: 453,786
Non-trainable params: 0
_________________________________________________________________
```



신경망 모델 구축 – visualkeras

conv2d_5_input	input:	[(None, 32, 32, 3)]
InputLayer	output:	[(None, 32, 32, 3)]

↓

conv2d_5	input:	(None, 32, 32, 3)
Conv2D	output:	(None, 30, 30, 64)

↓

max_pooling2d_2	input:	(None, 30, 30, 64)
MaxPooling2D	output:	(None, 15, 15, 64)

↓

conv2d_6	input:	(None, 15, 15, 64)
Conv2D	output:	(None, 13, 13, 32)

↓

flatten_2	input:	(None, 13, 13, 32)
Flatten	output:	(None, 5408)

↓

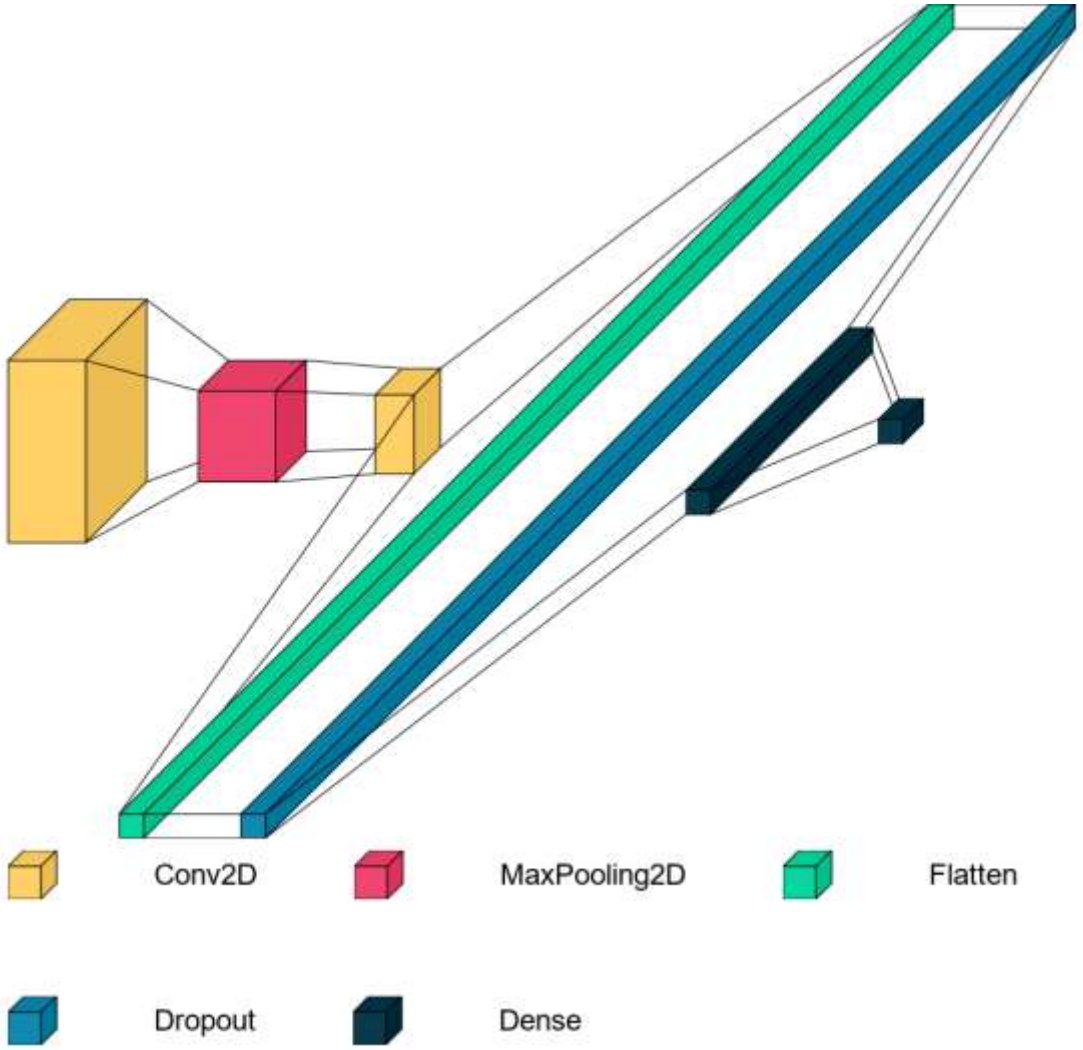
dropout	input:	(None, 5408)
Dropout	output:	(None, 5408)

↓

dense_4	input:	(None, 5408)
Dense	output:	(None, 80)

↓

dense_5	input:	(None, 80)
Dense	output:	(None, 10)



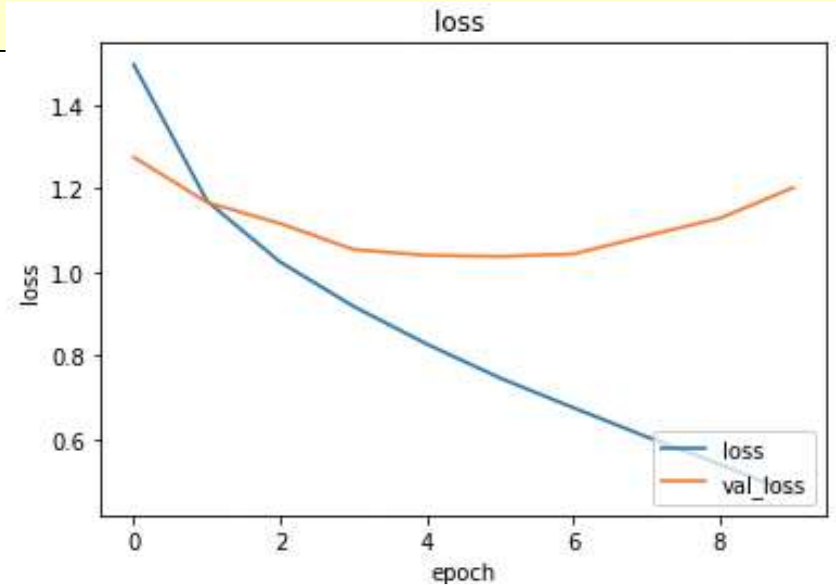
신경망 모델 구축

```
# 모델을 컴파일한다.
model.compile(optimizer = 'adam', loss = 'sparse_categorical_crossentropy',
              metrics = ['accuracy'])
# 모델을 훈련한다.
history = model.fit(X_train, y_train, epochs=10, verbose=1,
                    validation_split=0.3)
```

```
Epoch 10/10
1094/1094 [==============================] - 20s 18ms/step - loss: 0.4703 -
accuracy: 0.8339 - val_loss: 1.2012 - val_accuracy: 0.6419
```

손실값을 그래프로 그린다.

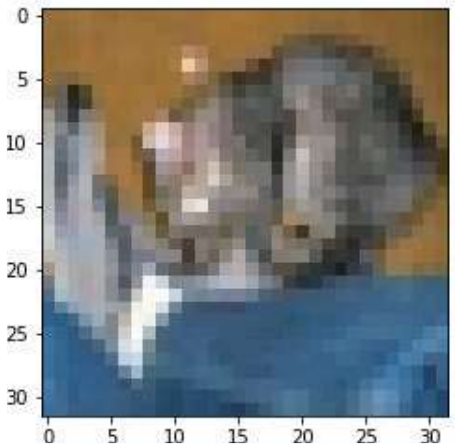
```
# 손실값을 그래프로 그린다.
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['loss', 'val_loss'], loc = 'lower right')
plt.show()
```



테스트

```
plt.figure()
plt.imshow(X_test[0])
y_pred = model.predict(X_test)
print("정답=", y_test[0])
print("예측값=", y_pred[0])
```

```
정답= [3]
예측값= [1.2904912e-12 5.1581086e-13 4.5277499e-14 9.9997449e-01 1.3750282e-09
4.7490963e-07 2.5059153e-05 8.8131386e-12 2.7819570e-11 2.6272799e-08]
```



데이터 증대 *(data augmentation)*

- 데이터 증대(data augmentation)는 한정된 데이터에서 여러 가지로 변형된 데이터를 만들어내는 기법이다.

라이브러리를 포함시킨다.

```
import tensorflow as tf
import matplotlib.pyplot as plt
from numpy import expand_dims
from tensorflow.keras.preprocessing.image import load_img, img_to_array

image = load_img("dog.jpg")
array = img_to_array(image)
sample = expand_dims(array, axis=0)
```

ImageDataGenerator()을 이용하여 영상 변환

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

datagen = ImageDataGenerator(rescale = 1./255,
    rotation_range=90, brightness_range=[0.8, 1.0],
    width_shift_range=0.2, zoom_range=[0.8, 1.2],
    height_shift_range=0.2)
```

- rotation_range=90: 회전 한도
- brightness_range=[0.2, 1.0]: 밝기 변형 비율
- width_shift_range=0.2: 좌우 이동 한도
- zoom_range=[0.2, 1.2]: 확대 한도

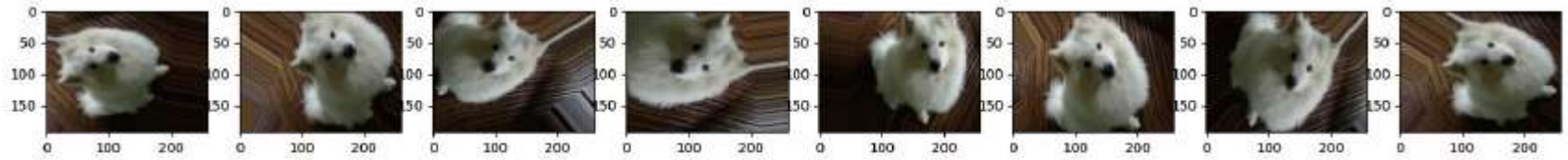
ImageDataGenerator로부터 제너레이터 객체

- ImageDataGenerator 출력은 파이썬의 제너레이터 형식이다. 영상을 생성하는 제너레이터 객체를 생성한다. 제너레이터는 next()가 호출되면 변형된 영상을 하나씩 보내게 된다.

```
obj = datagen.flow(sample, batch_size=1)
```

변형된 영상을 표시한다.

```
obj = datagen.flow(sample, batch_size=1)
fig = plt.figure(figsize=(20,5))

for i in range(8):
  plt.subplot(1,8,i+1)
  image = obj.next()
  plt.imshow(image[0])
```



모델 저장

- 가중치를 저장하여 재활용

가중치 저장과 전이 학습

- 우리는 이미 학습된 모델의 가중치를 저장할 수 있고, 필요할 때마다 가중치를 불러와서 바로 신경망이 예측을 할 수 있게 할 수 있다.

가중치 저장과 복원

- **저장**: model.save('mymodel')
- 저장되는 정보
  - 신경망 모델의 아키텍처 및 구성
  - 훈련 중에 학습된 모델의 가중치 값
  - 신경망 모델의 컴파일 정보
  - 옵티마이저와 현재 상태(훈련을 중단한 곳에서 다시 시작하기 위해)
- 저장 형식
  - TensorFlow SavedModel 형식
  - 이전 Keras H5 형식.
- **복원**: model = load_model('mymodel')

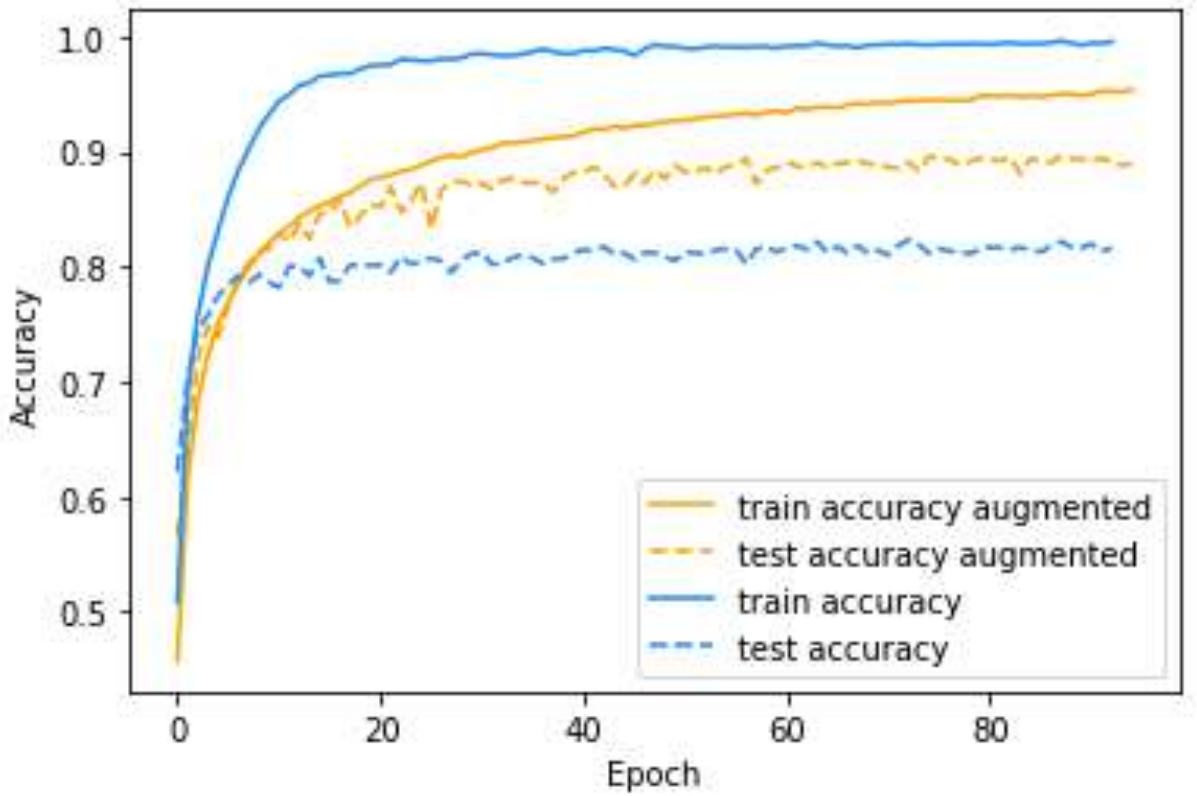
예제 : 가중치 저장과 이용

```
import numpy as np
import tensorflow as tf

# 난수로 훈련 예제를 만든다.
test_input = np.random.random((128, 32))
test_target = np.random.random((128, 1))

# 입력이 32, 출력이 1 노드인 신경망 모델을 구축한다. 함수형 API를 사용하였다.
inputs = tf.keras.Input(shape=(32,))
outputs = tf.keras.layers.Dense(1)(inputs)
model = tf.keras.Model(inputs, outputs)
model.compile(optimizer="adam", loss="mean_squared_error")

# 신경망을 3번 훈련시킨다.
model.fit(test_input, test_target, epochs=3)

# 3의 에포크를 수행한 모델을 저장한다.
model.save("my_model")

# 저장된 모델을 불러온다.
saved_model = tf.keras.models.load_model("my_model")

# 저장된 모델을 다시 학습시킨다.
saved_model.fit(test_input, test_target, epochs=3)
```

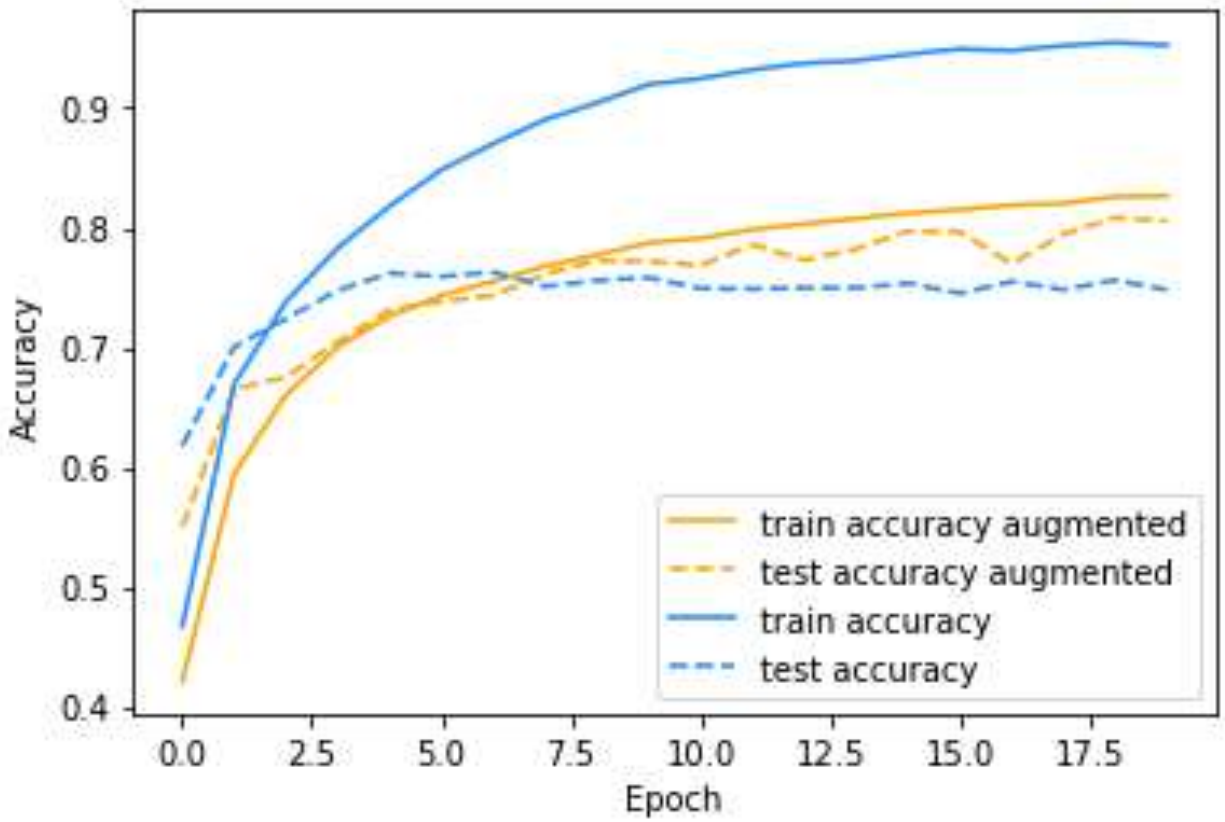
실행결과 → 저장 모델을 다시 훈련시킴. 결과는?

```
Epoch 1/3
4/4 [==============================] - 0s 499us/step - loss: 0.7929
Epoch 2/3
4/4 [==============================] - 0s 499us/step - loss: 0.7068
Epoch 3/3
4/4 [==============================] - 0s 498us/step - loss: 0.6226

INFO:tensorflow:Assets written to: my_model\assets

Epoch 1/3
4/4 [==============================] - 0s 748us/step - loss: 0.5518
Epoch 2/3
4/4 [==============================] - 0s 564us/step - loss: 0.4866
Epoch 3/3
4/4 [==============================] - 0s 505us/step - loss: 0.4354
```

데이터 증대 효과: *overfitting or not*

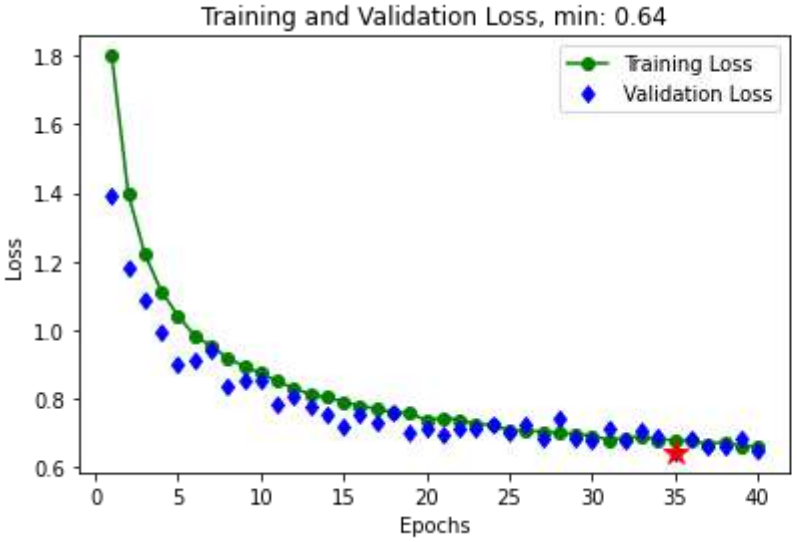


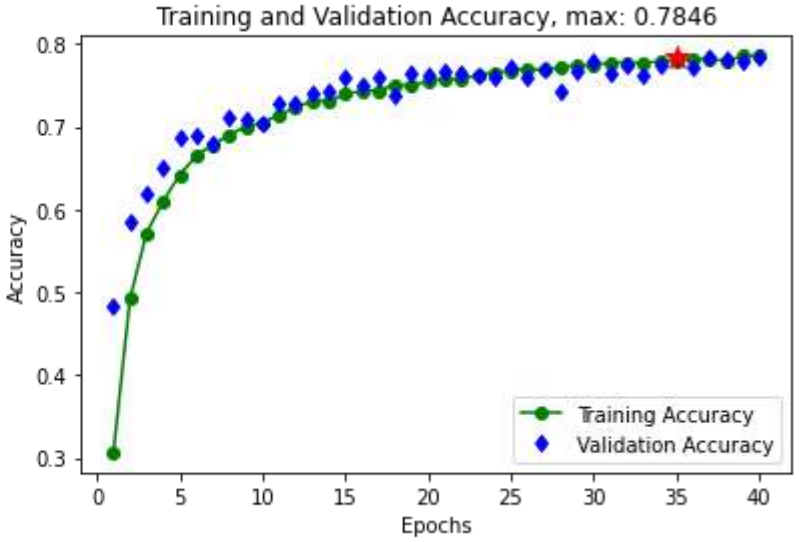
데이터 증대 효과: *overfitting or not*

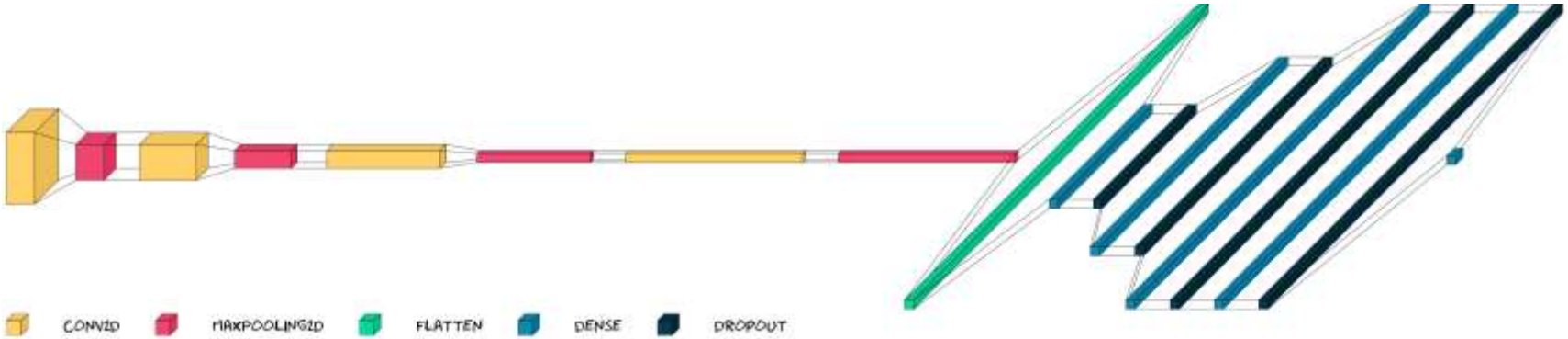


cifar10: 데이터증대의 예

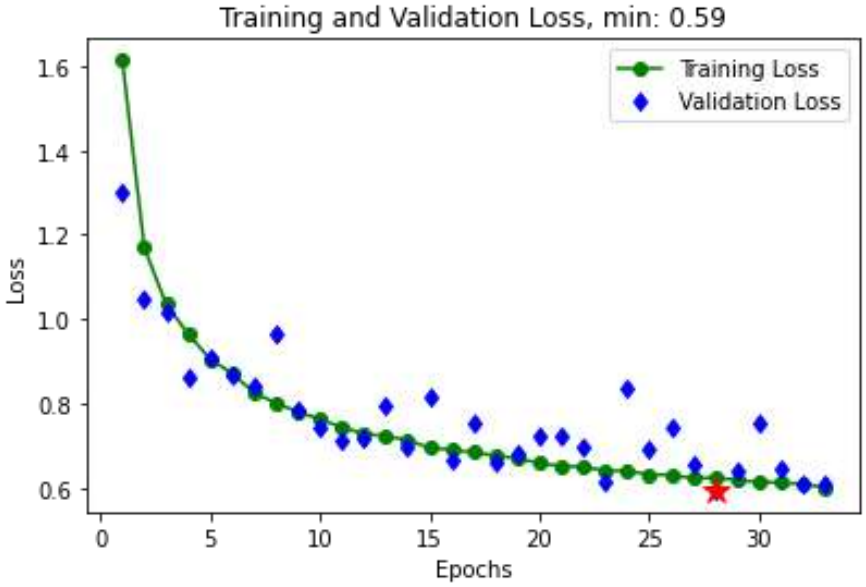
데이터 증대 효과: *overfitting or not*

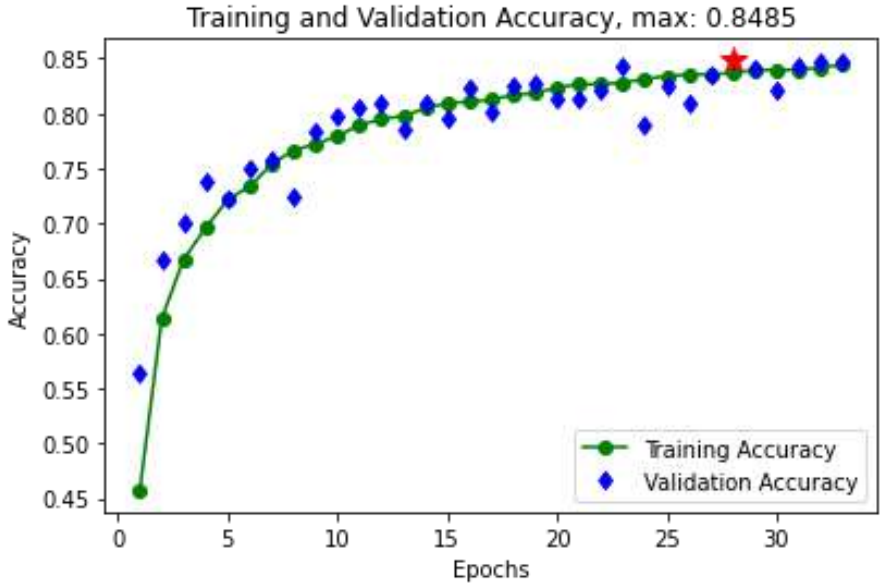


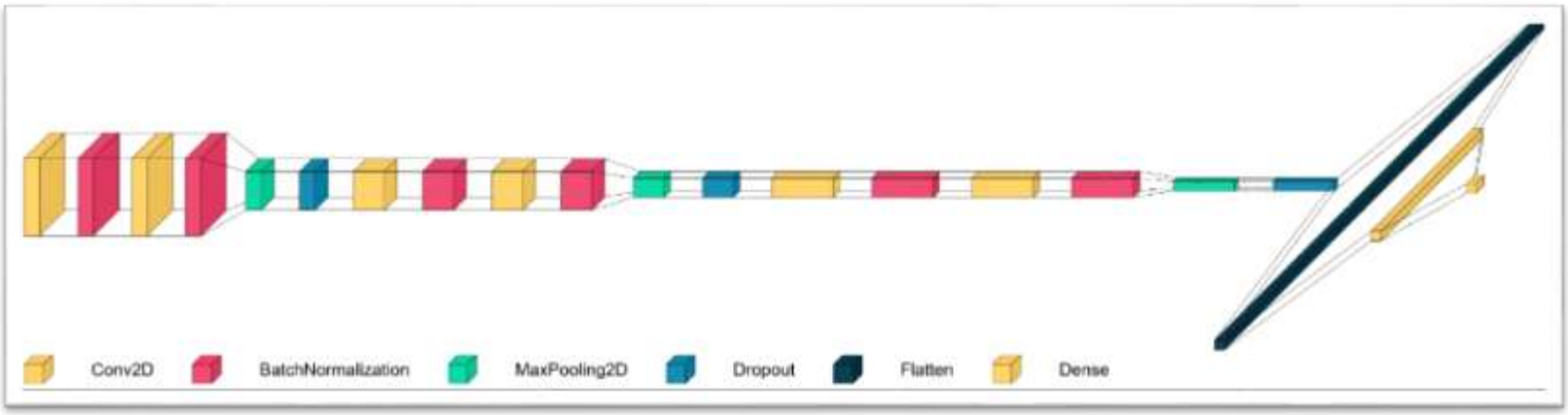




데이터 증대 효과: *overfitting or not*





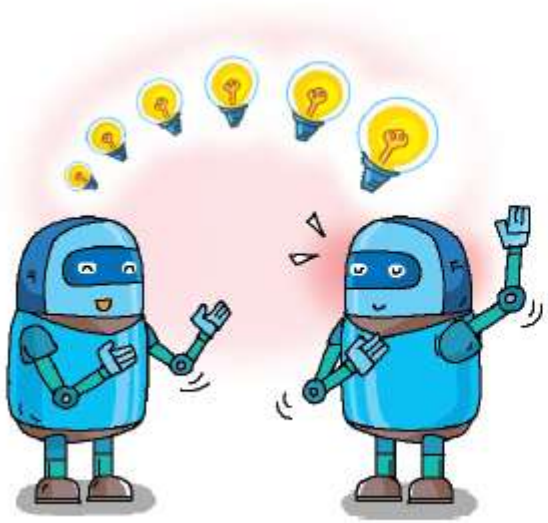


전이 학습

Transfer learning

전이학습 (transfer learning)

- 전이 학습(transfer learning)은 하나의 문제에 대해 학습한 신경망의 모델과 가중치를, 새로운 문제에 적용하는 것이다.



전이 학습

(a) 처음부터 심층 신경망을 훈련

(b) 전이 학습(사전 훈련된 모델의 미세 조정)

그림 10-12 (a) 처음부터 개발하는 방법 (b) 사전 훈련된 모델을 받아서 미세 조정하는 방법

사전 훈련된 신경망 모델

- 케라스는 사전 훈련된 딥러닝 모델들을 제공한다. 이것을 케라스에서는 “케라스 애플리케이션(keras applications)”이라고 부른다

모델	크기	Top-1 정확도	Top-5 정확도	매개 변수	깊이
Xception	88MB	0.790	0.945	22,910,480	126
VGG16	528MB	0.713	0.901	138,357,544	23
VGG19	549MB	0.713	0.900	143,667,240	26
ResNet50	98MB	0.749	0.921	25,636,712	–
ResNet101	171MB	0.764	0.928	44,707,176	–
ResNet152	232MB	0.766	0.931	60,419,944	–
ResNet50V2	98MB	0.760	0.930	25,613,800	–
ResNet101V2	171MB	0.772	0.938	44,675,560	–
ResNet152V2	232MB	0.780	0.942	60,380,648	–
InceptionV3	92MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215MB	0.803	0.953	55,873,736	572
MobileNet	16MB	0.704	0.895	4,253,864	88
MobileNetV2	14MB	0.713	0.901	3,538,984	88

Transfer Learning: models

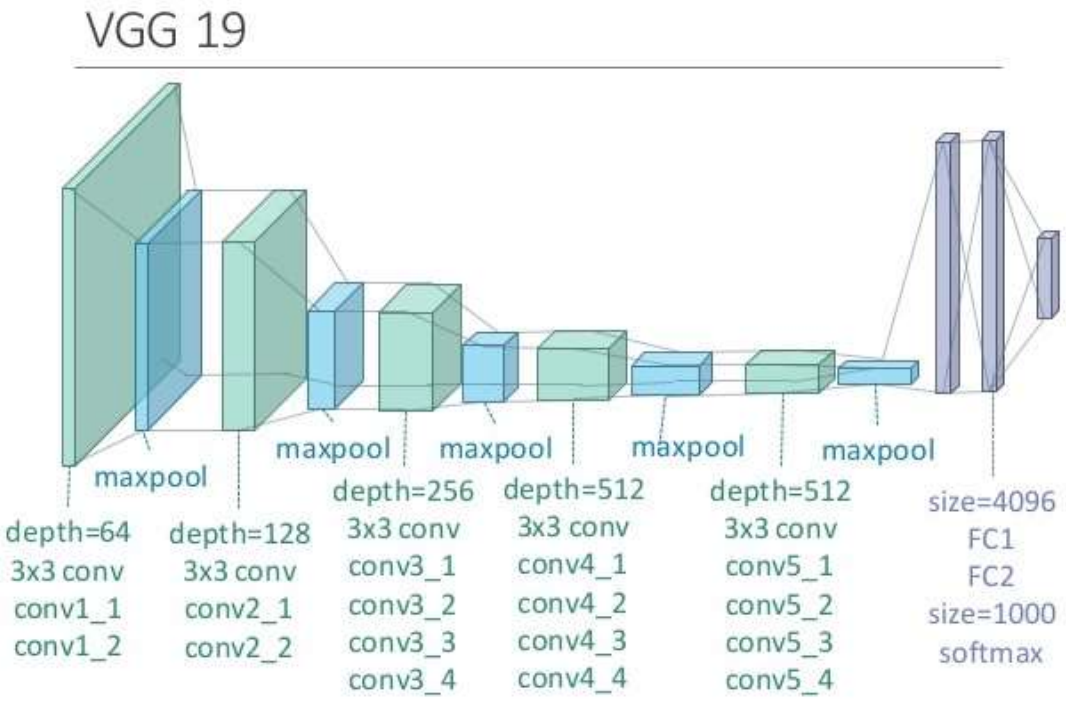
ImageNet dataset

- 1.4 million labeled images
- 1,000 different classes

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	98 MB	0.749	0.921	25,636,712	-
ResNet101	171 MB	0.764	0.928	44,707,176	-
ResNet152	232 MB	0.766	0.931	60,419,944	-
ResNet50V2	98 MB	0.760	0.930	25,613,800	-
ResNet101V2	171 MB	0.772	0.938	44,675,560	-
ResNet152V2	232 MB	0.780	0.942	60,380,648	-
InceptionV3	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	14 MB	0.713	0.901	3,538,984	88
DenseNet121	33 MB	0.750	0.923	8,062,504	121
DenseNet169	57 MB	0.762	0.932	14,307,880	169
DenseNet201	80 MB	0.773	0.936	20,242,984	201

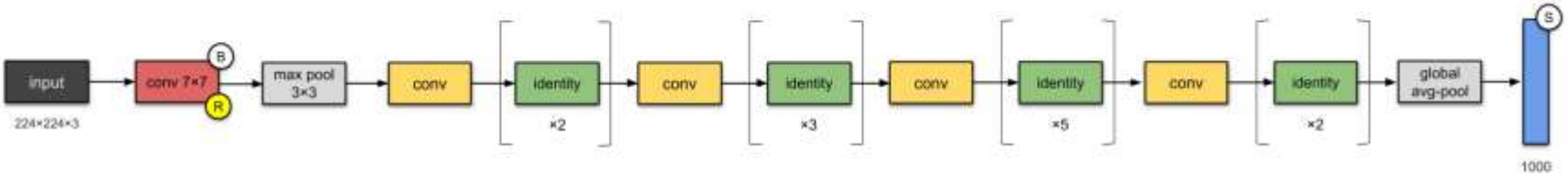
EfficientNet_V1,V2

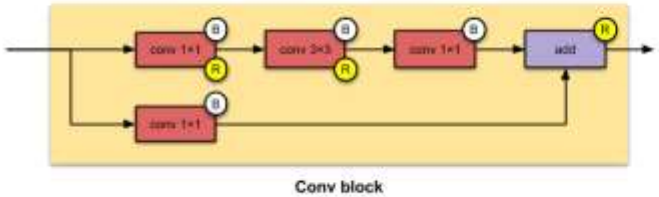
Transfer Learning: VGG19(2014)

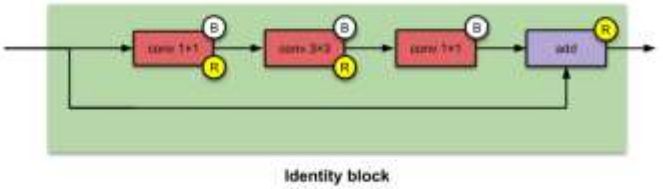


https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d

Transfer Learning: ResNet(2015)



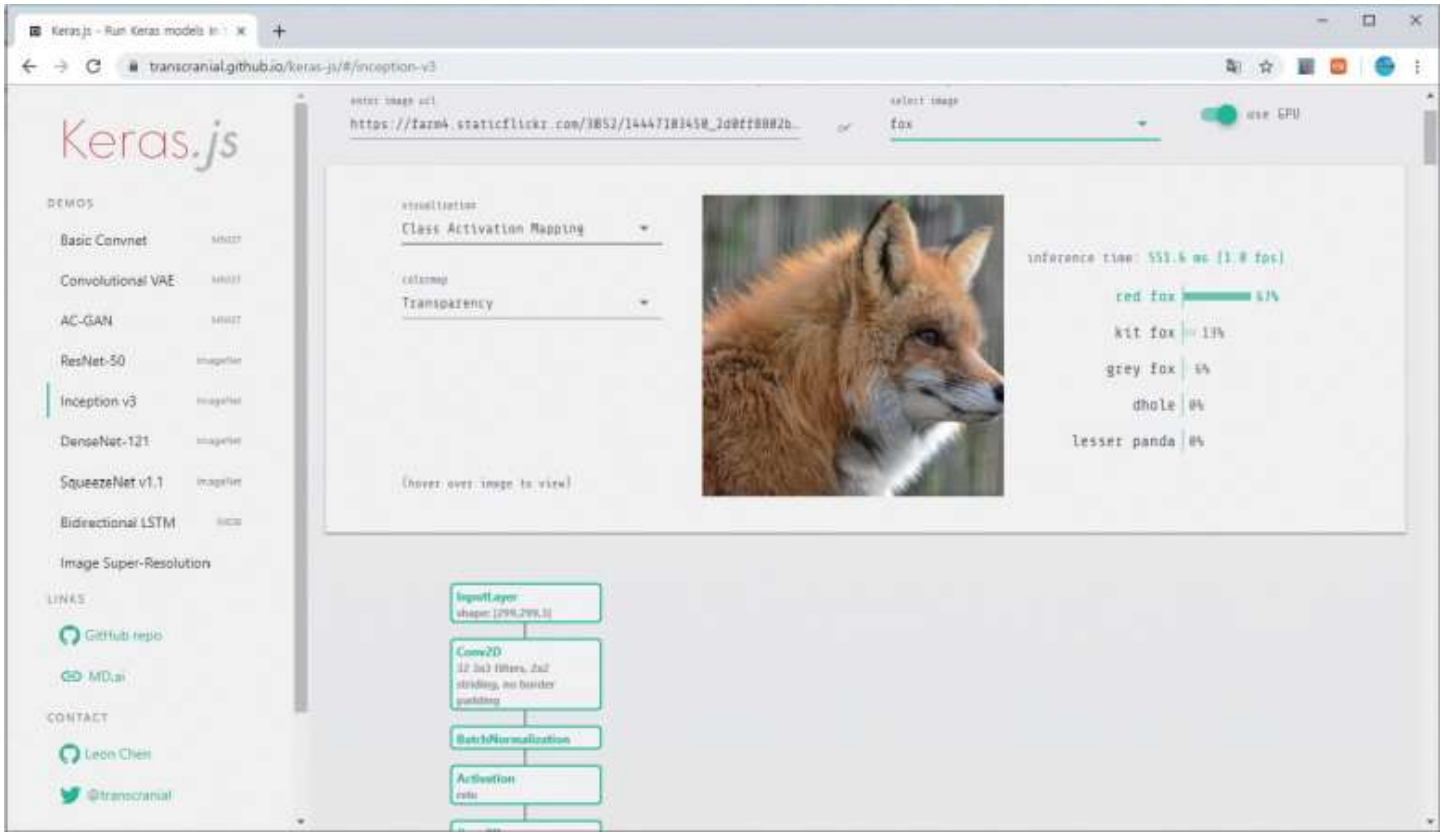




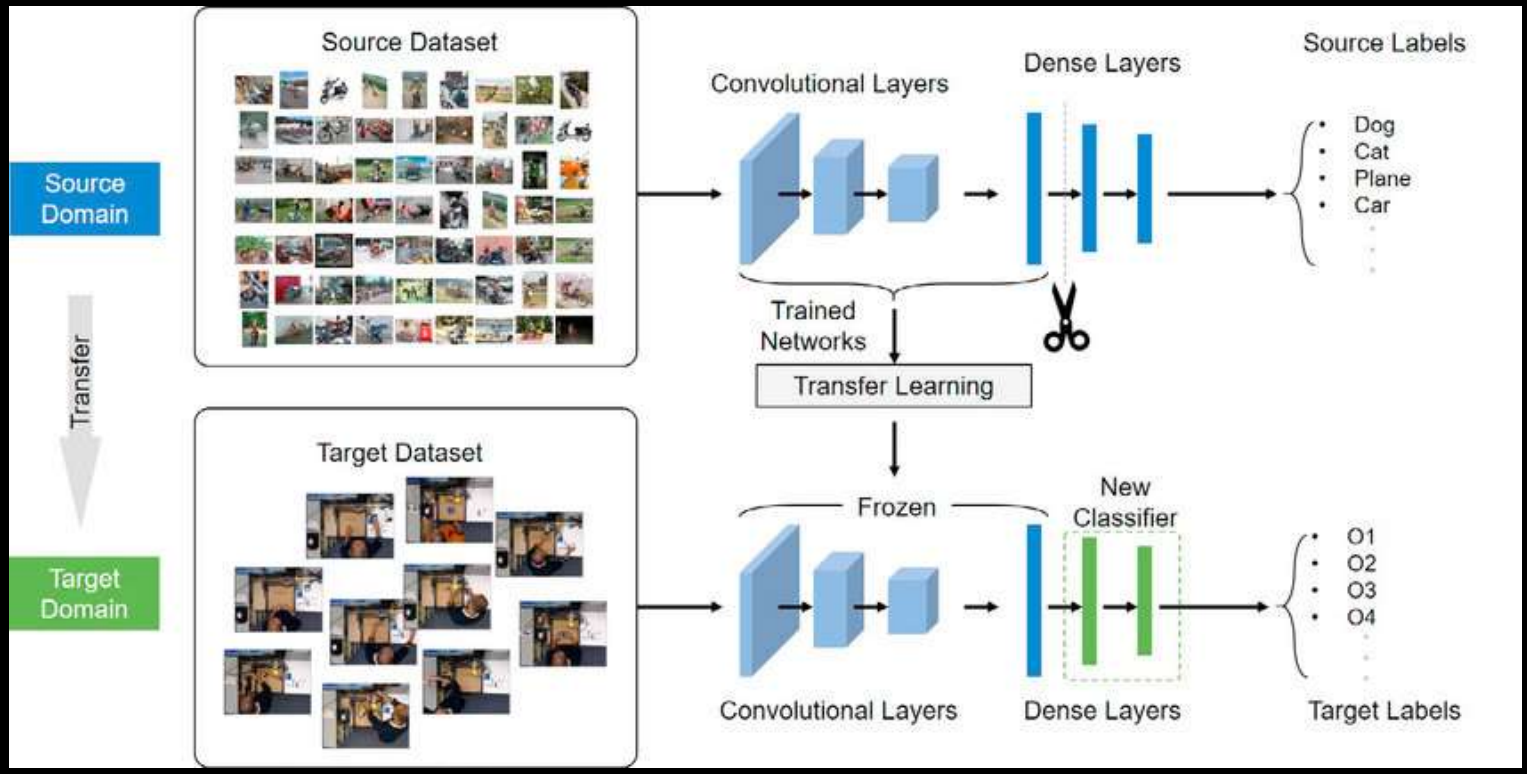
https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d

전이학습 체험

- https://transcranial.github.io/keras-js/#/resnet50

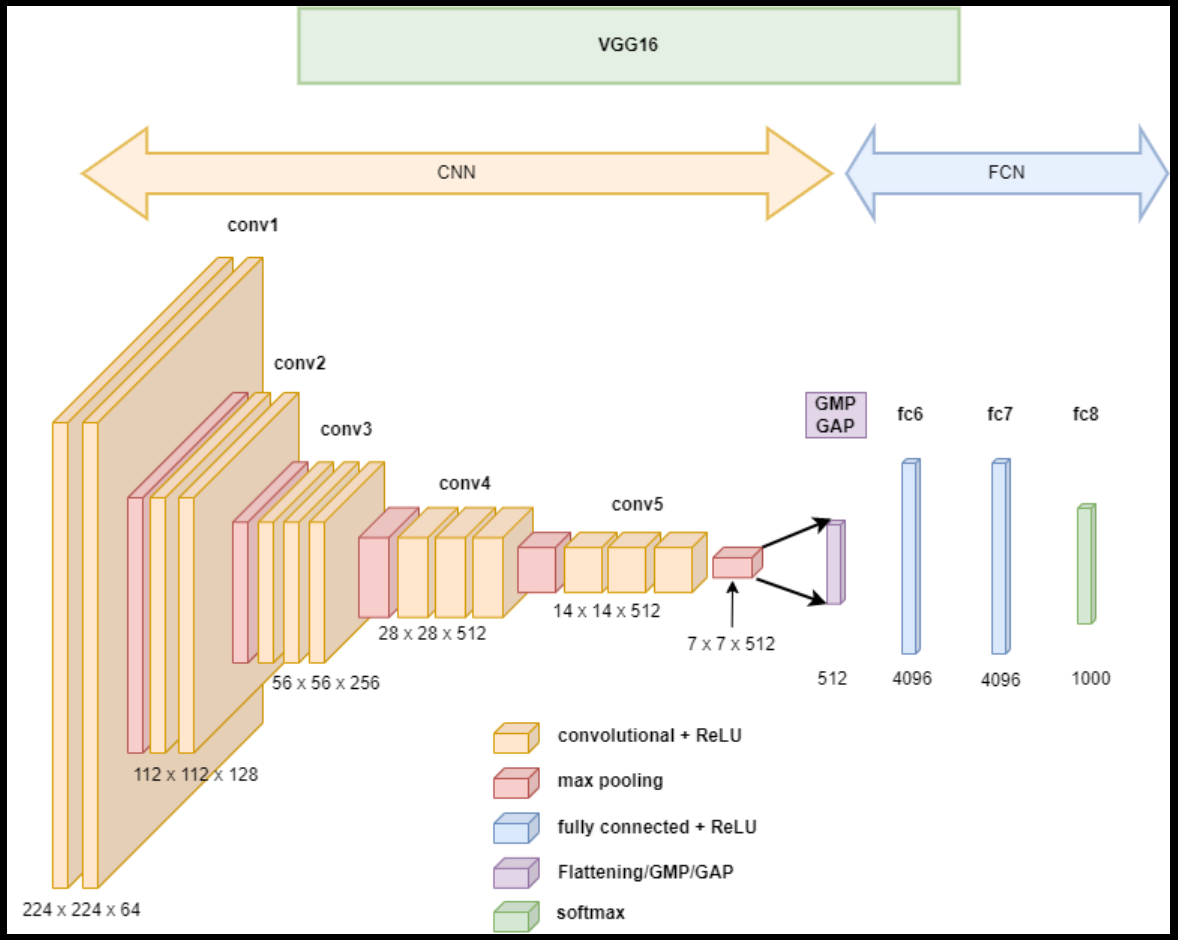


전이(전환) 학습 : ConvNet + FCN



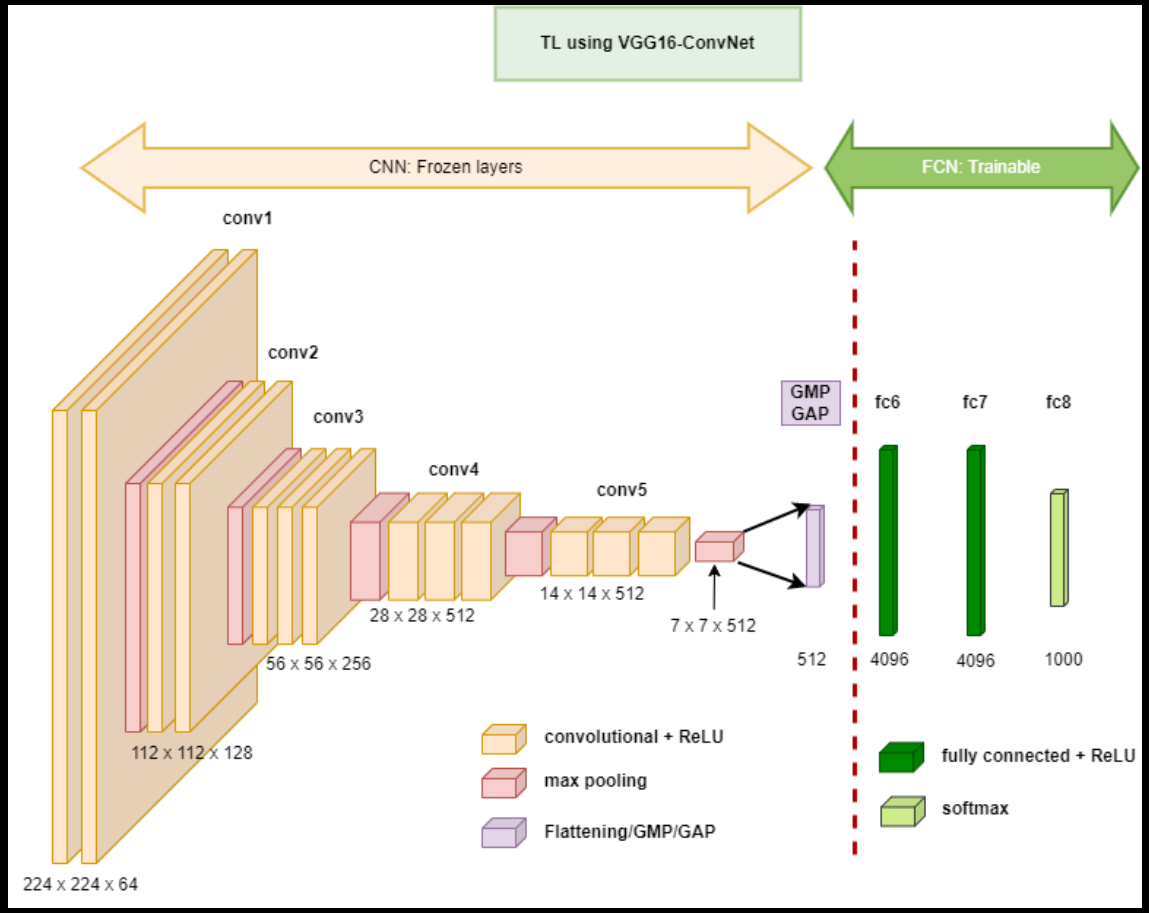
ConvNet– Very deep neural network models

ImageNet



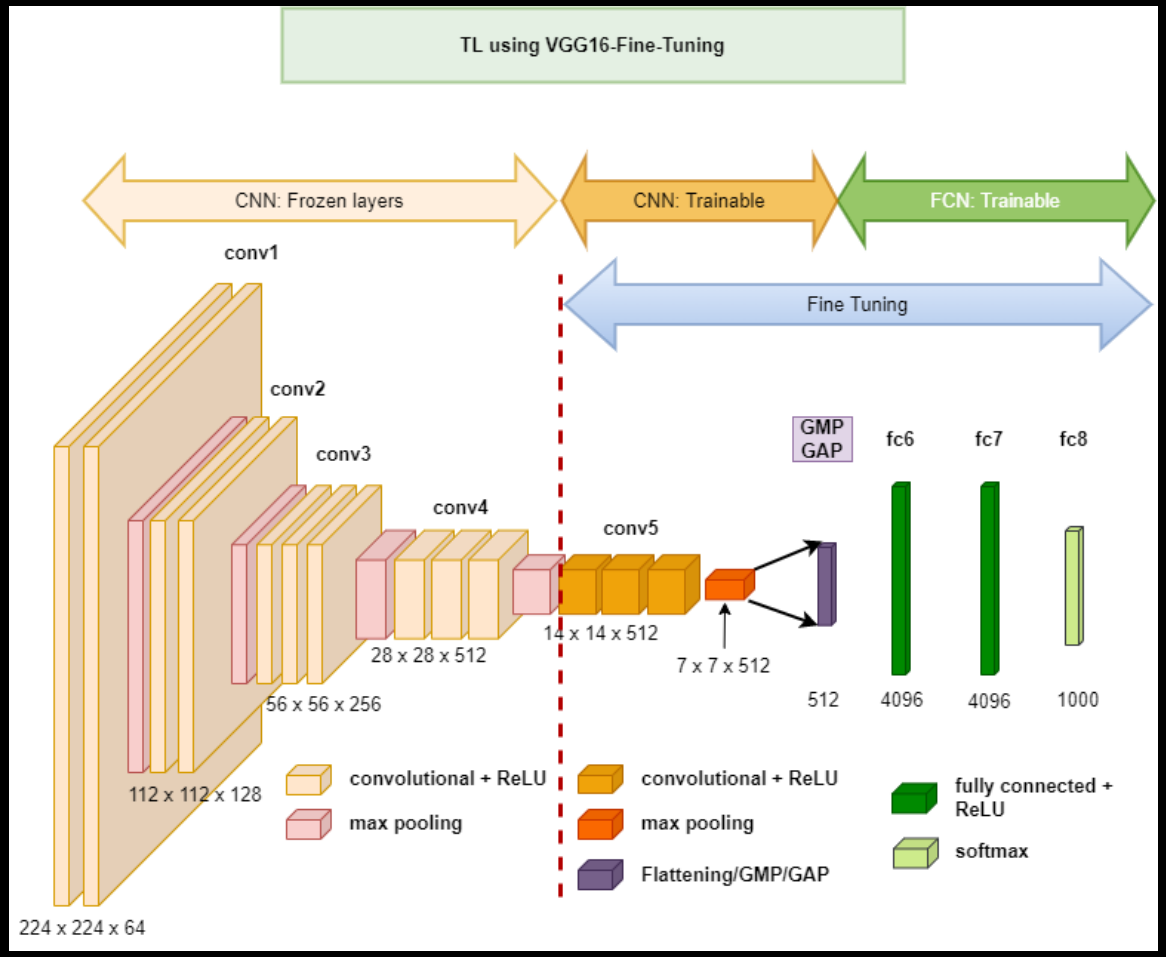
전이(전환) 학습 – Using ConvNet & training FCN

My small dataset



전이(전환) 학습 – Fine Tuning of ConvNet & FCN

My small dataset



사전 훈련된 모델을 내 프로젝트에 맞게 재정의하기

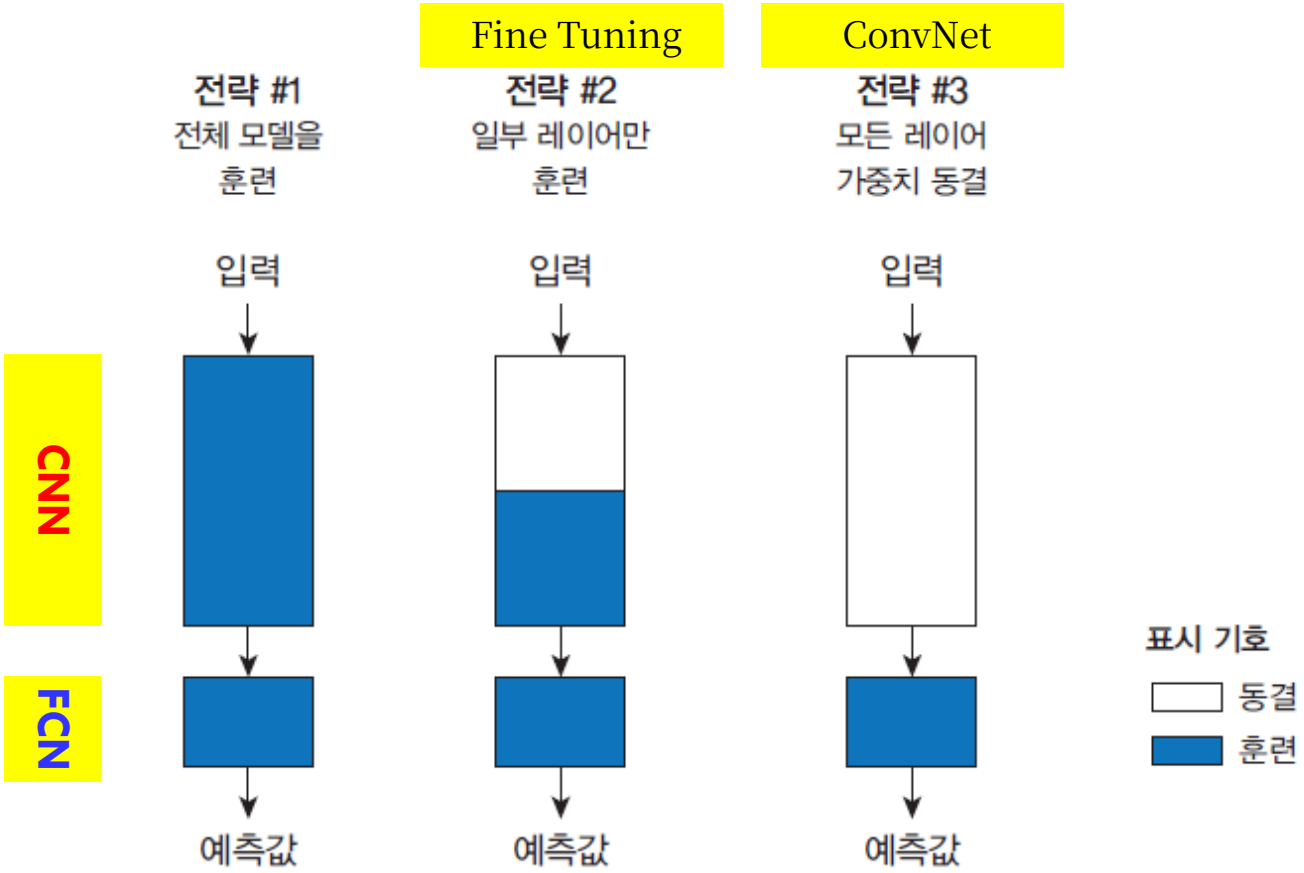


그림 10-13 사전 훈련 모델을 사용하는 3가지 방법

예제 #1

- 첫 번째 예제는 ResNet50을 다운로드받아서 변경하지 않고 그대로 사용해 보자. 구체적으로 인터넷에서 강아지 사진을 다운받아서 올바르게 인식하는지를 보자

전이학습: resnet50 이용

```
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.resnet50 import preprocess_input,
decode_predictions
import numpy as np

model = ResNet50(weights='imagenet')

img_path = 'dog.jpg'
img = image.load_img(img_path, target_size=(224, 224))  # 영상 크기를 변경하고
적재한다.
x = image.img_to_array(img)              # 영상을 넘파이 배열로 변환한다.
x = np.expand_dims(x, axis=0)            # 차원을 하나 늘인다. 배치 크기가 필요하다.
x = preprocess_input(x)       # ResNet50이 요구하는 전처리를 한다.

preds = model.predict(x)
print('예측:', decode_predictions(preds, top=3)[0])
```

```
예측: [('n02111889', 'Samoyed', 0.9557966), ('n02114548', 'white_wolf', 0.01857086),
('n02112018', 'Pomeranian', 0.00947881)]
```

예제 #2:사전 훈련 된 모델을 특징 추출기 전처리기로 사용

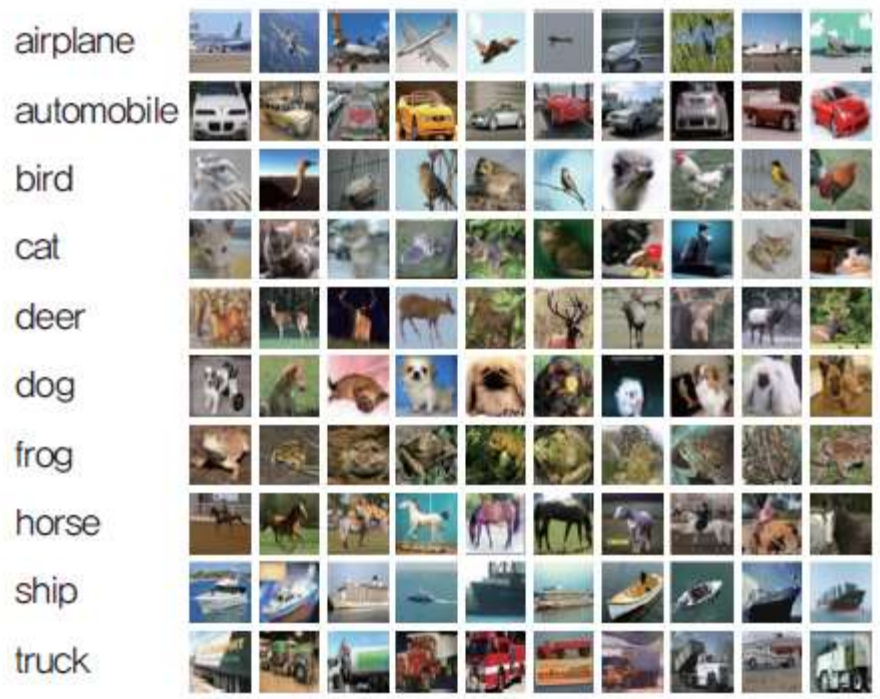
- 케라스가 제공하는 사전 훈련된 모델 중에서 MobileNet 또는 EfficientNet 을 다운로드받고 여기에 우리가 만든 분류기 레이어를 붙여서 새로운 신경망을 만든다.
- cifar10 데이터를 이용해서 전이학습.
- 이 신경망을 우리가 가지고 있는 강아지와 고양이 영상으로 학습/분류.
- 전이학습으로 생체 영상(lung X-ray)을 분류.

예제: CIFAR-10 영상 분류하기 - 전이학습

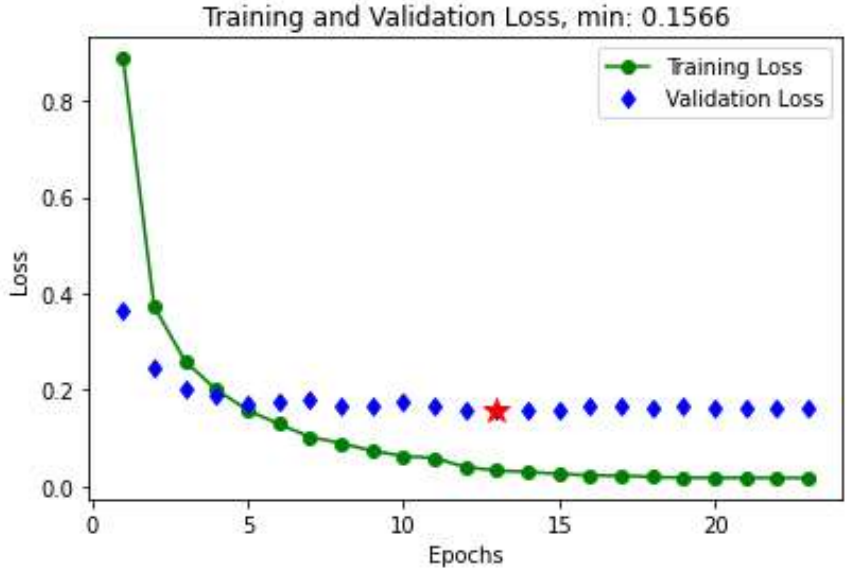
- CIFAR는 “Canadian Institute For Advanced Research”의 약자이다. CIFAR-10 데이터 세트는 CIFAR 연구소에서 CIFAR-100 데이터 세트와 함께 개발되었다.

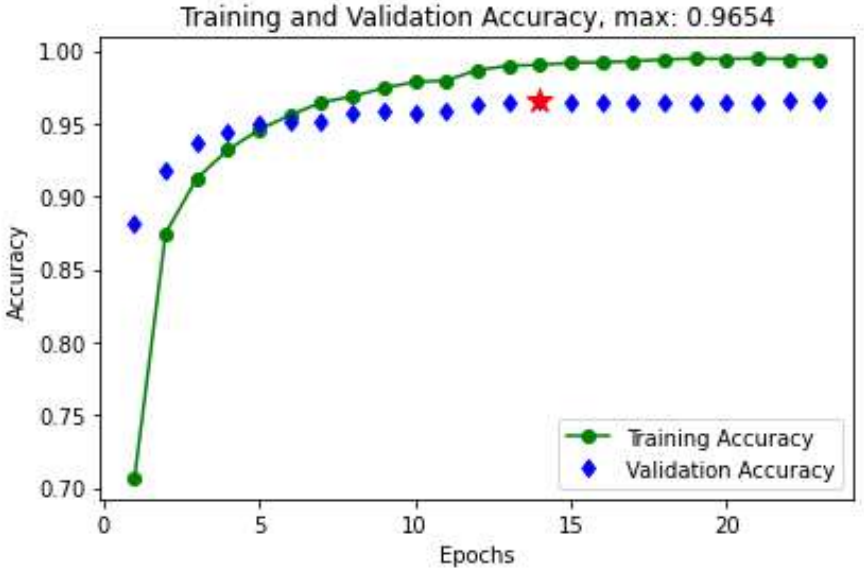
- 데이터 세트는
- (0: airplane, 1: automobile, 2: bird, 3: cat, 4: deer, 5: dog, 6: frog, 7: horse, 8: ship, 9: truck)의 10개 부류의 컬러 영상 60,000개로 이루어진다.

- 영상의 크기는 비교적 작아서 32×32×3 이다.



예제: *CIFAR-10* 영상 분류하기 - 전이학습



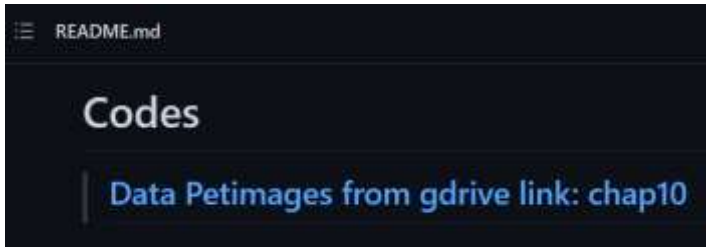


예제: 강아지와 고양이 구별하기

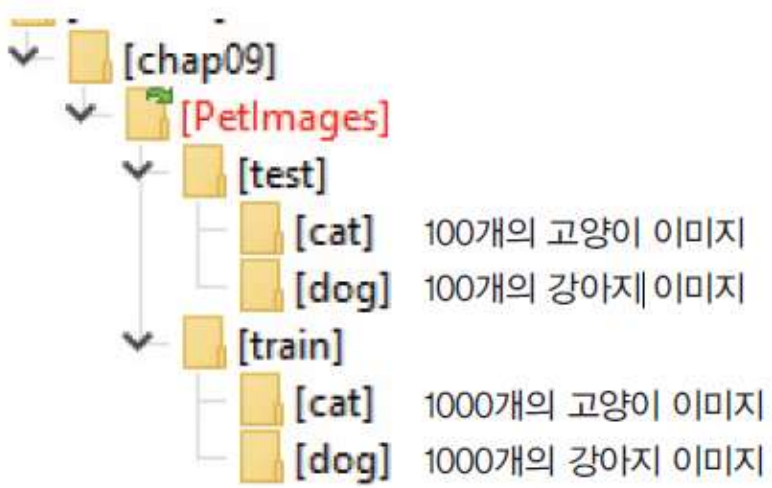
- 컨벌루션 신경망을 사용하여서 강아지와 고양이를 구별해보자. 강아지와 고양이를 구별하는문제는 2013년도에 Kaggle에서 컨테스트로 나온 적이 있다.
- Kaggle 사이트에서 데이터 세트를 받을 수도 있고 마이크로소프트사의 사이트에서 받아도 된다.

데이터 세트

- https://www.kaggle.com/c/dogs-vs-cats/data 사이트에서 다운로드받을 수 있다. 이미지들 중에서 약 2200장만 추리기로 하자. **2000장은 훈련 데이터로 사용하고 200장은 테스트용으로 사용**한다. 다음과 같이 디렉토리를 만들고 여기에 이미지들은 나누어서 저장한다. 각 이미지들은 크기가 다르다.



https://drive.google.com/file/d/1VEs5Hz2iThTqP_mRoxQgl8PrZVjwszod/view?usp=sharing



라이브러리 설치 (기 설치 시 불필요)

- 이 예제는 pillow라고 하는 이미지 처리 라이브러리를 사용한다.

```
(base) C:\Users\chun> activate deep
(deep) C:\Users\chun> pip install pillow
(deep) C:\Users\chun> pip install matplotlib
```

이미지 출력

- 1개의 이미지만 화면에 출력해보자.

```
import matplotlib.pyplot as plt
from matplotlib.image import imread

# sample image
image = imread('PetImages/train/dog/1.jpg')
image.shape
plt.imshow(image)
plt.show()
```

신경망 모델 생성

- 1개의 이미지만 화면에 출력해보자.

```
from tensorflow.keras import models, layers

train_dir = './Petimages/train'
test_dir = './Petimages/test'

model = models.Sequential()
model.add(layers.Conv2D(32,(3,3), activation='relu', input_shape=(128,128,3)))
model.add(layers.MaxPooling2D(2,2))
model.add(layers.Conv2D(64,(3,3), activation='relu'))
model.add(layers.MaxPooling2D(2,2))
model.add(layers.Flatten())
model.add(layers.Dense(units=512, activation='relu'))
model.add(layers.Dense(units=1, activation='sigmoid'))

model.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

이미지 전처리

① JPG 이미지 파일을 읽는다

② JPG 압축을 풀어서 RGB 형태로 픽셀값을 복원한다.

③ 픽셀값들은 실수형식의 넘파이 텐서로 변환한다.

④ 0~255 사이의 픽셀값들을 0.0~1.0 사이의 실수로 스케일링한다.

이미지 전처리 _ 데이터 증대

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

train_datagen = ImageDataGenerator(rescale = 1./255, shear_range = 0.2,
  zoom_range = 0.2, horizontal_flip = True)

test_datagen = ImageDataGenerator(rescale = 1./255) # train과의 차이를 주목

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(128, 128),
    batch_size=20,
    class_mode = 'binary')

test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(128, 128),
    batch_size=20,
    class_mode = 'binary')
```

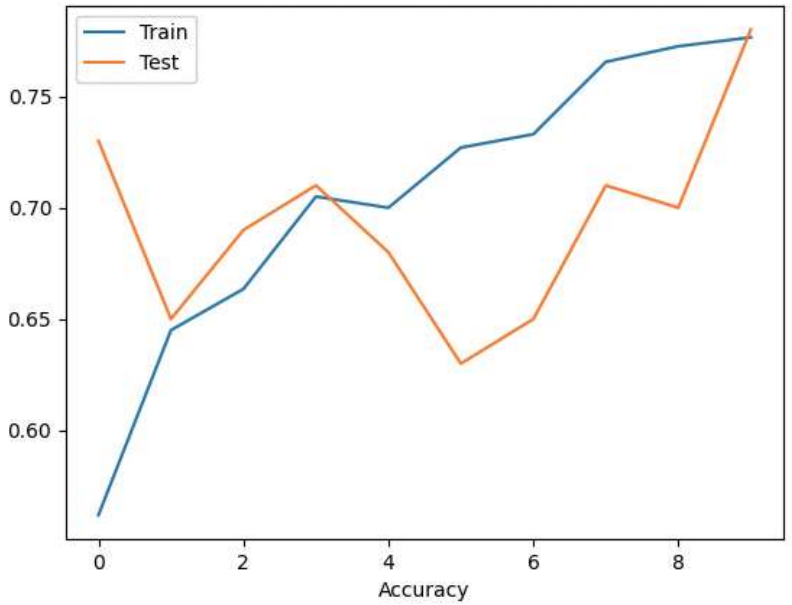
훈련

```
history = model.fit(
    train_generator, steps_per_epoch = 100, epochs=10,
    validation_data=test_generator, validation_steps=5)

# accuracy graph
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.xlabel('Epoch')
plt.xlabel('Accuracy')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```

```
Epoch 10/10
100/100 [==============================] - 19s 191ms/step - loss: 0.4509 -
accuracy: 0.7765 - val_loss: 0.5774 - val_accuracy: 0.7800
```

학습 결과 그래프

```
# accuracy graph
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.xlabel('Epoch')
plt.xlabel('Accuracy')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```



cats & dogs : **MobileNet**

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Dense,GlobalAveragePooling2D
from tensorflow.keras.applications import MobileNet
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.mobilenet import preprocess_input
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
```

cats & dogs : **MobileNet**

```
base_model=MobileNet(weights='imagenet',include_top=False) #imports the
mobilenet model and discards the last 1000 neuron layer.

x=base_model.output  # ConvNet
x=GlobalAveragePooling2D()(x)
# Training FCN
x=Dense(1024,activation='relu')(x)
x=Dense(1024,activation='relu')(x)
x=Dense(512,activation='relu')(x)
preds=Dense(2,activation='softmax')(x)

model=Model(inputs=base_model.input,outputs=preds)

# Fine Tuning
for layer in model.layers[:20]:
    layer.trainable=False
for layer in model.layers[20:]:
    layer.trainable=True
```

cats & dogs : **MobileNet**

```
train_datagen=ImageDataGenerator(preprocessing_function=preprocess_input)

train_generator=train_datagen.flow_from_directory('./Petimages/',
                                                  target_size=(128,128),
                                                  color_mode='rgb',
                                                  batch_size=32,
                                                  class_mode='categorical',
                                                  shuffle=True)


model.compile(optimizer='Adam',loss='categorical_crossentropy',metrics=['accuracy'])

step_size_train=train_generator.n//train_generator.batch_size
model.fit_generator(generator=train_generator,
                    steps_per_epoch=step_size_train,
                    epochs=5)
```

실행 결과: *cats & dogs* - **MobileNet**

```
WARNING:tensorflow:`input_shape` is undefined or non-square, or `rows` is not in
[128, 160, 192, 224]. Weights for input shape (224, 224) will be loaded as the
default.
Found 2200 images belonging to 2 classes.
Epoch 1/5
68/68 [==============================] - 30s 434ms/step - loss: 0.4500 -
accuracy: 0.8875
Epoch 2/5
68/68 [==============================] - 30s 443ms/step - loss: 0.2678 -
accuracy: 0.9096
Epoch 3/5
68/68 [==============================] - 29s 430ms/step - loss: 0.2181 -
accuracy: 0.9179
Epoch 4/5
68/68 [==============================] - 29s 431ms/step - loss: 0.1864 -
accuracy: 0.9290
Epoch 5/5
68/68 [==============================] - 29s 431ms/step - loss: 0.1646 -
accuracy: 0.9470
```

실행 결과: *cats & dogs* - **MobileNet**

Test data에 대한 평가

- Simple CNN : 70 %
- ConvNet[MobileNet] : 90 %

```
In [4]: test_datagen =
ImageDataGenerator(preprocessing_function=preprocess_input)

In [5]: test_generator = test_datagen.flow_from_directory(
   ...:     './Petimages/test',
   ...:     target_size=(128, 128),
   ...:     color_mode='rgb',
   ...:     batch_size=32,
   ...:     class_mode='categorical',
   ...:     shuffle=True)
Found 200 images belonging to 2 classes.

In [6]: model.evaluate(test_generator)
7/7 [==============================] - 1s 70ms/step - loss: 0.5604
- accuracy: 0.9000
Out[6]: [0.5603550672531128, 0.8999999761581421]
```

Summary

- 영상 인식(image recognition)란 영상 안의 물체를 인식하거나 분류하는 것이다.
- 영상 인식에 많이 사용되는 신경망은 컨볼루션 인공신경망(CNN)이다. 컨볼루션 신경망(CNN)은 동물의 조직에서 영감을 얻어서 만들어진 신경망이다. CNN에서 뉴런의 수용 공간은 다른 뉴론들과 약간 겹치게 된다. 컨볼루션 신경망은 영상 및 비디오 인식, 추천 시스템 및 자연 언어 처리 분야에서 폭넓게 응용되고 있다.
- 풀링(Pooling)이란 서브 샘플링이라고도 하는 것으로 입력 데이터의 크기를 줄이는 것이다.
- 데이터 증대(data augmentation)는 한정된 데이터에서 여러 가지로 변형된 데이터를 만들어내는 기법이다.
- 케라스에서는 save(), load_model() 메소드를 이용하여 가중치를 저장하거나 불러올 수 있다.
- 전이 학습(transfer learning)은 하나의 문제에 대해 학습한 신경망의 모델과 가중치를, 새로운 문제에 적용하는 것이다.

Q & A