2조

CNN 모델을 통한 이미지 분류

김용철, 박강인, 반위홍, 신철호, 오해윤

CNN모델을 통한 이미지 분류

Convolutional Neural Network

목차

프로젝트 설명

프로젝트 방향

이미지 전처리 및 증강 방법 구현

VGGNet, RESNet 학습 및 구현

CNN모델을 통한 이미지 분류

Convolutional Neural Network

사용한 데이터

UCF 101

UCF 101 란?

101 카테고리의 동작을 행동 인식을 위해 유튜브에서 실제 움직을 녹화한 데이터

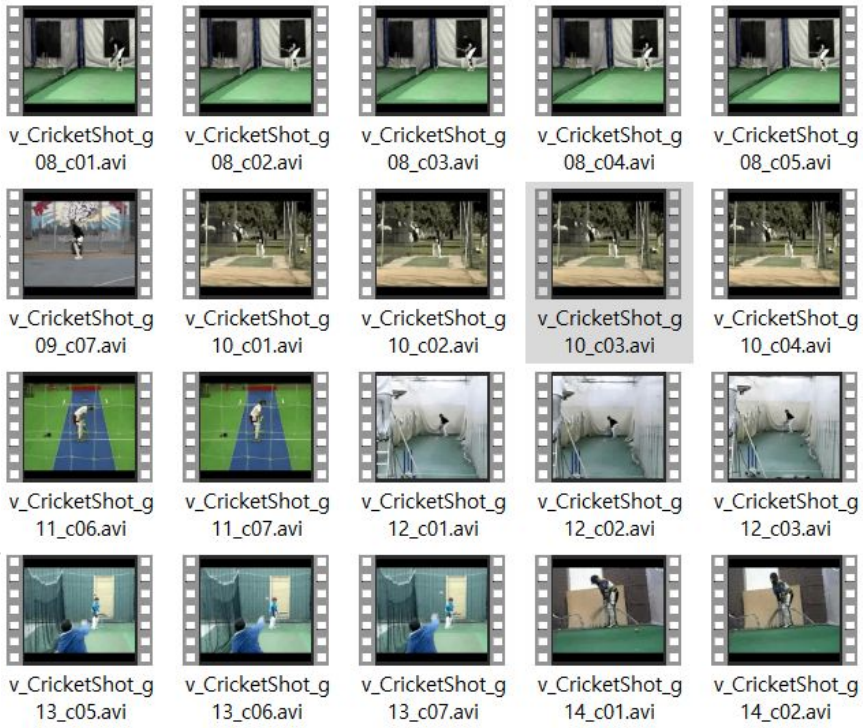
UCF 101은 다양성 동작을 제공하며 카메라 모션, 물체 모양 및 포즈, 물체 크기, 시점, 배경, 조명 등 현재까지 다양한 데이터 셋을 갖추고 있음

액션 카테고리 유형 (1) 사람-물체 간의 상호작용 (2) 신체 움직임 (3) 사람-사람 간의 상호작용 (4) 악기 연주 (5) 스포츠

101 카테고리는 눈 화장, 립스틱 바르기, 양궁, 기어가는 아기, 야구장, 농구, 벤치프레스, 자전거, 스쿼트, 양초불기 등

Cricket Shot, **Punch,** **Tennis Swing**

UCF CENTER FOR RESEARCH IN COMPUTER VISION



데이터 전처리

CNN모델을 통한 이미지 분류

Convolutional Neural Network

기존 강의 코드

1. 전처리

이미지 크기 설정

IMG_SIZE: 224*224

MAX_SEQ_LENGTH: 20

비디오 파일 설정

Label column 추가

```
for index, data in enumerate(train_df["label"].unique()):
    train_df["label"].replace(data, index, inplace=True)
```

인덱스	video_name	tag	label
0	v_CricketShor_g_c.avi	CricketShot	0
~	v_Punch_g_c.avi	Punch	1
356	v_TennisSwing_g_c.avi	TennisSwing	2

Index shuffle

```
train_df = sklearn.utils.shuffle(train_df)
```

CNN모델을 통한 이미지 분류

Convolutional Neural Network

기존 강의 코드

1. 전처리

이미지 가운데 부분 리턴

```
def crop_center_square(frame):
    y, x = frame.shape[0:2]
    min_dim = min(y, x)
    start_x = (x // 2) - (min_dim // 2)
    start_y = (y // 2) - (min_dim // 2)
    return frame[start_y : start_y + min_dim,
start_x : start_x + min_dim]
```

파일의 비디오를 각 프레임으로 리턴

```
def load_video(path, max_frames=20, resize=(IMG_SIZE, IMG_SIZE)):
    cap = cv2.VideoCapture(path)
    frames = []
    try:
        while True:
            ret, frame = cap.read()
            if not ret:
                break
            frame = crop_center_square(frame)
            frame = frame[:, :, [2, 1, 0]]
            frames.append(frame)
            if len(frames) == max_frames:
                break
    finally:
        cap.release()
    return np.array(frames)
```

비디오 파일의 이미지와 종류를 리턴

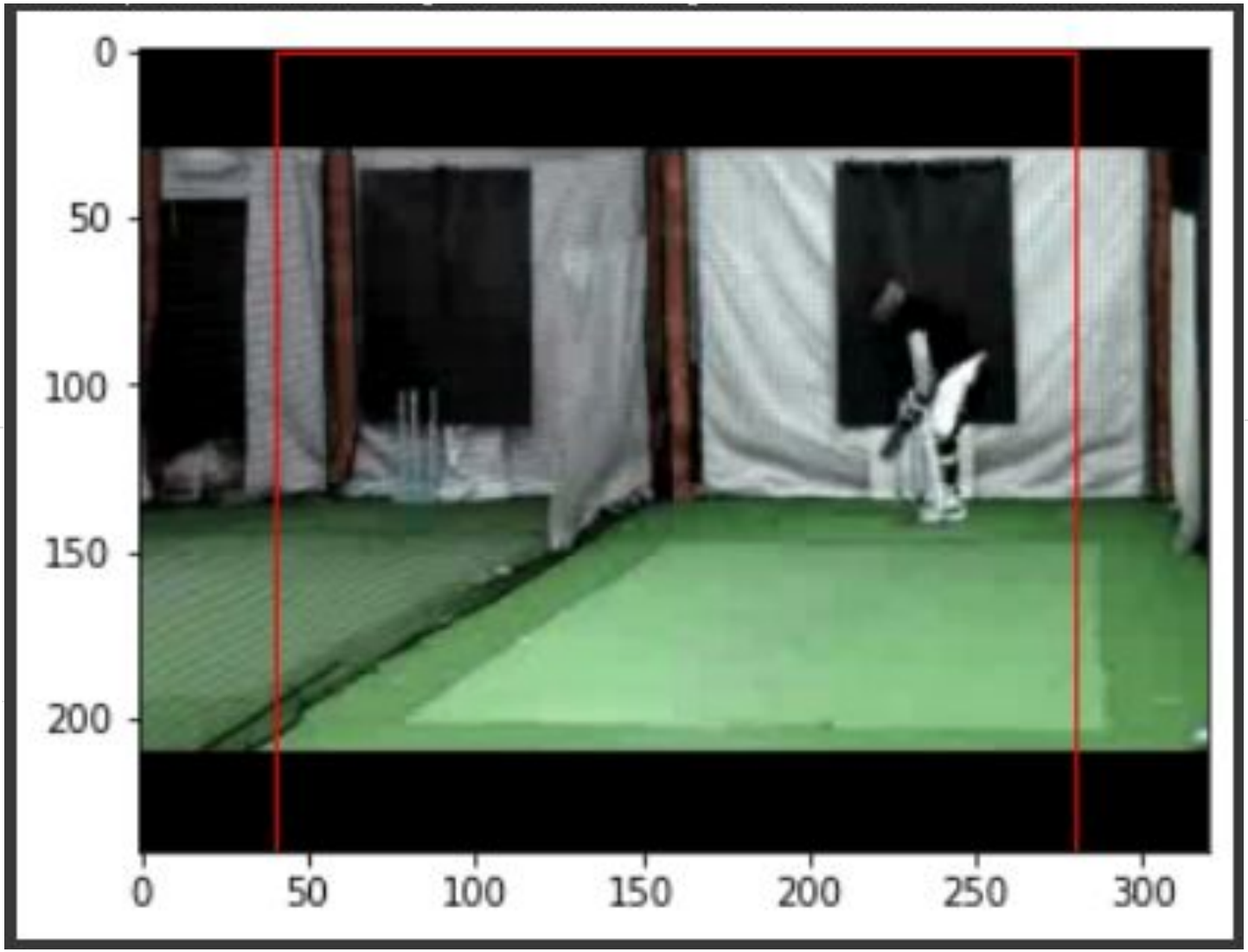
```
def prepare_all_videos(video_name , label, root_dir):
    num_samples = len(video_name)
    video_paths = video_name.values.tolist()
    labels = label.values
    labels = labels.reshape(-1,1)
    x = np.zeros(shape=(num_samples*MAX_SEQ_LENGTH, IMG_SIZE, IMG_SIZE, 3 ),
dtype="float32" )
    y = np.zeros(shape=(num_samples*MAX_SEQ_LENGTH), dtype="float32")
    index = 0
    for idx, path in enumerate(video_paths):
        frames = load_video(root_dir+ path)
        for i in range(len(frames)):
            x[index] = np.array(frames[i], dtype="float32")
            y[index]= np.array(labels[idx], dtype="float32")
            index += 1
    return (x,to_categorical(y))
```

Train_set를 prepare_all_vedeo 클래스에 적용

```
X_train, y_train =
prepare_all_videos(train_df["video_name"],
train_df["label"] , data_path + "train/")
```

CNN모델을 통한 이미지 분류

Convolutional Neural Network



CNN모델을 통한 이미지 분류

Convolutional Neural Network

데이터 증강

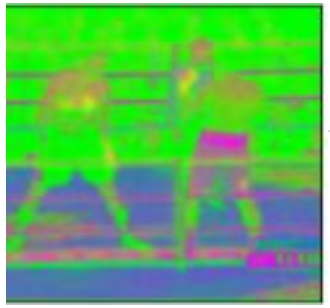
원본 이미지

horizontal flip

vertical flip

Affine 변환

Noise 추가



Color channel변환

데이터 증강

Cutmix

원본 이미지를 다른 이미지와 겹침으로서 CNN으로 하여금 이미지의 덜 중요한 부분까지 포커싱하게 만드는 regional dropout 전략

데이터 증강

cutmix 알고리즘 구현

우선 두 개의 이미지를 선택

원본 이미지

자를 이미지

CNN모델을 통한 이미지 분류

Convolutional Neural Network

데이터 증강

원본 이미지의 비율인 λ(0 ~ 1)를 sample하여 이미지 자름

λ = 0.75

원본 이미지

$W\sqrt{1-\lambda}$

$H\sqrt{1-\lambda}$

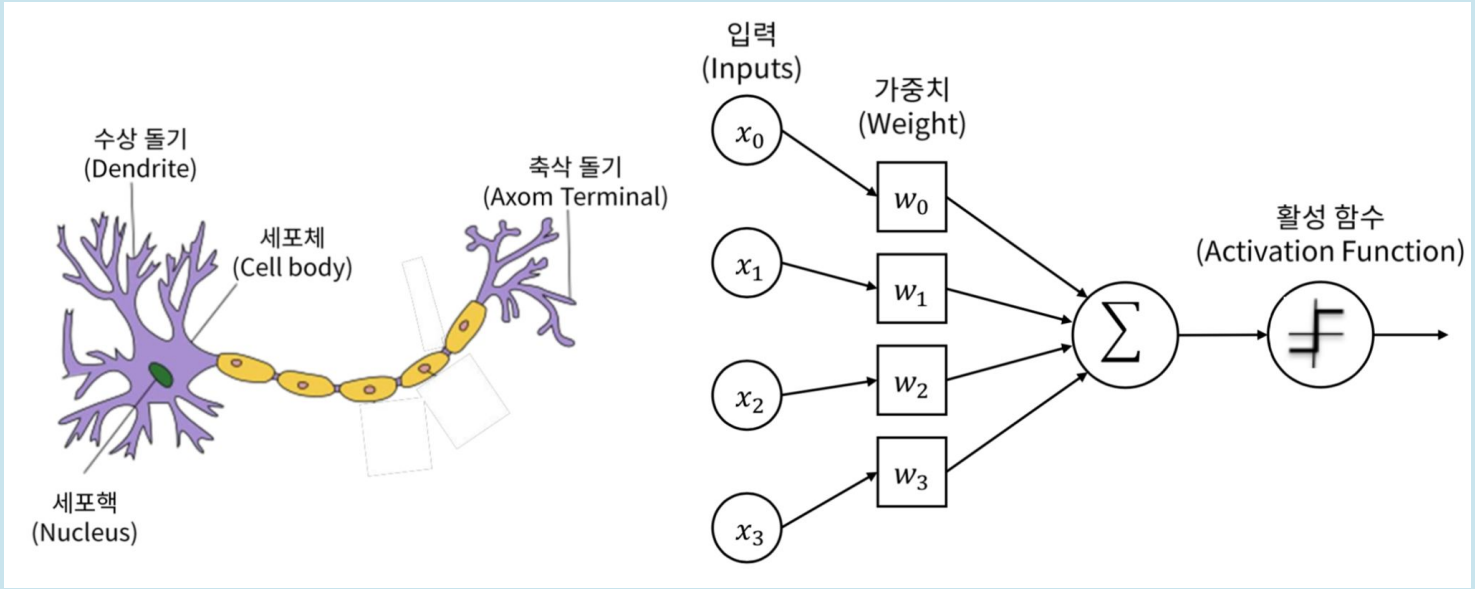
새로운 이미지 생성 후 비율에 따른 새로운 Label 생성

Label : 크리켓 0.75, 복싱 0.25

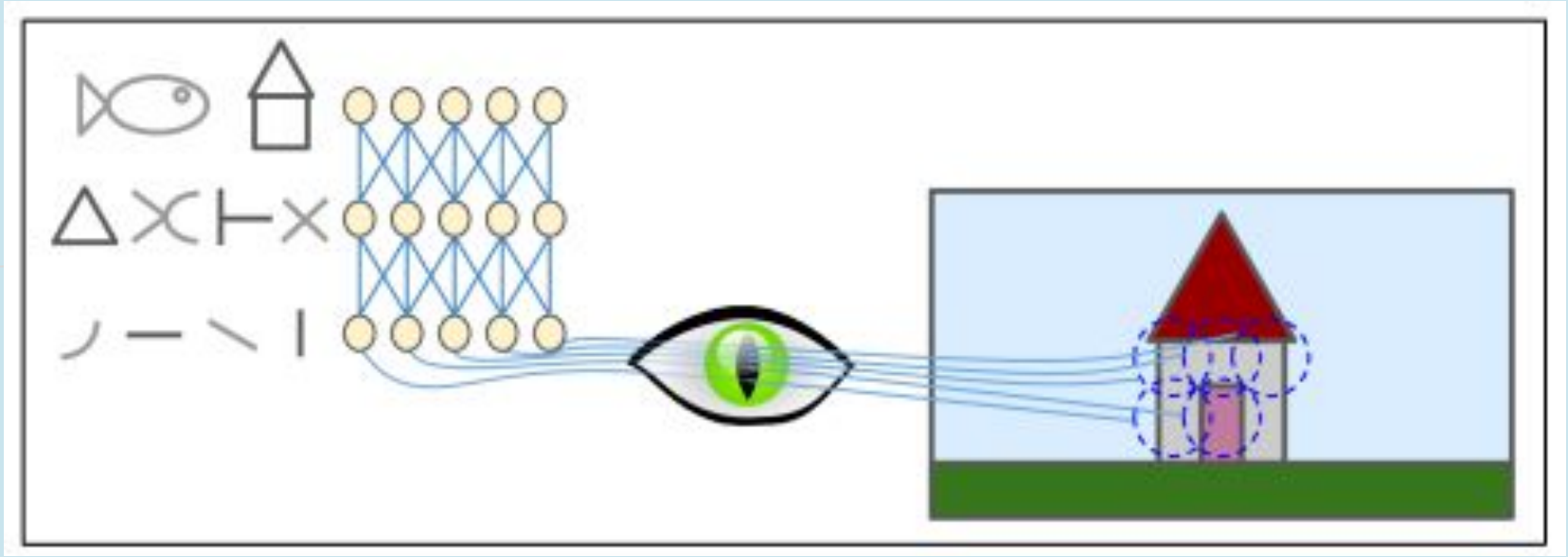
new image

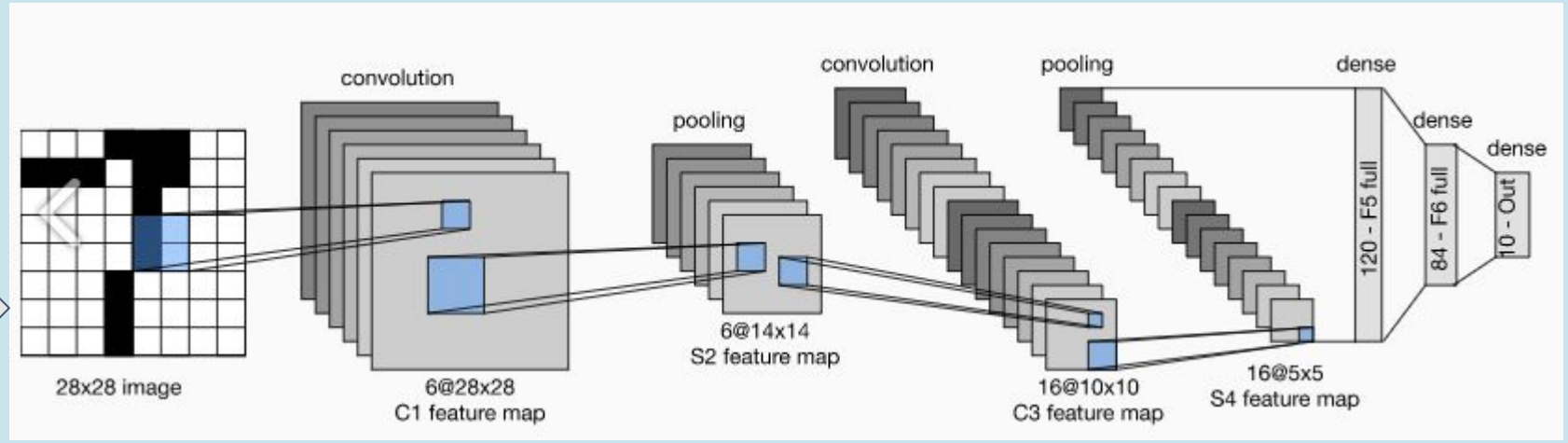
CNN 모델 학습

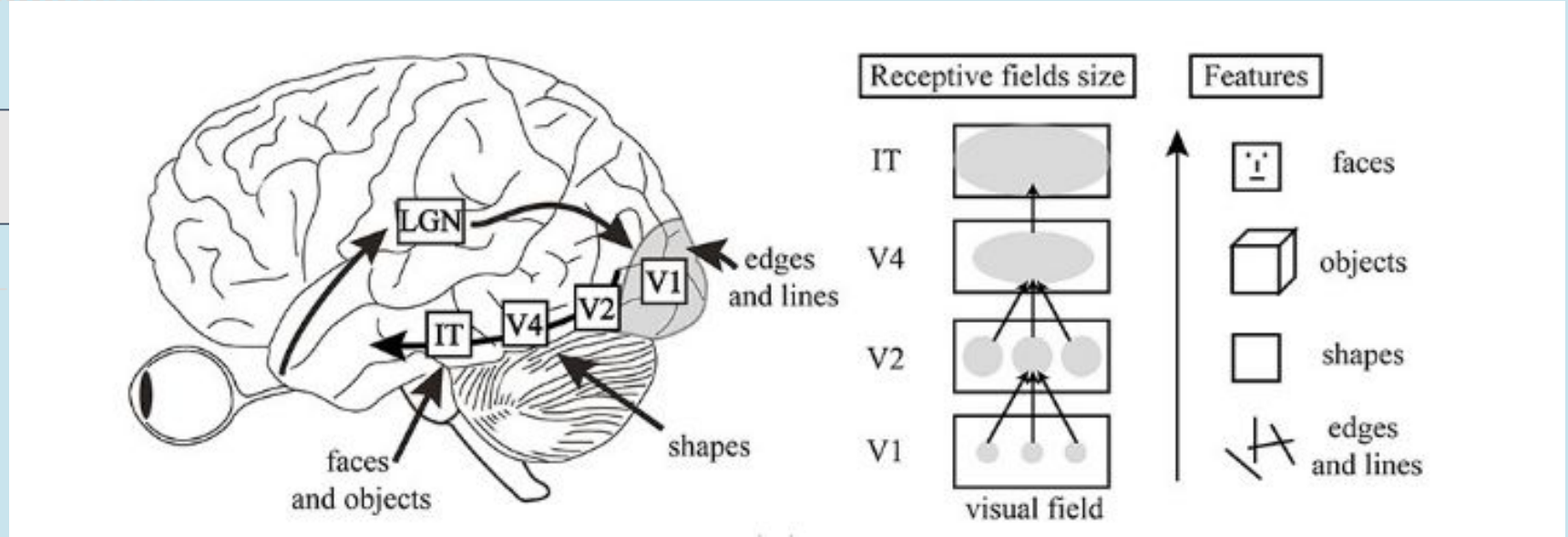
신경망의 원리



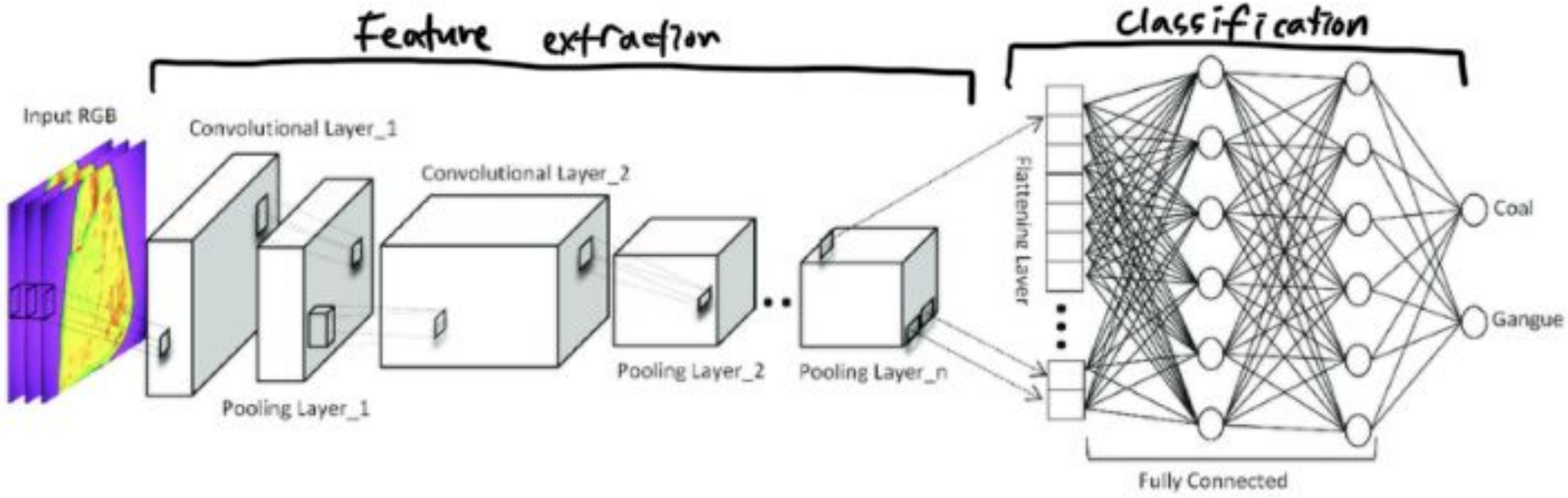
CNN 원리




convolution
pooling
convolution
pooling
dense
dense
dense
28x28 image
6@28x28
C1 feature map
6@14x14
S2 feature map
16@10x10
C3 feature map
16@5x5
S4 feature map
120 - F5 full
84 - F6 full
10 - Out


LGN
V1
V2
V4
IT
edges
and lines
shapes
faces
and objects
Receptive fields size
Features
IT
V4
V2
V1
visual field
faces
objects
shapes
edges
and lines

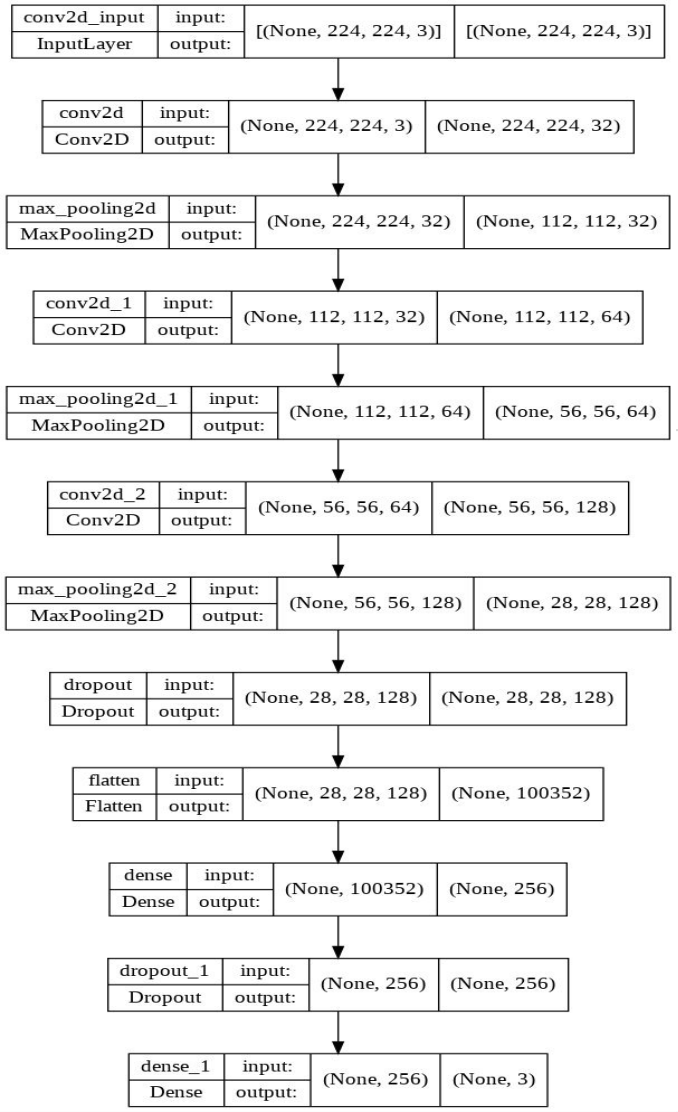
CNN 모델의 구조


Feature extraction
Classification
Input RGB
Convolutional Layer_1
Convolutional Layer_2
Pooling Layer_1
Pooling Layer_2
Pooling Layer_n
Flattening Layer
Coal
Gangue
Fully Connected

CNN모델을 통한 이미지 분류

Convolutional Neural Network

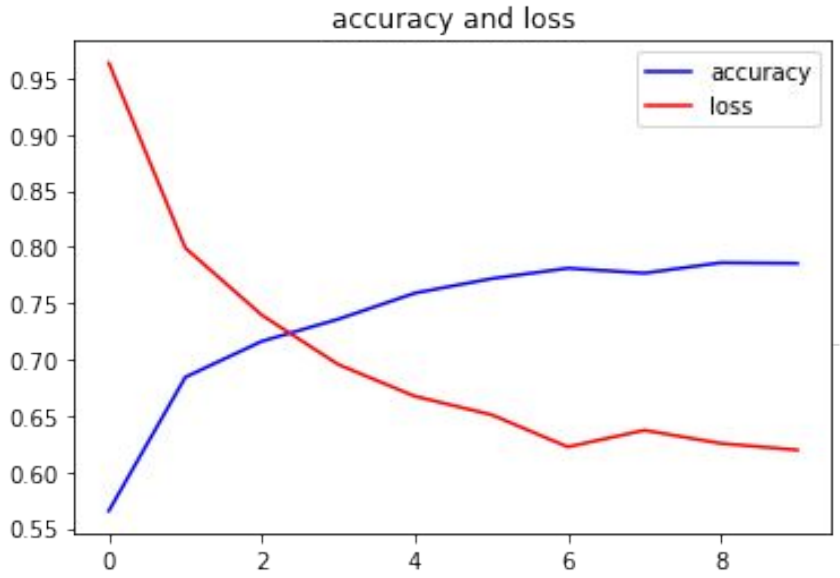
CNN 모델 Architecture




```
1 #생성된 모델 정보 출력
2 model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 224, 224, 32)      896

 max_pooling2d (MaxPooling2D  (None, 112, 112, 32)     0
 )

 conv2d_1 (Conv2D)           (None, 112, 112, 64)      18496

 max_pooling2d_1 (MaxPooling  (None, 56, 56, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 56, 56, 128)       73856

 max_pooling2d_2 (MaxPooling  (None, 28, 28, 128)      0
 2D)

 dropout (Dropout)           (None, 28, 28, 128)       0

 flatten (Flatten)           (None, 100352)            0

 dense (Dense)               (None, 256)               25690368

 dropout_1 (Dropout)         (None, 256)               0

 dense_1 (Dense)             (None, 3)                 771

=================================================================
Total params: 25,784,387
Trainable params: 25,784,387
Non-trainable params: 0
```

CNN 모델 성능 평가




```
1 print('CNN Model time of : ', end1)
```

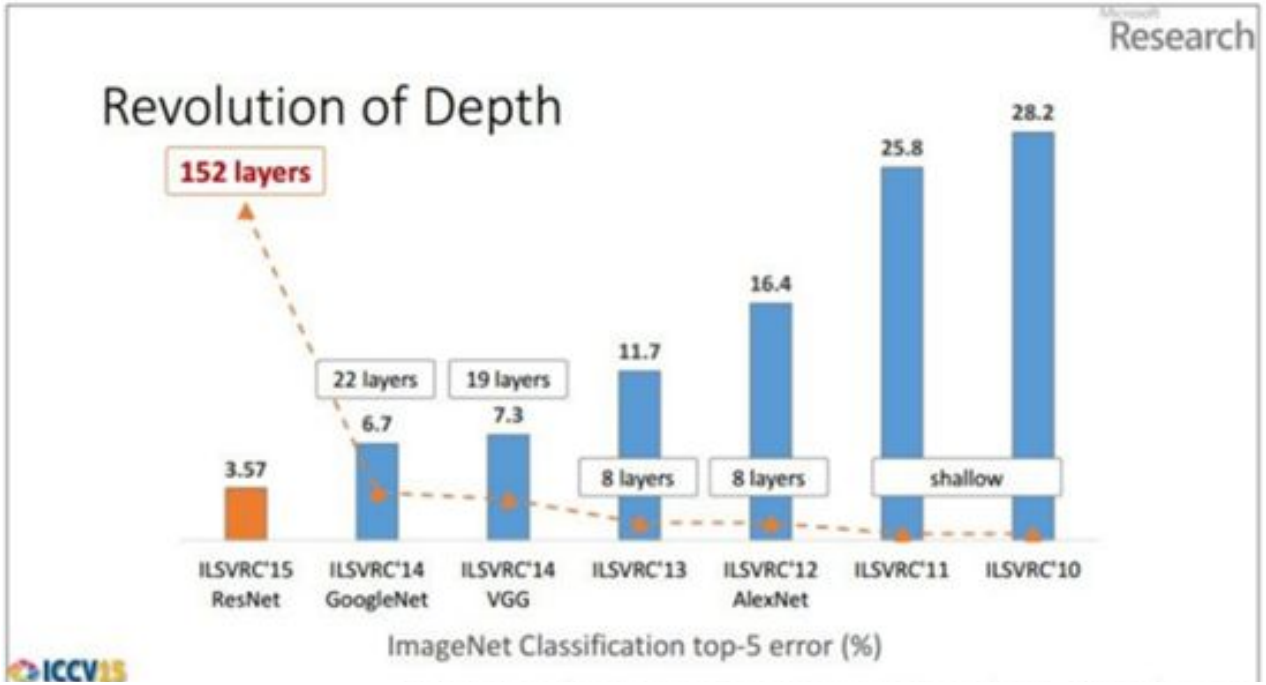
```
CNN Model time of :  1741.3608090877533
```

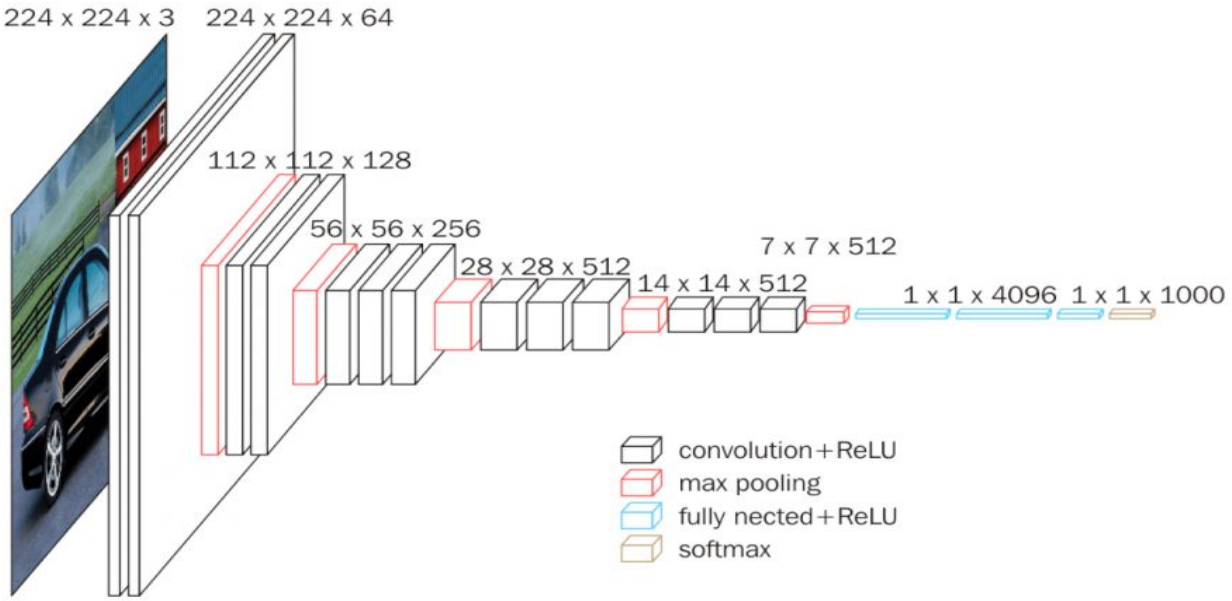
```
1 print('test_accuracy :', eval[1])
```

```
test_accuracy : 0.8467153310775757
```

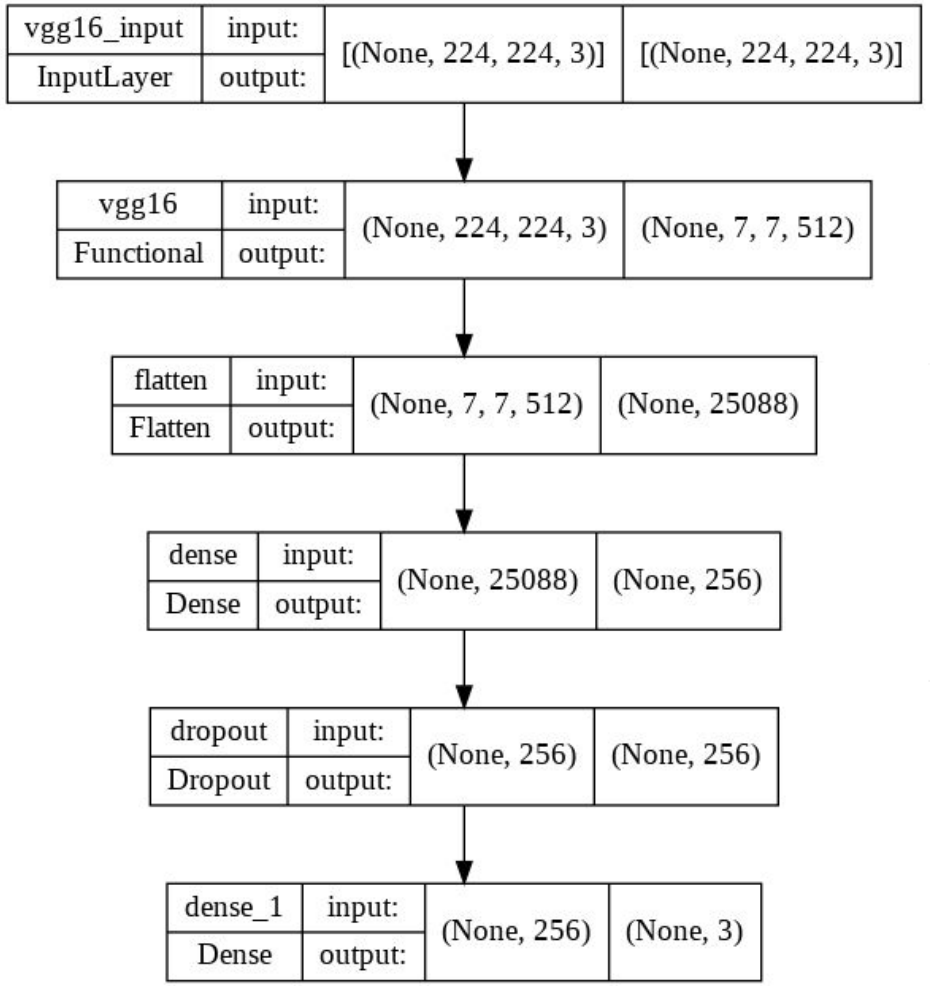
VGGNet 모델 구조

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**
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**
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**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					



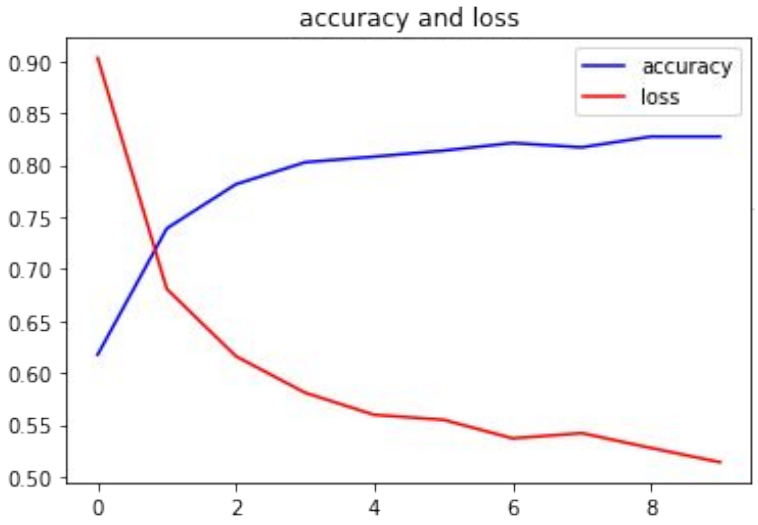


재구성한 VGG16 모델 Architecture




```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 256)               6422784

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 3)                 771

=================================================================
Total params: 21,138,243
Trainable params: 6,423,555
Non-trainable params: 14,714,688
_________________________________________________________________
```

VGG16 모델 성능 평가



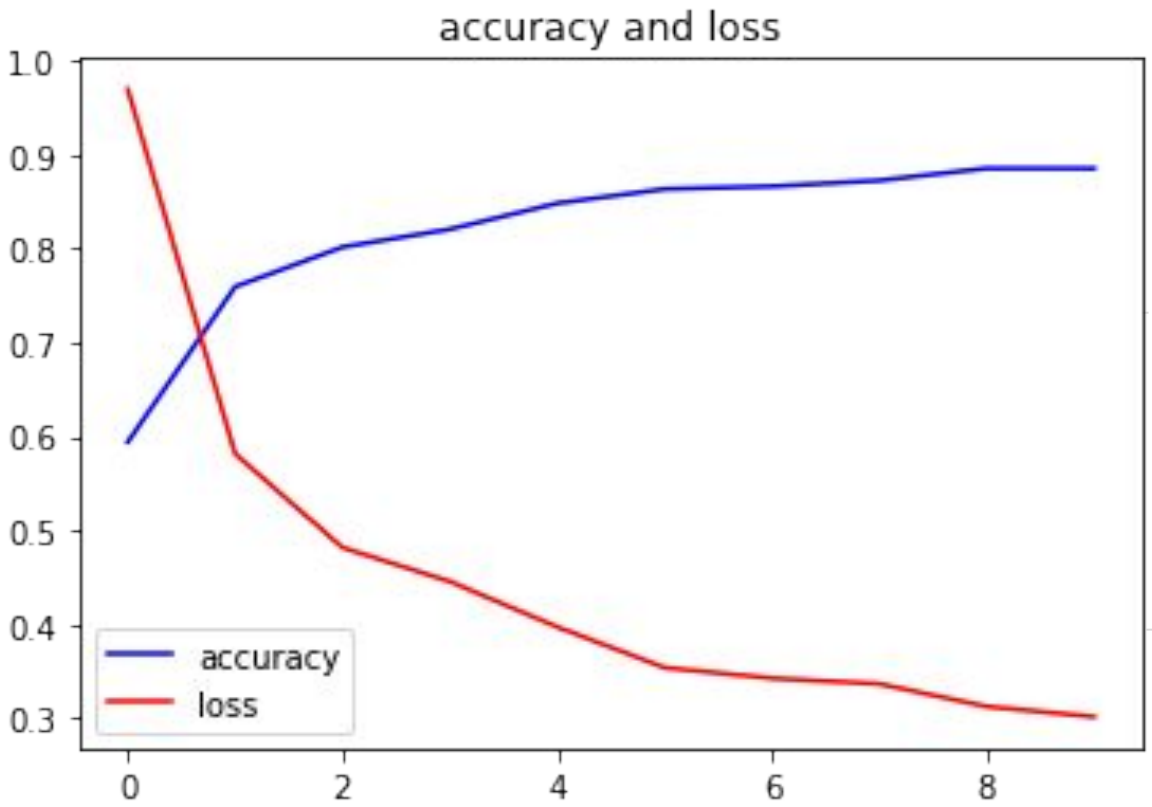

```
1  print('VGG16 Model time of : ', end1)
```

```
VGG16 Model time of :  1579.9311623573303
```

```
1  print('test_accuracy :', eval[1])
```

```
test_accuracy : 0.9171532988548279
```

VGG19 모델 성능평가



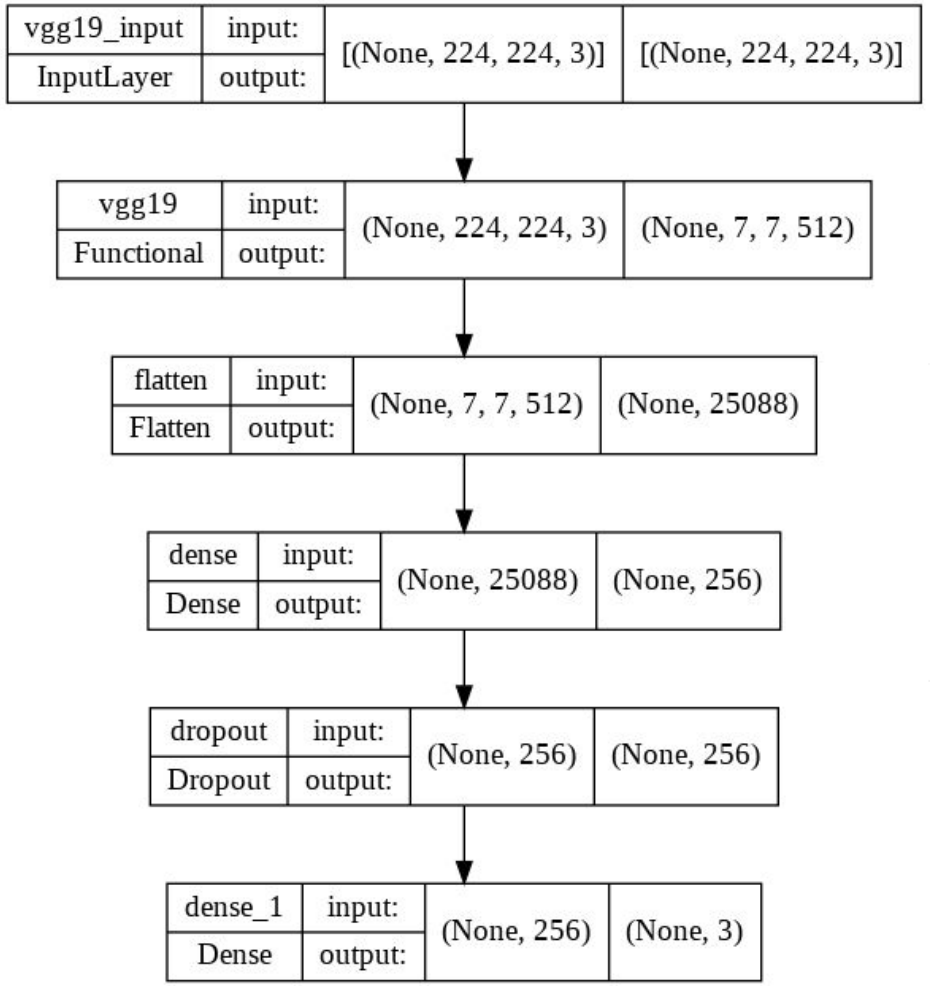

```
print('Time of VGG19: ', end1)
```

```
Time of VGG19:  1894.5123617649078
```

```
print('Accuracy of VGG19: ', eval[1])
```

```
Accuracy of VGG19:  0.9171532988548279
```

재구성한 VGG19 모델 Architecture



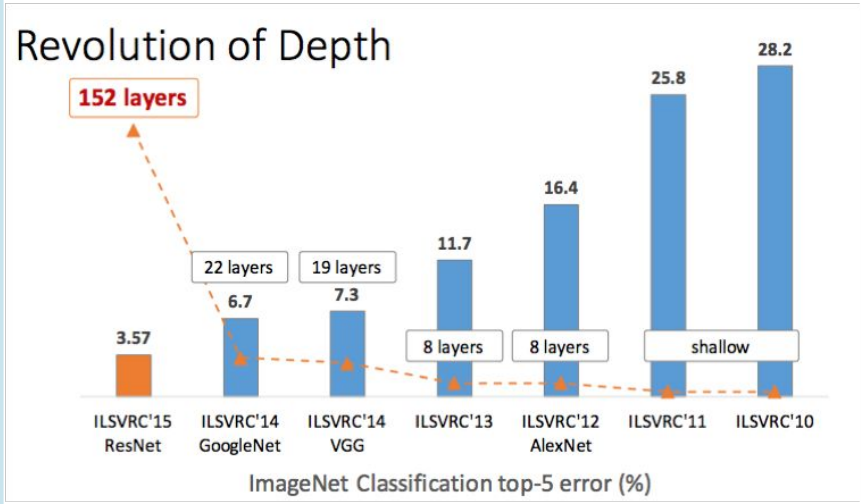

```
#생성된 모델 정보 출력
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg19 (Functional)           (None, 7, 7, 512)         20024384

flatten (Flatten)            (None, 25088)             0

dense (Dense)                (None, 256)               6422784

dropout (Dropout)            (None, 256)               0

dense_1 (Dense)              (None, 3)                 771

=================================================================
Total params: 26,447,939
Trainable params: 6,423,555
Non-trainable params: 20,024,384
_________________________________________________________________
```

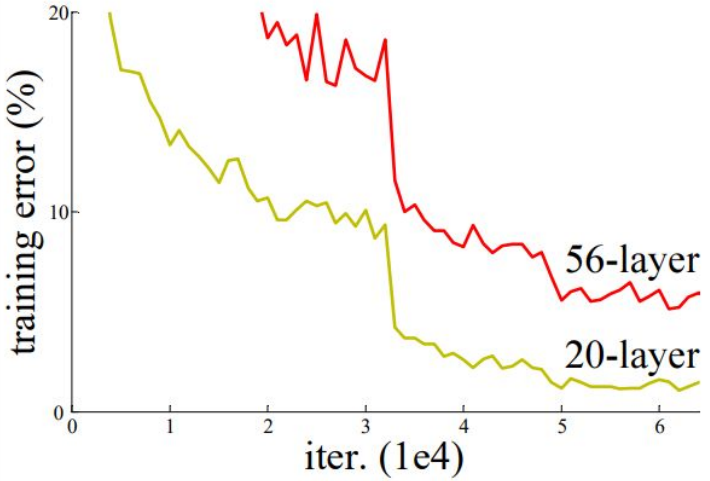
CNN모델을 통한 이미지 분류

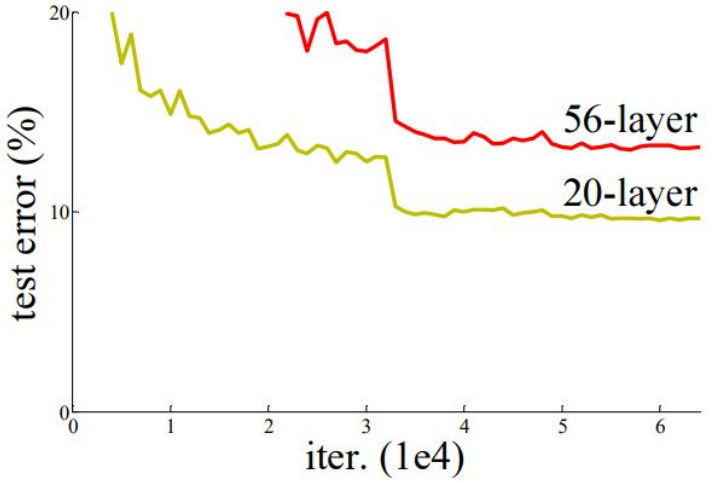
Convolutional Neural Network

RESNet 소개



층의 깊이에 따른 성능(모델별 비교)





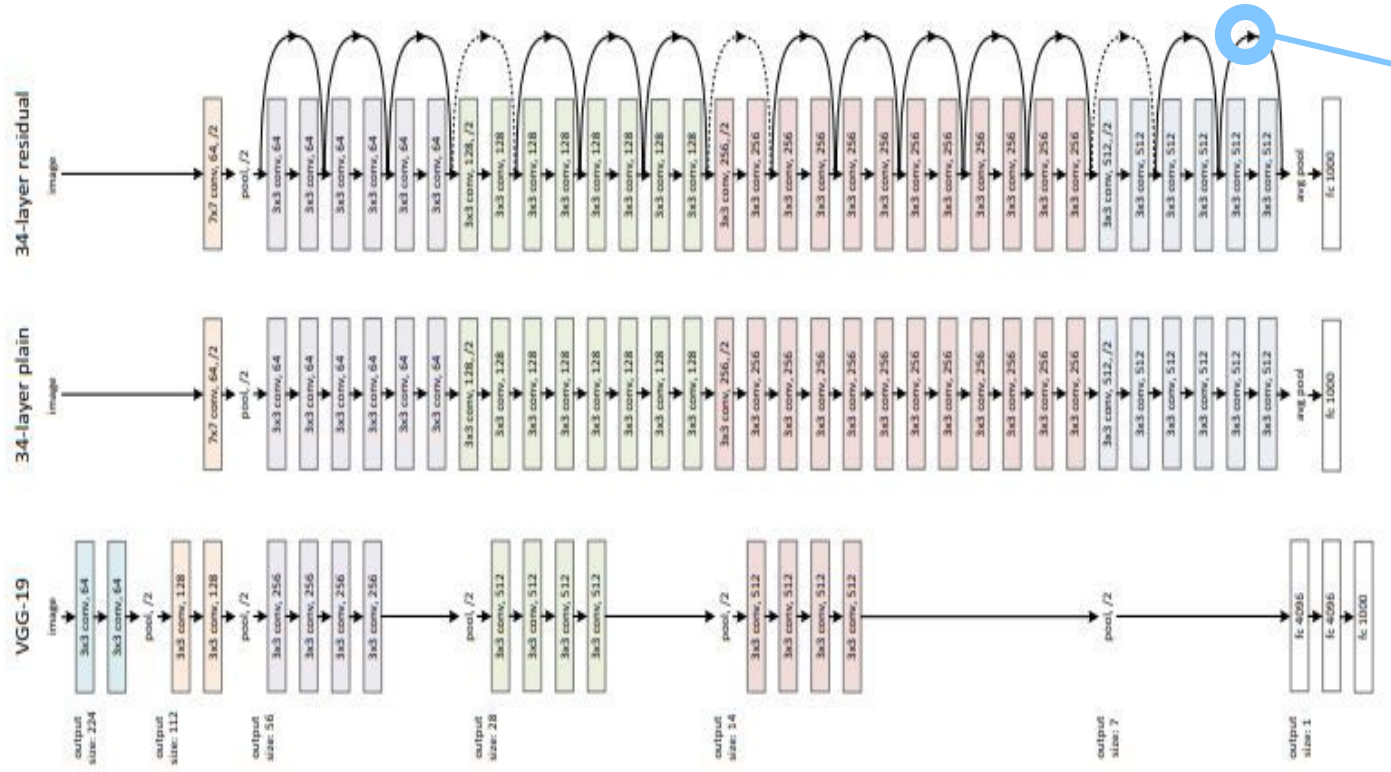
RESNet 저자 테스트

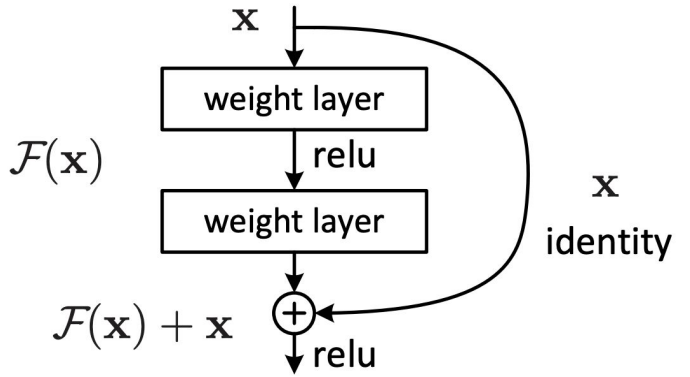
56층의 신경망의 성능이 좋아지다가 에포크 3회 이후 성능이 역전되어 떨어지는 것을 확인 할 수 있음

CNN모델을 통한 이미지 분류

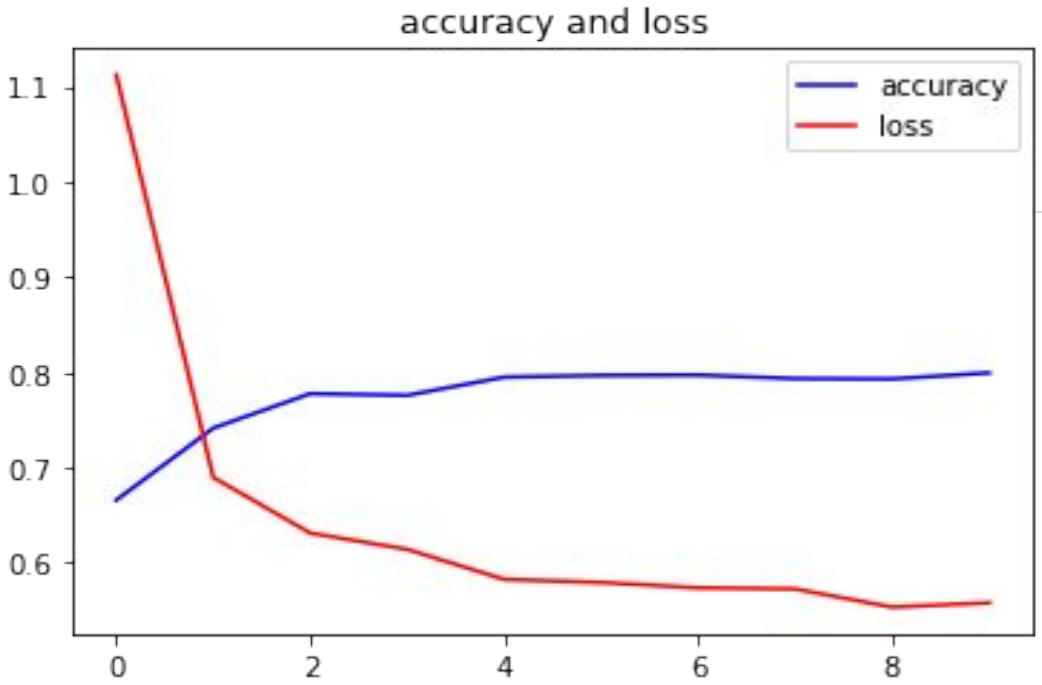
Convolutional Neural Network

RESNet 소개





ResNet50 모델 성능평가



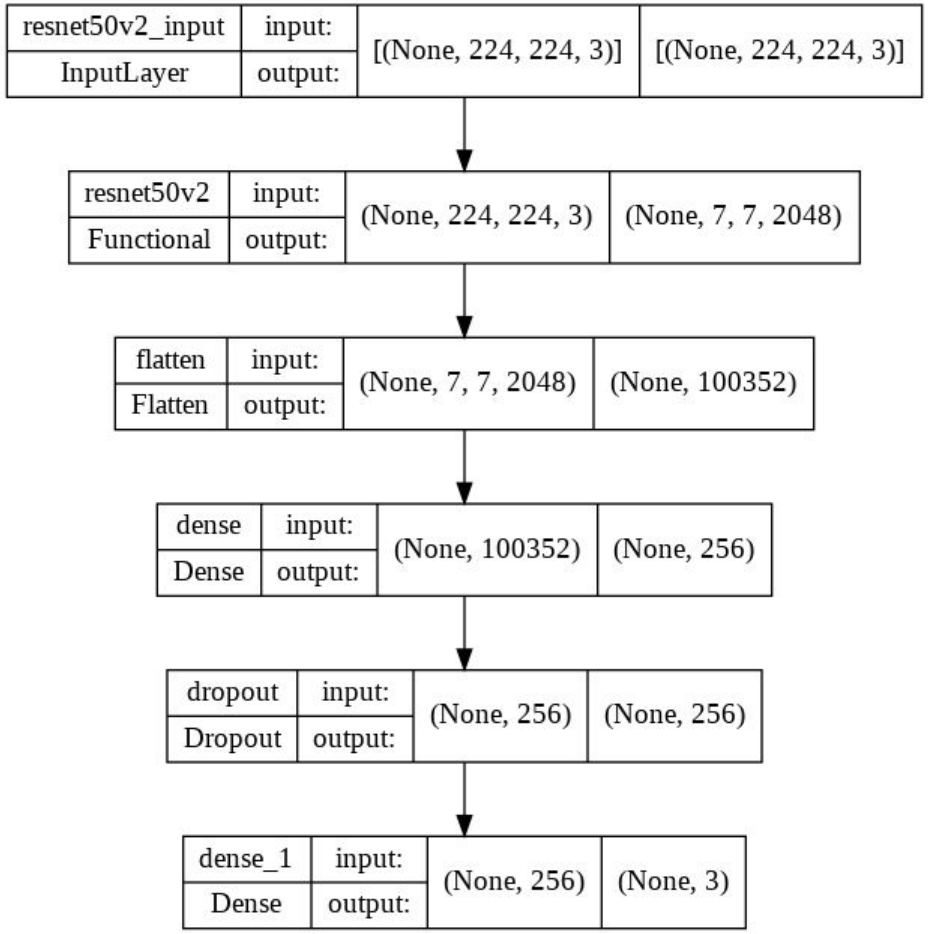

```
1 print('ResNet50v2 Model time of : ', end1)
```

```
ResNet50v2 Model time of :  1824.9995138645172
```

```
1  print('test_accuracy :', eval[1])
```

```
test_accuracy : 0.9171532988548279
```

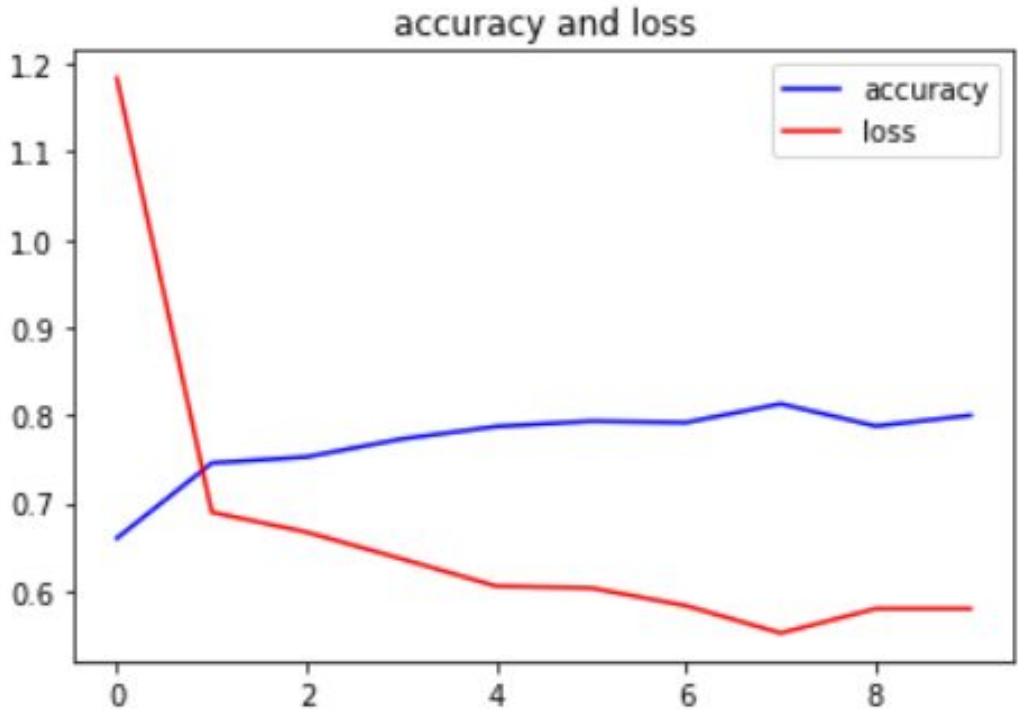
재구성한 ResNet50 모델 Architecture




```
1 #생성된 모델 정보 출력
2 model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50v2 (Functional)     (None, 7, 7, 2048)        23564800

 flatten (Flatten)           (None, 100352)            0

 dense (Dense)               (None, 256)               25690368

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 3)                 771

=================================================================
Total params: 49,255,939
Trainable params: 25,691,139
Non-trainable params: 23,564,800
_________________________________________________________________
```

ResNet101 모델 성능평가



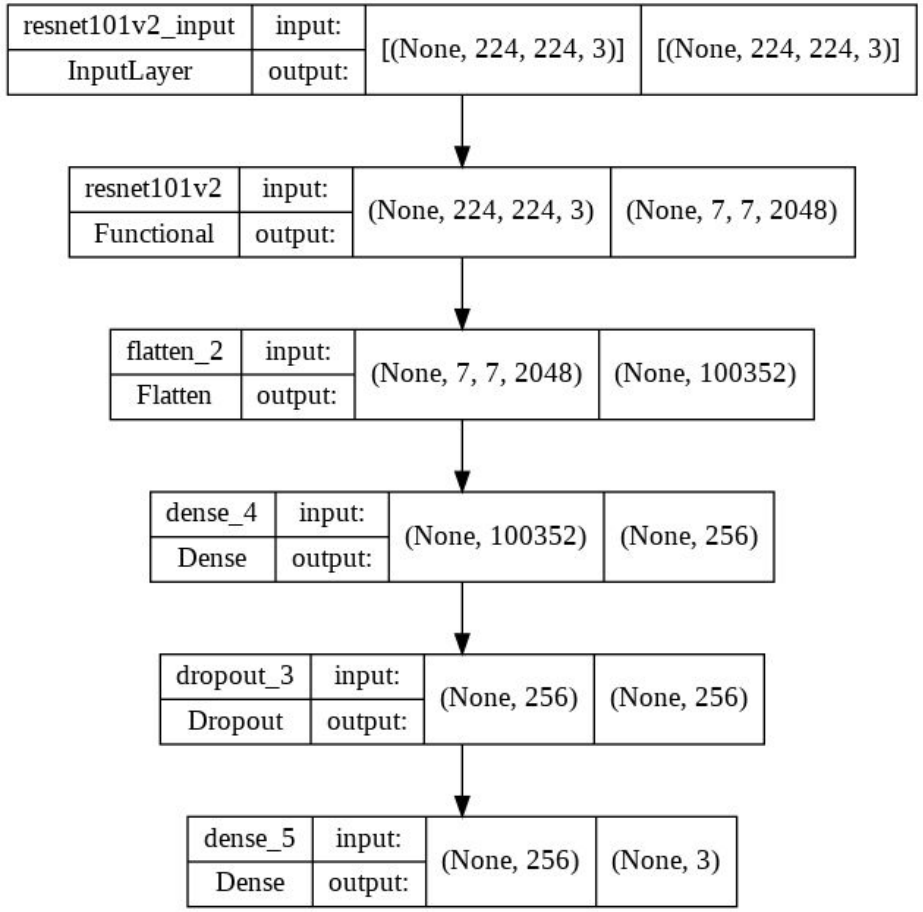

```
print('CNN Model time of : ', end1)
```

```
CNN Model time of :  1485.0741991996765
```

```
print('test_accuracy :', eval[1])
```

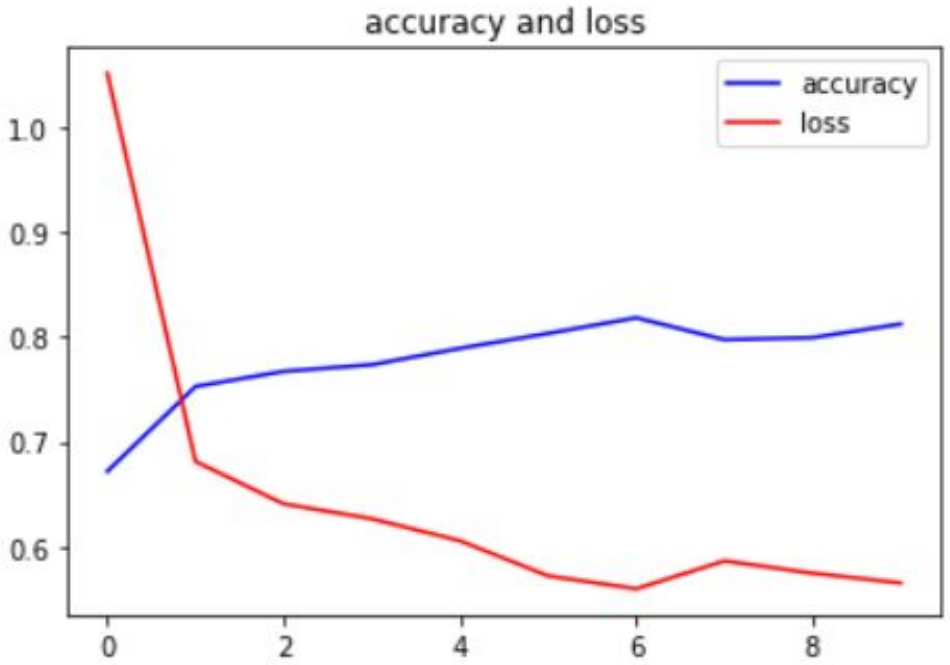
```
test_accuracy : 0.9766423106193542
```

재구성한 ResNet101 모델 Architecture




```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
resnet101v2 (Functional)     (None, 7, 7, 2048)        42626560

flatten_2 (Flatten)          (None, 100352)            0

dense_4 (Dense)              (None, 256)               25690368

dropout_3 (Dropout)          (None, 256)               0

dense_5 (Dense)              (None, 3)                 771

=================================================================
Total params: 68,317,699
Trainable params: 25,691,139
Non-trainable params: 42,626,560
_________________________________________________________________
```

ResNet152 모델 성능평가



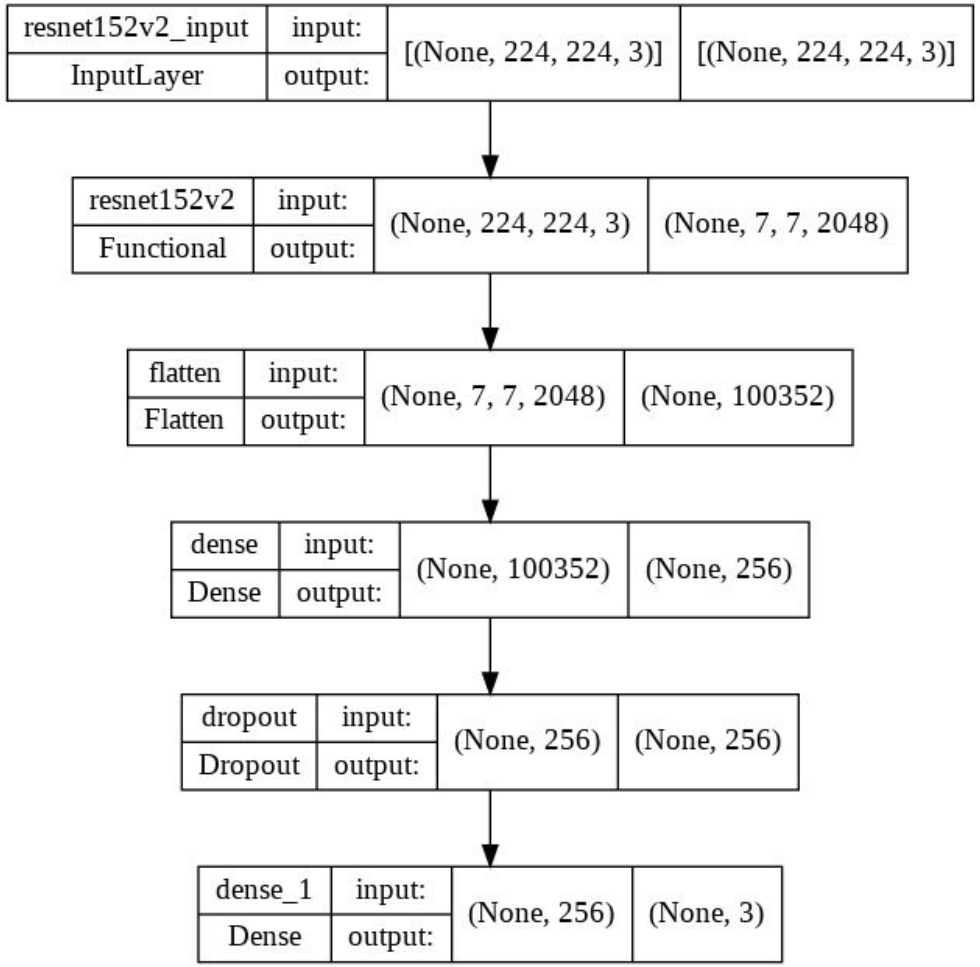

```
print('Resnet151V2 Model time of :', end1)
```

```
CNN Model time of : 1554.716659784317
```

```
print('test_accuracy :', eval[1])
```

```
test_accuracy : 0.9678832292556763
```

재구성한 ResNet152 모델 Architecture




```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet152v2 (Functional)    (None, 7, 7, 2048)        58331648

 flatten (Flatten)           (None, 100352)            0

 dense (Dense)               (None, 256)               25690368

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 3)                 771

=================================================================
Total params: 84,022,787
Trainable params: 25,691,139
Non-trainable params: 58,331,648
_________________________________________________________________
```

결론

데이터 증강의 방식에 따라 모델 성능에 차이가 있음

VGG 16	accuracy	loss
원본 데이터	0.8788	0.3487
데이터 증강	0.9186	0.2271
데이터 증강 + CutMix	0.9201	0.2062

모델에 따른 시간, 정확도 차이가 있음

Model	Time	evaluated accuracy	evaluated loss
VGGnet 16	1579.9311	0.9171	0.3253
VGGnet 19	1894.5123	0.9172	0.2301
Resnet 50 V2	1824.9995	0.9285	0.1993
Resnet 101 V2	**1485.0741**	**0.9766**	0.1266
Resnet 152 V2	1554.7166	0.9678	**0.1229**

감사합니다.