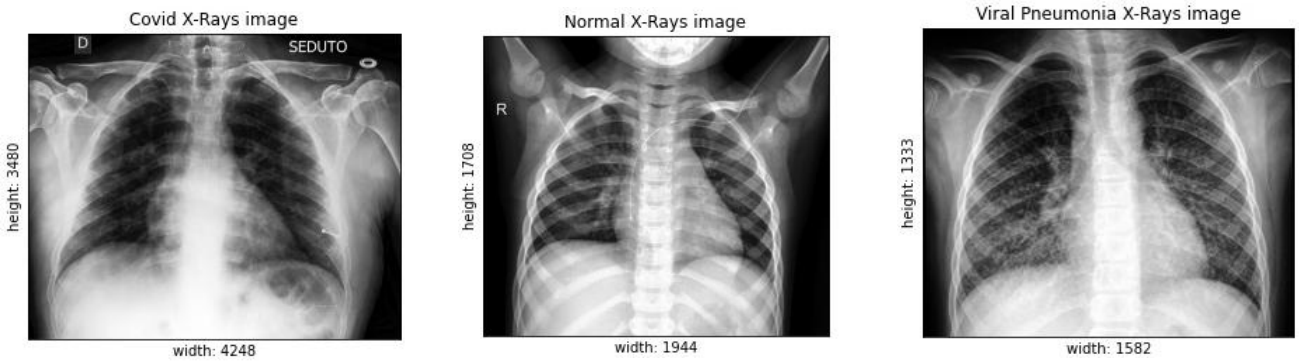
기말과제 보고서
류채은

요약:
VGG16 모델을 활용한 전이학습(Transfer Learning)을 통해 흉부 X-Rays 이미지가 정상적인 폐인지, 코로나가 감염된 폐인지, 바이러스성 폐렴(Viral Pneumonia)가 감염된 폐인지를 알려주는 Image Classification을 수행했다.

1. 데이터 설명

   코로나가 걸린 폐의 흉부 X-Ray 이미지, 바이러스성 폐렴이 걸린 폐의 흉부 X-ray 이미지, 정상적 폐의 흉부 X-Ray 이미지로 구성된 데이터이다. 훈련 데이터셋은 251개 이미지, 테스트 데이터셋은 66개의 이미지로 구성되어있다.

   1.1 이미지 레이블



   왼쪽부터 차례로 코로나에 감염된 폐의 이미지, 질병이 없는 정상적인 폐의 이미지, 바이러스성 폐렴이 걸린 폐의 이미지 예시다. 데이터는 이렇게 총 3개의 이미지 레이블로 구성되어있으며 이미지 별로 height와 width도 다 다르다.

   1.2 데이터 수(Dataset Size)와 분포(Distribution)

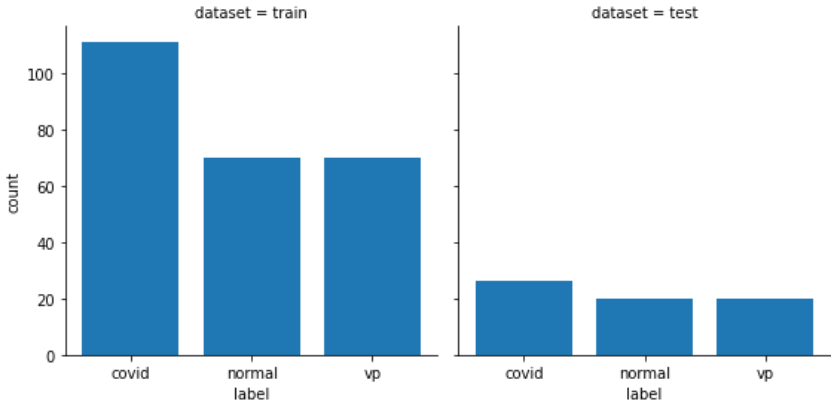


그림 1 데이터 셋 당 레이블 개수

   데이터 셋은 훈련 데이터셋(train)과 테스트 데이터셋(test)으로 나누어져 있으며, 각 데이터 셋마다 레이블 당 이미지 개수를 FacetGrid를 통해 막대 그래프로 나타냈다. 두 데이터 셋이 공유하는 y축(count)을 통해 훈련 데이터셋 수가 테스트 데이터셋보다 많음을 알 수 있다.

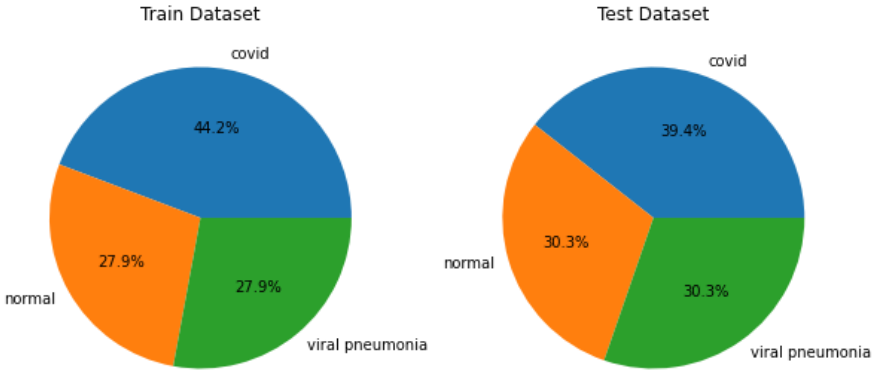


그림 2 레이블 별 각 데이터셋에서 차지하는 비율

레이블 별로 각 데이터셋에서 차지하는 비율을 파이 차트로 나타냈다. 대체적으로 코로나 이미지가 더 많으며, 각 레이블 당 이미지 수의 차이가 크지 않아 고르게 분포되어 있는 것을 알 수 있다.

1.3 레이블 당 이미지 크기(height x width)

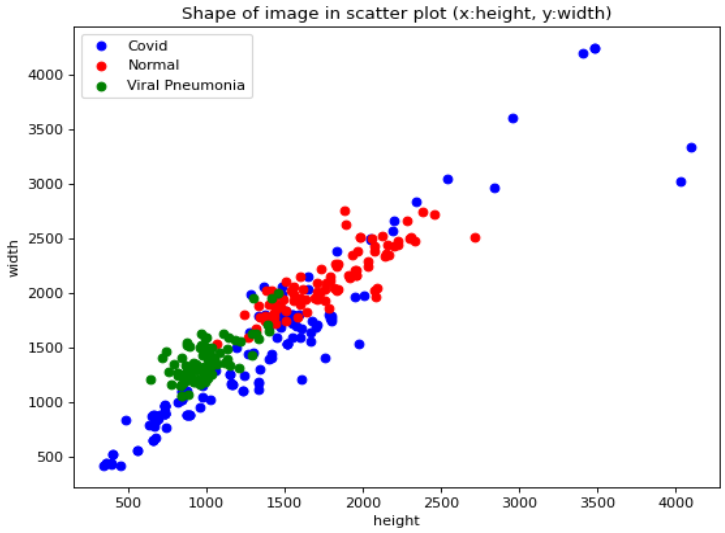


그림 3 레이블 별 이미지 크기(height x width)

이미지의 크기인 세로(height)와 width(가로)를 각각 x축과 y축으로 설정하여 산점도로 나타내었다. 이미지 크기는 다양하며, (0,0) 쪽에 아무것도 없는 것으로 보아 빈 이미지 파일이 존재하지 않음을 알 수 있다.

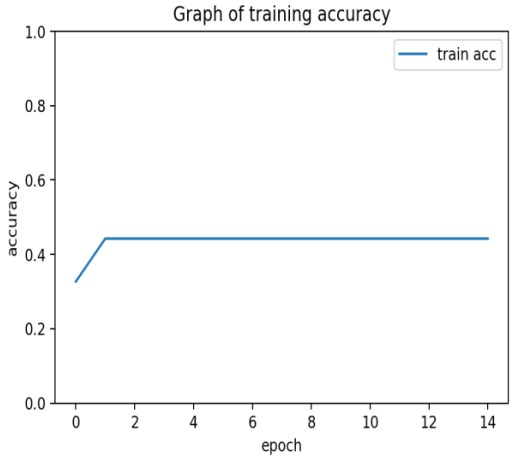
2. 기법

2.1 데이터 전처리(Data Preprocessing)

모델을 훈련시키기 전에 이미지 데이터를 전처리하는 과정이 필요했다. 이미지 데이터는 0~255사이의 값을 갖기에 훈련 데이터셋과 테스트 데이터셋을 이를 0~1 사이의 값으로 정규화(normalization)했다. 또한, 이미지 데이터 구조(shape)가 그림 3에서 보이듯 일괄적이지 않았기에 [224,224]로 통일하는 전처리를 진행했다.

2.2 모델

2.2.1 처음 활용하려던 모델 성능



초반엔 CNN(Convolutional Neural Network) 모델 중 하나로 이미지 분류에 뛰어난 성능을 보인다고 알려진 VGG16 모델의 구조를 빌려서 초기화된 가중치로 모델 훈련을 시도했다. 하지만, 0.44의 정확도로 너무 낮은 결과가 나왔다. 또한, 아무리 반복해서 모델을 훈련해보아도 성능이 좋아지지 않았다. 뿐만 아니라, GPU를 사용하지 않고 훈련을 하면 모델을 훈련시키는 속도가 데이터셋 크기에 비해 너무 느렸다.

2.2.2 이후 방안

따라서, 이를 해결할 수 있는 방법을 연구해보았고 학습 데이터 수가 적을 때 효과적이며 학습 속도가 빠른 전이학습(Transfer Learning)을 활용하였다. 전이 학습이란 특정 분야에서 학습된 신경망의 일부 지식을 새로운 분야를 학습하는데 활용하는 것을 의미한다. 모델의 이전 Keras에서 제공하는 사전에 학습된(pre-trained) VGG16모델의 ImageNet 가중치를 활용하여 모델을 훈련시켰다.

2.2.3 Model Architecture and Sequence Diagram

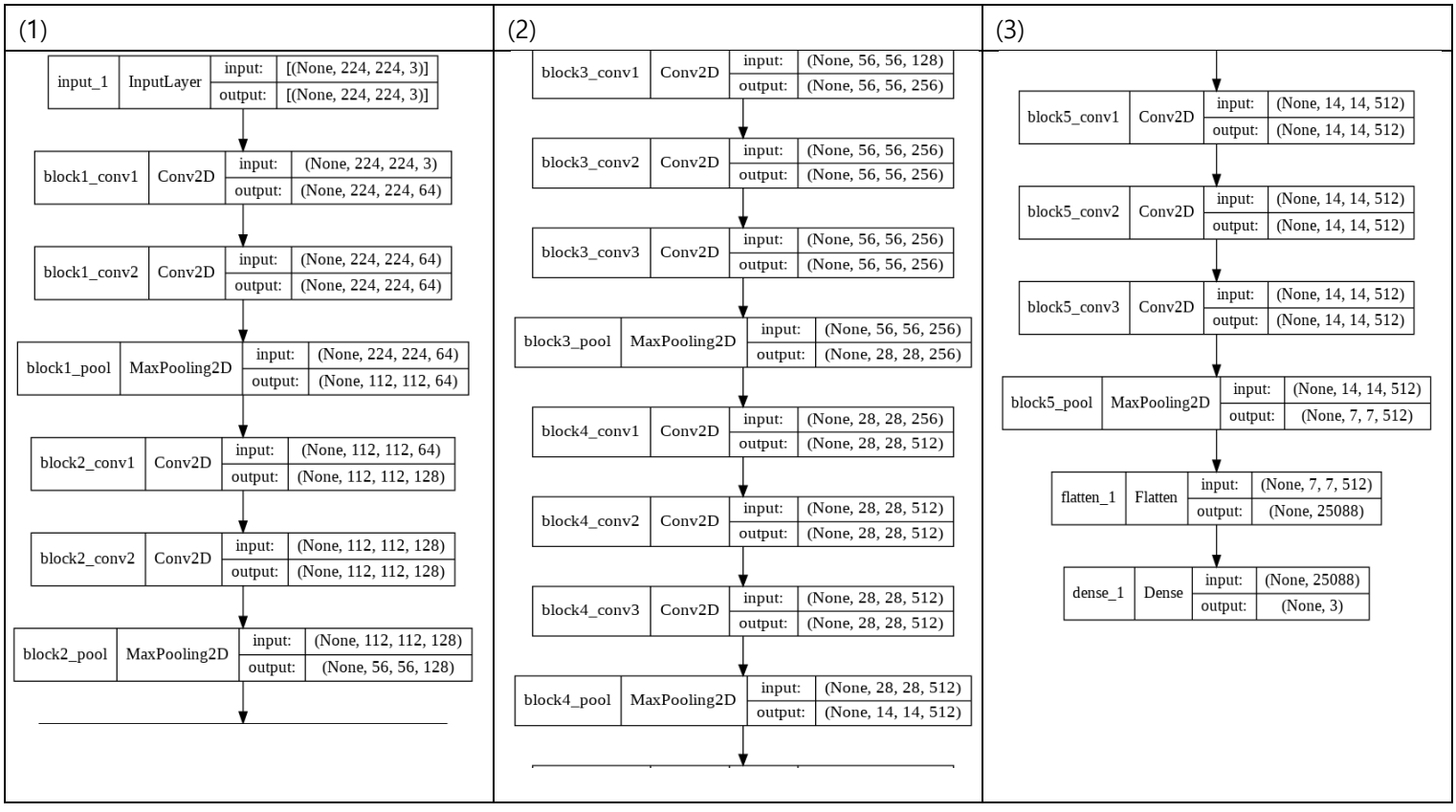


그림 4 Model architecture and sequence

모델 구조의 크기가 커서 한 모델을 3분할하여 순서대로 첨부했다. 모델의 구조는 VGG16에서 빌려온 모델로, VGG16 모델의 입력층(input_1)과 마지막 출력층(dense_1)을 이 데이터 셋에 맞게 조정하여 VGG16모델의 부분을 사용한 새로운 모델을 만들었다. VGG에 사전에 존재하던 층의 가중치를 활용하기 위해 훈련하는 동안 초기화되거나 변경되는 것을 layer.trainable = False 코드를 통해 막았다. 이후, VGG16모델에 사전에 존재하지 않던 층

(input_1, dense_1)만 가중치가 변경되도록 현재 데이터 셋에 맞게 훈련시키는 데에 집중했다.

2.3 훈련

다수의 레이블을 분류하는 작업을 수행하기에 모델을 훈련할 때 사용한 손실(loss)은 categorical crossentropy로 지정하였다. 또한, 옵티마이저는 Adam으로 사용하였으며, accuracy를 판단하였다. 에포크 횟수는 8로 정했다.

3. 결과

```
Epoch 1/8
8/8 [==============================] - 13s 2s/step - loss: 2.9067 - accuracy: 0.3546
Epoch 2/8
8/8 [==============================] - 12s 2s/step - loss: 1.8464 - accuracy: 0.5857
Epoch 3/8
8/8 [==============================] - 12s 2s/step - loss: 0.3101 - accuracy: 0.8566
Epoch 4/8
8/8 [==============================] - 12s 1s/step - loss: 0.5369 - accuracy: 0.8367
Epoch 5/8
8/8 [==============================] - 12s 2s/step - loss: 0.1673 - accuracy: 0.9482
Epoch 6/8
8/8 [==============================] - 12s 2s/step - loss: 0.1514 - accuracy: 0.9442
Epoch 7/8
8/8 [==============================] - 12s 1s/step - loss: 0.0508 - accuracy: 0.9801
Epoch 8/8
8/8 [==============================] - 12s 1s/step - loss: 0.0555 - accuracy: 0.9801
```

그림 5 Training Result

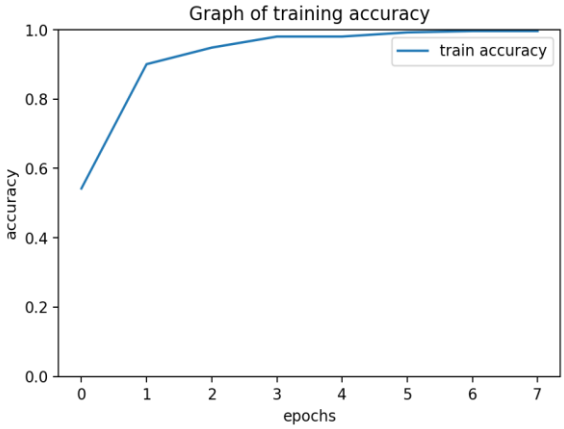


그림 7 Training accuracy

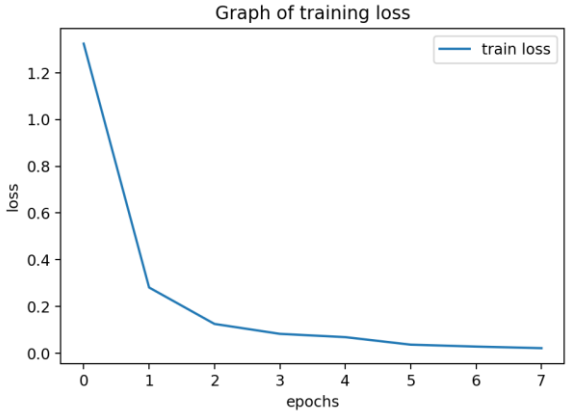


그림 6 Training loss

Epoch 횟수가 늘수록 훈련 정확도는 1.00에 근접하게 수렴하게 훈련 손실이 0에 가깝게 줄어드는 것을 알 수 있다. epoch횟수를 12 정도로 늘려보았을 땐 training accuracy가 epoch 10 정도부터 1.00이 되어 더 이상 훈련하는 것이 의미가 없었고 과대적합의 위험을 인지했다. 특히, 이 훈련 데이터셋은 데이터셋 크기가 작기에 훈련데이터 예측정확도가 테스트 데이터 예측 정확도에 비해 현저히 높은 오버피팅이 일어난다면 모델의 generalization ability가 현저히 떨어질 수 있기 때문에 과대적합을 막기 위해 1.00이 되지 않도록 epoch횟수를 8로 조정했다. 결과적으로 이 모델은 Train accuracy는 0.996, loss는 0.0488을 기록하며 뛰어난 훈련 결과를 도출했다.

4. 모델 성능

4.2 테스트 데이터셋에 적용한 결과

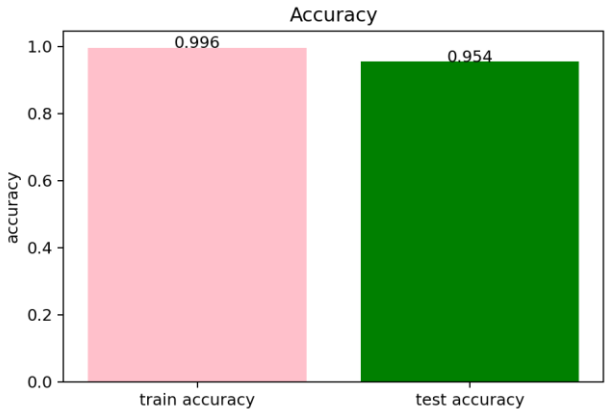


그림 8 훈련 정확도와 테스트 정확도

테스트 데이터셋을 통해 모델이 경험하지 않는 새로운 데이터인 test dataset에 모델의 정확도를 평가해보았고, 예측 정확도는 0.954 정도로 훈련 정확도보다는 낮지만 꽤 높은 성능을 보이는 것을 알 수 있었다.

4.3 분류 오류

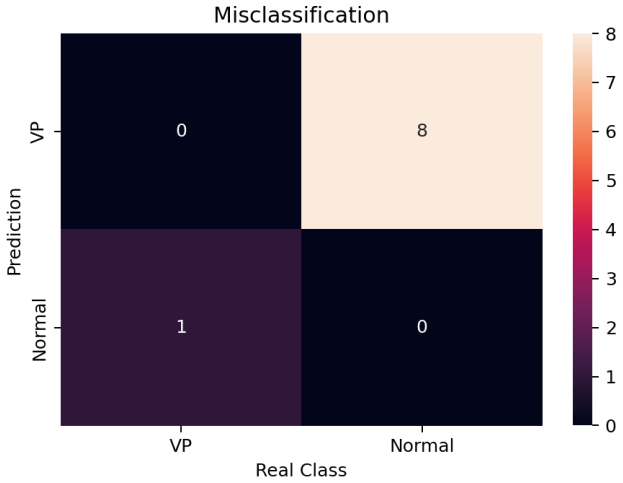


그림 9 잘못 분류한 경우

모델이 어떤 상황에서 분류 작업 수행에 낮은 성능을 보이는지 알기 위해서 예측한 이미지 레이블(Predicted label)과 실제 이미지 레이블(Real label)이 다를 경우를 출력해보았다. 훈련 데이터셋에선 총 3번 잘못 분류한 경우가 존재했는데, 정상(Normal)을 바이러스성 폐렴(Viral Pneumonia)로 잘못 분류하는 경우가 2번, 바이러스성 폐렴을 정상으로 잘못 분류하는 경우가 한 번 있었다. 테스트 데이터셋에선 총 6번 잘못 분류하였고 모두 정상을 바이러스성 폐렴으로 잘못 분류하는 경우였다. 훈련 데이터셋과 테스트 데이터셋에서 발생한 잘못 분류한 경우를 모두 모아 heatmap으로 나타내어 Prediction label과 Real Class 관계 당 오류가 발생하는 빈도를 표현하였다. 이를 통해 정상인 폐의 X-Ray사진을 바이러스성 폐렴(VP, Viral Pneumonia)로 잘못 분류하는 경우의 오류가 가장 많이 발생함을 알 수 있었다.

5. 코드

```
6. ###############Import Library####################
   #import necessary libraries
   import seaborn as sns
   import os
   import cv2
   from sklearn.model_selection import train_test_split
   import numpy as np
   import pandas as pd
   import math
   import matplotlib.pyplot as plt
   import tensorflow as tf
   from tensorflow import keras

   #set dataset directory
   train_data_path = "/content/drive/MyDrive/Colab Notebooks/Covid19-dataset/train"
   test_data_path = "/content/drive/MyDrive/Colab Notebooks/Covid19-dataset/test"

   ###############Data Visualization####################
   #show sample covid x-rays image
   img = plt.imread(os.path.join(train_data_path, "Covid/01.jpeg"))#read image from
   file
   plt.imshow(img)#show image
   height, width, dim = img.shape#get image shape
   plt.title('Covid X-Rays image')#set title
   plt.xlabel("width: {}".format(width))#set xlabel
   plt.ylabel("height: {}".format(height))#set ylabel
   plt.xticks([])#empty xticks
   plt.yticks([])#empty yticks
   print("size of image (h x w)",height,width)

   #show sample normal image
   img = plt.imread(os.path.join(train_data_path, "Normal/01.jpeg"))#read image
   plt.imshow(img)#show image
   height, width, dim = img.shape
   plt.title('Normal X-Rays image')#set title
   plt.xlabel("width: {}".format(width))
   plt.ylabel("height: {}".format(height))
   plt.xticks([])#empty xticks
   plt.yticks([])#empty yticks
   print("size of image (h x w)",height,width)

   #show sample Viral Pneumonia image
   img = plt.imread(os.path.join(train_data_path, "Viral Pneumonia/01.jpeg"))#read
   image
   plt.imshow(img)#show image
   height, width, dim = img.shape#get image shape
   plt.title('Viral Pneumonia X-Rays image')
   plt.xlabel("width: {}".format(width))
   plt.ylabel("height: {}".format(height))
   plt.xticks([])#empty xticks
   plt.yticks([])#empty yticks
   print("size of image (h x w)",height,width)

   import os, os.path

   #shapes of image
   train_shapes = []#shapes of images in train dataset
   test_shapes = []#shapes of images in test dataset

   #train
   path, dirs, files = next(os.walk("/content/drive/MyDrive/Colab
   Notebooks/Covid19-dataset/train/Covid"))
   file_count_train_covid = len(files)#count image files

   path, dirs, files = next(os.walk("/content/drive/MyDrive/Colab
   Notebooks/Covid19-dataset/train/Normal"))
```

```
file_count_train_normal = len(files)
path, dirs, files = next(os.walk("/content/drive/MyDrive/Colab
Notebooks/Covid19-dataset/train/Viral Pneumonia"))
file_count_train_vp = len(files)#count image files

#test
path, dirs, files = next(os.walk("/content/drive/MyDrive/Colab
Notebooks/Covid19-dataset/test/Covid"))
file_count_test_covid = len(files)
path, dirs, files = next(os.walk("/content/drive/MyDrive/Colab
Notebooks/Covid19-dataset/test/Normal"))
file_count_test_normal = len(files)
path, dirs, files = next(os.walk("/content/drive/MyDrive/Colab
Notebooks/Covid19-dataset/test/Viral Pneumonia"))
file_count_test_vp = len(files)#count image files

#make a dataframe for data visualization
df= []
#make a list containing number of images in each label per dataset
vals = [file_count_train_covid, file_count_train_normal, file_count_train_vp,
file_count_test_covid, file_count_test_normal, file_count_test_vp]
for i in range(6):#number of data folders
  tmp_df = []
  if i<3:
    tmp_df.append("train")
  else:
    tmp_df.append("test")
  if i%3 == 0:
    tmp_df.append("covid")
  elif i%3 == 1:
    tmp_df.append("normal")
  else:
    tmp_df.append("vp")
  tmp_df.append(vals[i])
  df.append(tmp df)

#dataframe that has number of labels in each dataset
df_to_plot = pd.DataFrame(df, columns=["dataset","label","count"])

#####Visualization 1: Use multi-plot to show the number of data in each folder
and dataset
#Use multi-plot to show the number of data in each folder and dataset
g = sns.FacetGrid(df_to_plot, col="dataset",size=4)#use facetgrid
g.map(plt.bar, "label","count")
plt.show()#show bar graph

#####Visualization 2: Use pie chart to show distribution of label in each folder
#Use Pie Chart to show
total_train = np.sum(vals[:3])#total number of images in train dataset
total_test = np.sum(vals[3:])#total number of images in test dataset
train_percentage = df_to_plot[df_to_plot["dataset"] ==
"train"]["count"]/total_train#get percentage of label for each label in train
data
test_percentage = df_to_plot[df_to_plot["dataset"] ==
"test"]["count"]/total_test#get percentage of label for each label in test data
labels =["covid","normal","viral pneumonia"]#list of labels

fig, (ax1,ax2) = plt.subplots(1,2,figsize=(10,10)) #ax1,ax2 refer to two pies
ax1.pie(train_percentage,labels = labels,autopct = '%1.1f%%') #plot first pie
about train dataset
ax1.set_title('Train Dataset')
ax2.pie(test_percentage,labels = labels,autopct = '%1.1f%%') #plot second pie
about second dataset
ax2.set_title('Test Dataset')
plt.show()#show pie chart
```

```
#visualize the size of each image
img_shapes = [[],[],[]]#shape of image of Covid, Normal, Viral Pneumonia
FILENAMES = ["Covid", "Normal", "Viral Pneumonia"]

for i in ([train_data_path, test_data_path]):#for both train and test dataset
  for j in range(len(FILENAMES)):
    d = os.path.join(i,FILENAMES[j])
    path, dirs, files = next(os.walk(d))

    for file in files:
      img = plt.imread(os.path.join(path,file))
      img_shapes[j].append(list(img.shape[:2]))#store height, width of image into
'img_shapes' list


covid_x = []#all height size of covid images
covid_y = []#all width size of covid images

for im in img_shapes[0]:
  covid_x.append(im[0])#height
  covid_y.append(im[1])#width

normal_x = []#all height size of normal images
normal_y = []#all width size of normal images

for im in img_shapes[1]:
  normal_x.append(im[0])#height
  normal_y.append(im[1])#width

vp_x = []#all width size of viral pneumonia images
vp_y = []#all width size of viral pneumonia images

for im in img_shapes[2]:
  vp_x.append(im[0])#height
  vp y.append(im[1])#width

#####Visualization 3: Shape of image in scatter plot
#scatter plot the shape of images per label
plt.scatter(covid_x,covid_y,c='blue',label="Covid")#plot height and width of
covid x-rays image
plt.scatter(normal_x, normal_y,c='red',label="Normal")#plot height and width of
normal x-rays image
plt.scatter(vp_x, vp_y,c='green',label="Viral Pneumonia")#plot height and width
of viral pneumonia x-rays image
plt.legend()#show legend
plt.title("Shape of image in scatter plot (x:height, y:width)")#set title
plt.xlabel("height")
plt.ylabel("width")
plt.show()#show scatter plot


################Data Preprocessing#####################
from keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator
train_Gen = ImageDataGenerator(rescale=1/255)#rescale the training data so that
the value is in range 0~1
test_Gen = ImageDataGenerator(rescale=1/255)#rescale the testing data so that
the value is in range 0~1

#prepare train dataset suitable for training
train_dataset = train_Gen.flow_from_directory(train_data_path,
                                    target_size = (224, 224),#resize the image
                                    batch_size = 32,
                                    class_mode = 'categorical',#dataset has 3
labels
                                    shuffle = False)#don't shuffle the data
```

```
#prepare test dataset suitable for training
test_dataset = test_Gen.flow_from_directory(test_data_path,
                                    target_size= (224, 224),#resize the image
                                    batch_size = 32,
                                   class_mode = 'categorical',#dataset has 3
labels
                                    shuffle = False)#don't shuffle the data

train_dataset.class_indices#print labels of train dataset
train_dataset.image_shape#get shape of train dataset


################Set Model#####################
from keras.applications.vgg16 import VGG16#get pre-trained VGG16 model
IMAGE_SIZE = [224,224]
#adjust the input layer of the model suitable for this dataset
vgg = VGG16(input_shape=IMAGE_SIZE + [3], weights='imagenet',
include_top=False)#set imagenet weights
for layer in vgg.layers:
  layer.trainable = False#do not modify the pre-trained weights while training

from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Lambda, Dense, Flatten
labels = ["Covid", "Normal", "Viral Pneumonia"]#labels
x = Flatten()(vgg.output)
#adjust the last layer of the model suitable for this dataset
last_layer=Dense(len(labels),activation='softmax')(x)#shape the output size
suitable for this dataset, and modify the last layer
new_model = Model(inputs=vgg.input, outputs=last_layer)#build model with the new
input layer and output layer

new_model.summary()#summary of this model


################Train the Model#####################
new_model.compile(#model compile
  loss='categorical_crossentropy',#suitable for image classification of many
labels
  optimizer='adam',
  metrics=['accuracy']
)

#train model
r = new_model.fit_generator(
  train_dataset,
  epochs=8,#number of epochs
  steps_per_epoch=len(train_dataset)#8
)


################Plot training results (Visualization)#####################
#plot training loss
plt.plot(r.history['loss'], label='train loss')#show in line graph
plt.legend()
plt.title("Graph of training loss")#set title
plt.xlabel("epochs")
plt.ylabel("loss")
plt.show()#show graph

#plot train accuracy
plt.plot(r.history['accuracy'], label='train accuracy')#show in line graph
plt.legend()
plt.title("Graph of training accuracy")
plt.xlabel("epochs")
plt.ylabel("accuracy")
plt.ylim(0,1)#set limit of y axis between 0 and 1
plt.show()#show graph
```

```
###############Test the Model####################
#test the model
t = new_model.evaluate_generator(test_dataset)#test
train_accuracy = r.history['accuracy'][-1]#train_accuracy
test_accuracy = t[1]#get test accuracy


###############Visualization of results####################
#show train accuracy and test accuracy in bar chart
x_ticks = ["train accuracy","test accuracy"]#set x ticks
acc_list =[train_accuracy, test_accuracy]
up = [float(str(train_accuracy)[:5]), float(str(test_accuracy)[:5])]#the
accuracy value to put on top of each bar
plt.bar(x_ticks, acc_list, color = ['pink','g'])#show in bar graph
for i in range(len(x_ticks)):
  plt.text(i, up[i],up[i],ha="center")#plot the accuracy value on top of each bar
plt.title("Accuracy")
plt.ylabel("accuracy")
plt.show()#show in bar chart

###############Look at the performance of model in depth####################

#predict the image labels for images in train dataset
pred=new_model.predict_generator(train_dataset,verbose=1)#predict the images by
the model
predicted_class_indices=np.argmax(pred,axis=1)#get the label with the highest
percentage(most likely to be the label)
labels = (train_dataset.class_indices)
labels = dict((v,k) for k,v in labels.items())
predictions = [labels[k] for k in predicted_class_indices]#convert the numbers
that represent label into real category labels in string

filenames=train_dataset.filenames
real class = []#get the real label by the folder name
for f in filenames:
  real_c, _ = f.split("/")#preprocess the foldername to convert it to label
  real_class.append(real_c)#append the label into real label list

#make dataframe of predicted labels and real labels for each image in train
dataset
results=pd.DataFrame({"Real Class":real_class,
                      "Predictions":predictions})
results.head()#show the dataframe

# misclassification(classification error) in training set
print("misclassification happened in {} training
images".format(len(results[results["Real Class"] != results["Predictions"]])))

#store the dataframe where real label is not same as predicted label in train
dataset
train_misclassification = results[results["Real Class"] !=
results["Predictions"]]

#predict the labels of image in test dataset
test_pred=new_model.predict_generator(test_dataset,verbose=1)#predict with the
model
test_predicted_class_indices=np.argmax(test_pred,axis=1)#get the most likely
label based on percentage
test_labels = (test_dataset.class_indices)
test_labels = dict((v,k) for k,v in test_labels.items())
test_predictions = [test_labels[k] for k in
test_predicted_class_indices]#convert the digit into string labels

filenames=test_dataset.filenames#file names of images in test dataset
real_class = []#make a list containing real labels of test dataset images
for f in filenames:
```

```
    real_c, _ = f.split("/")
    real_class.append(real_c)#real label

#make dataframe of predicted labels and real labels for each image in test
dataset
test_results=pd.DataFrame({"Real Class":real_class,
                   "Predictions":test_predictions})
test_results.head()

print("misclassification happened in {} test
images".format(len(test_results[test_results["Real Class"] !=
test_results["Predictions"]])))
test_misclassification = test_results[test_results["Real Class"] !=
test_results["Predictions"]]
test_misclassification##store the dataframe where real label is not same as
predicted label in test dataset

#concatename two misclassification dataframes(train and test) into one dataframe
misclassification = pd.concat([train_misclassification, test_misclassification])

from sklearn.metrics import confusion_matrix
conf_matrix = confusion_matrix(misclassification["Real Class"],
misclassification["Predictions"])#make confusion matrix with misclassification
sns.heatmap(conf_matrix, annot=True, yticklabels=['VP','Normal'],
xticklabels=['VP','Normal'])#show the confusion matrix by heat map
plt.title("Misclassification")#set title
plt.ylabel("Prediction")
plt.xlabel("Real Class")
plt.show()
```