Pneumonia Diagnosis app

TheDarkKinght-21th

okwjm0107@gmail.com

<Table of Content>

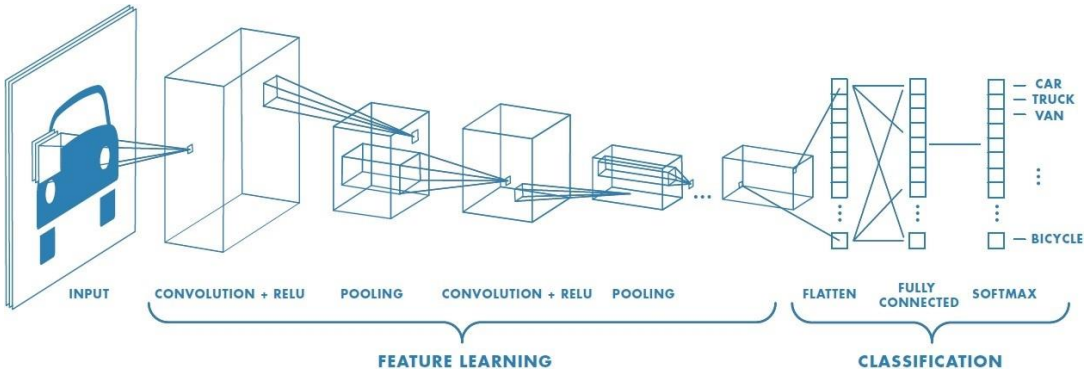
<Table of Content>

1. Introduction

이번 프로젝트의 목표는 x-ray 폐 사진들을 학습 데이터로 설정하여 CNN 모델을 만들고 그 모델로 가지고 학습 데이터로 학습하여 폐렴인지 정상인지 진단을 내리는 모델을 만드는 것이다.
데이터는 총 3 개의 폴더(Train, test, val)로 구성되며, 각 이미지 범주(Pneumonia/Normal)에 대한 하위 폴더가 포함된다. 5,863 개의 X 선 영상(JPEG)과 2 개의 범주(폐렴/정상)가 있다. 각각 train 은 5216 개(정상 1341 개, 폐렴 3875 개), val 은 16 개(정상, 폐렴 각각 8 개) 그리고 test 는 624 개(정상 234 개, 폐렴 390 개)이다.
흉부 X 선 영상(앞-뒤)은 광저우에 거주하는 여성 및 어린이 의료 센터의 1~5 세 소아 환자의 회고적 코호트에서 선택되었다. 모든 흉부 X 선 촬영은 환자의 일상적인 임상 치료의 일환으로 수행되었다.
흉부 X 선 영상 분석을 위해 모든 흉부 방사선 사진은 처음에 저 품질 또는 판독 불가능한 스캔을 모두 제거하여 품질 관리를 위해 선별되었다.

2. Background knowledge

2.1 CNN



"Convolutional Neural Networks"의 약자로 딥러닝에서 주로 이미지나 영상 데이터를 처리할 때 쓰이며 이름에서 알 수 있다시피 Convolution 이라는 전처리 작업이 들어가는 Neural Network 모델이다.

2.2 Necessity of CNN

일반 DNN(Deep Neural Network)의 문제점에서부터 출발한다. 일반 DNN 은 기본적으로 1 차원 형태의 데이터를 사용한다. 때문에 (예를들면 1028x1028 같은 2 차원 형태의)이미지가 입력값이 되는 경우, 이것을 flatten 시켜서 한 줄 데이터로 만들어야 하는데 이 과정에서 이미지의 공간적/지역적 정보(spatial/topological information)가 손실되게 된다. 또한 추상화과정 없이 바로 연산과정으로 넘어가 버리기 때문에 학습시간과 능률의 효율성이 저하됩니다.

이러한 문제점에서부터 고안한 해결책이 CNN 입니다. CNN 은 이미지를 날것(raw input) 그대로 받음으로써 공간적/지역적 정보를 유지한 채 특성(feature)들의 계층을 만든다. CNN 의 중요 포인트는 이미지 전체보다는 부분을 보는 것, 그리고 이미지의 한 픽셀과 주변 픽셀들의 연관성을 살리는 것이다.

2.3 ImageDataGenerator (keras)

데이터 분석에서 데이터 증강(“data argument”)은 기존 데이터의 약간 수정된 복사본이나 기존 데이터에서 새로 생성된 합성 데이터를 추가하여 데이터 양을 증가시키는 데 사용되는 기술이다. 그것은 정규화 역할을 하며 머신 러닝 모델을 훈련할 때 과적합을 줄이는 데 도움이 된다. 이것은 데이터 분석의 오버샘플링과 밀접한 관련이 있다. keras 는 이러한 이미지 전처리 라이브러리를 제공한다. 바로 “ImageDataGenerator”이다.

“CNN”은 영상의 2 차원 변환인 회전(Rotation), 크기(Scale), 밀림(Shearing), 반사(Reflection), 이동(Translation)과 같은 2 차원 변환인 Affine Transform 에 취약하여 변환된 사진 또는 영상을 다른 자료로 인식한다.

그것에 추가로 "ImageDataGenerator"는 Noise 삽입, 색상, 밝기 변형 등을 활용하여 실시간으로 데이터 증강을 사용하고, 텐서 이미지 데이터 배치를 생성하여 데이터 학습의 정확도를 높이는 역할을 한다. 본 프로젝트에서는 위 라이브러리를 사용할 것이다.

3. Source Code

3.1 Development environment

프로젝트의 파일 확장자는 ".ipynb"이고 개발환경은 "Google Colab pro"이다. 이 환경에서의 스팩은 다음과 같다

.

CPU : Intel(R) Xeon(R) CPU @ 2.20GHz (single core)
GPU : Nvidia tesla t4 (vram :16GB)
Ram(memory) size : 32GB

3.2 CNN Model

3.2.1 Libraries

```
# Import Libraries
import numpy as np # linear algebra
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import cv2 # OpenCV

from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report,confusion_matrix
import tensorflow as tf
import keras
from keras.models import Sequential
from keras.layers import Dense, Conv2D , MaxPool2D , Flatten , Dropout , BatchNormalization
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ReduceLROnPlateau
import os
from tqdm import tqdm
```

3.2.2 Load the data

```
LABELS = ('PNEUMONIA','NORMAL')
IMG_SIZE=160
def get_data(data_dir):
    data = []
    for label in LABELS:
        path = os.path.join(data_dir, label)
        class_num = LABELS.index(label)
        for img in tqdm(os.listdir(path)):
            try:
                img_arr = cv2.imread(os.path.join(path, img),
cv2.IMREAD_GRAYSCALE)
                resized_arr = cv2.resize(img_arr, (IMG_SIZE,
IMG_SIZE)) # Reshaping Images
                data.append([resized_arr, class_num])
            except Exception as e:
                print(e)
    return np.array(data)
train_raw =
get_data('/content/drive/MyDrive/long/chest_xray/train')
test_raw =
get_data('/content/drive/MyDrive/long/chest_xray/test')
val_raw =
get_data('/content/drive/MyDrive/long/chest_xray/val')
```

이미지 크기는 (160,160)이다. 데이터를 가져오는 방식은 여라가지가 있다. 하지만 나는 데이터의 경로로부터 사진 pixel 정보를 하나씩 가져오는 방법을 선택했다. (이때 사진 데이터만 저장하는 것이 아니라 class 까지 **같이** 저장한다.)

그 이유는 모델 학습 시간을 절약하기 위해서이다. Keras 에서 제공하는 "ImageDataGenerator" 모듈의 **"flow_from_dirctory()"**함수가 있다. 이는 사진 파일이 담긴 폴더 디렉토리만 설정해주면 사진을 가져오는 것과 프로그래머가 설정한 포맷의 데이터 증강을 함수가 한 번에 처리해준다. 이 함수의 단점은 데이터를 위와 같이 하나하나 가져올 때 보다 데이터 학습 속도가 많이 느리다는 점이다.

3.2.3 Split the data

```
#Splitting data
X_train = []
y_train = []

X_val = []
y_val = []

X_test = []
y_test = []

for image, label in train_raw:
    X_train.append(image)
    y_train.append(label)

for image, label in val_raw:
    X_val.append(image)
    y_val.append(label)

for image, label in test_raw:
    X_test.append(image)
    y_test.append(label)
```

3.2.1 에서 받은 데이터를 데이터와 class 로 쪼개어 주는 코드이다.
이미지 데이터와 클래스 데이터를 각각 리스트에 넣어준다.

3.2.4 Normalize the data

```
# Normalize the data
X_train = np.array(X_train) / 255
X_val = np.array(X_val) / 255
X_test = np.array(X_test) / 255
```

쪼갰던 데이터를 각각을 RGB 최대값인 255 로 나누어 정규화를 한다.

3.2.5 Resize the data

```
# Resize data for deep learning
X_train = X_train.reshape(-1, IMG_SIZE, IMG_SIZE, 1)
y_train = np.array(y_train)

X_val = X_val.reshape(-1, IMG_SIZE, IMG_SIZE, 1)
y_val = np.array(y_val)

X_test = X_test.reshape(-1, IMG_SIZE, IMG_SIZE, 1)
y_test = np.array(y_test)
```

모델의 학습을 위해 입력 값의 사이즈를 재정의 한다.

3.2.6 Data augmentation

```
datagen = ImageDataGenerator(
       featurewise_center=False,  # set input mean to 0 over the
dataset
       samplewise_center=False,  # set each sample mean to 0
       featurewise_std_normalization=False,  # divide inputs by
std of the dataset
       samplewise_std_normalization=False,  # divide each input
by its std
       zca_whitening=False,
       rotation_range = 30,  # randomly rotate images in the
range (degrees, 0 to 180)
       zoom_range = 0.2, # Randomly zoom image
       width_shift_range=0.1,  # randomly shift images
horizontally (fraction of total width)
       height_shift_range=0.1,  # randomly shift images
vertically (fraction of total height)
       horizontal_flip = True,  # randomly flip images
       vertical_flip=False)  # randomly flip images

datagen.fit(X_train)
```

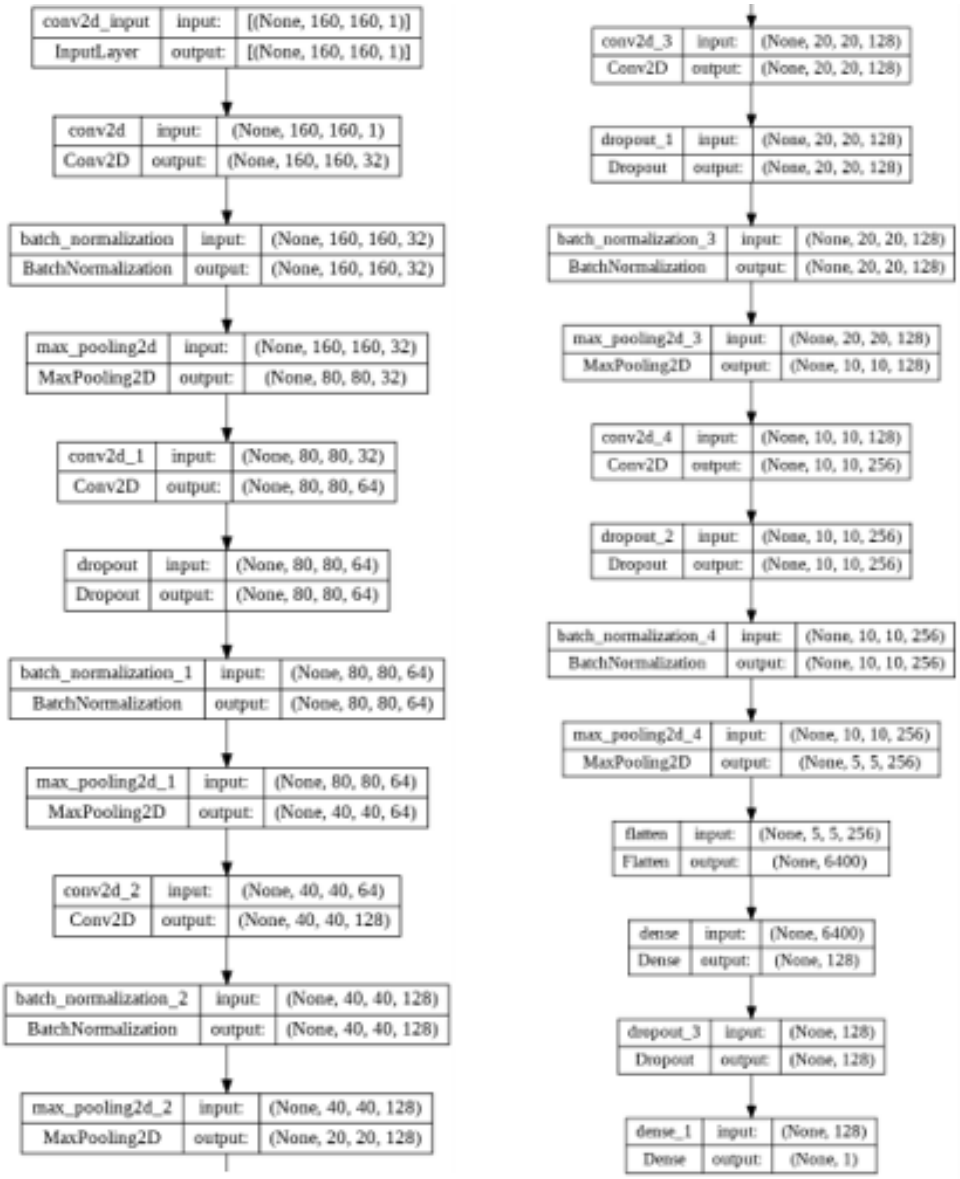
모든 argument 를 다 사용하는 것이 아닌 몇몇의 필요한 argument 만 사용했다.

3.2.7 Model of CNN and neural network Structure

```
np.random.seed = 18
tf.random.set_seed(0)
NUM_EPOCHS = 18

model = tf.keras.models.Sequential()
model.add(Conv2D(32 , (3,3) , strides = 1 , padding = 'same' ,
activation = 'relu' , input_shape = (160,160,1)))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2) , strides = 2 , padding = 'same'))

model.add(Conv2D(64 , (3,3) , strides = 1 , padding = 'same' ,
activation = 'relu'))
model.add(Dropout(0.1))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2) , strides = 2 , padding = 'same'))

model.add(Conv2D(128 , (3,3) , strides = 1 , padding = 'same' ,
activation = 'relu'))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2) , strides = 2 , padding = 'same'))

model.add(Conv2D(128 , (3,3) , strides = 1 , padding = 'same' ,
activation = 'relu'))
model.add(Dropout(0.2))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2) , strides = 2 , padding = 'same'))
```

```
model.add(Conv2D(256 , (3,3) , strides = 1 , padding = 'same' ,
activation = 'relu'))
model.add(Dropout(0.2))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2) , strides = 2 , padding = 'same'))
model.add(Flatten())



model.add(Dense(units = 128 , activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(units = 1 , activation = 'sigmoid'))

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

<model structure-1>



<model structure-2>

```
=================================================================
Total params: 1,357,313
Trainable params: 1,356,097
Non-trainable params: 1,216
_________________________________________________________________
```

모델 설명

이 Neural Network 모델은 총 6 개 layer 를 가진다. 5 개의 레이어는 CNN layer 이고 나머지 1 개는 DNN 이다.
각각의 CNN 레이어는 합성곱 연산(convolution)과 MaxPooling 을 했고 과대 적합을 방지하기 위해 "Dropout"과 " BatchNormalization"을 했다. 그 후, 데이터를 flatten 했고 128 개의 노드로 neural network 설정했다.
마지막으로 비용함수는 "binary cross entropy"함수이고 최적화 방식은 "adam"을 선택했다.

3.2.8 Model fit

```
#%%
learning_rate_reduction =
ReduceLROnPlateau(monitor='val_accuracy',
                                        patience = 2,
                                        verbose=1,
                                        factor=0.3,
                                        min_lr=0.000001)

history = model.fit(datagen.flow(X_train, y_train, batch_size =
32),
                    epochs = NUM_EPOCHS,
                    validation_data = datagen.flow(X_val, y_val),
                    callbacks = [learning_rate_reduction]
                   )
```

(Training 마지막 5 개 epoch 출력 화면)

```
Epoch 14/18
163/163 [==============================] - 11s 67ms/step - loss: 0.1000 - accuracy: 0.9684 - val_loss: 0.6014 - val_accuracy: 0.6875 - lr: 8.1000e-06
Epoch 15/18
163/163 [==============================] - ETA: 0s - loss: 0.0762 - accuracy: 0.9720
Epoch 15: ReduceLROnPlateau reducing learning rate to 2.4299999495269736e-06.
163/163 [==============================] - 11s 67ms/step - loss: 0.0762 - accuracy: 0.9720 - val_loss: 0.8844 - val_accuracy: 0.6250 - lr: 8.1000e-06
Epoch 16/18
163/163 [==============================] - 11s 68ms/step - loss: 0.0824 - accuracy: 0.9693 - val_loss: 0.8878 - val_accuracy: 0.6250 - lr: 2.4300e-06
Epoch 17/18
163/163 [==============================] - ETA: 0s - loss: 0.0806 - accuracy: 0.9737
Epoch 17: ReduceLROnPlateau reducing learning rate to 1e-06.
163/163 [==============================] - 11s 67ms/step - loss: 0.0806 - accuracy: 0.9737 - val_loss: 0.6107 - val_accuracy: 0.6250 - lr: 2.4300e-06
Epoch 18/18
163/163 [==============================] - 11s 67ms/step - loss: 0.0813 - accuracy: 0.9724 - val_loss: 0.7300 - val_accuracy: 0.6250 - lr: 1.0000e-06
```

3.2.9 Test dataset loss and accuracy

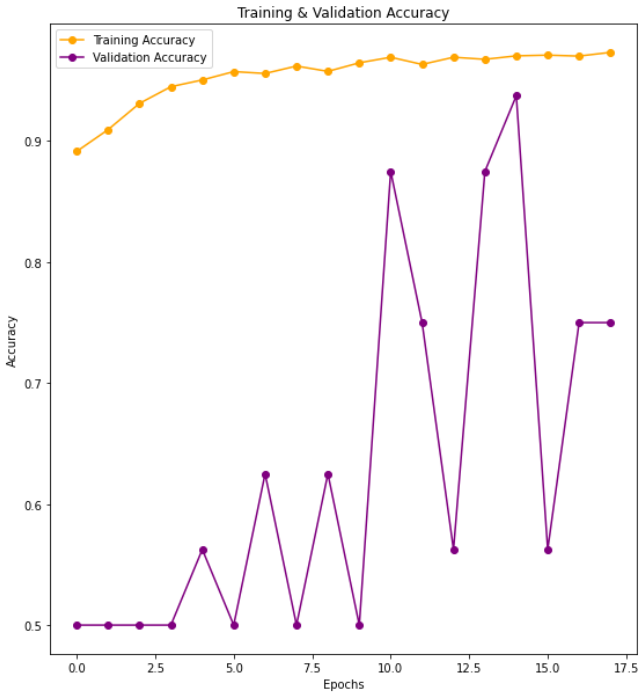
```
print("Loss of the model is - " ,
model.evaluate(X_test,y_test)[0])
print("Accuracy of the model is - " ,
model.evaluate(X_test,y_test)[1]*100 , "%")
```

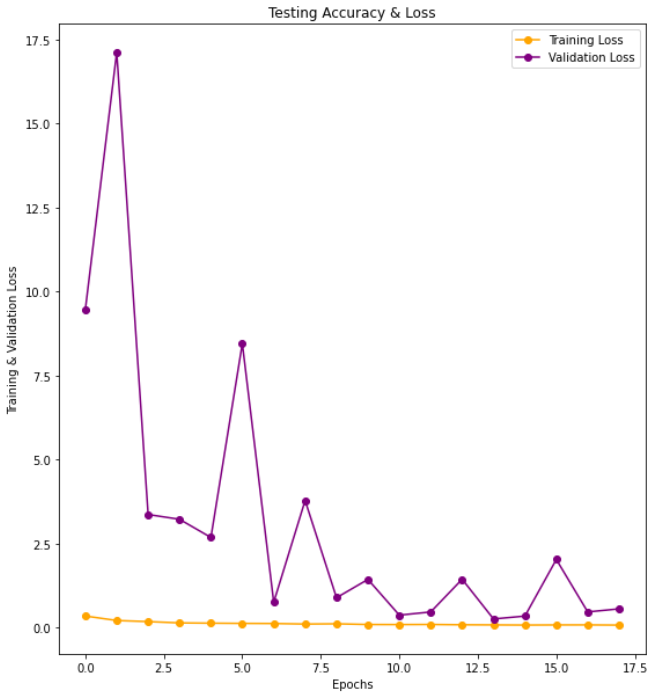
(Test loss 와 accuracy 출력화면)

```
20/20 [==============================] - 0s 12ms/step - loss: 0.2406 - accuracy:
0.9231
Loss of the model is -  0.24063491821289062
20/20 [==============================] - 0s 12ms/step - loss: 0.2406 - accuracy:
0.9231
Accuracy of the model is -  92.30769276618958 %
```

모델의 test 데이터 셋에 대한 정확도는 약 92.3% 그리고 손실은 약 0.24 였다. (모델을 정의 할 때 마다 정확도와 손실에 약간의 오차가 있을 수 있다.)

3.2.10 Training accuracy and loss graph





그래프는 학습 데이터와 검증데이터의 정확도와 손실 그래프이다. 위 정확도 그래프를 보면, 학습 데이터에 대한 정확도는 점점 증가하고 약 0.96~0.97 으로 수렴하는 것을 확인할 수 있다. 반면에 검증 데이터에 대한 정확도는 증가하는 추세이긴 하지만 학습데이터에 대한 정확도보다 작고 학습에 따른 정확도 증감하는 정도가 학습 데이터보다 상대적으로 큰 것을 확인할 수 있다.

또한, 손실 그래프를 보면 검증데이터가 학습데이터 보다 손실이 크다. 그리고 둘 다 손실이 감소하지만 검증 데이터의 손실은 증감하는 정도가 학습 데이터 보다 크다.

이것에 대한 이유는 바로 **과대적합(overfitting)**이다. 그렇다고 이 모델이 의미가 없는 모델은 아니다. 왜냐하면 과대 적합이 나 올 수밖에 없기 때문이다.

그 이유는 훈련 세트와 검증세트의 크기 차이가 매우 크고 그 검증세트의 크기가 작기 때문이다. (데이터 셋의 크기는 "Introduce"에서 소개했다.)

그러므로 학습데이터의 train data 의 정확도 약 97%와 test data 의 정확도 약 92%(3.2.9 참고)를 봤을 때, 의미 있는 모델이라고 할 수 있다.

3.3 Pneumonia Diagnostic App

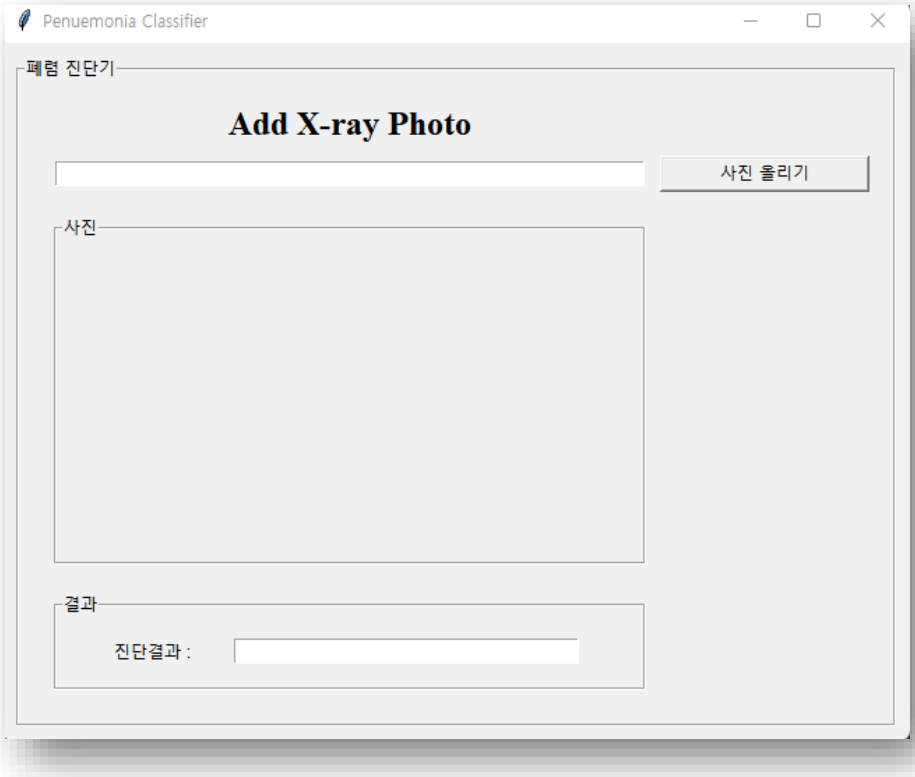
3.3.1 Source Code

```
import tkinter as tk
from tkinter import filedialog
from PIL import Image, ImageTk
import cv2

my_w = tk.Tk()
my_w.geometry("650x500")  # Size of the window
my_w.title('Penuemonia Classifier')
my_font1=('times', 18, 'bold')
frm1 = tk.LabelFrame(my_w, text="폐렴 진단기", pady=15, padx=15)
frm1.grid(row=0, column=0, pady=10, padx=10, sticky="nswe")
my_w.columnconfigure(0, weight=1)   # 프레임 (0,0)은 크기에 맞춰
늘어나도록
my_w.rowconfigure(0, weight=1)

lb1 = tk.Label(frm1,text='Add X-ray
Photo',width=20,font=my_font1)
lb1.grid(row=0,column=0)
image_x=0

listb1 = tk.Listbox(frm1,width=60,height=1)
listb1.grid(row=1, column=0,padx = 10, pady=10)

b1 = tk.Button(frm1, text='사진 올리기',
   width=20,command = lambda:upload_file())
b1.grid(row=1,column=1)

frm2 = tk.LabelFrame(frm1, text="사진", pady=15,
padx=15,width=250,height=250)
frm2.grid(row=2, column=0, pady=10, padx=10, sticky="nswe")

frm3 = tk.LabelFrame(frm1, text="Result", pady=15,
padx=15,width=250,height=2)
frm3.grid(row=3, column=0, pady=10, padx=10, sticky="nswe")

lb2 = tk.Label(frm3, text="진단결과 : ", width=15, height=1)
lb2.grid(row=0, column=0)

listb2 = tk.Listbox(frm3, width=35, height=1)
listb2.grid(row=0, column=1)

from keras.models import load_model

model =
load_model('C:\\Users\\Mr.wi\\Desktop\\long\\cnn_model.h5')
print("log : model upload complete")

def upload_file():
```

```
    global img
    global img2
    print("log : clicked upload button")

    f_types = [('Jpeg Files', '*.jpeg')]
    filename = filedialog.askopenfilename(filetypes=f_types)
    listb1.delete(0,"end")
    listb1.insert(0,filename)

    image = Image.open(filename)
    # The (450, 350) is (height, width)
    img2 = cv2.imread(filename, cv2.IMREAD_GRAYSCALE)
    img2 = cv2.resize(img2, (160,160))
    img2 = img2/255
    img2 = img2.reshape(1,160,160,1)
    image = image.resize((200, 200), Image.ANTIALIAS)
    img = ImageTk.PhotoImage(image)


    b2 =tk.Label(frm2,image=img,width=200,height=200) # using
Button
    b2.grid(row=0,column=0)

    res = model.predict(img2,verbose = 1)
    print("log : Diagnosis complete")
    res = res[0][0]
    if res <= 0.5:
        listb2.delete(0, "end")
        listb2.insert(0,("폐렴 (정상 : {0:.2f}%, 폐렴 :
{1:.2f}%)".format(res*100,(1-res)*100)))
        print("log : Diagnosis = 폐렴")
    elif res > 0.5:
        listb2.delete(0, "end")
        listb2.insert(0,("정상 (정상 : {0:.2f}%, 폐렴 :
{1:.2f}% )".format(res*100,(1-res)*100)))
        print("log : Diagnosis = 정상")

my_w.mainloop()  # Keep the window open
print("log : app exit")
```

위 코드는 폐렴 진찰 앱에 대한 소스코드다.
언어는 파이썬으로, Tkinter 라이브러리로 만들었다. 구성은 다음과 같다. 사진을 업로드하기 위해 upload button 을 설정했고, 어떤 경로에서 파일을 가져왔는지 확인하기 위해, list bar 를 upload button 옆에 설정했다.
그 list bar 와 button 아래는 업로드한 사진이 나타나게 했고 그 사진 밑에는 모델이 진단 한 결과를 나타나게 하기 위해 Label 과 list bar 를 배치했다.
이 때, 모델은 이미지를 통하여 출력값(진단)을 할 때, 0 과 1 이 아니라 시그모이드 함수의 출려값(0 과 1 사이에 값)이 출력된다. 그래서 앱에서는 0.5 이하값은 폐렴, 0.5 초과 값은 정상으로 출력하도록 했다.

3.3.2 App Executing



<최초 실행>

<폐렴 사진 진단>

<정상 사진 진단>

위 3 개의 사진은 폐렴 진단 앱을 실행했을 때 화면이다.
사진 경로를 살펴보면 test 데이터 셋(폴더)에 사진을 알 수 있고 이 사진이 normal 폴더 사진과 pneumonia 폴더 사진 중에 어떤 곳에서 업로드한 사진인지 알 수 있다.
진단 결과 옆에는 정상과 폐렴에 대한 백분율을 표시했는데 그 이유는 모델이 **100%로 결과를 예측하는 것이 아닐 수 있기 때문이다**.
예를 들어서, 모델이 사진을 정상 사진으로 진단했지만, 정상이 60% 그리고 폐렴이 40%이면, 더 정확한 진단을 위해 모델의 진찰 말고도 다른 더 정확한 진단이 필요하다.

```
log : model upload complete
log : clicked upload button
1/1 [==============================] - 0s 211ms/step
log : Diagnosis complete
log : Diagnosis => 폐렴
log : clicked upload button
1/1 [==============================] - 0s 27ms/step
log : Diagnosis complete
log : Diagnosis => 정상
log : app exit
```

<앱 log 기록>

앱이 어떻게 동작하는지 표현 하기위해, 앱의 동작 log 도 표현했다.

4. Personal Feeling (소감)

이 번 프로젝트는 처음부터 쉽지 않았던 것 같다. 아무래도 현재가 내가 가지고 있는 pc 와 노트북의 사양이 그리 좋지 않다 보니 구글 코랩에서 진행했다. 하지만 아무래도 이번 프로젝트의 데이터가 이미지 데이터이다 보니 메모리 사용량 많았다. 그러므로 더 좋은 그래픽 카드 와 더 메모리 사이즈가 큰 환경에서 프로젝트를 진행하기 위해 코랩 유료 버전을 결제하게 되었다.
처음에는 이미지 전 처리 없이, 오직 이미지의 픽셀 값만 255 로 나누었다. 그 데이터를 가지고 학습한 결과 학습데이터와 검증 데이터에 대한 정확도는 0.99 와 1 로 엄청난 정확도를 보였다. 하지만 test 데이터에 대한 정확도는 0.74 로 낮은 상대적으로 낮은 정확도를 보였다.
나는 이런 **과대적합** 문제에 직면했다. 아무래도 인공지능 경진 대회들을 나가봤던 경험을 미루어 짐작해봤을 때, 이러게 많이 차이나는 과대적합을 해결하기 위해서는 모델의 디자인을 변경하기 앞서서 더 많은 의미있는 데이터 전처리가 중요하다고 생각했다.
보통 csv 파일의 데이터의 전처리는 여러가지 있는 것을 알고 있었지만, 이런 미지에 대한 전처리에 대해서는 잘 알지 못했다. 그래서 공부하던 중 keras 에서 제공하는 "ImageDataGenerator"라는 모듈을 알게 되었다. (이 모듈에 대한 설명은 2.3 을 참고) 그 모듈을 통해 과대 적합 문제를 어느 정도 해결할 수 있었다.
그 다음은 입력값의 크기 조절과 모델 구조를 편집을 했다. 입력값을 원본에 가까운 크기로 설정하고 싶었으나 크기를 크게 할수록 업로드 하는 과정에서 메모리 사용 초과에 의해 런타임이 계속 초기화 되었다. 그래서 어느정도 모델의 크기에 대해 타협을 해야 했다. 그래서 150~300 사이에 값으로 입력값을 했을 때 위의 문제를 해결할 수 있었다.
다음은 모델의 구조 편집이었다. 처음에는 cnn layer 10 개층 그리고 dnn layer 6 개 층으로 시작을 했다. 여기서 입력값 크기에 맞춰서 convolution 과 maxpooling 의 각각 파라미터를 조절했다. 이렇게 무수히 많은 조합에 맞춰서 정확도를 평가를 했다. 그 결과, 많은 CNN 레이어는 오히려 이미지 데이터의 의미 있는 feature 들을 뽑아내지 못했고 많은 dnn layer 는 학습을 과대적합으로 만들거나 오히려 방해를 하는 효과를 낼 수 있다는 것을 알게 되었다.
이렇게 해서 결국 test 데이터에 대해 92.6%라는 **정확도**를 뽑아 낼 수 있었다.

정확도를 높이기 위한 이러한 과정과 이번 프로젝트를 통해 이미지 데이터에 대해 분류 모델을 만들기 위해선 많은 시간과 비용이 든다는 사실을 깨닫게 되었고 이 강의를 듣기 전에 cnn 에 대해 잘 몰랐었는데, 강의를 듣고 프로젝트를 진행한 후 보다 cnn 에 대해 친숙하게 되고 더 잘 알게 되었다. 그리고 위와 같은 이유들로 이번 프로젝트 나에게 있어서 너무나 유익하고 의미 있는 프로젝트였다.