

딥러닝 – Part2_v1

2020-12-08 이동환



목차

Part2 – CNN : 딥러닝 고려사항(keras)

1. Preprocessing Data
2. Train, Test, Validation Split
3. Image Data Generator
4. K-fold Cross Validation
5. Weight Initialization
6. Regularization
7. Batch Normalization
8. Activation Function
9. Dropout
10. Loss Function
11. Optimization
12. Batch, Epoch, Iteration
13. Callbacks
14. Transfer Learning
15. Grad-CAM(Class Activation Map)

Preprocessing Data

1. Raw Image
(.zip, .csv ..)



Load images, Grayscale, Reshape ...

2. Encoding

from sklearn.preprocessing import __

- LabelEncoder : 0부터 n까지 정수로 변환
- OneHotEncoder, LabelBinarizer : 두 결과는 비슷
(사용법 참고)

```
Data: ['cold' 'cold' 'warm' 'cold' 'hot' 'hot' 'warm' 'cold' 'warm' 'hot']
Label Encoder: [0 0 2 0 1 1 2 0 2 1]
OneHot Encoder: [[ 1.  0.  0.]
 [ 0.  0.  1.]
 [ 1.  0.  0.]
 [ 0.  1.  0.]
 [ 0.  1.  0.]
 [ 0.  0.  1.]
 [ 1.  0.  0.]
 [ 0.  0.  1.]
 [ 0.  1.  0.]]
Label Binarizer: [[1 0 0]
 [0 0 1]
 [1 0 0]
 [0 1 0]
 [0 1 0]
 [0 0 1]
 [1 0 0]
 [0 0 1]
 [0 1 0]]
```

Train, Test, Validation Split



[사용 이유]

overfitting 방지

[사용법]

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, Y_train, Y_test = train_test_split()
```

- Validation set은 위 함수를 한번 더 사용하여 생성함
- Default : 0.25%-test, 0.75%-train
- 6:3:1(train:validation:test)

Image Data Generator

[Option]

rotation_range : 원본 이미지 회전

width(height)_shift_range : 원본 이미지 이동

rescale : 0~1 사이의 값으로 변환

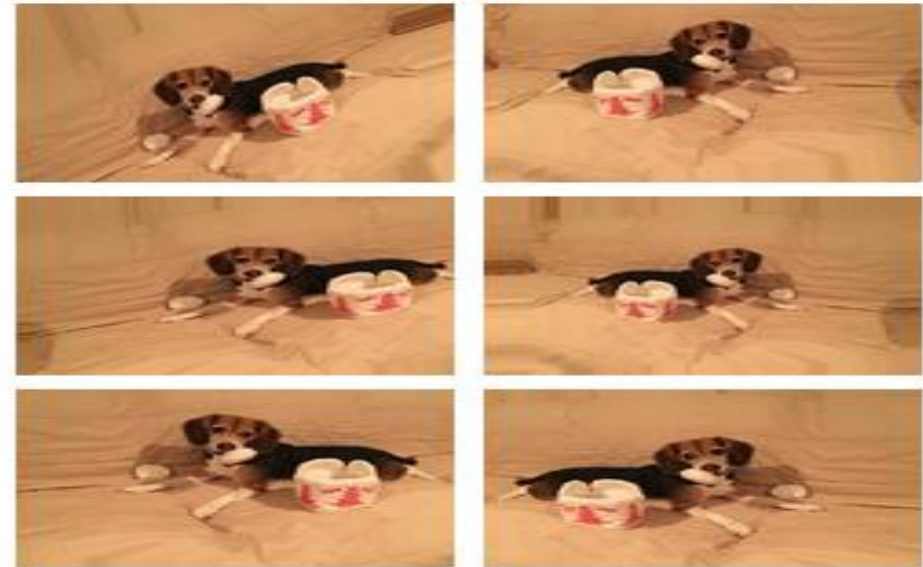
zoom_range : 원본 이미지 확대/축소

horizontal_flip : 수평방향 뒤집기

vertical_flip : 수직방향 뒤집기

fill_mode : 새로운 공간 채우는 방법

Augmented Images



[사용 이유]

부족한 데이터셋 보충하여 성능 향상, overfitting 방지

[사용법]

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

K-fold Cross Validation

| | A | B | C | D | E |
|------------------------------|-------|-------|-------|-------|-------|
| Cross Validation Iteration 1 | Test | Train | Train | Train | Train |
| Cross Validation Iteration 2 | Train | Test | Train | Train | Train |
| Cross Validation Iteration 3 | Train | Train | Test | Train | Train |
| Cross Validation Iteration 4 | Train | Train | Train | Test | Train |
| Cross Validation Iteration 5 | Train | Train | Train | Train | Test |

[사용 이유]

[Train으로 학습, Validation으로 evaluate, test에 대한 predict]-K번 반복, accuracy 획득(평균)

목적 : fold에서 **최고의 모델을 뽑아내는 것이 아니라**,

여러 모델 중 데이터 분포에 대한 평균 정확도 비교를 통해 모델 선택을 위해 사용함

Kfold(독립적이고, 동일한 분포 데이터), StratifiedKFold(동일 분포 아닌 경우) 등을 사용

[사용법]

```
from sklearn.model_selection import Kfold
```

Weight Initialization

◦ Xavier Normal Initialization

$$W \sim N(0, Var(W))$$

$$Var(W) = \sqrt{\frac{2}{n_{in} + n_{out}}}$$

(n_{in} : 이전 layer(input)의 노드 수, n_{out} : 다음 layer의 노드 수)

```
# Xavier initialization
# Glorot et al. 2010
W = np.random.randn(fan_in, fan_out)/np.sqrt(fan_in)
```

Xavier(Glorot) Initialization
- sigmoid, tanh

[사용 이유]

기울기 소실 문제(Part1 참고) 해결을 위한 방법

[사용법]

```
kernel_initializer = tensorflow.keras.initializers.he_normal()
```

◦ He Normal Initialization

$$W \sim N(0, Var(W))$$

$$Var(W) = \sqrt{\frac{2}{n_{in}}}$$

(n_{in} : 이전 layer(input)의 노드 수)

```
# He et al. 2015
W = np.random.randn(fan_in, fan_out)/np.sqrt(fan_in/2)
```

He Initialization
- ReLU

Regularization

$$||w||_1 = \sum_{i=1}^n |w_i|$$

L1 노름

가중치의 절대값에 비례하는 비용을
Loss function에 추가

$$||w||_2 = \sqrt{\sum_{i=1}^n |w_i|^2}$$

L2 노름

가중치의 제곱에 비례하는 비용을
Loss function에 추가

[사용 이유]

overfitting 방지, weight가 작아지도록 학습(outlier : 비이상적 가중치 영향 감소를 위함)

[사용법]

```
from tensorflow.keras.regularizers import l2
```


Batch Normalization

[사용 이유]

기울기 소실 문제(Part1 참고) 해결을 위한 방법

[사용법]

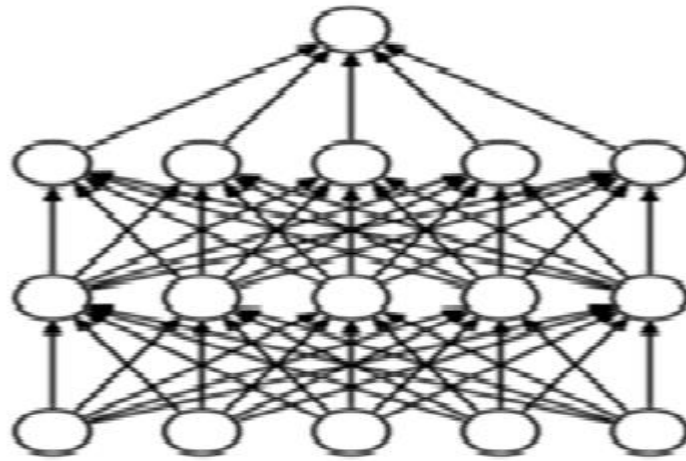
```
from tensorflow.keras.layers import BatchNormalization  
model.add(BatchNormalization())
```

Activation Function

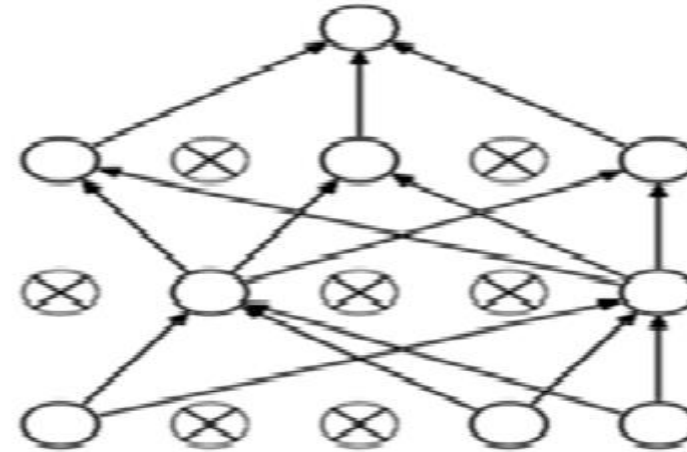
| Activation Function | 사용법 |
|---------------------|---|
| Sigmoid | activation='sigmoid' |
| Tanh | activation='tanh' |
| ReLU | activation='relu' |
| selu | activation='selu' |
| Swish | <pre>from keras import backend K from keras.utils.generic_utils import get_custom_objects from keras.layers import Activation def swish(x) : return K.sigmoid(x) * x get_custom_objects().update({'swish': Activation(swish)})</pre> 이후, 위와 같이 사용 |
| Mish | <pre>def mish(x) : return x * K.tanh(K.softplus(x))</pre> |

[사용 이유]
기울기 소실 문제(Part1 참고) 해결을 위한 방법

Dropout



(a) Standard Neural Net



(b) After applying dropout.

학습 시 전체 Node 중에서 일부 Node를 쉬게 한다.
(Test에는 Dropout 적용하지 않음)

[사용 이유]

overfitting 방지, 성능 향상

[사용법]

```
from tensorflow.keras.layers import Dropout
```

Loss Function

회귀

- 평균 제곱 오차 : `mean_square_error`
- 평균 절대 오차 : `mean_absolute_error`

분류

- 이진 분류 : `binary_crossentropy`(결과 : 0과 1로 Encoding)
- 3개 이상 분류 : `categorical_crossentropy`(결과 : 0과 1로 Encoding)
- 3개 이상 분류 : `sparse_categorical_crossentropy`(결과 : 0,1,2와 같은 정수 형태로 Encoding)

[사용 이유]

높은 정확도를 가진 `weight`와 `bias`를 구하기 위함

[사용법]

`loss = 'categorical_crossentropy'`

Optimization

| 고급 경사 하강법 | 사용법 |
|--|--|
| 확률적 경사 하강법 (SGD) + 모멘텀 (Momentum) + 네스테로프 모멘텀 (NAG) | from tensorflow.keras.optimizers sgd = optimizers.SGD(lr=0.01, momentum=0.9, nesterov=True, decay=1e-6) - decay : 0보다 크거나 같은 float값. 업데이트 마다 적용되는 학습률의 감소율 |
| 아다그라드 (Adagrad) | adagrad = optimizers.Adagrad (default lr = 0.01) |
| 알엠에스프롭 (RMSProp) | rmsprop = optimizers.RMSprop (default lr = 0.001) |
| 아담 (Adam) | adam = optimizers.Adam (default lr = 0.001) |

[사용 이유]

정확하고 빠르게 Loss Function을 최소화 하는 weight 탐색

Batch, Epoch, Iteration



Ex) 데이터 셋 : 2000개, Epoch : 20, batch_size = 500

1 Epoch = 2000/500, 4번의 iteration으로 구성되어 있음

각 Epoch마다 4번의 iteration을 수행

즉, 전체 데이터셋에 대해 20번의 학습 수행

$20 \times 4 = 80$ 번의 학습 수행(iteration기준)

[사용 이유]

데이터가 너무 많은 경우 여러 번으로 나눠 학습 수행

[사용법]

epochs = OO , batch_size = OO

Callbacks

EarlyStopping

→ 더 좋은 성능의 모델이 발견되지 않는 경우 학습 중단

ReduceLROnPlateau

→ 모델 성능이 향상되지 않는 경우 Learning rate 조정

ModelCheckpoint

→ 가장 좋은 성능인 경우의 모델을 저장

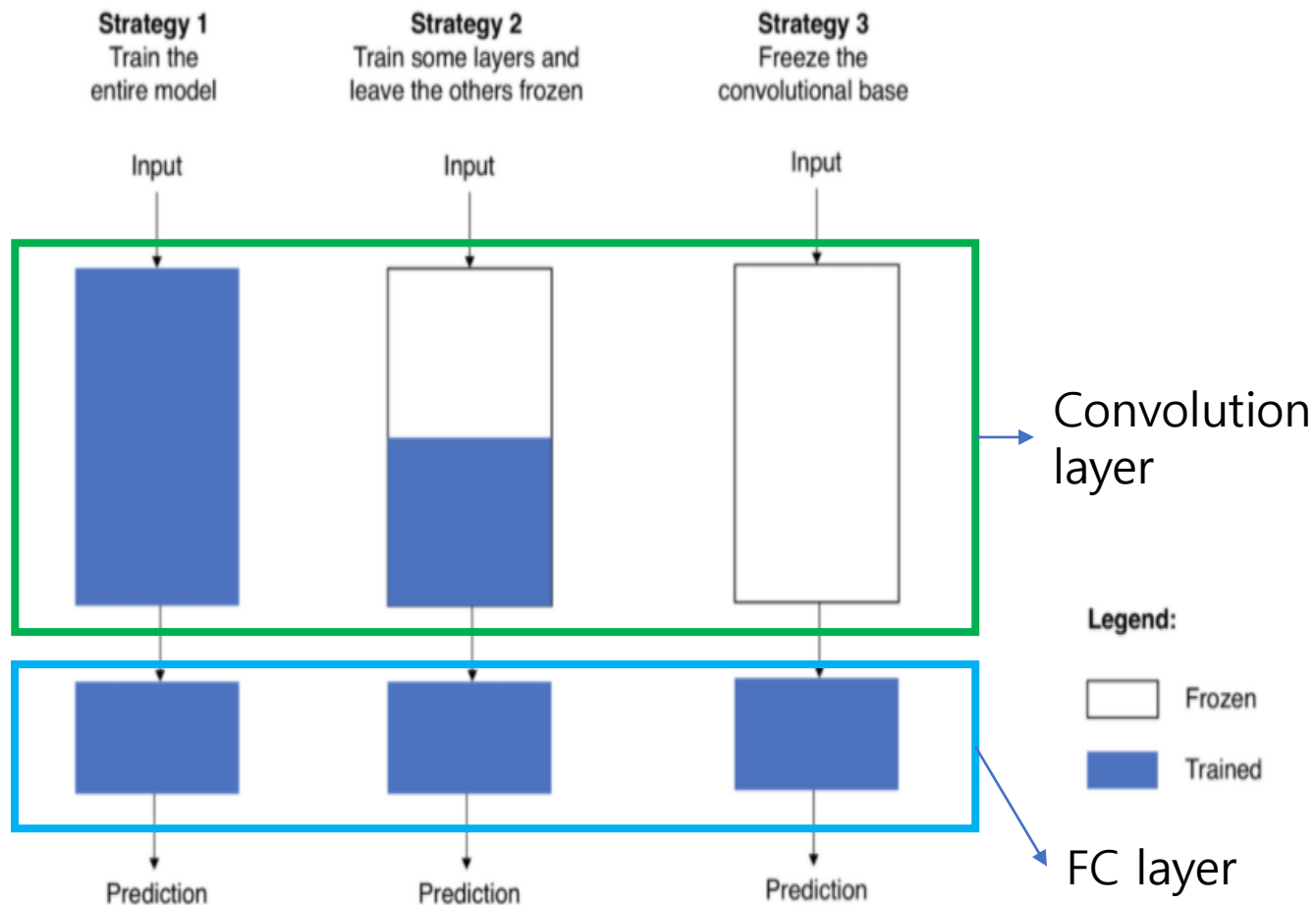
[사용 이유]

학습 과정의 특정 단계에서 적용하여 상태 파악

[사용법]

```
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelCheckpoint
```

Transfer Learning



- 전략 1 : 사전학습 모델 구조만 사용하고, 내 데이터셋에 맞게 전부 새로 학습 시키는 방법
 - 데이터셋 많고, 유사성 작은 경우
- 전략 2 : Convolution layer low level 계층 (일반적인 특징 추출)은 고정시키고, high level는 다시 학습(구체적인 특징)
 - 데이터셋이 작고 유사성 작은 경우
 - High level 범위 증가
 - 데이터셋이 많고 유사성 높은 경우
 - High level 범위 감소
- 전략 3 : Convolution layer 전체는 고정시키고, FC layer만 다시 학습시키는 방법
 - 데이터셋 작고, 유사성 높은 경우

전이학습 사용법(전략1)

```
# init the models
vgg_model = vgg16.VGG16(weights='imagenet')
```

1. 모델 load

```
# load an image in PIL format
original = load_img(filename, target_size=(224, 224))
print('PIL image size', original.size)
plt.imshow(original)
plt.show()

# convert the PIL image to a numpy array
# IN PIL - image is in (width, height, channel)
# In Numpy - image is in (height, width, channel)
numpy_image = img_to_array(original)
plt.imshow(np.uint8(numpy_image))
plt.show()
print('numpy array size', numpy_image.shape)

# Convert the image / images into batch format
# expand_dims will add an extra dimension to the data at a particular axis
# We want the input matrix to the network to be of the form (batchsize, height, width, channels)
# Thus we add the extra dimension to the axis 0
image_batch = np.expand_dims(numpy_image, axis=0)
print('image batch size', image_batch.shape)
plt.imshow(np.uint8(image_batch[0]))
```

2. 이미지 크기 및 형태 변경

```
# prepare the image for the VGG model
processed_image = vgg16.preprocess_input(image_batch.copy())

# get the predicted probabilities for each class
predictions = vgg_model.predict(processed_image)
# print predictions
# convert the probabilities to class labels
# we will get top 5 predictions which is the default
label_vgg = decode_predictions(predictions)
# print VGG16 predictions
for prediction_id in range(len(label_vgg[0])):
    print(label_vgg[0][prediction_id])

('n02123597', 'Siamese_cat', 0.3093419)
('n01877812', 'wallaby', 0.080341235)
('n02326432', 'hare', 0.075098425)
('n02325366', 'wood_rabbit', 0.05053069)
('n03223299', 'doormat', 0.04817361)
```

3. 예측

전이학습 사용법(전략2) – VGG16 load

```
from tensorflow.python.keras.applications.vgg16 import VGG16

model_vgg16 = VGG16(include_top=False, weights='imagenet', input_shape=(150,150,3))
```

| | | |
|-------------------------------|---------------|-----------|
| flatten (Flatten) | (None, 25088) | 0 |
| fc1 (Dense) | (None, 4096) | 102764544 |
| fc2 (Dense) | (None, 4096) | 16781312 |
| predictions (Dense) | (None, 1000) | 4097000 |
| ===== | | |
| Total params: 138,357,544 | | |
| Trainable params: 138,357,544 | | |
| Non-trainable params: 0 | | |

| | | |
|----------------------------------|-----------------------|---------|
| Model: "vgg16" | | |
| Layer (type) | Output Shape | Param # |
| ===== | | |
| input_1 (InputLayer) | [(None, 150, 150, 3)] | 0 |
| block1_conv1 (Conv2D) | (None, 150, 150, 64) | 1792 |
| block1_conv2 (Conv2D) | (None, 150, 150, 64) | 36928 |
| block1_pool1 (MaxPooling2D) | (None, 75, 75, 64) | 0 |
| block2_conv1 (Conv2D) | (None, 75, 75, 128) | 73856 |
| block2_conv2 (Conv2D) | (None, 75, 75, 128) | 147584 |
| block2_pool1 (MaxPooling2D) | (None, 37, 37, 128) | 0 |
| block3_conv1 (Conv2D) | (None, 37, 37, 256) | 295168 |
| block3_conv2 (Conv2D) | (None, 37, 37, 256) | 590080 |
| block3_conv3 (Conv2D) | (None, 37, 37, 256) | 590080 |
| block3_pool1 (MaxPooling2D) | (None, 18, 18, 256) | 0 |
| block4_conv1 (Conv2D) | (None, 18, 18, 512) | 1180160 |
| block4_conv2 (Conv2D) | (None, 18, 18, 512) | 2359808 |
| block4_conv3 (Conv2D) | (None, 18, 18, 512) | 2359808 |
| block4_pool1 (MaxPooling2D) | (None, 9, 9, 512) | 0 |
| block5_conv1 (Conv2D) | (None, 9, 9, 512) | 2359808 |
| block5_conv2 (Conv2D) | (None, 9, 9, 512) | 2359808 |
| block5_conv3 (Conv2D) | (None, 9, 9, 512) | 2359808 |
| block5_pool1 (MaxPooling2D) | (None, 4, 4, 512) | 0 |
| ===== | | |
| Total params: 14,714,688 | | |
| Trainable params: 0 | | |
| Non-trainable params: 14,714,688 | | |

include_top = False

- 나의 데이터셋은 1000개의 class를 분류하지 않기 때문에, FC layer는 불러오지 않는다.
- FC layer는 내가 새로 정의하여 사용하겠다.

(이미지 : FC layer)

weight = 'imagenet'

- 가중치 초기화 시 이미지넷을 학습한 가중치를 사용하겠다.

전이학습 사용법(전략2) – FC 추가

```
from tensorflow.keras import models, layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
```

```
input_shape = model_vgg16.output_shape[1]
# model_vgg16.add(layers.Flatten())
additional_model = models.Sequential()
additional_model.add(model_vgg16)
additional_model.add(layers.Flatten())
additional_model.add(layers.Dense(4096, activation='relu'))
additional_model.add(layers.Dense(2048, activation='relu'))
additional_model.add(layers.Dense(1024, activation='relu'))
additional_model.add(layers.Dense(4, activation='softmax'))
```

```
additional_model.summary()
```

```
Model: "sequential_3"
```

| Layer (type) | Output Shape | Param # |
|---------------------|-------------------|----------|
| vgg16 (Functional) | (None, 4, 4, 512) | 14714688 |
| flatten_1 (Flatten) | (None, 8192) | 0 |
| dense_4 (Dense) | (None, 4096) | 33558528 |
| dense_5 (Dense) | (None, 2048) | 8390656 |
| dense_6 (Dense) | (None, 1024) | 2098176 |
| dense_7 (Dense) | (None, 4) | 4100 |

```
Total params: 58,766,148
```

```
Trainable params: 44,051,460
```

```
Non-trainable params: 14,714,688
```

분류하려는 Class 개수

나머지 과정(학습 및 결과 예측)은 동일함

전이학습 사용법(전략2) – 학습 layer 설정

```
import pandas as pd
pd.set_option('max_colwidth', -1)
layers = [(layer, layer.name, layer.trainable) for layer in model.layers]
pd.DataFrame(layers, columns=['Layer Type', 'Layer Name', 'Layer Trainable'])
```

시각화

```
model.trainable = False
for layer in model.layers :
    layer.trainable = False
```

전체 모델 layer 중 학습하지 않고(freeze)
사용할 layer 설정(전체)

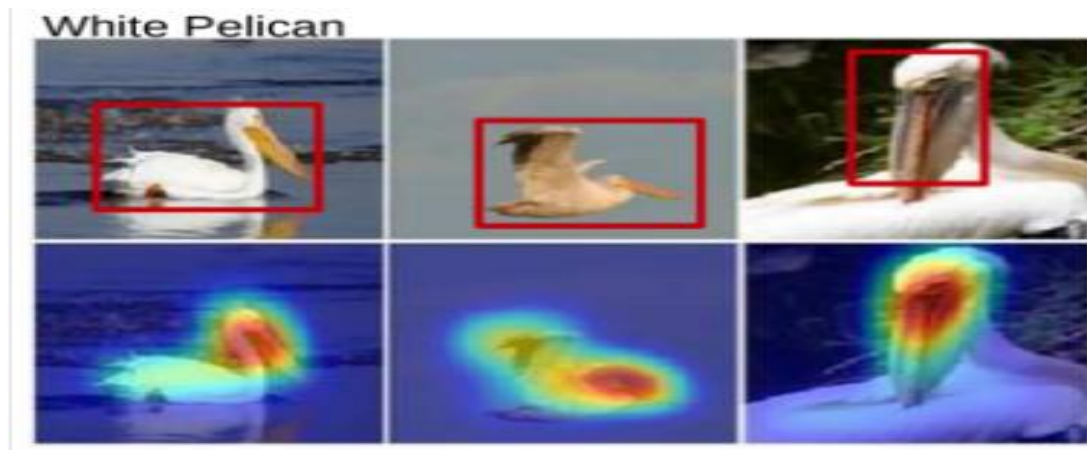
```
set_trainable = False
for layer in model.layers :
    if layer.name == 'block5_conv1' :
        set_trainable = True
    if set_trainable :
        layer.trainable = True
    else :
        layer.trainable = False
```

전체 모델 layer 중 새롭게 학습할 layer 설정(일부)
(초록 box : True로 변함)

| | Layer Type | Layer Name | Layer Trainable |
|----|--|--------------|-----------------|
| 0 | <tensorflow.python.keras.engine.input_layer.InputLayer object at 0x000001ECFBA83488> | input_5 | False |
| 1 | <tensorflow.python.keras.layers.convolutional.Conv2D object at 0x000001ECFBA83608> | block1_conv1 | False |
| 2 | <tensorflow.python.keras.layers.convolutional.Conv2D object at 0x000001ED04901FC8> | block1_conv2 | False |
| 3 | <tensorflow.python.keras.layers.pooling.MaxPooling2D object at 0x000001ED04900688> | block1_pool | False |
| 4 | <tensorflow.python.keras.layers.convolutional.Conv2D object at 0x000001ECFBA96808> | block2_conv1 | False |
| 5 | <tensorflow.python.keras.layers.convolutional.Conv2D object at 0x000001ED048F60C8> | block2_conv2 | False |
| 6 | <tensorflow.python.keras.layers.pooling.MaxPooling2D object at 0x000001ED0490EBC8> | block2_pool | False |
| 7 | <tensorflow.python.keras.layers.convolutional.Conv2D object at 0x000001ED08243048> | block3_conv1 | False |
| 8 | <tensorflow.python.keras.layers.convolutional.Conv2D object at 0x000001ED082489C8> | block3_conv2 | False |
| 9 | <tensorflow.python.keras.layers.convolutional.Conv2D object at 0x000001ED0824B908> | block3_conv3 | False |
| 10 | <tensorflow.python.keras.layers.pooling.MaxPooling2D object at 0x000001ED08254248> | block3_pool | False |
| 11 | <tensorflow.python.keras.layers.convolutional.Conv2D object at 0x000001ED0825CC48> | block4_conv1 | False |
| 12 | <tensorflow.python.keras.layers.convolutional.Conv2D object at 0x000001ED08265648> | block4_conv2 | False |
| 13 | <tensorflow.python.keras.layers.convolutional.Conv2D object at 0x000001ED08268448> | block4_conv3 | False |
| 14 | <tensorflow.python.keras.layers.pooling.MaxPooling2D object at 0x000001ED086A5548> | block4_pool | False |
| 15 | <tensorflow.python.keras.layers.convolutional.Conv2D object at 0x000001ED086A6148> | block5_conv1 | False |
| 16 | <tensorflow.python.keras.layers.convolutional.Conv2D object at 0x000001ED086B1FC8> | block5_conv2 | False |
| 17 | <tensorflow.python.keras.layers.convolutional.Conv2D object at 0x000001ED086B4548> | block5_conv3 | False |
| 18 | <tensorflow.python.keras.layers.pooling.MaxPooling2D object at 0x000001ED086BF708> | block5_pool | False |

Grad-CAM(Class Activation Map)

- Convolution Neural Network는 어떻게 사물을 예측할까?
 - 예측 모델을 사용하는 현업자들이 이해하기 쉽게
 - 구축한 예측 모델 결과가 타당함을 직관적으로 보이게
 - 예측 결과에 대한 원인을 분석하고 향후 대처 할 수 있게



Zhou, Bolei, et al. "Learning deep features for discriminative localization." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.