Keras와 함께 하는 딥러닝 기초

의료 데이터를 활용하는 딥러닝 예제 실습

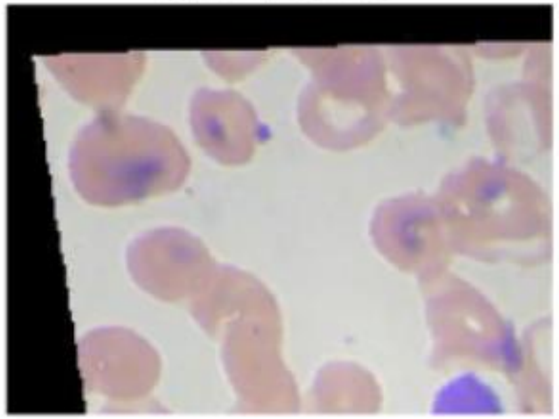
- 백혈구(white blood cell) 분류(classification)
- Skin Cancer Detection Model
- Chest X-Ray Images(Pneumonia) Dataset Classification

Problem

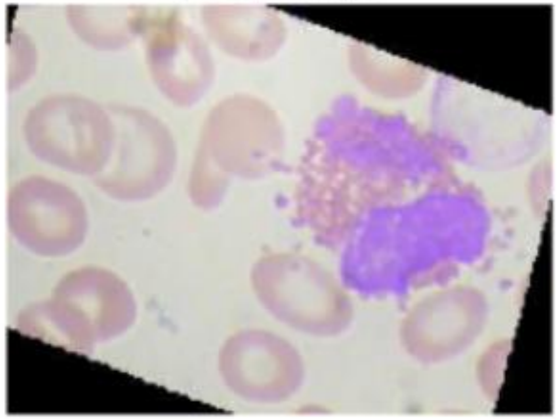
- 백혈구(white blood cell)를 분류(classification)하는 문제

- 종류
  - NEUTROPHIL : 호중구
  - EOSINOPHIL : 호산구
  - MONOCYTE : 단핵구
  - LYMPHOCYTE : 림프구

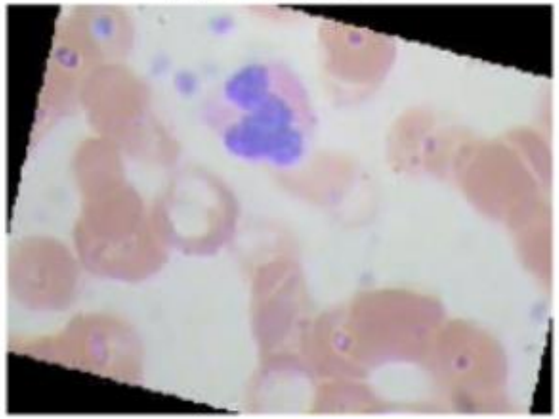
LYMPHOCYTE



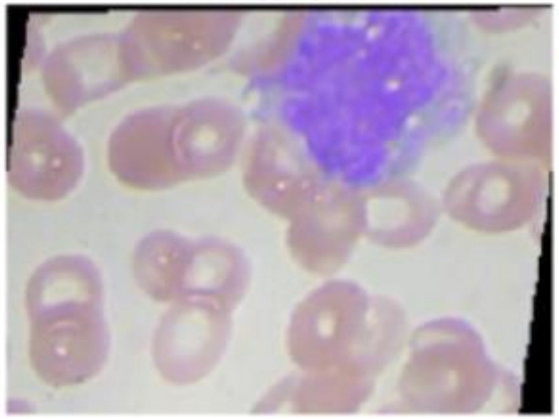
EOSINOPHIL



NEUTROPHIL



MONOCYTE



Environment

Datasets in Kaggle
https://www.kaggle.com/paultimothymooney/blood-cells

데이터셋을 받고

Google Colab
https://colab.research.google.com/

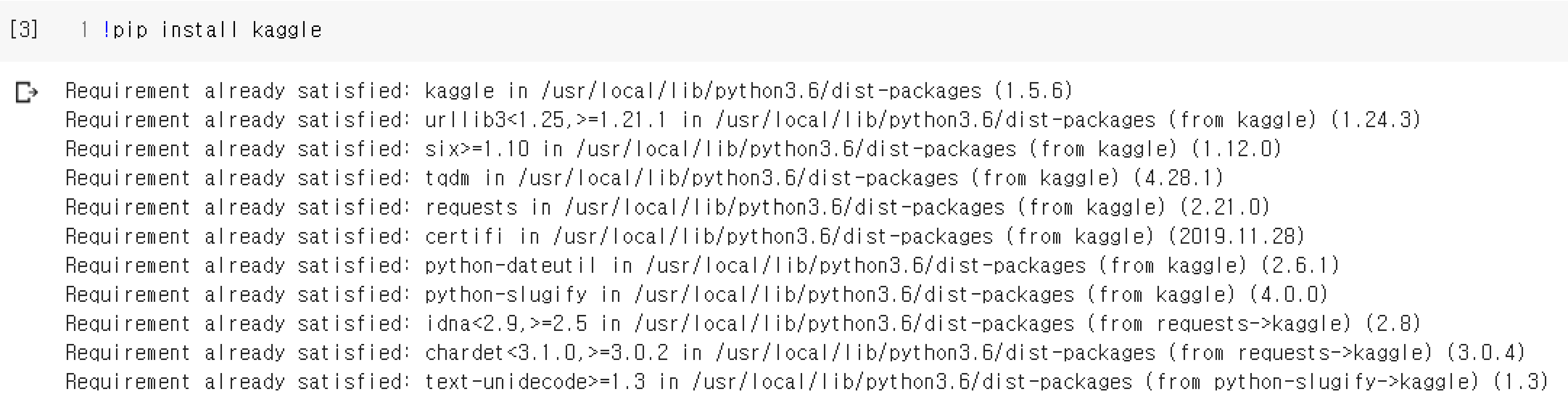
Colab에 넣어준다!

CODE
https://github.com/jjeamin/kaggle/tree/master/Blood Cells

Python

Keras(Deep Learning Framework)

Kaggle API

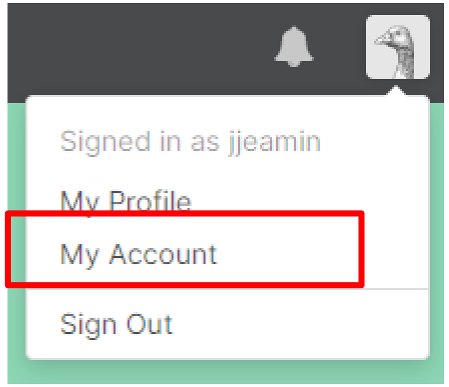


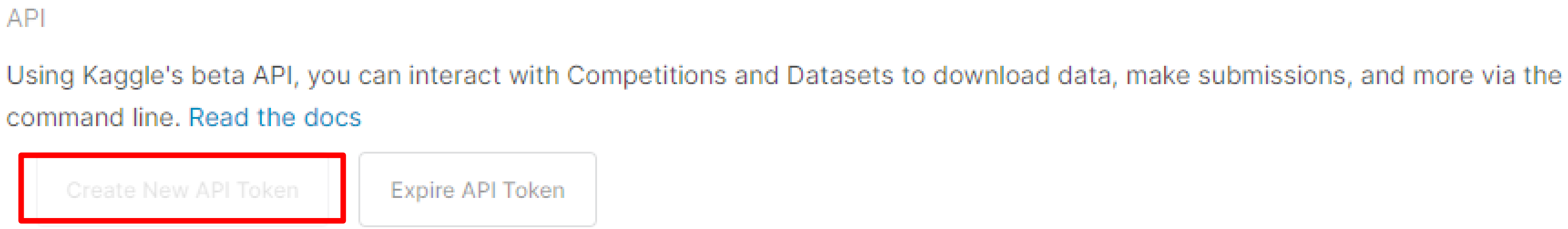
```
[3] !pip install kaggle
```

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.6/dist-packages (1.5.6)
Requirement already satisfied: urllib3<1.25,>=1.21.1 in /usr/local/lib/python3.6/dist-packages (from kaggle) (1.24.3)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.6/dist-packages (from kaggle) (1.12.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.6/dist-packages (from kaggle) (4.28.1)
Requirement already satisfied: requests in /usr/local/lib/python3.6/dist-packages (from kaggle) (2.21.0)
Requirement already satisfied: certifi in /usr/local/lib/python3.6/dist-packages (from kaggle) (2019.11.28)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.6/dist-packages (from kaggle) (2.6.1)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.6/dist-packages (from kaggle) (4.0.0)
Requirement already satisfied: idna<2.9,>=2.5 in /usr/local/lib/python3.6/dist-packages (from requests->kaggle) (2.8)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /usr/local/lib/python3.6/dist-packages (from requests->kaggle) (3.0.4)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.6/dist-packages (from python-slugify->kaggle) (1.3)
```

Kaggle API Token

https://www.kaggle.com



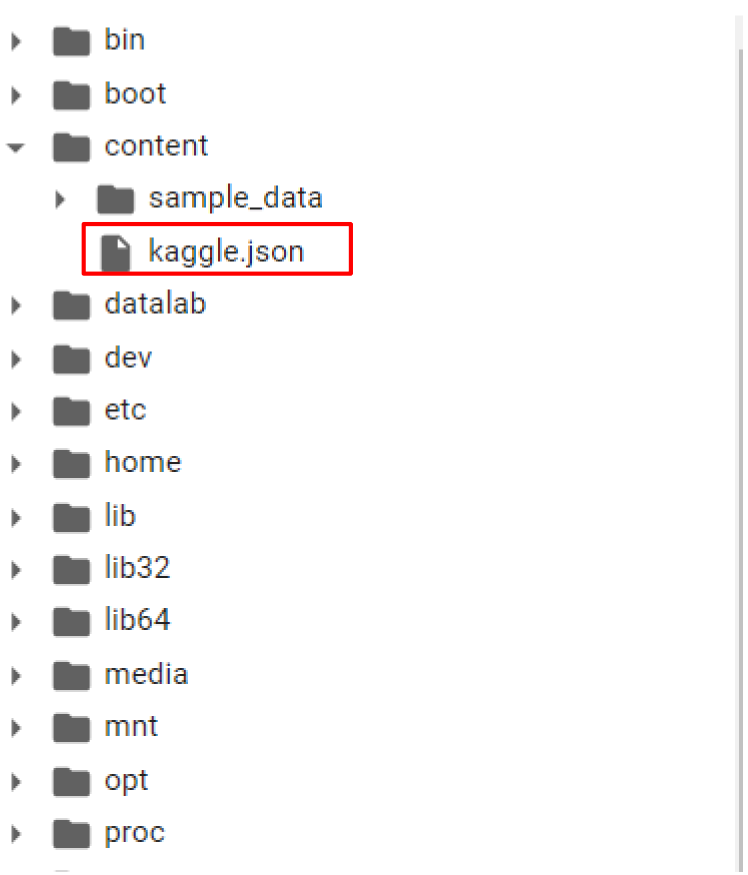


API

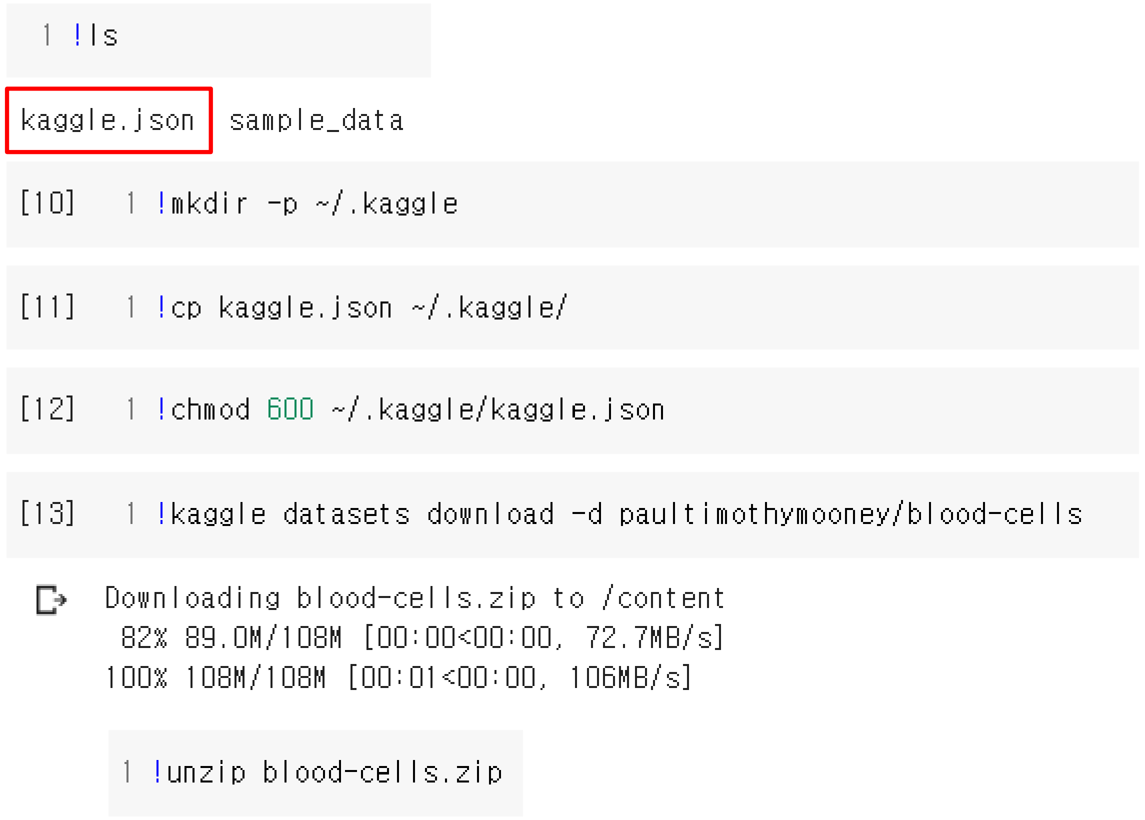
Using Kaggle's beta API, you can interact with Competitions and Datasets to download data, make submissions, and more via the command line. Read the docs

1. **kaggle에 로그인을 한다.**
2. **My Account에 접속한다.**
3. **API -> Create New API Token을 다운로드 받는다.**
4. **다운로드 받은 json 파일을 content 폴더에 Drag & Drop 한다.**

Dataset Download



```
bin
boot
content
    sample_data
    kaggle.json
datalab
dev
etc
home
lib
lib32
lib64
media
mnt
opt
proc
```



```
!ls
```

```
kaggle.json  sample_data
```

```
[10]  !mkdir -p ~/.kaggle
```

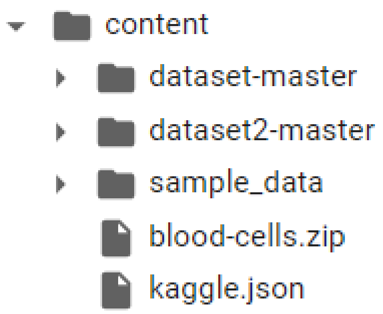
```
[11]  !cp kaggle.json ~/.kaggle/
```

```
[12]  !chmod 600 ~/.kaggle/kaggle.json
```

```
[13]  !kaggle datasets download -d paultimothymooney/blood-cells
```

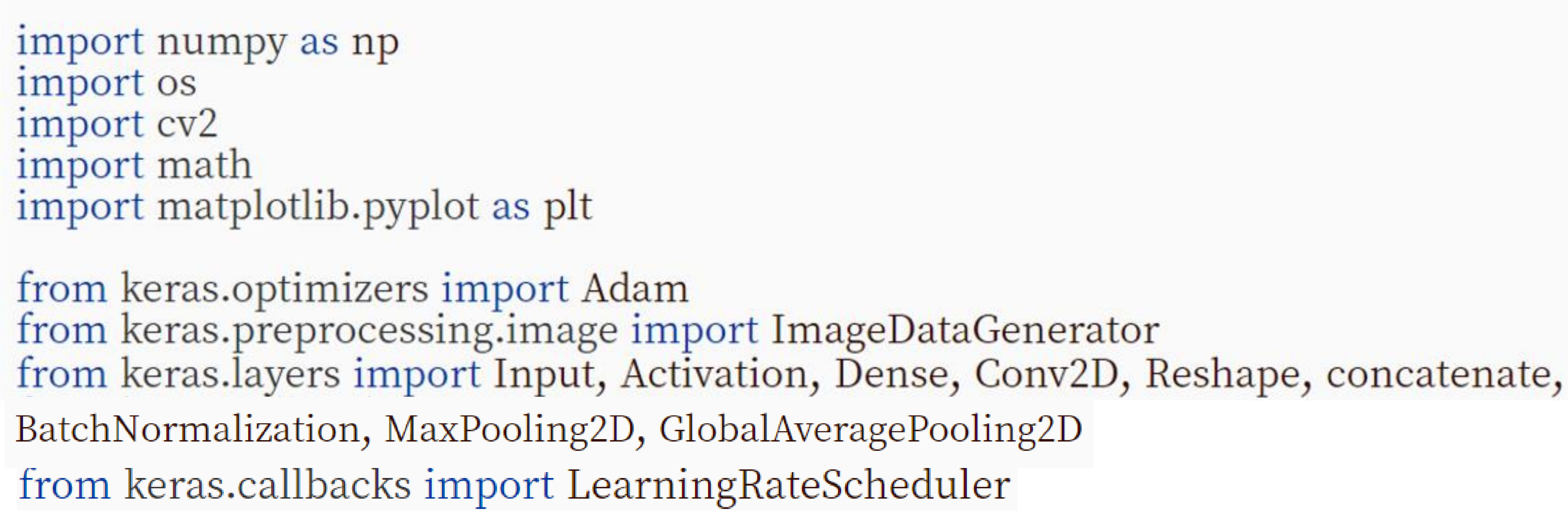
```
Downloading blood-cells.zip to /content
 82% 89.0M/108M [00:00<00:00, 72.7MB/s]
100% 108M/108M [00:01<00:00, 106MB/s]
```

```
!unzip blood-cells.zip
```



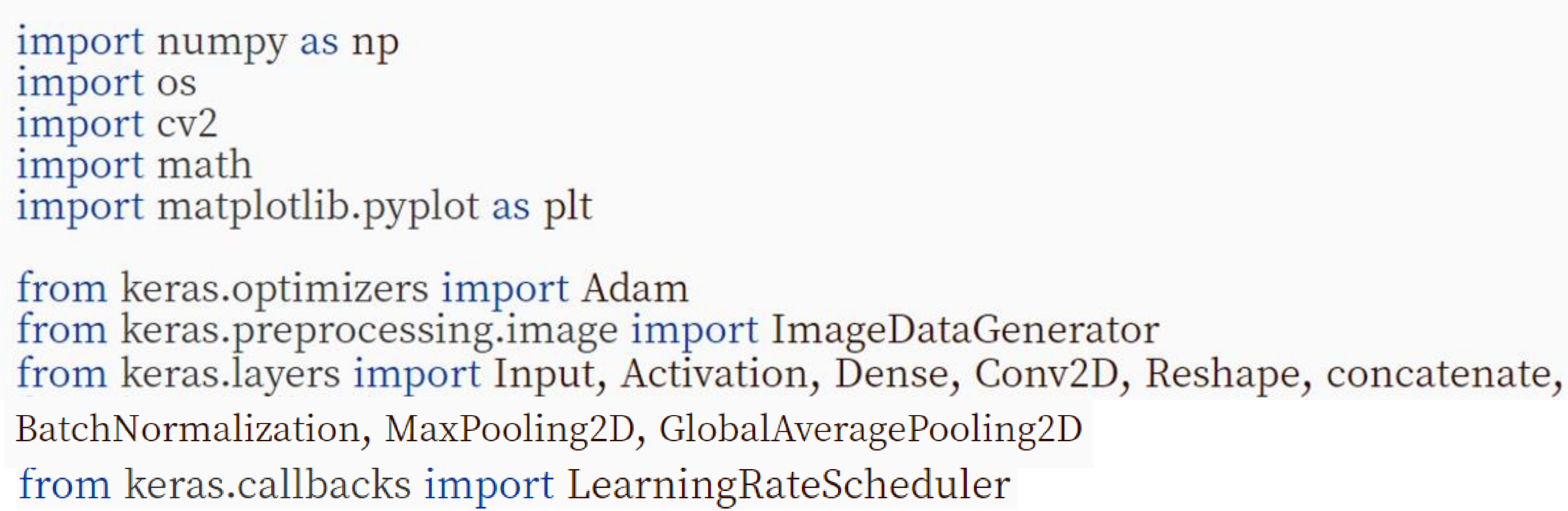
```
content
    dataset-master
    dataset2-master
    sample_data
    blood-cells.zip
    kaggle.json
```

Library Import



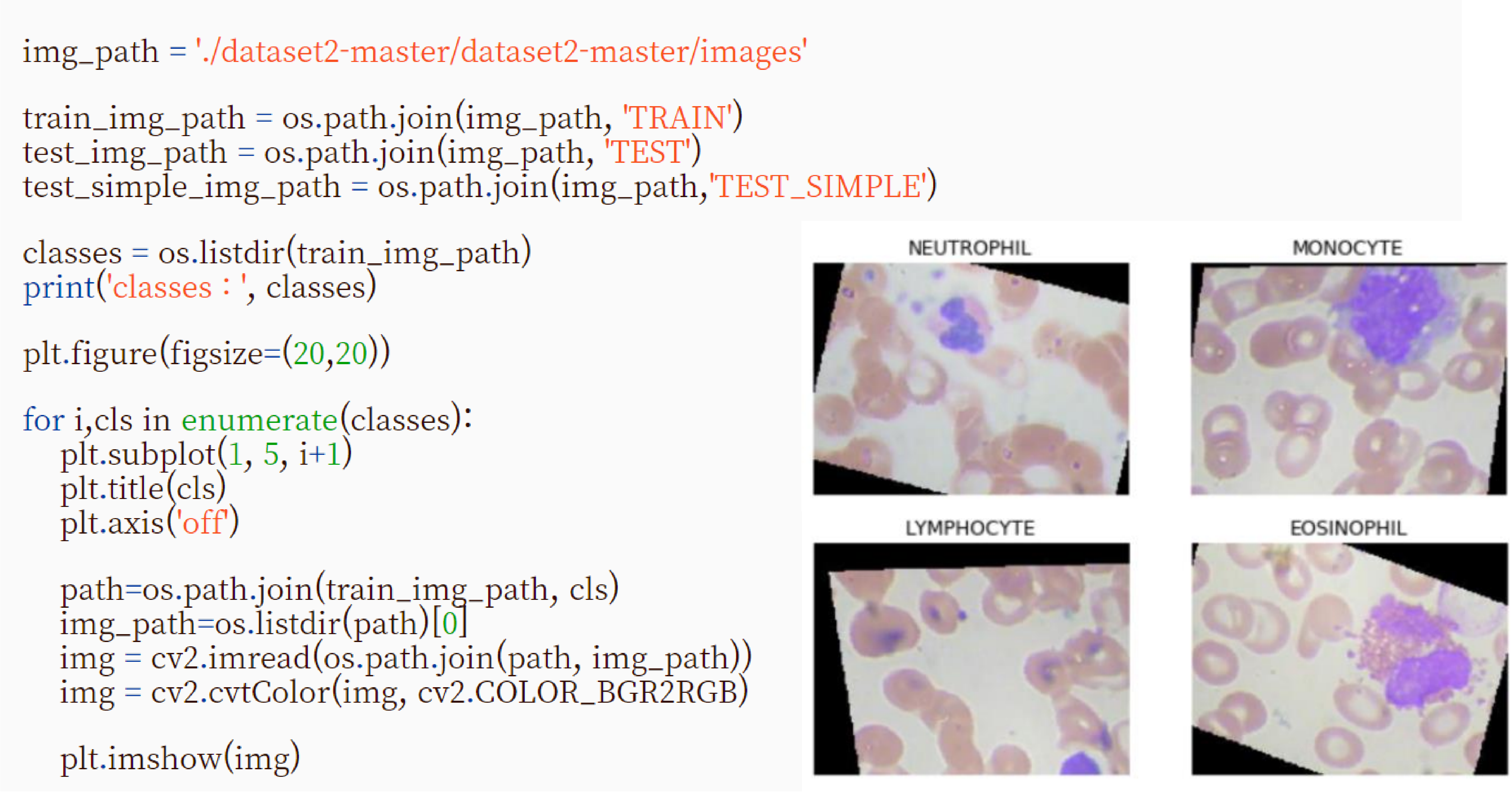
```
import numpy as np
import os
import cv2
import math
import matplotlib.pyplot as plt

from keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator
from keras.layers import Input, Activation, Dense, Conv2D, Reshape, concatenate,
BatchNormalization, MaxPooling2D, GlobalAveragePooling2D
from keras.callbacks import LearningRateScheduler
```

Library Import



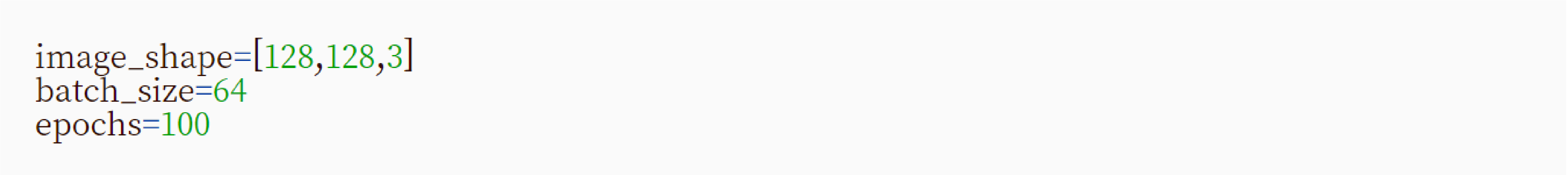
```
import numpy as np
import os
import cv2
import math
import matplotlib.pyplot as plt

from keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator
from keras.layers import Input, Activation, Dense, Conv2D, Reshape, concatenate,
BatchNormalization, MaxPooling2D, GlobalAveragePooling2D
from keras.callbacks import LearningRateScheduler
```

DataSet



```
img_path = './dataset2-master/dataset2-master/images'

train_img_path = os.path.join(img_path, 'TRAIN')
test_img_path = os.path.join(img_path, 'TEST')
test_simple_img_path = os.path.join(img_path,'TEST_SIMPLE')

classes = os.listdir(train_img_path)
print('classes : ', classes)

plt.figure(figsize=(20,20))

for i,cls in enumerate(classes):
    plt.subplot(1, 5, i+1)
    plt.title(cls)
    plt.axis('off')

    path=os.path.join(train_img_path, cls)
    img_path=os.listdir(path)[0]
    img = cv2.imread(os.path.join(path, img_path))
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    plt.imshow(img)
```

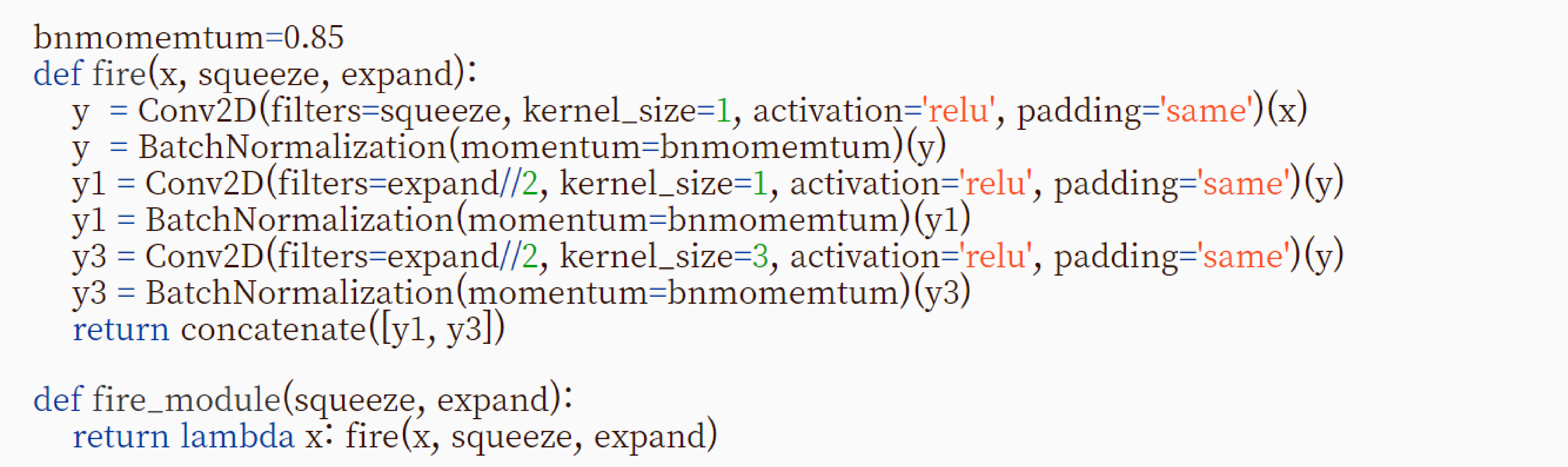
Hyper Parameters



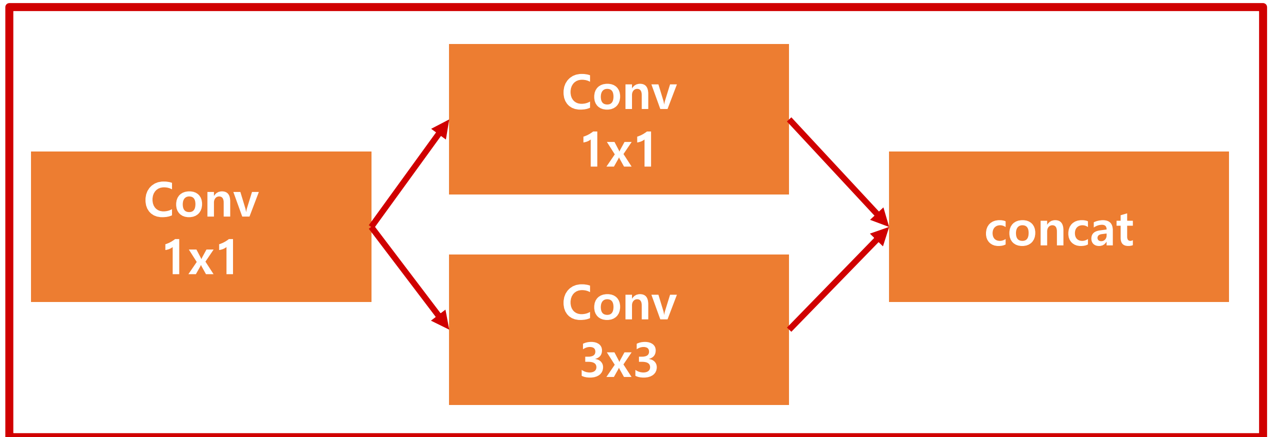
```
image_shape=[128,128,3]
batch_size=64
epochs=100
```

learning rate값을 정의해야하지만 뒤에 learning rate decay를 scheduler로 진행하기 위해서 뒤에 정의한다.

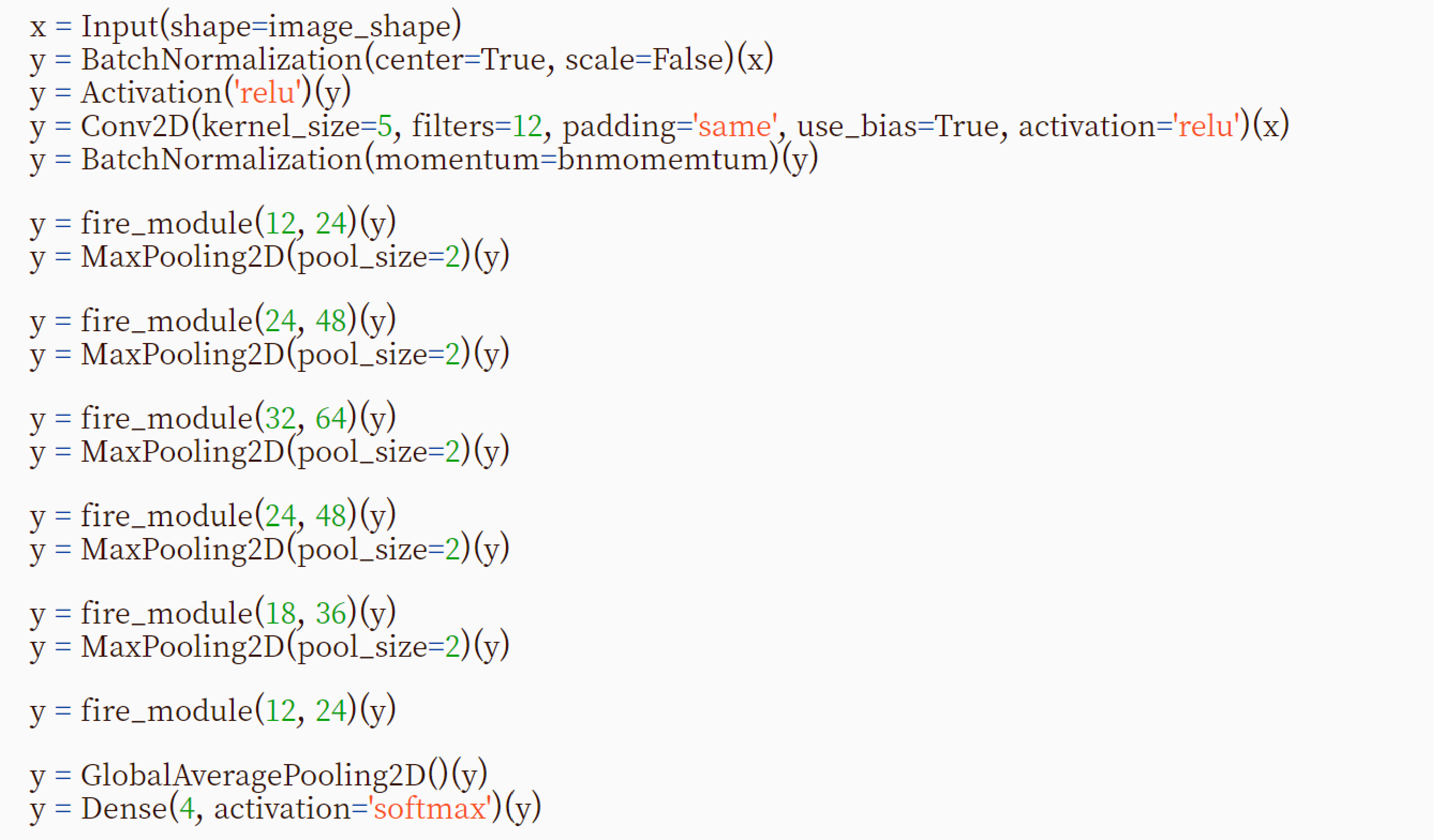
Module in DNN model



```
bnmomemtum=0.85
def fire(x, squeeze, expand):
    y  = Conv2D(filters=squeeze, kernel_size=1, activation='relu', padding='same')(x)
    y  = BatchNormalization(momentum=bnmomemtum)(y)
    y1 = Conv2D(filters=expand//2, kernel_size=1, activation='relu', padding='same')(y)
    y1 = BatchNormalization(momentum=bnmomemtum)(y1)
    y3 = Conv2D(filters=expand//2, kernel_size=3, activation='relu', padding='same')(y)
    y3 = BatchNormalization(momentum=bnmomemtum)(y3)
    return concatenate([y1, y3])

def fire_module(squeeze, expand):
    return lambda x: fire(x, squeeze, expand)
```

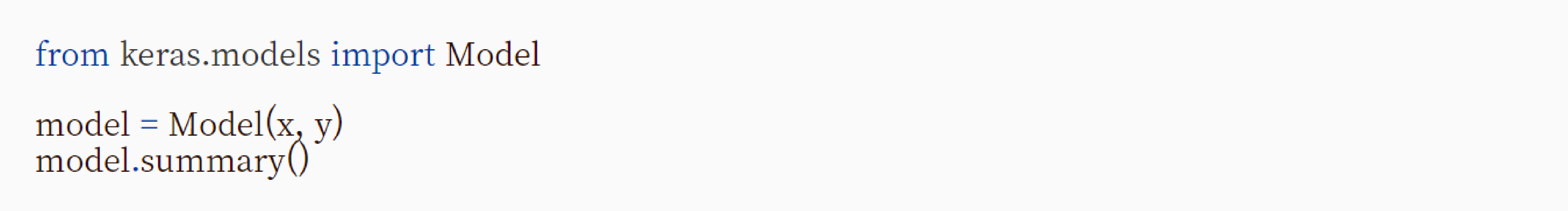


Model



```
x = Input(shape=image_shape)
y = BatchNormalization(center=True, scale=False)(x)
y = Activation('relu')(y)
y = Conv2D(kernel_size=5, filters=12, padding='same', use_bias=True, activation='relu')(x)
y = BatchNormalization(momentum=bnmomemtum)(y)

y = fire_module(12, 24)(y)
y = MaxPooling2D(pool_size=2)(y)

y = fire_module(24, 48)(y)
y = MaxPooling2D(pool_size=2)(y)

y = fire_module(32, 64)(y)
y = MaxPooling2D(pool_size=2)(y)

y = fire_module(24, 48)(y)
y = MaxPooling2D(pool_size=2)(y)

y = fire_module(18, 36)(y)
y = MaxPooling2D(pool_size=2)(y)

y = fire_module(12, 24)(y)

y = GlobalAveragePooling2D()(y)
y = Dense(4, activation='softmax')(y)
```

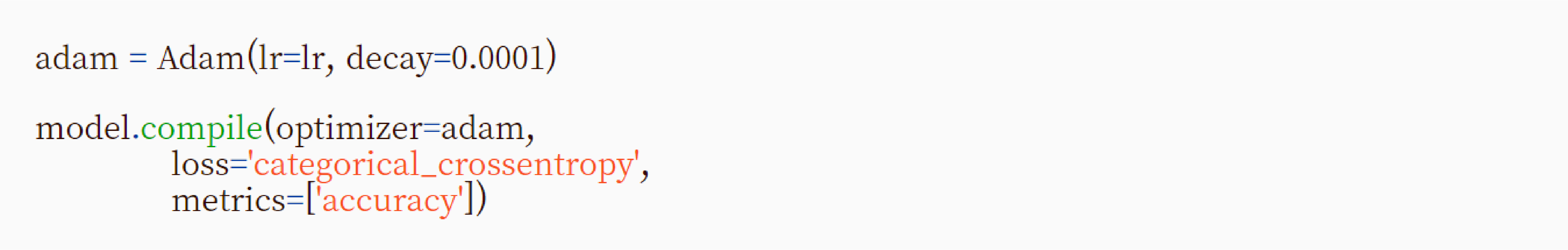
Model Summary



```
from keras.models import Model

model = Model(x, y)
model.summary()
```

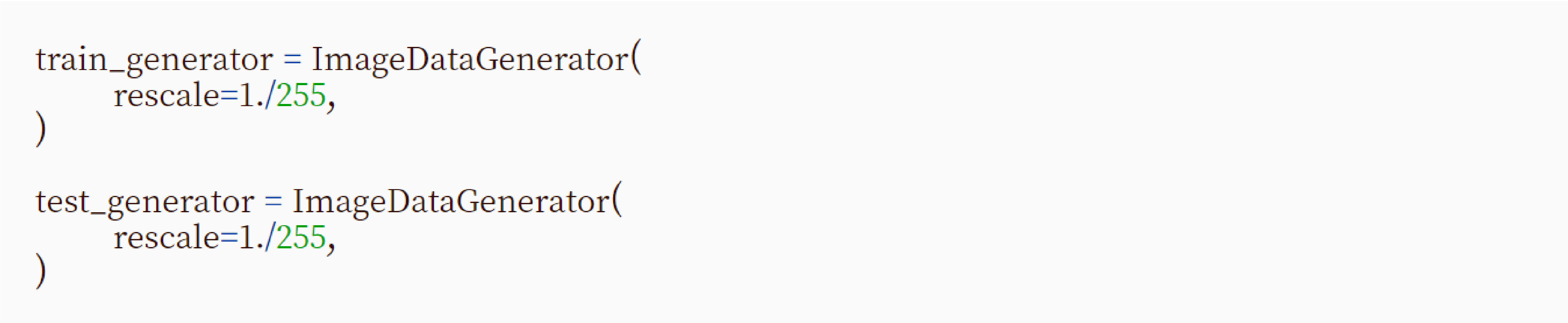
summary함수를 호출하면 model의 구조를 한눈에 볼수 있다.

Model Compile



```
adam = Adam(lr=lr, decay=0.0001)

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

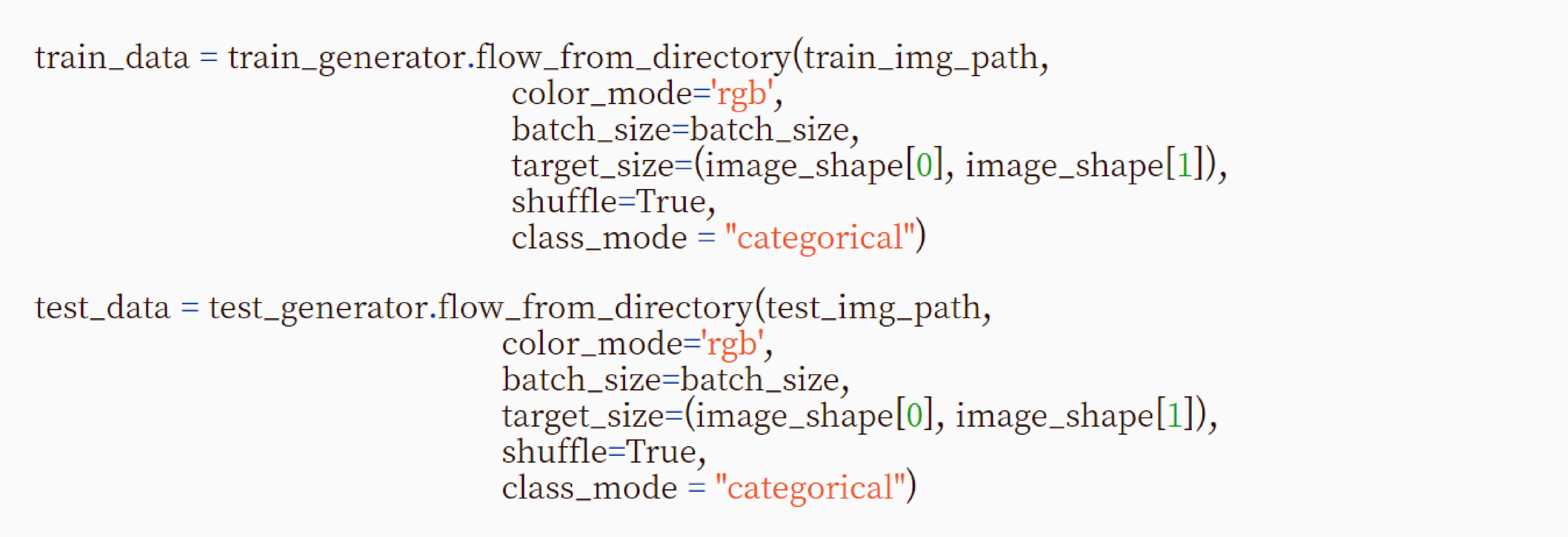
Preprocessing input images



```
train_generator = ImageDataGenerator(
        rescale=1./255,
)

test_generator = ImageDataGenerator(
        rescale=1./255,
)
```

여기서 image augmentation도 함께 진행할 수 있지만 여기서는 사용하지 않고 scale만 조절한다.

Data Loader

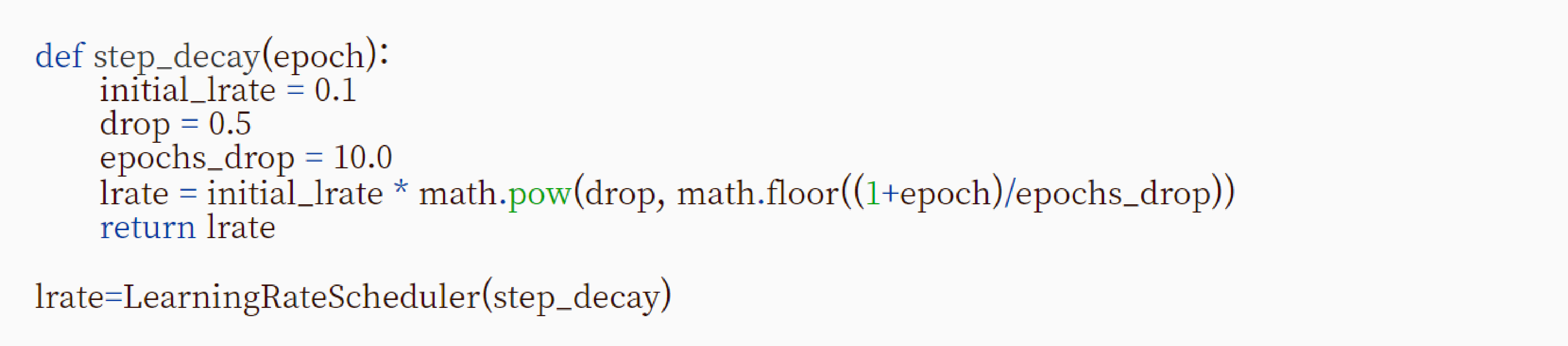


```
train_data = train_generator.flow_from_directory(train_img_path,
                                                 color_mode='rgb',
                                                 batch_size=batch_size,
                                                 target_size=(image_shape[0], image_shape[1]),
                                                 shuffle=True,
                                                 class_mode = "categorical")

test_data = test_generator.flow_from_directory(test_img_path,
                                               color_mode='rgb',
                                               batch_size=batch_size,
                                               target_size=(image_shape[0], image_shape[1]),
                                               shuffle=True,
                                               class_mode = "categorical")
```

매번 데이터를 ImageDataGenerator를 이용해서 호출하기 위해서 Loader를 만든다.(Train / Valid)

- **color_mode** : rgb color
- **target_size** : input image size를 조절한다.
- **shuffle** : image random shuffle

Learning rate scheduler



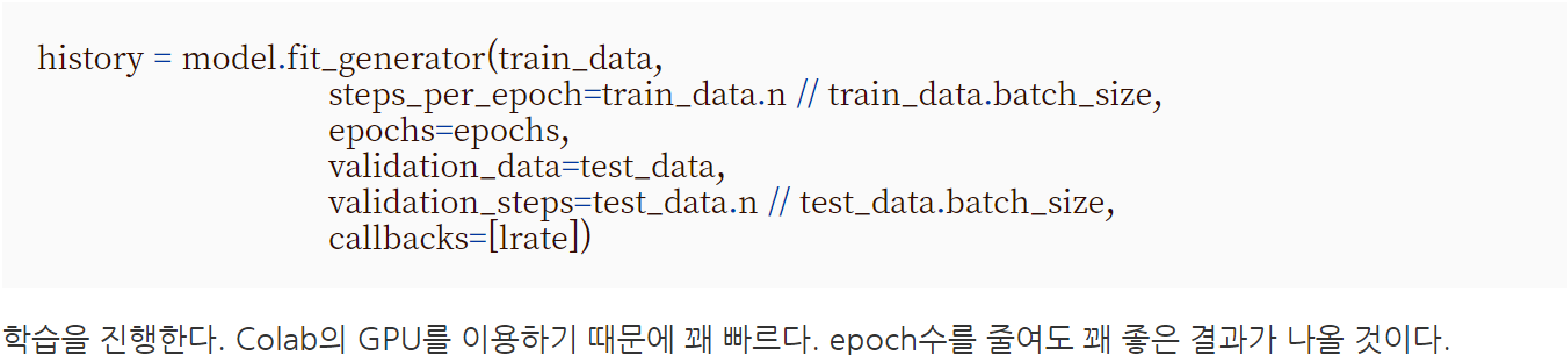
```
def step_decay(epoch):
    initial_lrate = 0.1
    drop = 0.5
    epochs_drop = 10.0
    lrate = initial_lrate * math.pow(drop, math.floor((1+epoch)/epochs_drop))
    return lrate

lrate=LearningRateScheduler(step_decay)
```

매 epoch마다 learning rate를 조절하기 위한 callback함수를 만든다.

Learning rate : 0.1

10 epochs 마다 0.5씩 곱해서 학습률을 감소시킨다.

Training



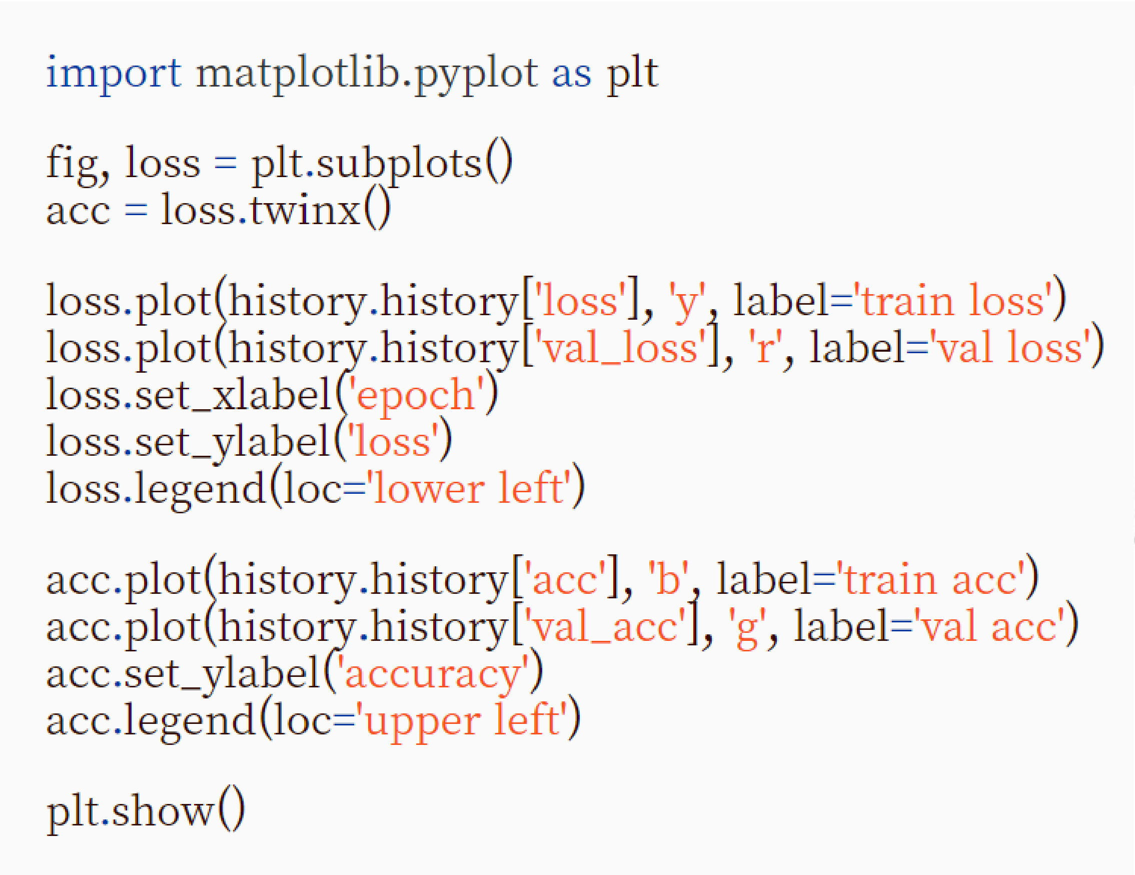
```
history = model.fit_generator(train_data,
                              steps_per_epoch=train_data.n // train_data.batch_size,
                              epochs=epochs,
                              validation_data=test_data,
                              validation_steps=test_data.n // test_data.batch_size,
                              callbacks=[lrate])
```

학습을 진행한다. Colab의 GPU를 이용하기 때문에 꽤 빠르다. epoch수를 줄여도 꽤 좋은 결과가 나올 것이다.

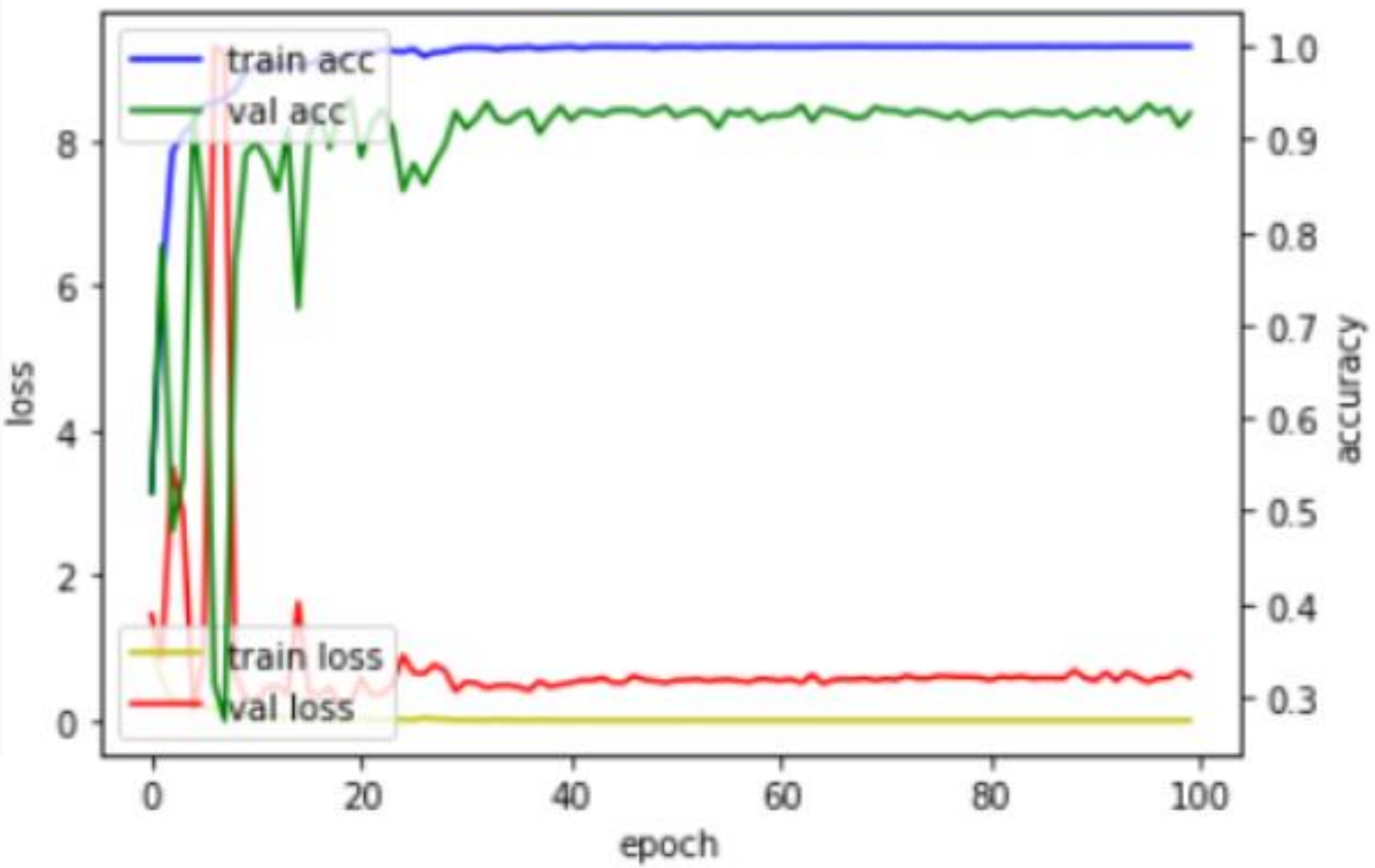
학습시간이 오래걸리기 때문에 미리 학습을 시켜놓은 Pretrained model을 사용한다.

이 부분은 넘어가고 Save/Load로 간다.

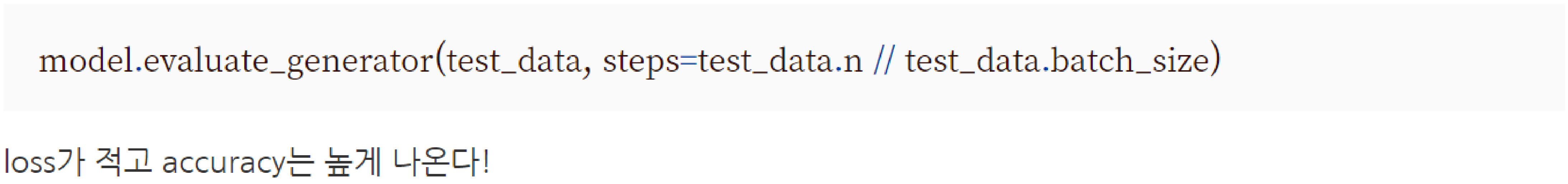
Graph



```
import matplotlib.pyplot as plt

fig, loss = plt.subplots()
acc = loss.twinx()

loss.plot(history.history['loss'], 'y', label='train loss')
loss.plot(history.history['val_loss'], 'r', label='val loss')
loss.set_xlabel('epoch')
loss.set_ylabel('loss')
loss.legend(loc='lower left')

acc.plot(history.history['acc'], 'b', label='train acc')
acc.plot(history.history['val_acc'], 'g', label='val acc')
acc.set_ylabel('accuracy')
acc.legend(loc='upper left')

plt.show()
```



Testing



```
model.evaluate_generator(test_data, steps=test_data.n // test_data.batch_size)
```

loss가 적고 accuracy는 높게 나온다!

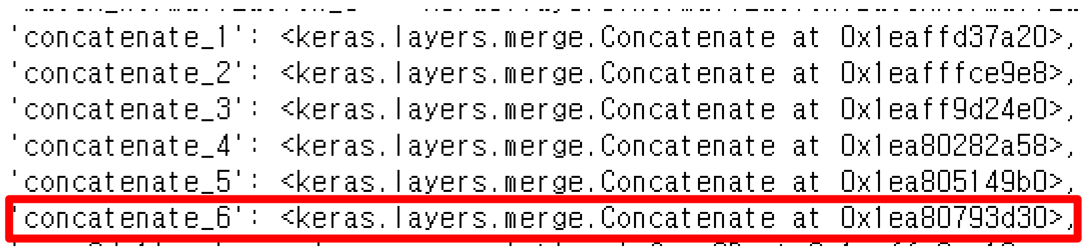
CAM – Visualizing Classification

```
class_weights = model.layers[-1].get_weights()[0]
```

softmax로 들어오는 weights를 가져온다.

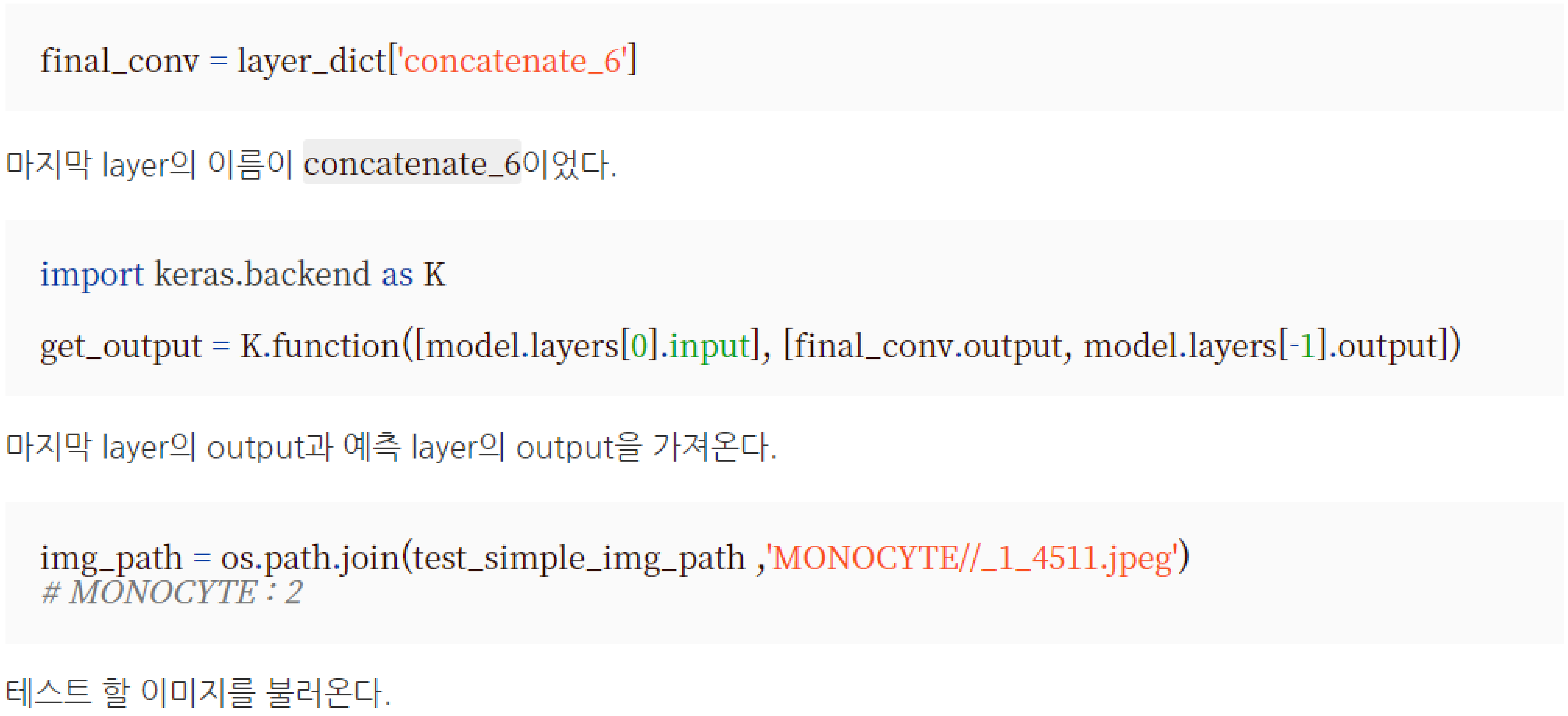
```
layer_dict = dict([(layer.name, layer) for layer in model.layers])
print(layer_dict)
```

layer를 확인한다. GlobalAveragePooling을 하기 전에 output을 가져와야하기 때문에 layer_dict로 이름을 확인한뒤 가져와야한다.



```
'concatenate_1': <keras.layers.merge.Concatenate at 0x1eaffd37a20>,
'concatenate_2': <keras.layers.merge.Concatenate at 0x1eafffce9e8>,
'concatenate_3': <keras.layers.merge.Concatenate at 0x1eaff9d24e0>,
'concatenate_4': <keras.layers.merge.Concatenate at 0x1ea80282a58>,
'concatenate_5': <keras.layers.merge.Concatenate at 0x1ea805149b0>,
'concatenate_6': <keras.layers.merge.Concatenate at 0x1ea80793d30>,
```

CAM – Visualizing Classification



```
final_conv = layer_dict['concatenate_6']
```

마지막 layer의 이름이 concatenate_6이었다.

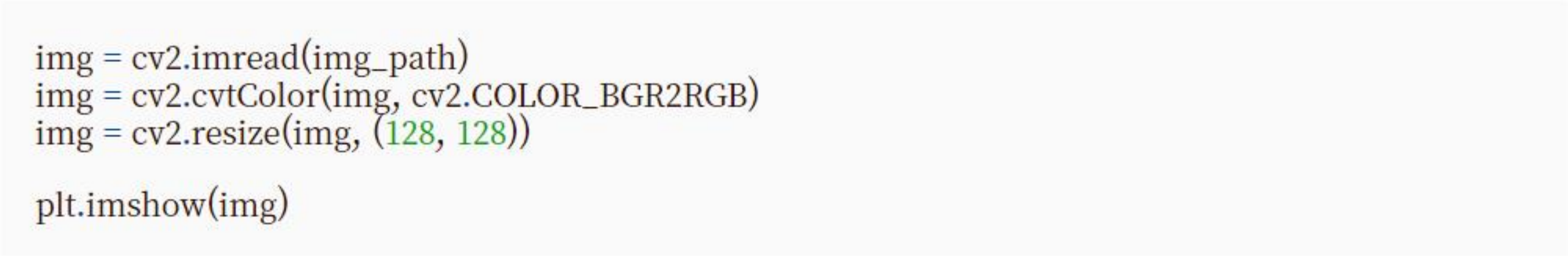
```
import keras.backend as K

get_output = K.function([model.layers[0].input], [final_conv.output, model.layers[-1].output])
```

마지막 layer의 output과 예측 layer의 output을 가져온다.

```
img_path = os.path.join(test_simple_img_path ,'MONOCYTE//_1_4511.jpeg')
# MONOCYTE : 2
```

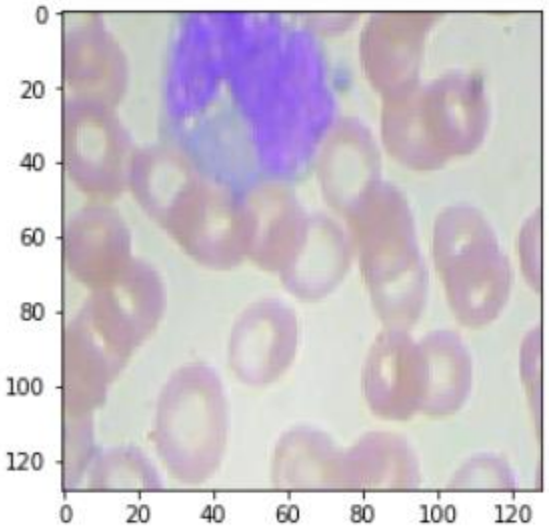
테스트 할 이미지를 불러온다.

CAM – Visualizing Classification

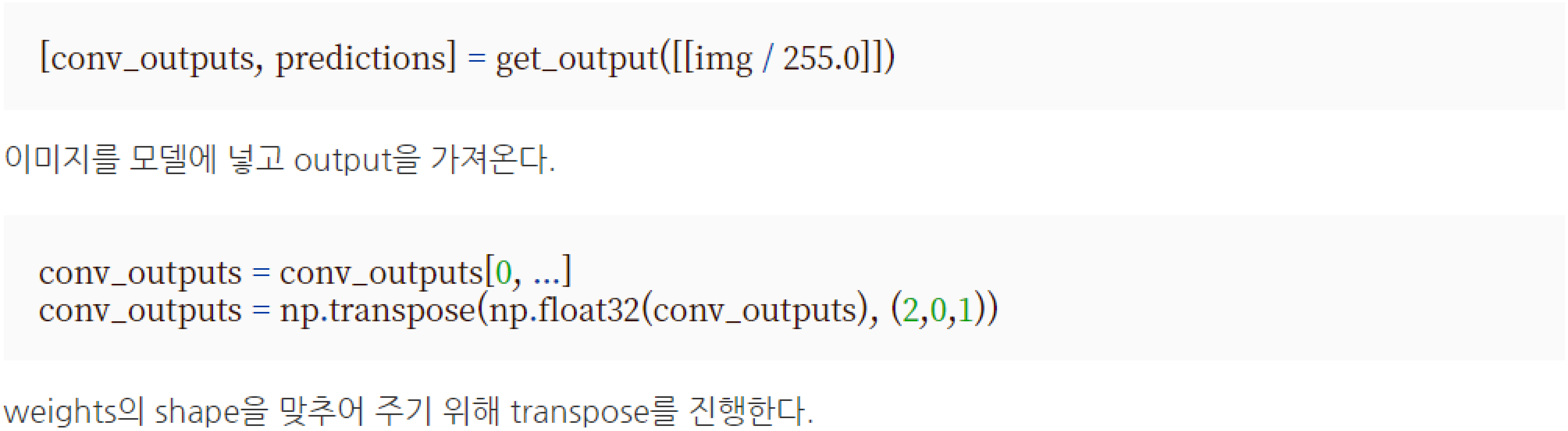


```
img = cv2.imread(img_path)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
img = cv2.resize(img, (128, 128))

plt.imshow(img)
```

이미지를 확인한다.



CAM – Visualizing Classification



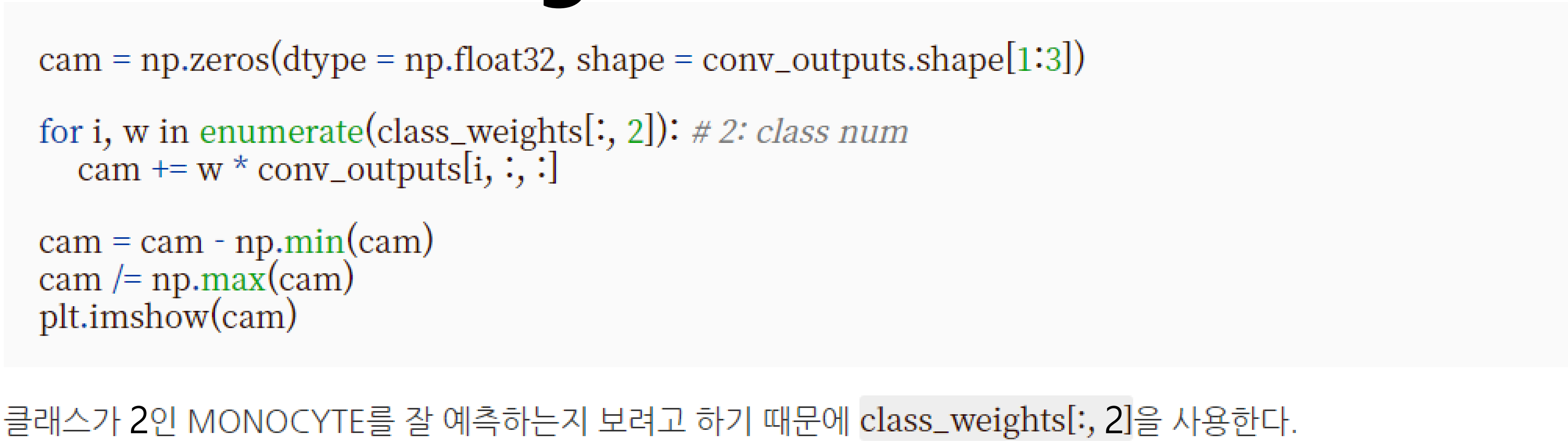
```
[conv_outputs, predictions] = get_output([[img / 255.0]])
```

이미지를 모델에 넣고 output을 가져온다.

```
conv_outputs = conv_outputs[0, ...]
conv_outputs = np.transpose(np.float32(conv_outputs), (2,0,1))
```

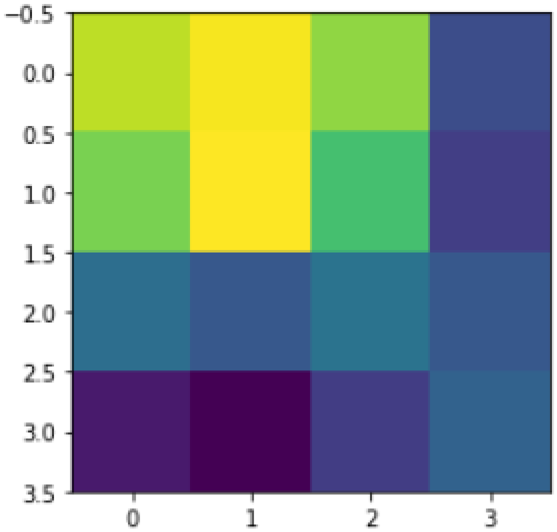
weights의 shape을 맞추어 주기 위해 transpose를 진행한다.

CAM – Visualizing Classification

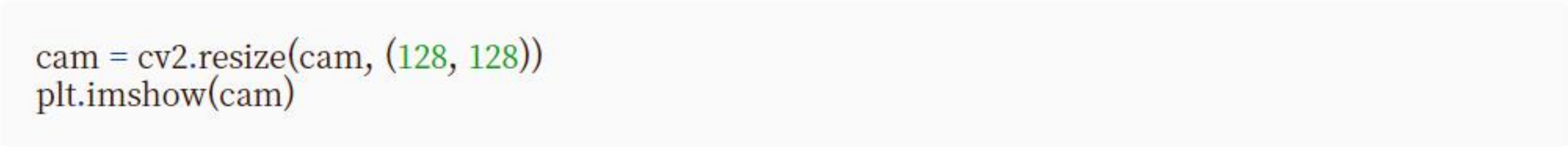


```
cam = np.zeros(dtype = np.float32, shape = conv_outputs.shape[1:3])

for i, w in enumerate(class_weights[:, 2]): # 2: class num
    cam += w * conv_outputs[i, :, :]

cam = cam - np.min(cam)
cam /= np.max(cam)
plt.imshow(cam)
```

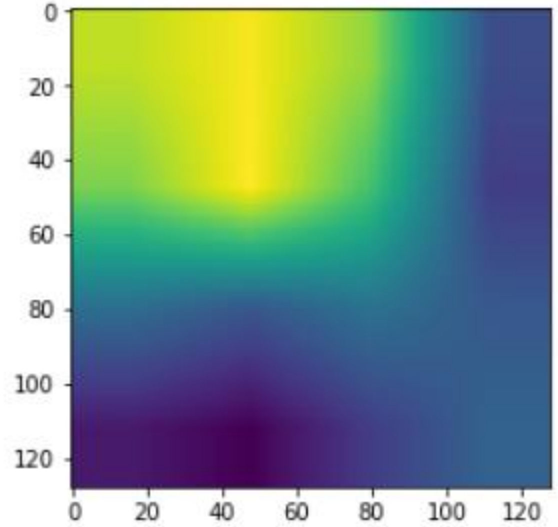
클래스가 2인 MONOCYTE를 잘 예측하는지 보려고 하기 때문에 class_weights[:, 2]을 사용한다.



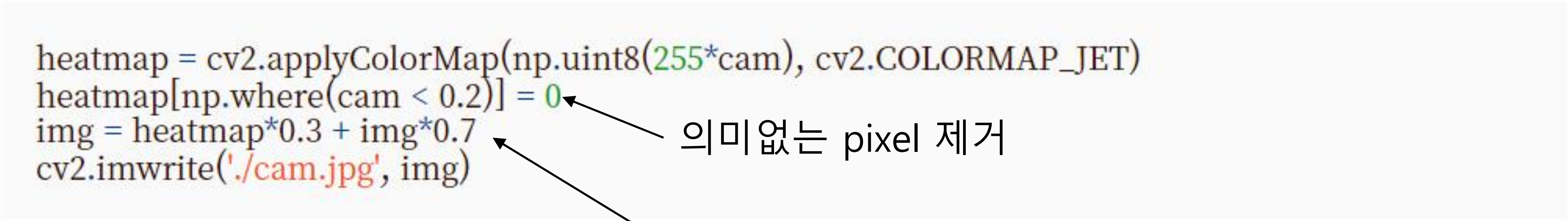
CAM – Visualizing Classification



```
cam = cv2.resize(cam, (128, 128))
plt.imshow(cam)
```



CAM – Visualizing Classification

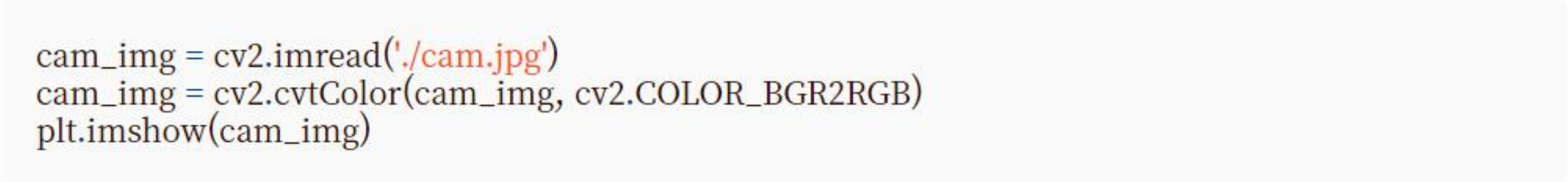


```
heatmap = cv2.applyColorMap(np.uint8(255*cam), cv2.COLORMAP_JET)
heatmap[np.where(cam < 0.2)] = 0
img = heatmap*0.3 + img*0.7
cv2.imwrite('./cam.jpg', img)
```

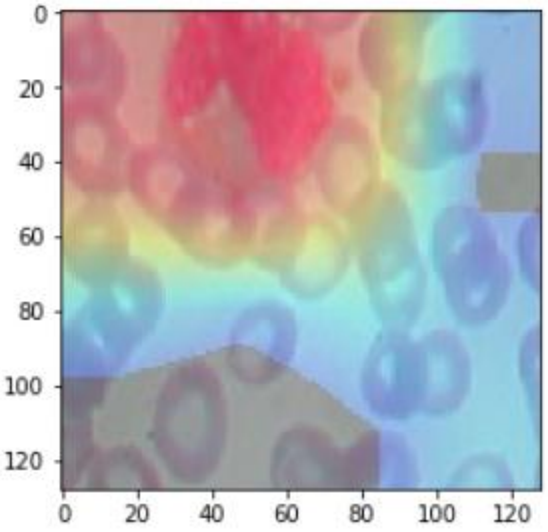
의미없는 pixel 제거

overlap 비율

이미지 저장하기



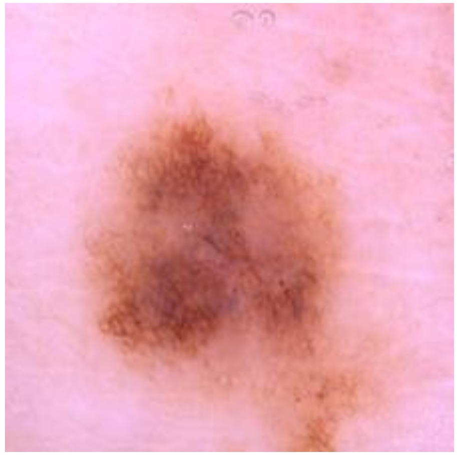
```
cam_img = cv2.imread('./cam.jpg')
cam_img = cv2.cvtColor(cam_img, cv2.COLOR_BGR2RGB)
plt.imshow(cam_img)
```



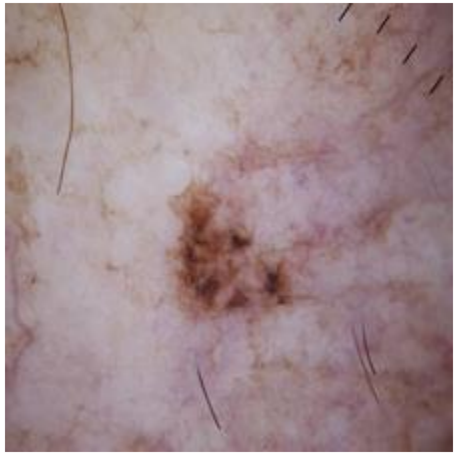
Skin Cancer Detection Model

DataSet

- Total : 3297 images
  - Test
    - Benign : 360 images
    - Malignant : 300 images
  - Train
    - Benign : 1440 images
    - Malignant : 1197 images
- Size : 224 X 224

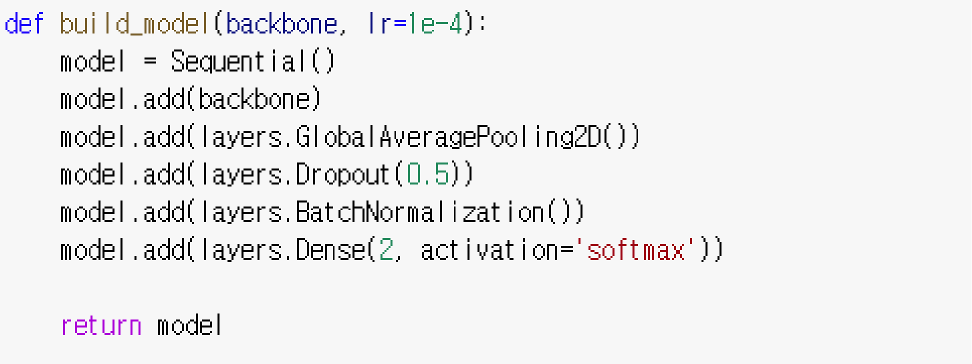


Benign

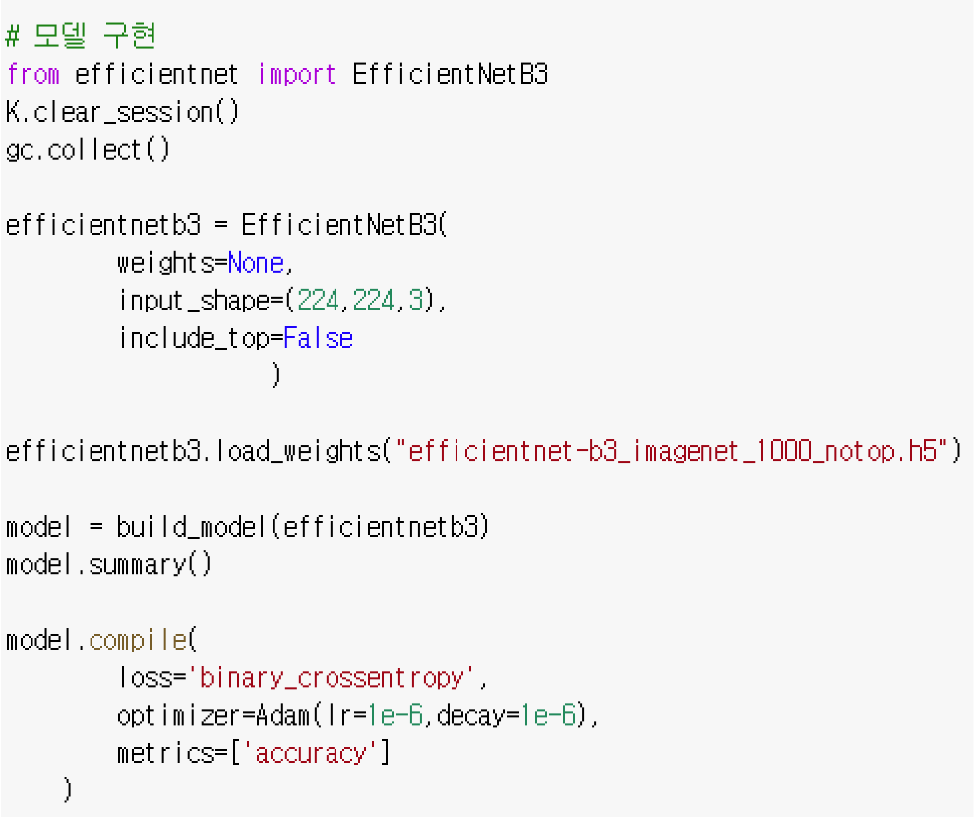


Malignant

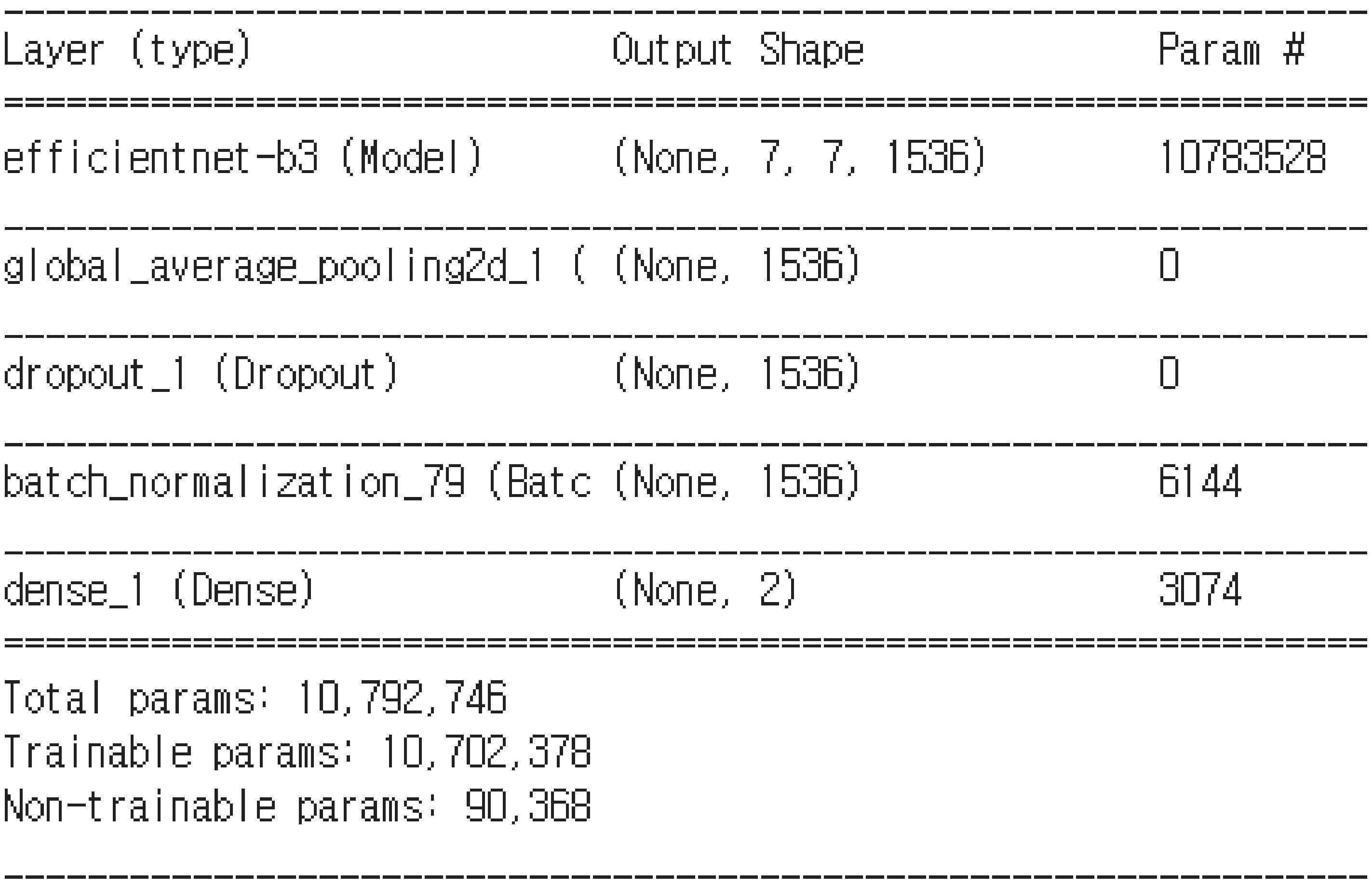
Model



```
def build_model(backbone, lr=1e-4):
    model = Sequential()
    model.add(backbone)
    model.add(layers.GlobalAveragePooling2D())
    model.add(layers.Dropout(0.5))
    model.add(layers.BatchNormalization())
    model.add(layers.Dense(2, activation='softmax'))

    return model
```

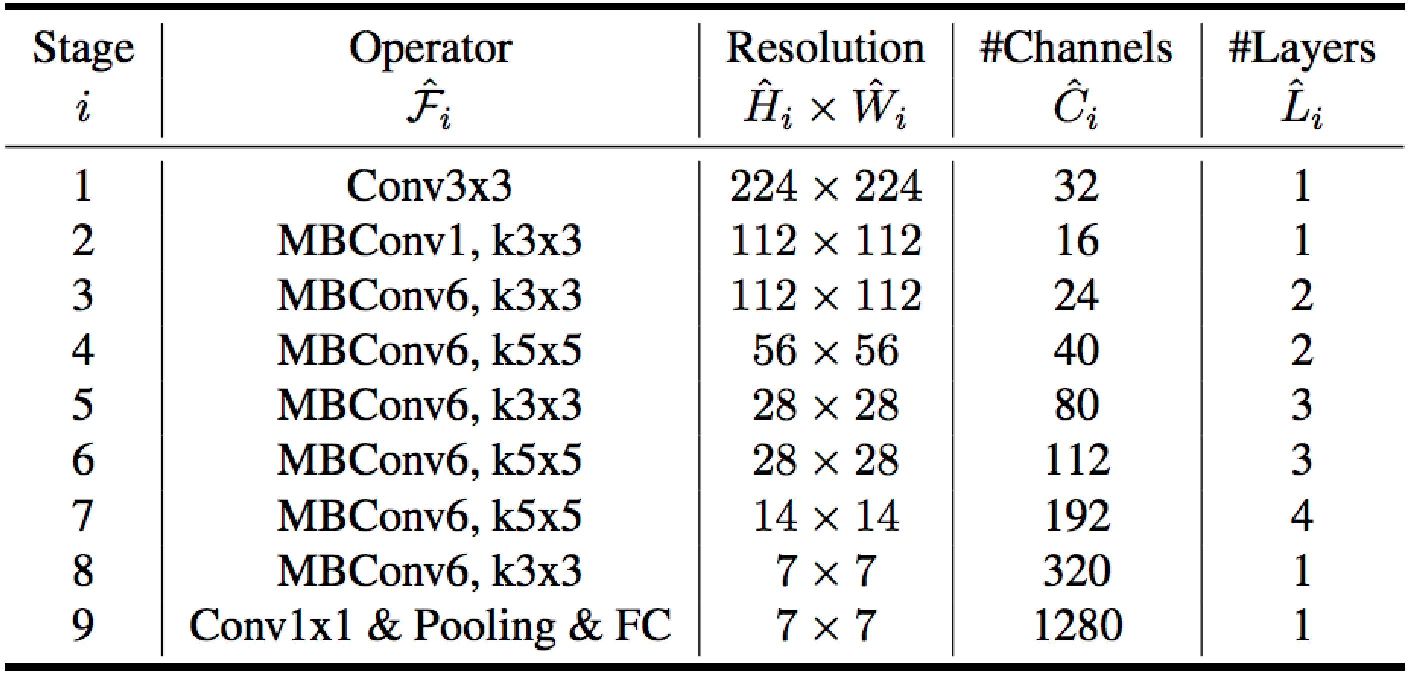


```
# 모델 구현
from efficientnet import EfficientNetB3
K.clear_session()
gc.collect()

efficientnetb3 = EfficientNetB3(
        weights=None,
        input_shape=(224,224,3),
        include_top=False
                   )

efficientnetb3.load_weights("efficientnet-b3_imagenet_1000_notop.h5")

model = build_model(efficientnetb3)
model.summary()

model.compile(
        loss='binary_crossentropy',
        optimizer=Adam(lr=1e-6,decay=1e-6),
        metrics=['accuracy']
    )
```



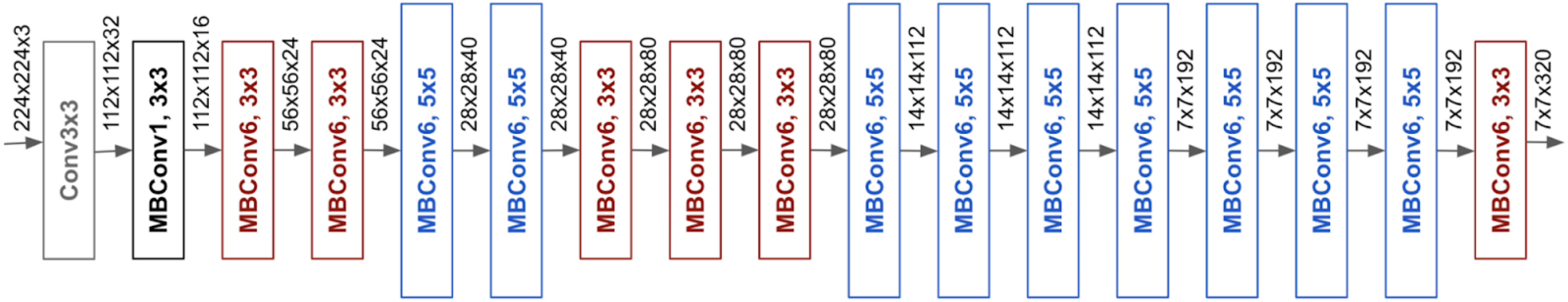
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
efficientnet-b3 (Model)      (None, 7, 7, 1536)        10783528
_________________________________________________________________
global_average_pooling2d_1 ( (None, 1536)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 1536)              0
_________________________________________________________________
batch_normalization_79 (Batc (None, 1536)              6144
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 3074
=================================================================
Total params: 10,792,746
Trainable params: 10,702,378
Non-trainable params: 90,368
_________________________________________________________________
```

Model



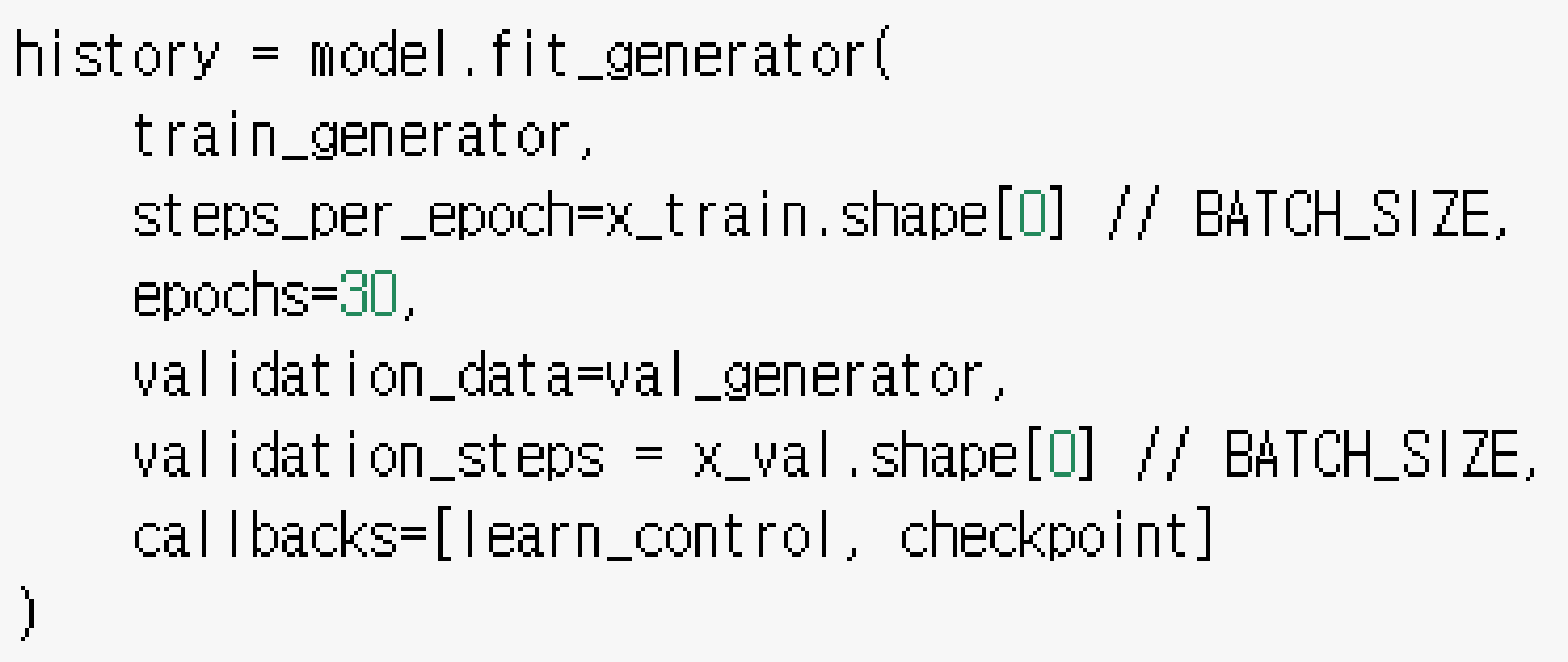
Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224 × 224	32	1
2	MBConv1, k3x3	112 × 112	16	1
3	MBConv6, k3x3	112 × 112	24	2
4	MBConv6, k5x5	56 × 56	40	2
5	MBConv6, k3x3	28 × 28	80	3
6	MBConv6, k5x5	28 × 28	112	3
7	MBConv6, k5x5	14 × 14	192	4
8	MBConv6, k3x3	7 × 7	320	1
9	Conv1x1 & Pooling & FC	7 × 7	1280	1

*MBConv = Inverted Residual Block



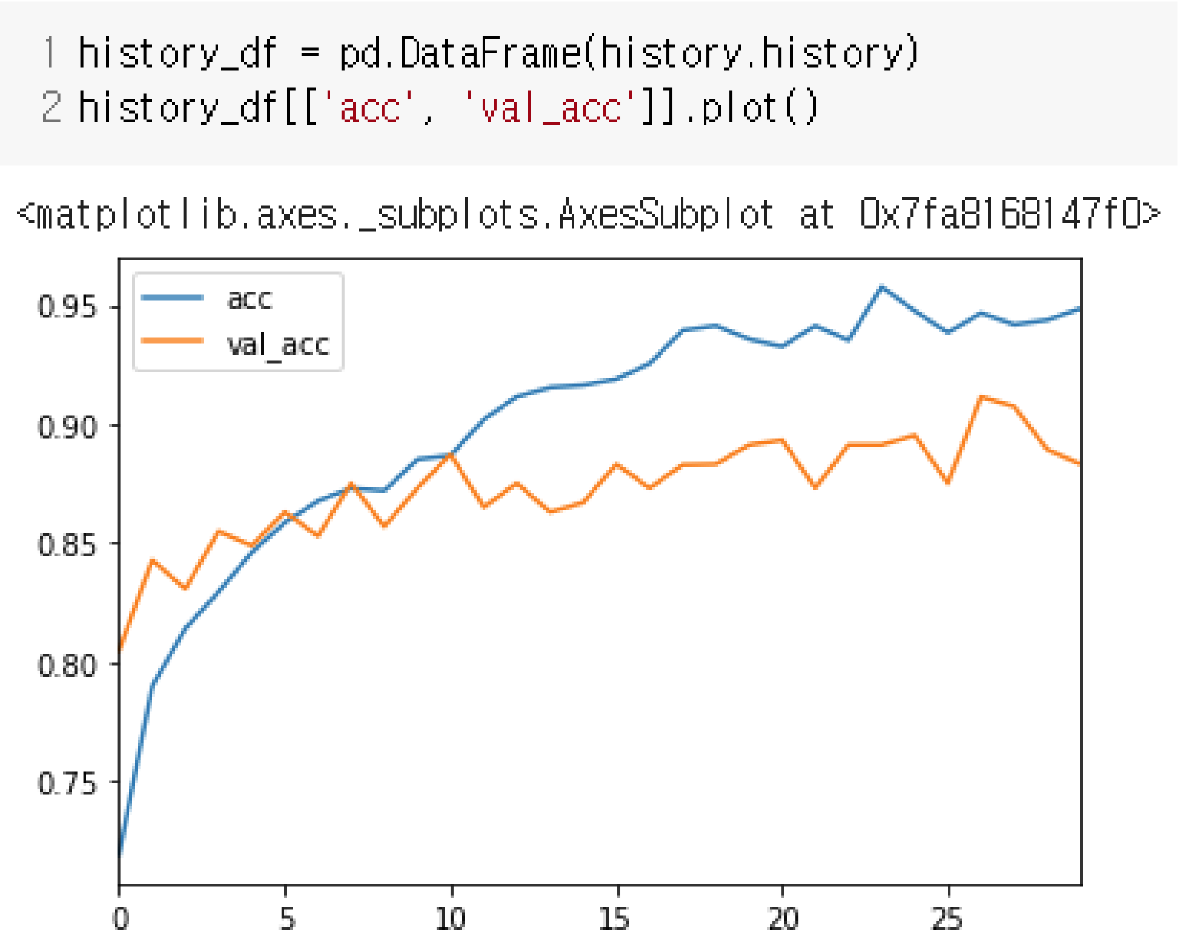
Training

- Batch size = 32
- epoch = 30



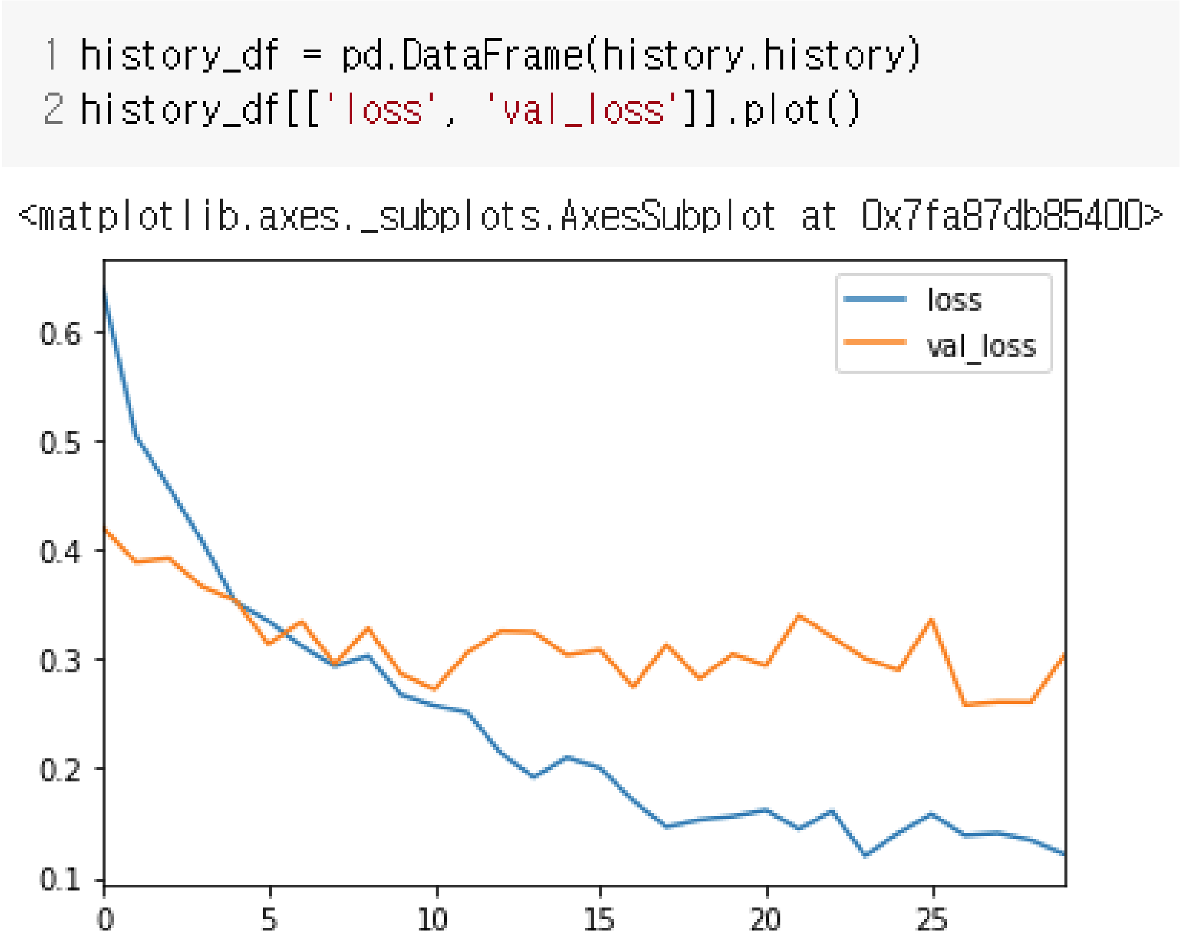
```
history = model.fit_generator(
    train_generator,
    steps_per_epoch=x_train.shape[0] // BATCH_SIZE,
    epochs=30,
    validation_data=val_generator,
    validation_steps = x_val.shape[0] // BATCH_SIZE,
    callbacks=[learn_control, checkpoint]
)
```

Result



```
history_df = pd.DataFrame(history.history)
history_df[['acc', 'val_acc']].plot()
```

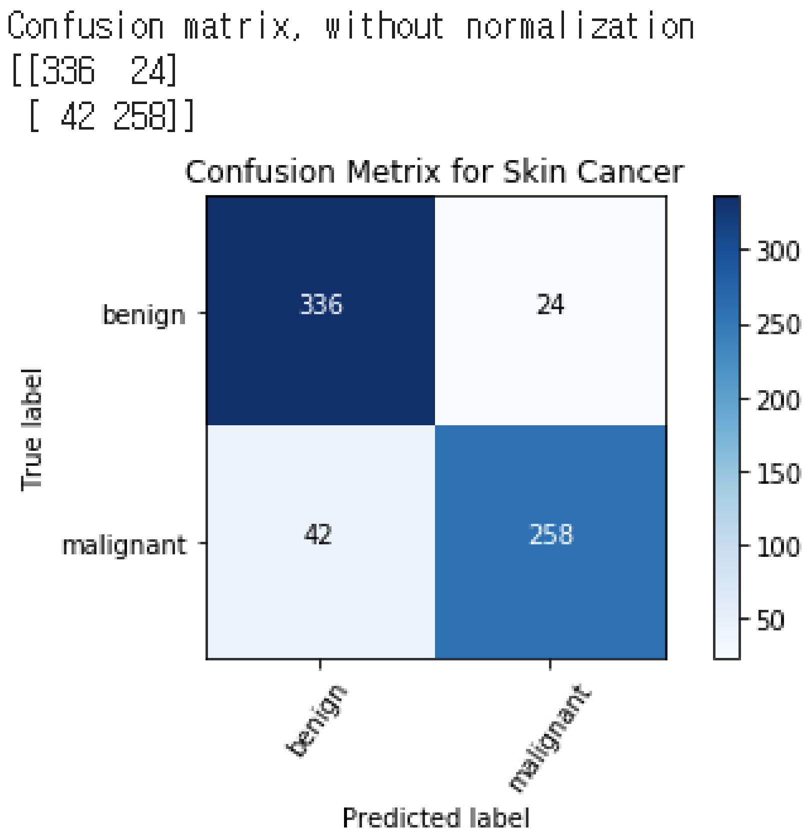
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa8168147f0>
```



```
history_df = pd.DataFrame(history.history)
history_df[['loss', 'val_loss']].plot()
```

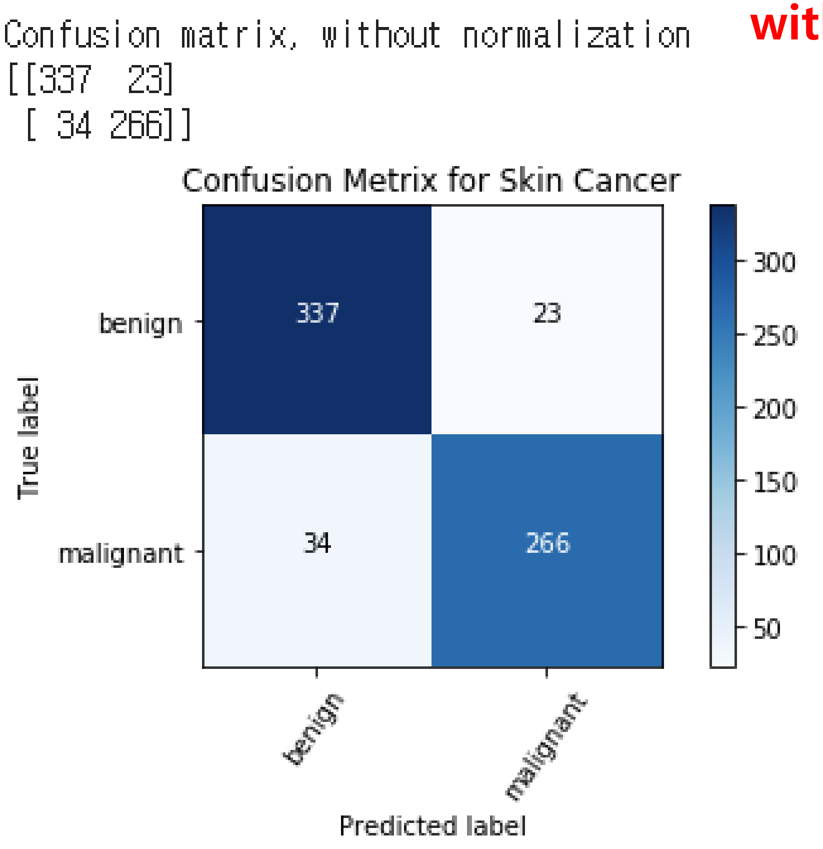
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa87db85400>
```

Result



```
Confusion matrix, without normalization
[[336  24]
 [ 42 258]]
```

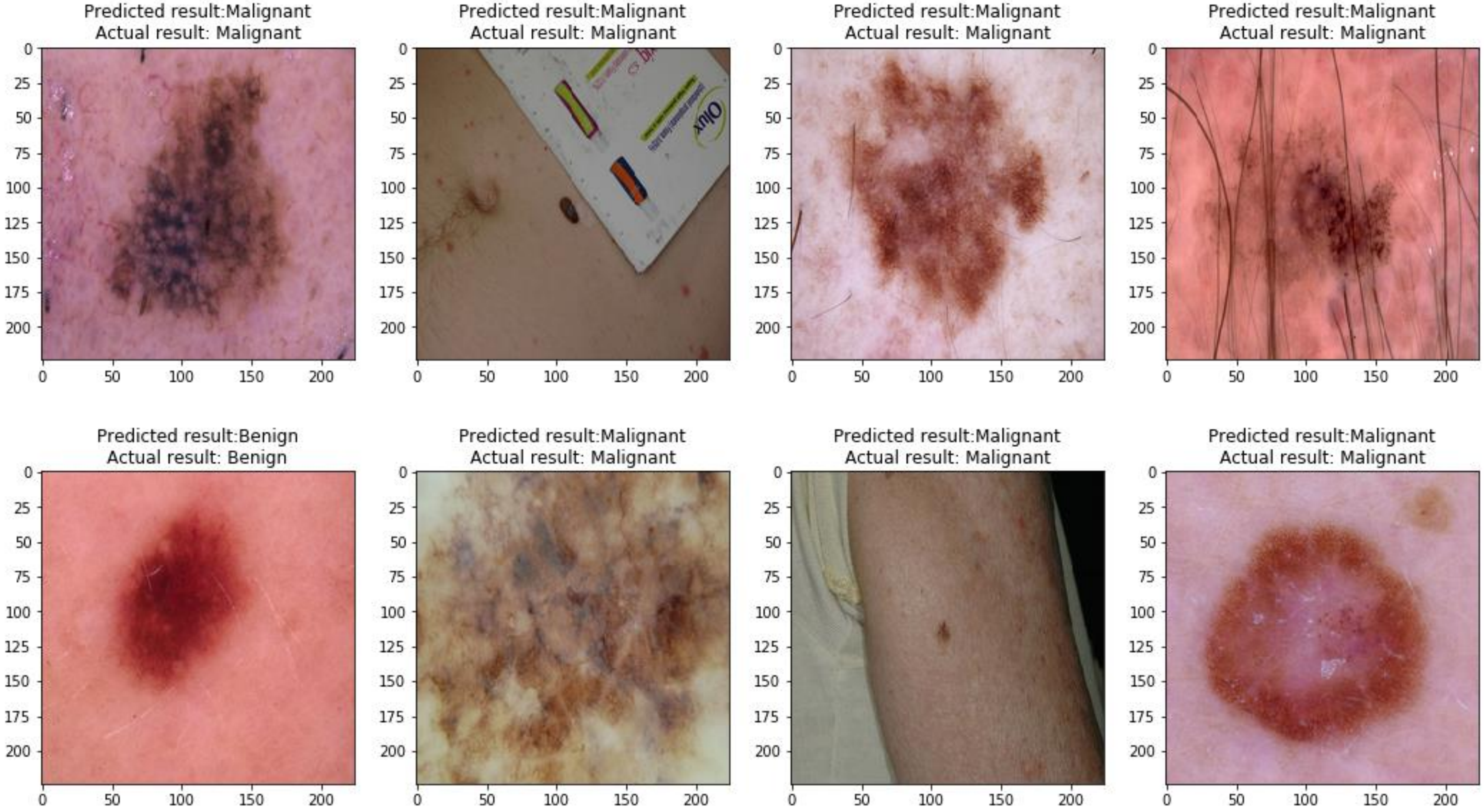
with TTA



```
Confusion matrix, without normalization
[[337  23]
 [ 34 266]]
```

	precision	recall	f1-score
Benign	0.91	0.94	0.92
Malignant	0.92	0.89	0.90
accuracy			0.91

Result



Predicted result:Malignant
Actual result: Malignant

Predicted result:Malignant
Actual result: Malignant

Predicted result:Malignant
Actual result: Malignant

Predicted result:Malignant
Actual result: Malignant

Predicted result:Benign
Actual result: Benign

Predicted result:Malignant
Actual result: Malignant

Predicted result:Malignant
Actual result: Malignant

Predicted result:Malignant
Actual result: Malignant

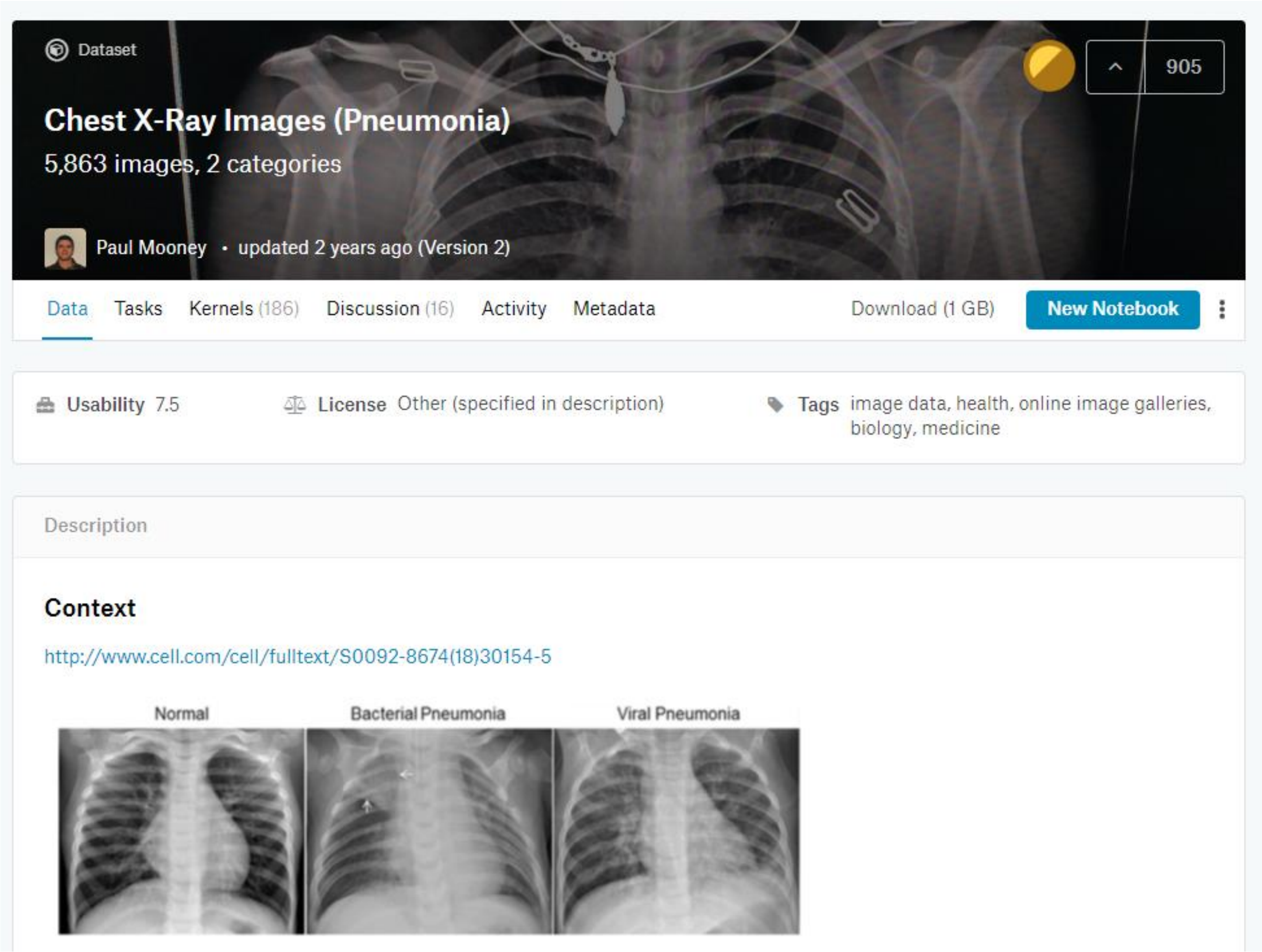
Chest X-Ray Images(Pneumonia) Dataset Classification

Image Classification

- Image Classification이란
  - 입력으로 이미지를 받아서 해당 이미지를 분류하는 것
  - 일반적으로 softmax를 통해서 one-hot encoding을 함
  - 예)
    - 정상일 때 [1 , 0] = 0 을 의미
    - 비정상일 때 [0 , 1] = 1 을 의미

Kaggle

- Kaggle : https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia



Chest X-Ray Images (Pneumonia)

5,863 images, 2 categories

Paul Mooney • updated 2 years ago (Version 2)

Usability 7.5

License Other (specified in description)

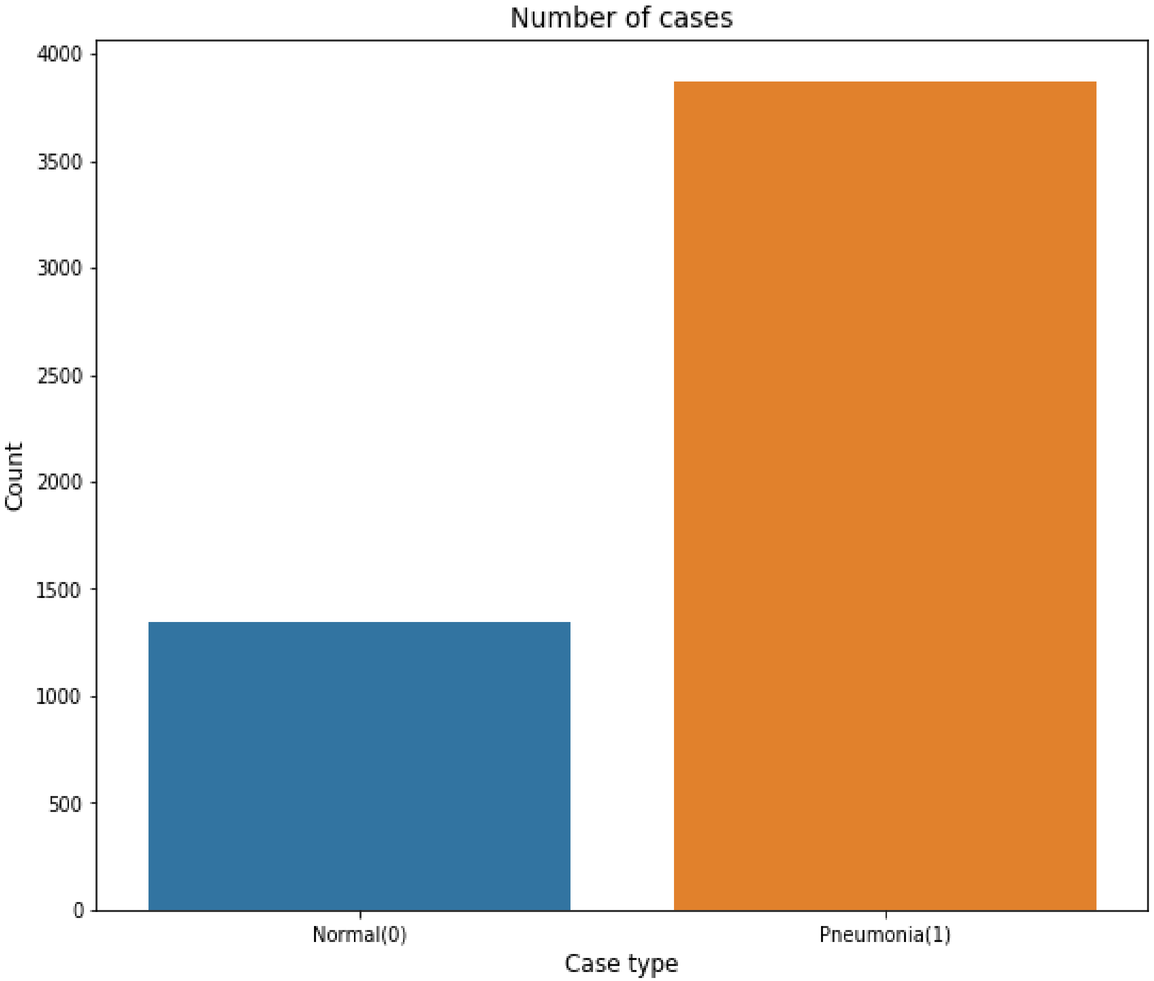
Tags image data, health, online image galleries, biology, medicine

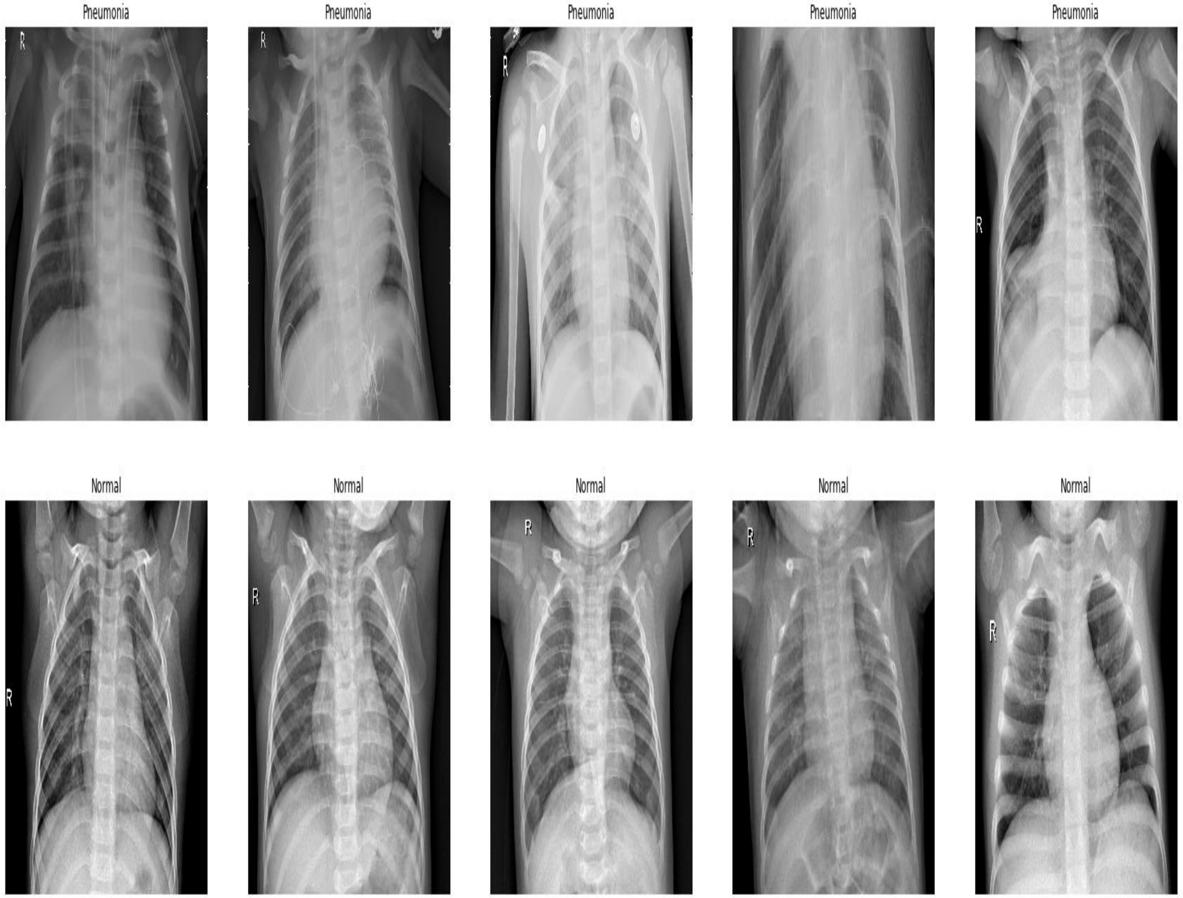
Description

Context

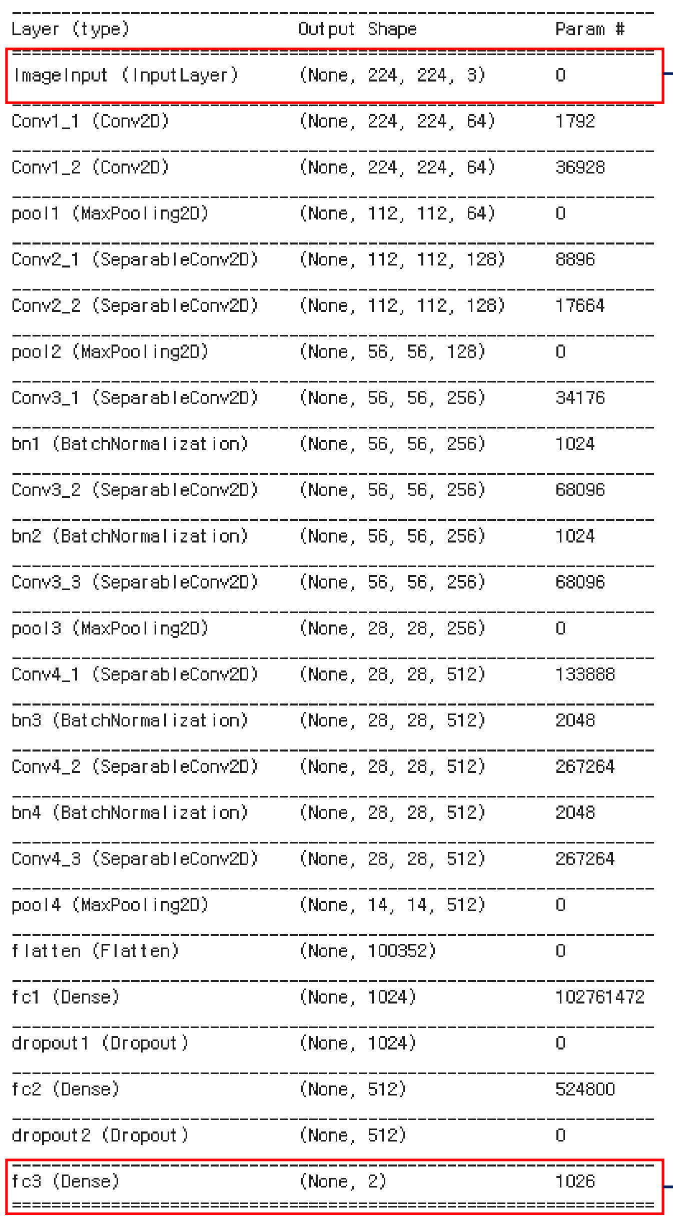
http://www.cell.com/cell/fulltext/S0092-8674(18)30154-5

DataSet


Number of cases
Count
4000
3500
3000
2500
2000
1500
1000
500
0
Normal(0)
Pneumonia(1)
Case type


Pneumonia
Pneumonia
Pneumonia
Pneumonia
Pneumonia
Normal
Normal
Normal
Normal
Normal

Model (VGG16)

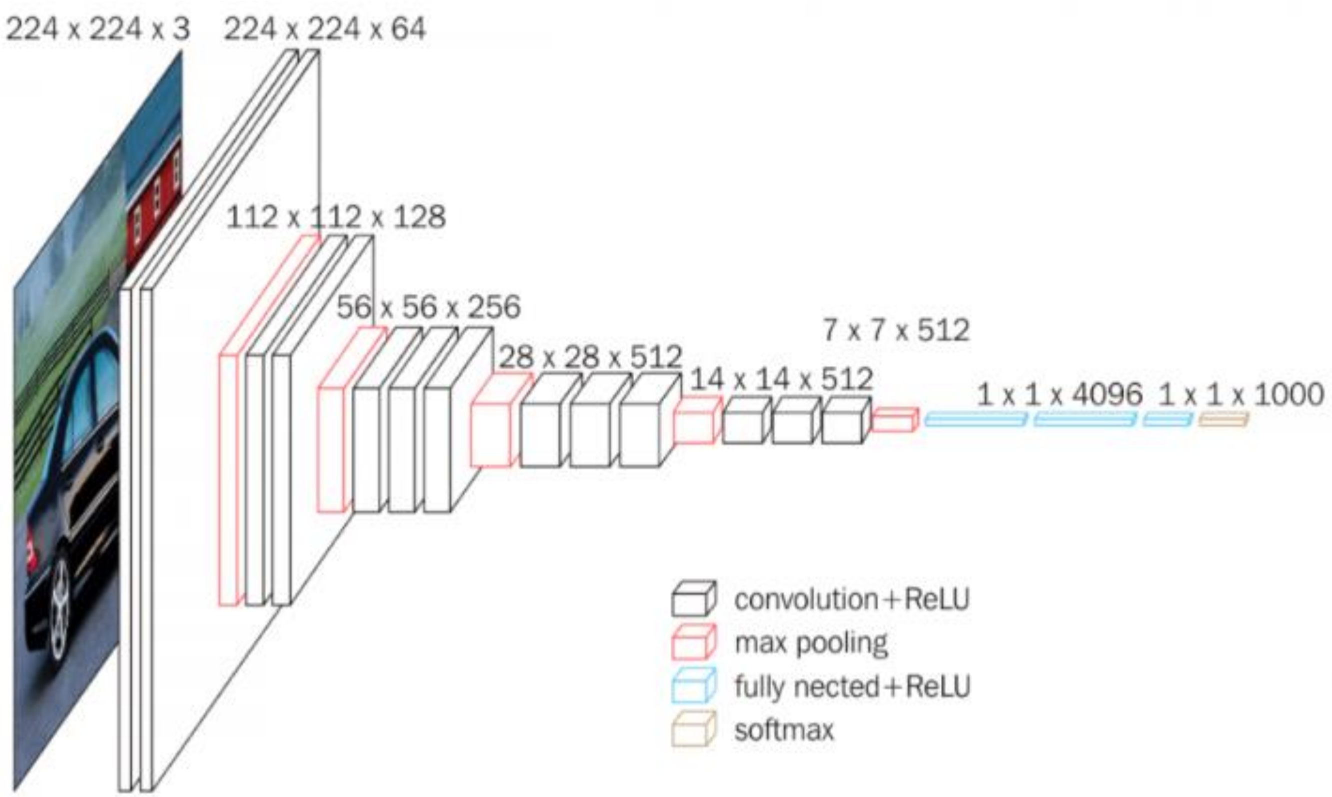


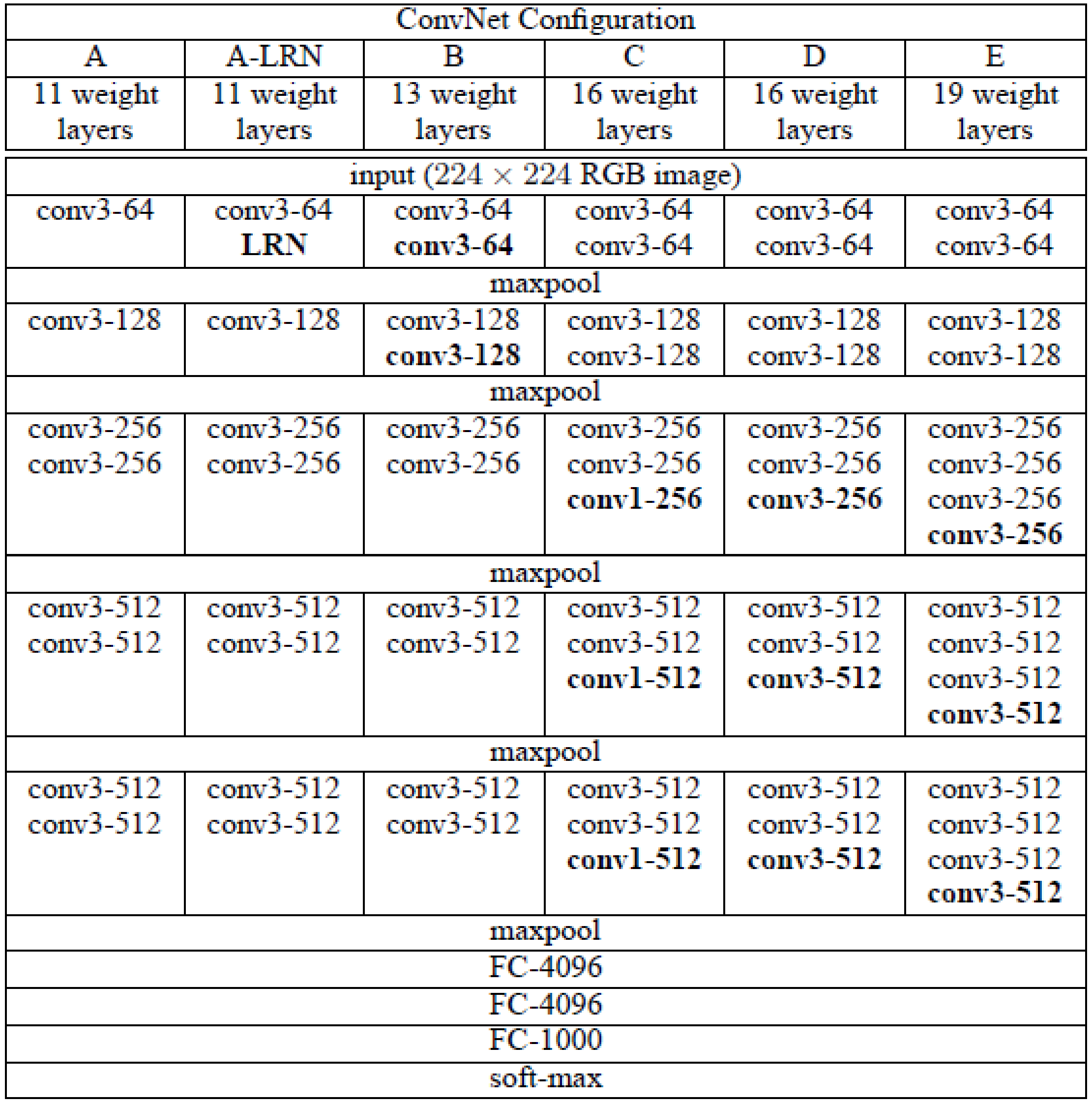
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
ImageInput (InputLayer)      (None, 224, 224, 3)       0
_________________________________________________________________
Conv1_1 (Conv2D)             (None, 224, 224, 64)      1792
_________________________________________________________________
Conv1_2 (Conv2D)             (None, 224, 224, 64)      36928
_________________________________________________________________
pool1 (MaxPooling2D)         (None, 112, 112, 64)      0
_________________________________________________________________
Conv2_1 (SeparableConv2D)    (None, 112, 112, 128)     8896
_________________________________________________________________
Conv2_2 (SeparableConv2D)    (None, 112, 112, 128)     17664
_________________________________________________________________
pool2 (MaxPooling2D)         (None, 56, 56, 128)       0
_________________________________________________________________
Conv3_1 (SeparableConv2D)    (None, 56, 56, 256)       34176
_________________________________________________________________
bn1 (BatchNormalization)     (None, 56, 56, 256)       1024
_________________________________________________________________
Conv3_2 (SeparableConv2D)    (None, 56, 56, 256)       68096
_________________________________________________________________
bn2 (BatchNormalization)     (None, 56, 56, 256)       1024
_________________________________________________________________
Conv3_3 (SeparableConv2D)    (None, 56, 56, 256)       68096
_________________________________________________________________
pool3 (MaxPooling2D)         (None, 28, 28, 256)       0
_________________________________________________________________
Conv4_1 (SeparableConv2D)    (None, 28, 28, 512)       133888
_________________________________________________________________
bn3 (BatchNormalization)     (None, 28, 28, 512)       2048
_________________________________________________________________
Conv4_2 (SeparableConv2D)    (None, 28, 28, 512)       267264
_________________________________________________________________
bn4 (BatchNormalization)     (None, 28, 28, 512)       2048
_________________________________________________________________
Conv4_3 (SeparableConv2D)    (None, 28, 28, 512)       267264
_________________________________________________________________
pool4 (MaxPooling2D)         (None, 14, 14, 512)       0
_________________________________________________________________
flatten (Flatten)            (None, 100352)            0
_________________________________________________________________
fc1 (Dense)                  (None, 1024)              102761472
_________________________________________________________________
dropout1 (Dropout)           (None, 1024)              0
_________________________________________________________________
fc2 (Dense)                  (None, 512)               524800
_________________________________________________________________
dropout2 (Dropout)           (None, 512)               0
_________________________________________________________________
fc3 (Dense)                  (None, 2)                 1026
=================================================================
```

Input = 224X224X3(color)이미지가 모델의 입력

Output = 2로 0 = 정상 , 1 = 폐렴으로 판단

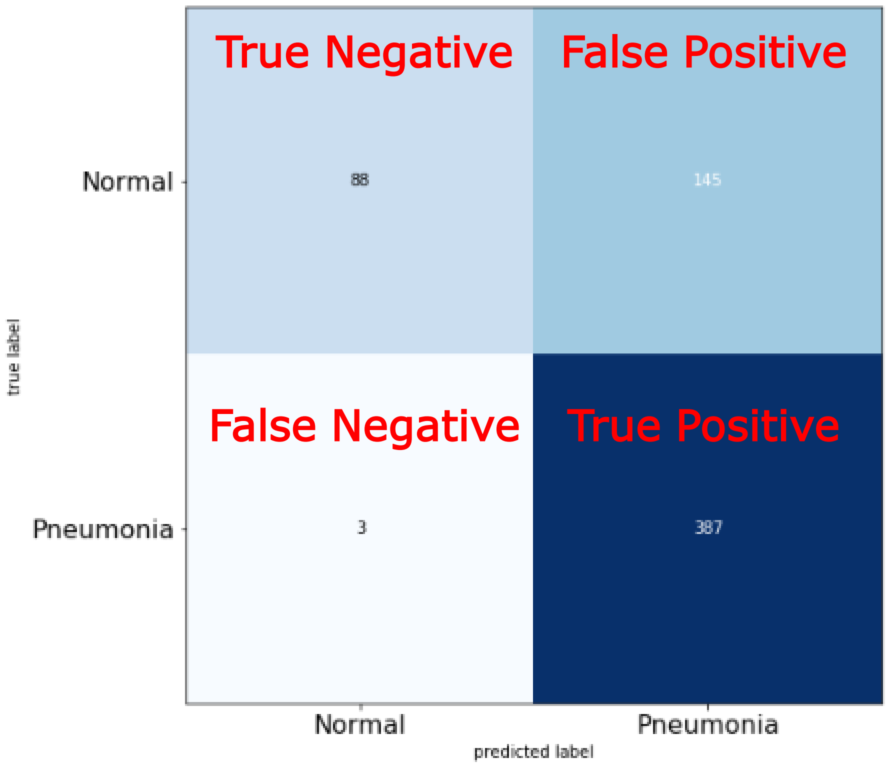
VGG Network Architecture





ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

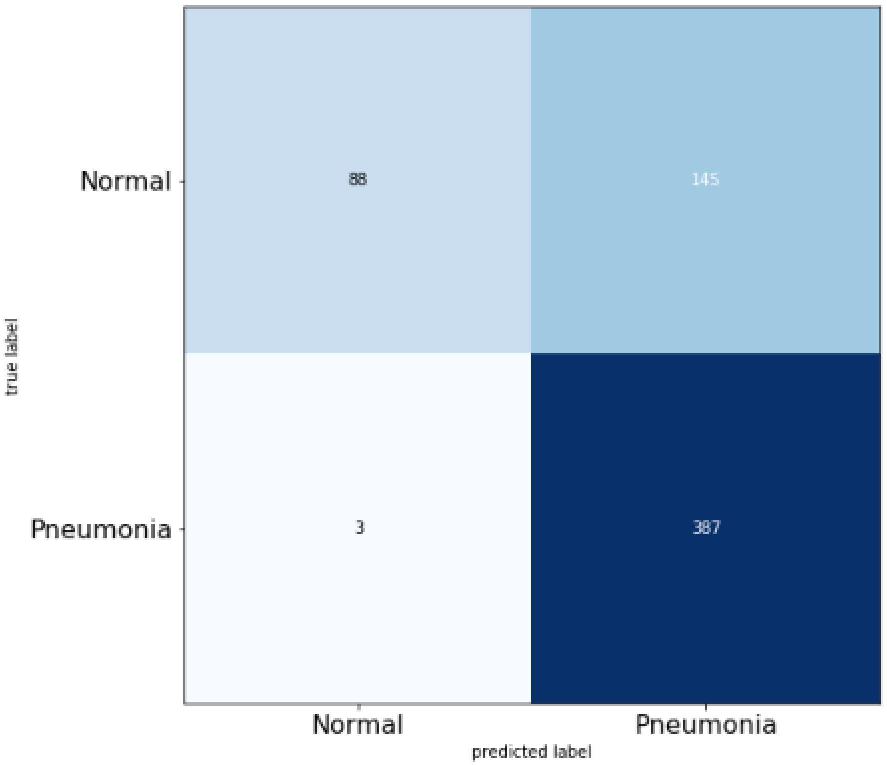
Test Confusion Matrix



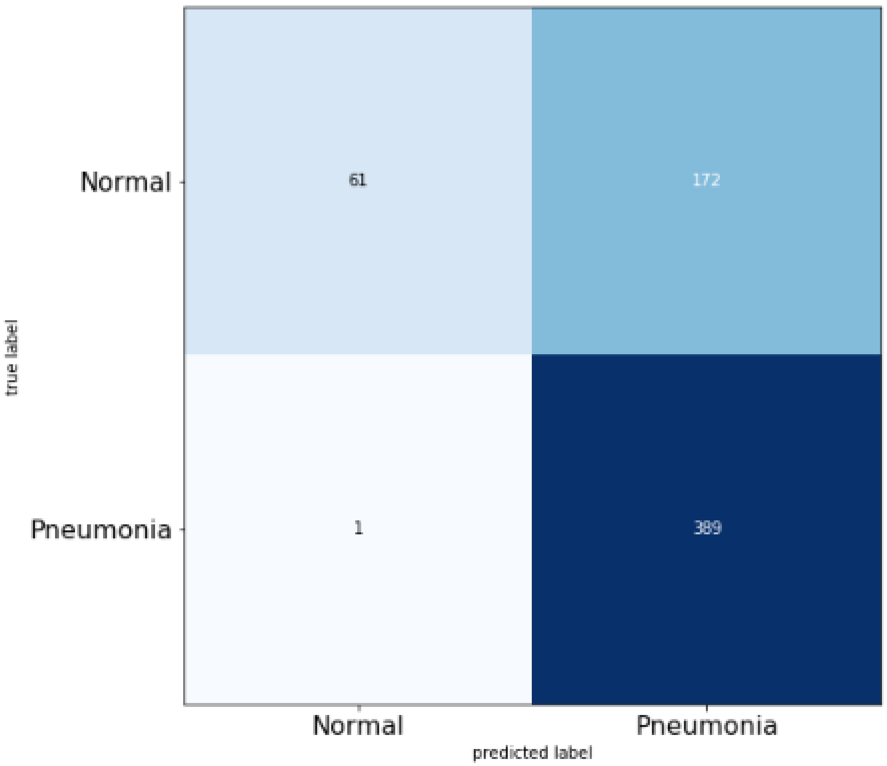
- Precision = TP/(TP+FP)
- Recall = TP/(TP+FN)

Test Confusion Matrix

VGG19 결과



VGG16 결과


Normal
Pneumonia
true label
Normal
Pneumonia
predicted label