한국외국어대학교
HANKUK UNIVERSITY OF FOREIGN STUDIES

Machine Learning Final Report

VGG16 & SimpleCNN result comparison

2021-2 기계학습 Prof. 윤일동
202130131 컴퓨터공학과 이준민

Content

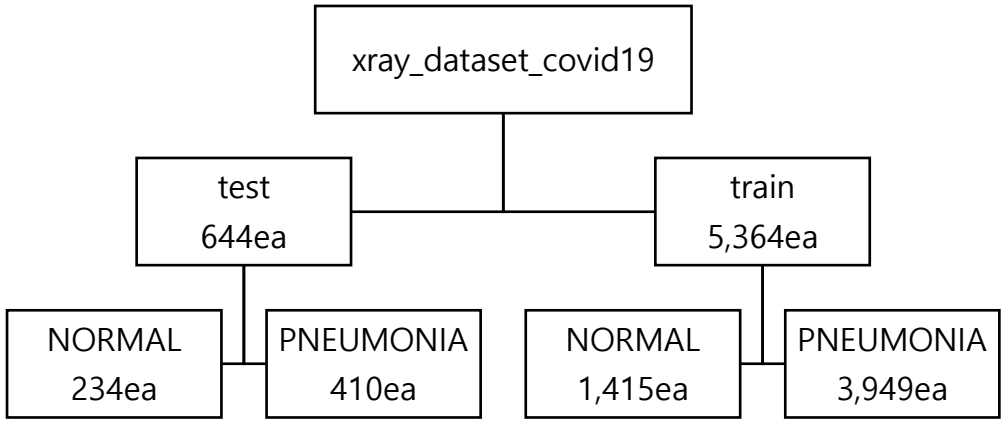
한국외국어대학교
HANKUK UNIVERSITY OF FOREIGN STUDIES

- Project Outline
- VGG16 and SimpleCNN result comparison
  - Data & Preprocessing
  - VGG16 Network Architecture
  - Run and Evaluate
  - Result Analysis
  - Simple CNN Network Architecture
  - Result Analysis
- Conclusion

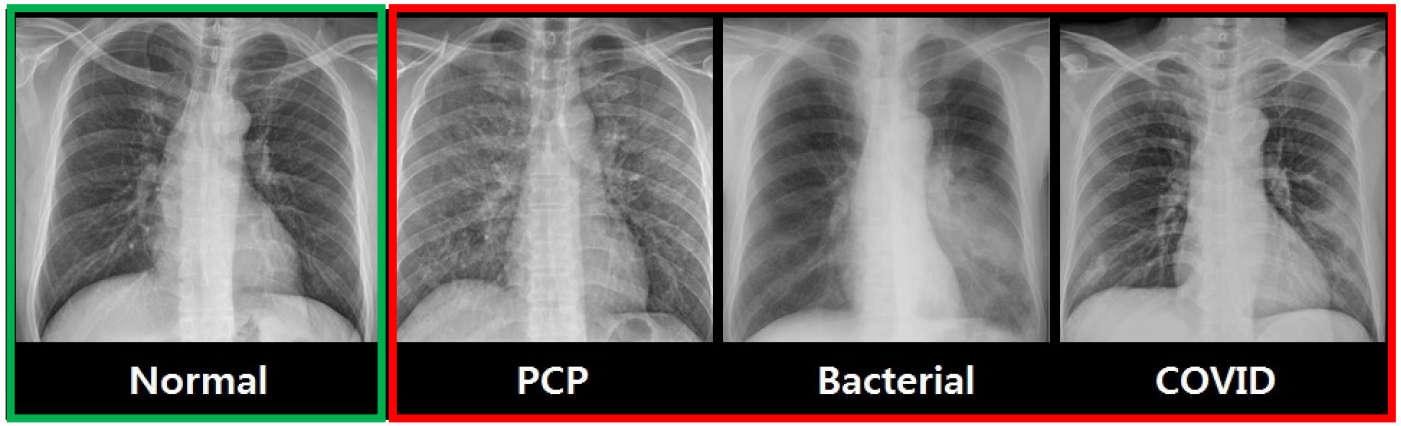
Project Outline

- VGGNet은 CNN 알고리즘 중, 이미지 분류용 알고리즘으로 2014년 이미지넷 이미지 인식 **대회에서 준우승**을 한 모델입니다.
- VGGNet 모델부터 시작해서 **네트워크가 깊어지기 시작했으며, 성능이 좋아졌음**을 다른 깊은 모델들을 통해서 증명하기도 하였습니다.
- 위와 같은 내용들을 보고 **깊은 네트워크** VGGNet을 사용해보고 성능을 관찰하고자, 이번 기말 프로젝트를 통해 VGG16 모델을 사용하여 **단순한 분류 데이터셋을 입력으로** 제공하고 학습 시켰을 때 보이는 결과 분석, 발견 된 문제점을 해결 하기 위한 연구를 진행하고자 합니다.
- 데이터셋은 현재 전세계적으로 기승을 부리고 있는 Covid-19 관련 이미지 데이터입니다. 이미지는 Kaggle에서 환자의 **폐렴 유/무 X-ray** 이미지를 다운로드하고 취합하였습니다.
- 다른 어려운 데이터셋 보다 단순한 데이터셋을 선택한 이유는 '**복잡한 네트워크에 단순한 입력을 넣었을 때의 결과**'를 확인하기 위함입니다.

Data



<표1> 실제 데이터 디텍토리 구조



<그림1> 왼쪽 부터 정상(초록박스), 폐포자충폐렴, 전형성 폐렴, 우한 폐렴(빨간박스) X-ray 이미지

- 데이터는 총 **6,008**개 - NORMAL, PNEUMONIA로 구분됩니다.
- NORMAL은 정상 폐의 이미지, PNEUMONIA는 폐렴을 가지고 있는 폐의 이미지 입니다.
- 그림1의 빨간색 박스 안에 있는 X-ray 이미지는 모두 PNEUMONIA로 분류하였습니다.

Data Preprocessing for VGG16

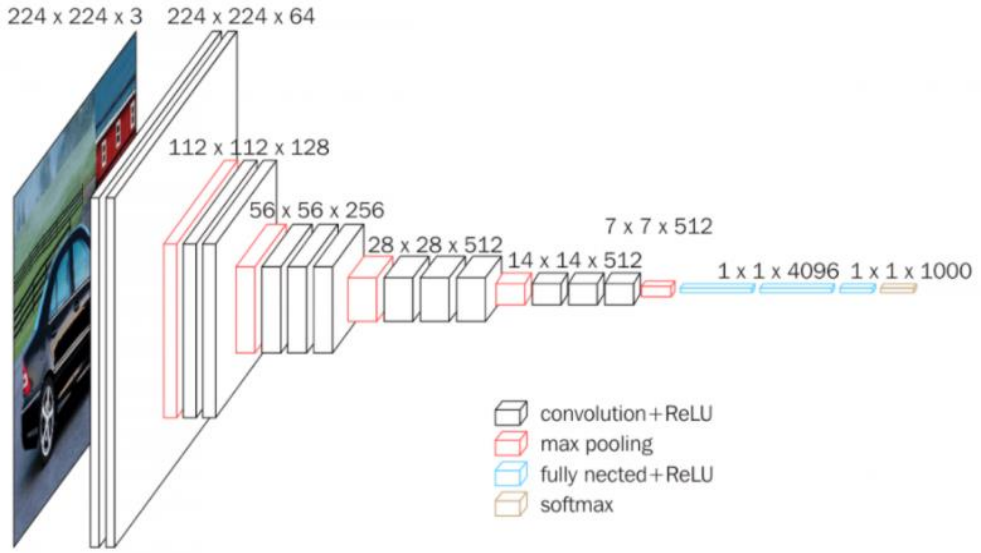
```python
from keras.preprocessing.image import ImageDataGenerator

new = ImageDataGenerator(rescale = 1.0/255.0,
                         horizontal_flip = True,
                         zoom_range = 0.2,
                         shear_range = 0.2,
                         width_shift_range = 0.01,
                         height_shift_range = 0.01)

train = new.flow_from_directory(train,
                                target_size = (224,224),
                                class_mode = 'binary',
                                color_mode = 'grayscale',
                                batch_size = 32)

valid = new.flow_from_directory(test,
                                target_size = (224,224),
                                class_mode = 'binary',
                                color_mode = 'grayscale',
                                batch_size = 32)
```

```
(array([[[[0.        ],
          [0.        ],
          [0.        ],
          ...,
          [0.        ],
          [0.00539174],
          [0.00477583]],

         [[0.        ],
          [0.        ],
          [0.        ],
          ...,
          [0.        ],
          [0.        ],
          [0.06392078]],

         [[0.        ],
          [0.        ],
          [0.        ],
          ...,
          [0.00214176],
          [0.0012387 ],
          [0.48455906]],
```

<코드1> 데이터 전처리 및 전처리 결과

- 다운로드 한 기존의 X-ray 이미지는 RGB로 구성되어있습니다.
- Keras ImageDataGenerator를 사용하여 이미지의 수평반전, 리스케일, 줌, 기울기등을 설정해주었습니다.
- flow_from_directory를 이용해 디렉토리에서 이미지를 하나씩 가져와서 각각 train과 valid 변수에 담아 주었습니다.
- 클래스 타입을 binary으로 바꿔 주었으며, 색상 또한 grayscale로 처리 하였습니다.

VGG16 Network Architecture

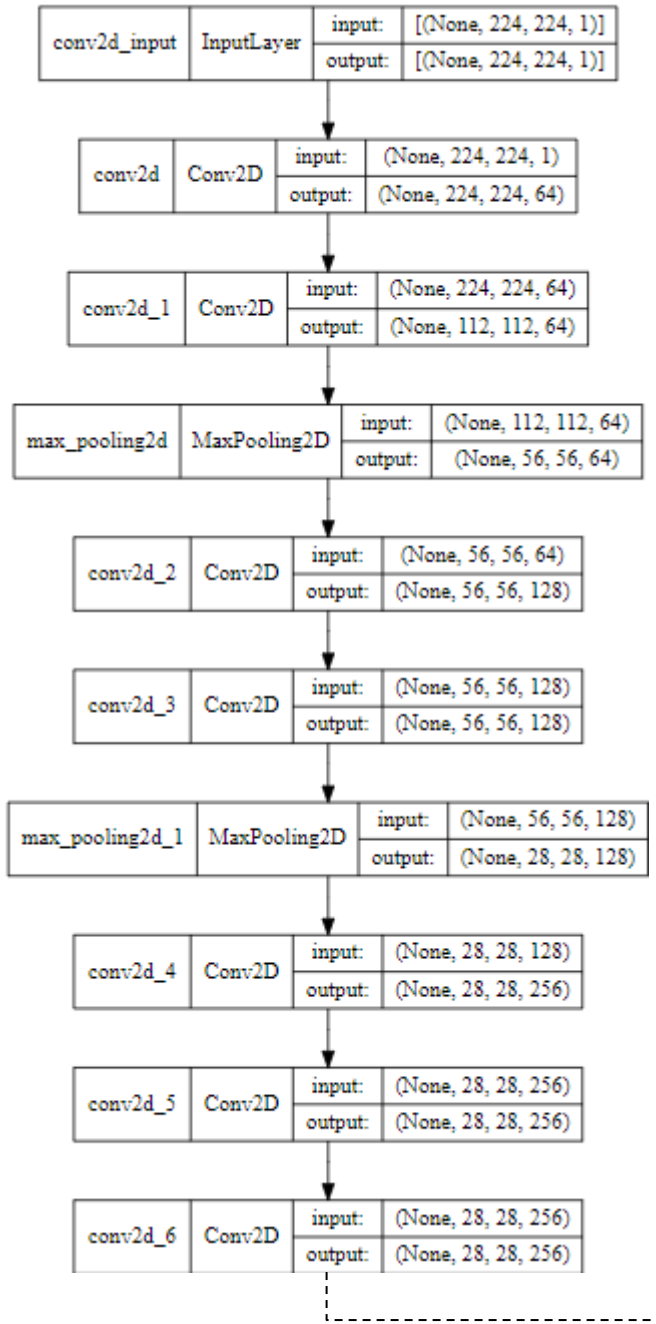


<그림2> 기존 VGG16 네트워크 구조

```python
# CNN model (VGG16)
model1 = Sequential()
model1.add(Conv2D(64, (3,3), kernel_initializer=w_initializer,activation=f_activation, input_shape=(224,224,1),
                  padding='same'))
model1.add(Conv2D(64,kernel_size=(3,3), strides=(2,2),kernel_initializer=w_initializer,activation=f_activation,
                  padding='same'))
model1.add(MaxPooling2D(strides=(2,2)))
model1.add(Conv2D(128,kernel_size=(3,3),kernel_initializer=w_initializer,activation=f_activation,padding='same'))
model1.add(Conv2D(128,kernel_size=(3,3),kernel_initializer=w_initializer,activation=f_activation,padding='same'))
model1.add(MaxPooling2D(strides=(2,2)))
model1.add(Conv2D(256,kernel_size=(3,3),kernel_initializer=w_initializer,activation=f_activation,padding='same'))
model1.add(Conv2D(256,kernel_size=(3,3),kernel_initializer=w_initializer,activation=f_activation,padding='same'))
model1.add(Conv2D(256,kernel_size=(3,3),kernel_initializer=w_initializer,activation=f_activation,padding='same'))
model1.add(MaxPooling2D(strides=(2,2)))
model1.add(Conv2D(512,kernel_size=(3,3),kernel_initializer=w_initializer,activation=f_activation,padding='same'))
model1.add(Conv2D(512,kernel_size=(3,3),kernel_initializer=w_initializer,activation=f_activation,padding='same'))
model1.add(Conv2D(512,kernel_size=(3,3),kernel_initializer=w_initializer,activation=f_activation,padding='same'))
model1.add(MaxPooling2D(strides=(2,2)))

model1.add(Flatten())
model1.add(Dense(4096,activation=f_activation))
model1.add(Dense(4096,activation=f_activation))
model1.add(Dense(1,activation='sigmoid'))
model1.compile(optimizer = SGD(.001),loss='binary_crossentropy',metrics=['accuracy'])

w_initializer = 'he_uniform'
f_activation = 'LeakyReLU'
```

<코드2> VGG16 모델 생성

- 기존 VGG16의 틀을 깨지 않고 Sequential 모델을 생성해줍니다. 전 페이지의 전처리 과정 때문에 발생한 차원 감소 때문에 input_shape의 3번째 인자 값을 변경해주었습니다.
- weight_initializer는 he_uniform, activation_function은 LeakyReLU, optimizer는 SGD, loss function은 binary_crossentropy 를 사용하였습니다.



<그림3> 작성한 VGG16 모델 네트워크 구조

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 224, 224, 64)	640
conv2d_1 (Conv2D)	(None, 112, 112, 64)	36928
max_pooling2d (MaxPooling2D)	(None, 56, 56, 64)	0
conv2d_2 (Conv2D)	(None, 56, 56, 128)	73856
conv2d_3 (Conv2D)	(None, 56, 56, 128)	147584
max_pooling2d_1 (MaxPooling2D)	(None, 28, 28, 128)	0
conv2d_4 (Conv2D)	(None, 28, 28, 256)	295168
conv2d_5 (Conv2D)	(None, 28, 28, 256)	590080
conv2d_6 (Conv2D)	(None, 28, 28, 256)	590080
max_pooling2d_2 (MaxPooling2D)	(None, 14, 14, 256)	0
conv2d_7 (Conv2D)	(None, 14, 14, 512)	1180160
conv2d_8 (Conv2D)	(None, 14, 14, 512)	2359808
conv2d_9 (Conv2D)	(None, 14, 14, 512)	2359808
max_pooling2d_3 (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
dense (Dense)	(None, 4096)	102764544
dense_1 (Dense)	(None, 4096)	16781312
dense_2 (Dense)	(None, 1)	4097

Total params: 127,184,065
Trainable params: 127,184,065
Non-trainable params: 0

한국외국어대학교
Hankuk University of Foreign Studies

Train and Evaluate

```
hist1 = model1.fit(train,validation_data=valid,epochs=10,batch_size=1)
```

```
Epoch 1/10
168/168 [==============================] - 48s 245ms/step - loss: 0.4946 - accuracy: 0.7936 - val_loss: 0.4182 - val_accuracy: 0.8261
Epoch 2/10
168/168 [==============================] - 35s 210ms/step - loss: 0.3020 - accuracy: 0.8730 - val_loss: 0.3391 - val_accuracy: 0.8618
Epoch 3/10
168/168 [==============================] - 35s 207ms/step - loss: 0.2515 - accuracy: 0.8956 - val_loss: 0.3382 - val_accuracy: 0.8587
Epoch 4/10
168/168 [==============================] - 35s 208ms/step - loss: 0.2311 - accuracy: 0.9021 - val_loss: 0.5062 - val_accuracy: 0.7811
Epoch 5/10
168/168 [==============================] - 35s 207ms/step - loss: 0.2132 - accuracy: 0.9135 - val_loss: 0.3517 - val_accuracy: 0.8447
Epoch 6/10
168/168 [==============================] - 35s 208ms/step - loss: 0.1922 - accuracy: 0.9228 - val_loss: 0.4308 - val_accuracy: 0.8199
Epoch 7/10
168/168 [==============================] - 35s 208ms/step - loss: 0.1902 - accuracy: 0.9211 - val_loss: 0.4618 - val_accuracy: 0.8183
Epoch 8/10
168/168 [==============================] - 35s 209ms/step - loss: 0.1772 - accuracy: 0.9323 - val_loss: 0.3347 - val_accuracy: 0.8587
Epoch 9/10
168/168 [==============================] - 35s 207ms/step - loss: 0.1650 - accuracy: 0.9323 - val_loss: 0.3836 - val_accuracy: 0.8385
Epoch 10/10
168/168 [==============================] - 35s 207ms/step - loss: 0.1687 - accuracy: 0.9342 - val_loss: 0.2995 - val_accuracy: 0.8820
```

```
evaluate1 = model1.evaluate(valid)
print(evaluate1[1])
```

```
21/21 [==============================] - 6s 174ms/step - loss: 0.6832 - accuracy: 0.6366
0.6366459727287292
```

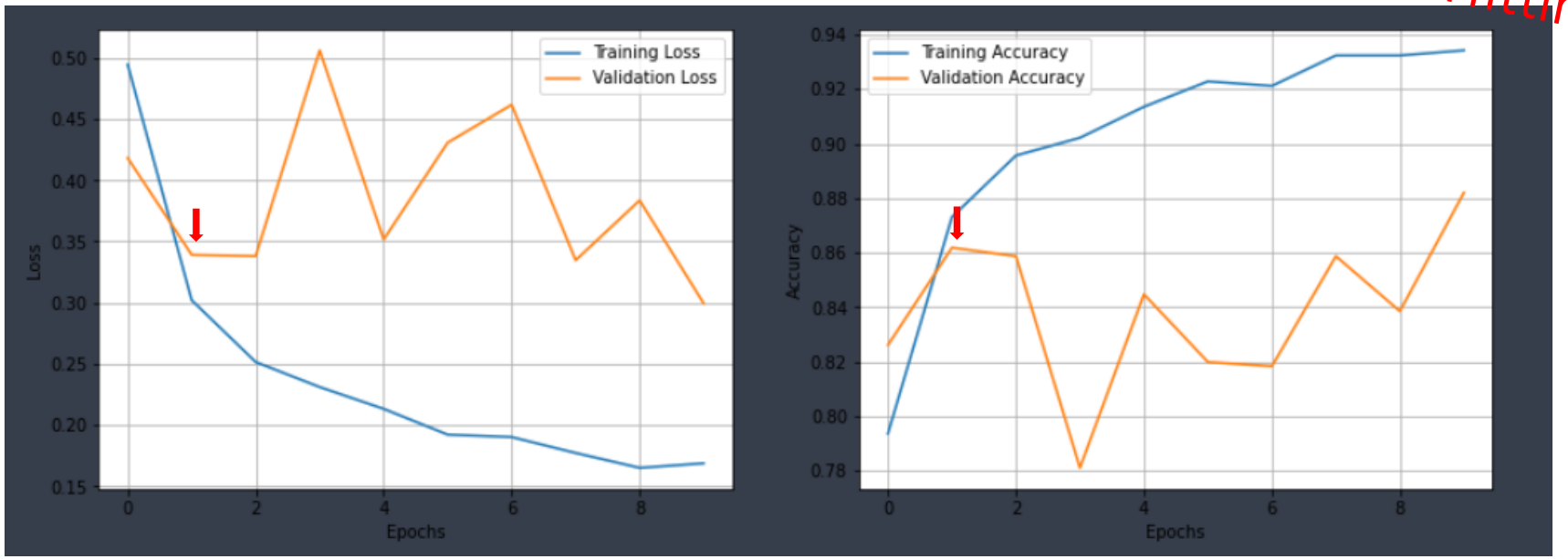
<코드3> VGG16 모델 Train and evaluate

학습 환경
운영체제 : windows10 **CPU** : i7-9700K @ 3.60GHz **RAM** : 32 GB **GPU** : NVDIA GeForce RTX 2080 **개발환경** : Jupyter Notebook **개발언어** : Python 3.8 **개발도구** : Tensorflow 2.x, Keras, Matplot, IPython, numpy
총 10번의 epoch, batch_size=1로 설정, Train and Validation을 진행 하였습니다. 이어서 Evaluate를 진행하여 모델에 대한 최종적인 loss와, accuracy를 얻었습니다. 학습시간 : 약 **5분**, **GPU 사용**

<표2>학습 환경 및 학습 변수 설정 정보

Result Analysis

Overfitting



<그림4> 모델 Training and Validation Loss, Accuracy 결과 그래프

- 층이 깊은 모델에 단순한 데이터셋을 입력으로 넣어 훈련을 시키고 평가를 한 결과, Training은 epoch가 수가 늘어남에 따라 천천히 Loss가 감소하는 것을 확인할 수 있었다. 하지만 Validation의 Loss가 epoch 2(빨간화살표) 부터 온전하지 못한 것으로 보아, "**과적합**"이라고 생각 된다.
- 본 문제의 해결 방안으로는 모델을 단순하게 만들거나, 복잡한 데이터를 입력, 드롭아웃이라고 생각 되었습니다. 따라서 **모델을 단순하게 만들어서 학습**을 다시 시키는 방향으로 진행하였습니다.

Data Preprocessing for Simple CNN

```
image_data = []
labels = []

label_dict = {
    'PNEUMONIA' : 0,
    'NORMAL' : 1
}

from keras.preprocessing import image

for i in folders :
    path = os.path.join(trainDir,i)
    for j in os.listdir(path):
        img = image.load_img(os.path.join(path,j),target_size=(224,224))
        img_array = image.img_to_array(img)
        image_data.append(img_array)
        labels.append(label_dict[i])
```

```
['NORMAL', 'PNEUMONIA']
```

```
print(len(image_data),len(labels))
```

```
5364 5364
```

```
import random
combined = list(zip(image_data,labels))
random.shuffle(combined)
image_data[:],labels[:]=zip(*combined)
```

```
print(labels)
```

```
[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1,
0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0,
0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0,
0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0,
1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0]
```

```
from keras.utils import np_utils

y_train = np_utils.to_categorical(y_train)
print(x_train.shape,y_train.shape)
```

```
(5364, 224, 224, 3) (5364, 2)
```

```
from keras.preprocessing.image import ImageDataGenerator
augment = ImageDataGenerator(
                             rotation_range=20,
                             width_shift_range=0.01,
                             height_shift_range=0.01,
                             horizontal_flip=False,
                             vertical_flip=False,
                            )
augment.fit(x_train)
```

<코드4> 데이터 전처리 및 전처리 결과

- VGG16 모델에 입력했을 때와는 다른 방식으로 데이터를 전처리 해보았습니다. 흑백 이미지로 넣는 것 보다 각 이미지를 numpy로 수치화 해주고 라벨과 combine 해주게 된다면 좀 더 정확하게 모델이 데이터셋을 읽고 처리할 수 있을 거라고 생각하였습니다.
- ImageDataGenerator를 사용하여 전처리를 진행하였지만, 색상에 **grayscale** 값을 주지 않았습니다.

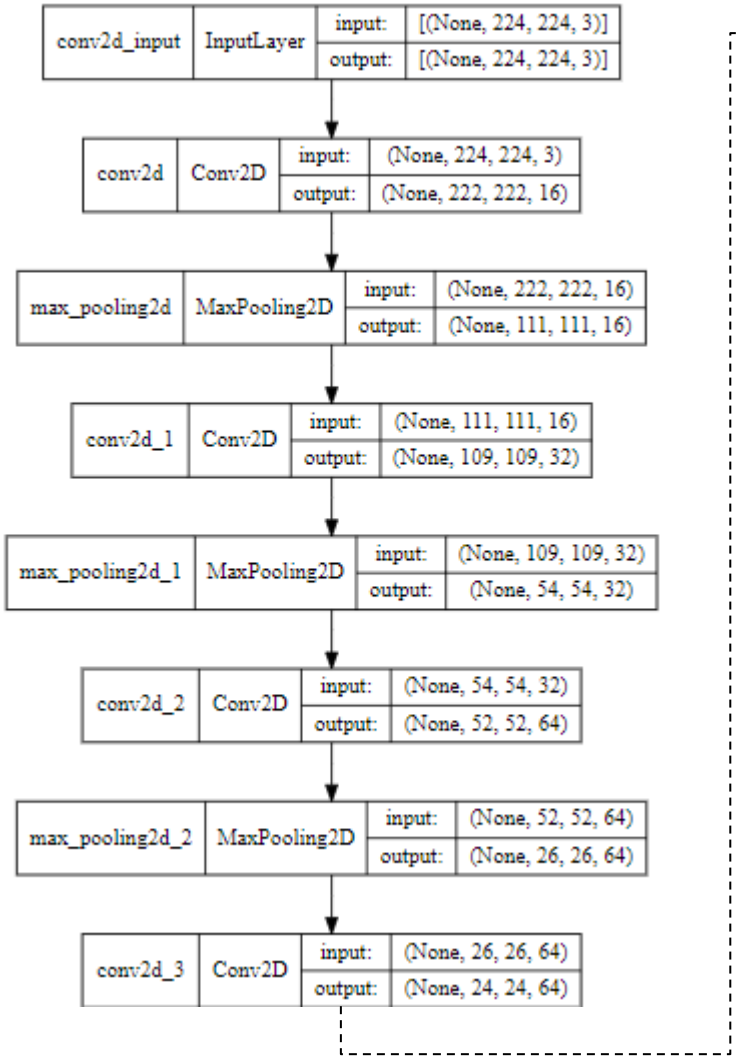
Simple CNN Network Architecture

한국외국어대학교
HANKUK UNIVERSITY OF FOREIGN STUDIES

```python
# Simple CNN model
model = Sequential()
model.add(Conv2D(filters=16,kernel_size=(3,3),input_shape=(224,224,3),activation='relu',kernel_regularizer=l2(0.01
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Conv2D(filters=32,kernel_size=(3,3),input_shape=(224,224,3),activation='relu',kernel_regularizer=l2(0.01
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Conv2D(filters=64,kernel_size=(3,3),input_shape=(224,224,3),activation='relu',kernel_regularizer=l2(0.01
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Conv2D(filters=64,kernel_size=(3,3),input_shape=(224,224,3),activation='relu',kernel_regularizer=l2(0.01
model.add(MaxPool2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(224))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(2))
model.add(Activation('sigmoid'))

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

<코드5> Simple CNN 모델 생성

- 기존 VGG16과 비교했을 때 더 적은 층을 가지고 있는 Sequential CNN 모델을 생성해줍니다.
- activation_function은 ReLU, optimizer는 adam, loss function은 binary_crossentropy 를 사용하였습니다.
- 위의 모델과 다르게 Dropout을 추가 함으로써 Overffiting을 발생을 최소화 시켰습니다.



<그림5> 작성한 Simple CNN 모델 네트워크 구조

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 222, 222, 16)	448
max_pooling2d_4 (MaxPooling 2D)	(None, 111, 111, 16)	0
conv2d_5 (Conv2D)	(None, 109, 109, 32)	4640
max_pooling2d_5 (MaxPooling 2D)	(None, 54, 54, 32)	0
conv2d_6 (Conv2D)	(None, 52, 52, 64)	18496
max_pooling2d_6 (MaxPooling 2D)	(None, 26, 26, 64)	0
conv2d_7 (Conv2D)	(None, 24, 24, 64)	36928
max_pooling2d_7 (MaxPooling 2D)	(None, 12, 12, 64)	0
flatten_1 (Flatten)	(None, 9216)	0
dense_2 (Dense)	(None, 224)	2064608
activation_2 (Activation)	(None, 224)	0
dropout_1 (Dropout)	(None, 224)	0
dense_3 (Dense)	(None, 2)	450
activation_3 (Activation)	(None, 2)	0

Total params: 2,125,570
Trainable params: 2,125,570
Non-trainable params: 0

한국외국어대학교
Hankuk University of Foreign Studies

Train and Evaluate

```
hist = model.fit(x_train,y_train,batch_size=32, epochs = 25, validation_split = 0.10,callbacks=callbacks_list)
```

```
151/151 [==============================] - ETA: 0s - loss: 0.2001 - accuracy: 0.9822
Epoch 00020: val_loss improved from 0.28975 to 0.25959, saving model to covid_detection.hdf5
151/151 [==============================] - 3s 22ms/step - loss: 0.2001 - accuracy: 0.9822 - val_loss: 0.2596 - val_accuracy: 0.9702
Epoch 21/25
151/151 [==============================] - ETA: 0s - loss: 0.1807 - accuracy: 0.9847
Epoch 00021: val_loss did not improve from 0.25959
151/151 [==============================] - 3s 21ms/step - loss: 0.1807 - accuracy: 0.9847 - val_loss: 0.3218 - val_accuracy: 0.9460
Epoch 22/25
151/151 [==============================] - ETA: 0s - loss: 0.1727 - accuracy: 0.9855
Epoch 00022: val_loss did not improve from 0.25959
151/151 [==============================] - 3s 21ms/step - loss: 0.1727 - accuracy: 0.9855 - val_loss: 0.2609 - val_accuracy: 0.9590
Epoch 23/25
151/151 [==============================] - ETA: 0s - loss: 0.1553 - accuracy: 0.9861
Epoch 00023: val_loss improved from 0.25959 to 0.22929, saving model to covid_detection.hdf5
151/151 [==============================] - 3s 22ms/step - loss: 0.1553 - accuracy: 0.9861 - val_loss: 0.2293 - val_accuracy: 0.9665
Epoch 24/25
151/151 [==============================] - ETA: 0s - loss: 0.1393 - accuracy: 0.9898
Epoch 00024: val_loss did not improve from 0.22929
151/151 [==============================] - 3s 21ms/step - loss: 0.1393 - accuracy: 0.9898 - val_loss: 0.2655 - val_accuracy: 0.9628
Epoch 25/25
151/151 [==============================] - ETA: 0s - loss: 0.1342 - accuracy: 0.9880
Epoch 00025: val_loss improved from 0.22929 to 0.22032, saving model to covid_detection.hdf5
151/151 [==============================] - 3s 22ms/step - loss: 0.1342 - accuracy: 0.9880 - val_loss: 0.2203 - val_accuracy: 0.9683
```

```
evaluate1 = model.evaluate(x_test,y_test)
print(evaluate1[1])
```

```
21/21 [==============================] - 0s 10ms/step - loss: 1.3036 - accuracy: 0.8168
0.8167701959609985
```

<코드6> Simple CNN 모델 Train and evaluate

학습 환경
운영체제 : windows10 **CPU** : i7-9700K @ 3.60GHz **RAM** : 32 GB **GPU** : NVDIA GeForce RTX 2080 **개발환경** : Jupyter Notebook **개발언어** : Python 3.8 **개발도구** : Tensorflow 2.x, Keras, Matplot, IPython, numpy
총 25번의 epoch, batch_size=32로 설정, Train and Validation을 진행 하였습니다. 이어서 Evaluate를 진행하여 모델에 대한 최종적인 loss와, accuracy를 얻었습니다. 학습시간 : 약 **1분**, **GPU 사용**

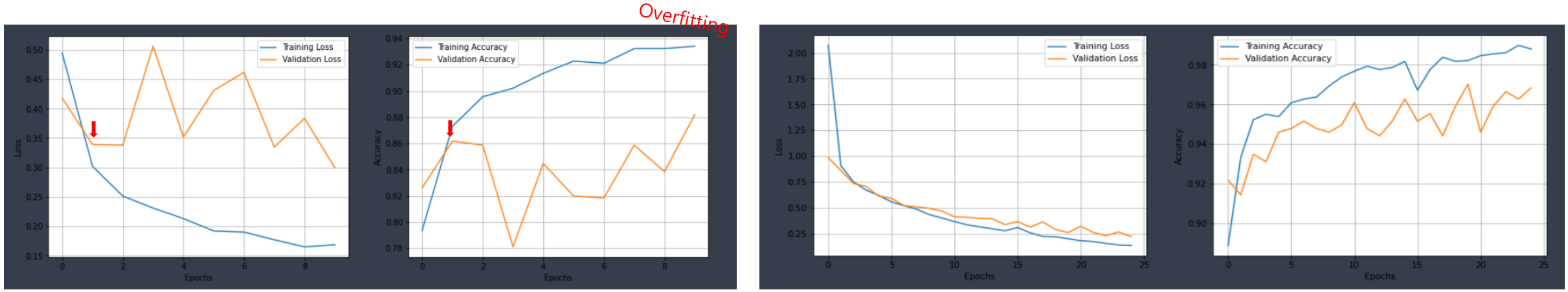
<표2>학습 환경 및 학습 변수 설정 정보

Result Analysis

<그림6> 모델 Training and Validation
Loss, Accuracy 결과 그래프

- Simple CNN은 Train과 Validation이 잘 되었다고 볼 수 있는 그래프를 보여줍니다. 약 epoch 2 지점부터 Loss와 Accuracy의 값이 점차 수렴이 되어가는 모습을 확인 할 수 있었습니다.
- 모델을 간단하게 만들고, Dropout을 사용하여 깊은 층을 가지고 있는 VGG16 모델에서 보여졌던 과적합 현상은 사라진 것으로 생각됩니다.

Conclusion



<그림7> VGG16 모델(좌), Simple CNN 모델(우) Training and Validation Loss, Accuracy 결과 그래프

- 본 실험을 통해 깊은 층을 사용하는 모델은 간단하거나 단순한 데이터셋에 대해서는 높은 정확도를 보여주지 않는다는 결론입니다.
- 과적합 발생 가능성이 있기 때문에 단순한 데이터셋을 사용하는 경우에는 모델의 사이즈, Dropout 과 같은 기법들을 사용해주어야 한다는 결과를 얻었습니다.
- 하지만 데이터가 "단순하다"라는 의미를 어떻게 정의하냐에 따라 다른 결과가 나올 것으로 생각됩니다. 예) 치와와와 머핀(전체적으로 복잡), 푸들과 치킨(전체적인 복잡), X-ray(세부적인 복잡)

Reference

https://www.kaggle.com/tolgadincer/labeled-chest-xray-images
https://www.kaggle.com/khoongweihao/covid19-xray-dataset-train-test-sets
https://bskyvision.com/504
https://www.tensorflow.org/tutorials/images/cnn?hl=ko
https://www.tensorflow.org/tutorials/images/classification?hl=ko
https://keras.io/api/models/model_training_apis/

모든 자료는 github에 업로드 하였습니다.