تمرین سوم: دسته بندی تصاویر اشعه ایکس قفسه سینه

توسط: مهدی مهدیانی

ایمیل:mahdimahdiani@ymail.com , m.mahdiani@stu.usc.ac.ir

استاد: آقای دکتر شهسوار حقیقی

دستیار آموزش: آقای مهندس صالح

۱- آماده سازی و پیش پردازش داده ها

سوال ۱ :

برای پاسخ به این سوال اول لازم به ذکر است که هر ناحیه کوچک از عکس یک ماتریسی از اعداد است و در فرآیند آموزش، وقتی یک عکس را ۲۰ بار به مدل نشان می دهیم باعث می شود مدل به سمت بیش برازش برود. برای این مساله ما تلاش می کنیم هر بار تغییری در عکس ایجاد کنیم تا از این مساله جلوگیری کنیم. از تکنیک های افزایش داده می توان به موارد زیر اشاره کرد.

Rotation – Translation – Scaling – Cropping – Flipping – Distortion – Shearing – Stretching – Color shifting (Color jitter) – Brightness and contrast adjustment – add noise

سوال ۲ :

انتخاب وضوح تصویر دلایل مختلفی را می تواند داشته باشد. برای مثال هر چه تصویر با کیفیت تر باشد ابعاد بزرگتری دارد در نتیجه به منابع محاسباتی(GPU / CPU) بیشتری نیاز دارد که می تواند فرآیند آموزش ما را کند تر کند. ثانیا هرچه تصویر با کیفیت تر باشد باعث می شود الگو های پیچیده تری پیدا شود و پایین بودن کیفیت تصویر باعث می شود یک سری جزئیات تصویر از بین برود و دقت مدل کاهش یابد. البته باید توجه داشت که انتخاب وضوح بسیار بالا ممکن است مدل را به جزئیات بسیار ریز و نویز ها حساس کند و در نتیجه مدل دچار بیش برازش شود.

سوال ۳ :

در مرحله پیش پردازش و افزایش داده ها ابتدا عکس های مجموعه داده را به ابعاد ۱۲۸*۱۲۸ تغییر سایز داده ایم.

لازم به ذکر است که اگر لایه های FC را قرار است دور بریزیم باید، اولا در تعریف مدل ابعاد را مشخص کنیم، ثانیا باید توجه داشته باشیم که عکس ها باید دارای ۳ بعد باشند. در ادامه به بررسی تکنیک های افزایش داده ای که توسط این مقاله انجام شده خواهیم پرداخت.

الف) Re-scale : البته این مورد جز پیش پردازش داده های به حساب می آید که در این مقاله با استفاده از این لایه داده ها (تصاویر) را نرمال کرده و از [۰،۲۵۵] به نرمال شده ی بین [۰،۱] تبدیل کرده است . برای این کار مقدار

پارامتر scale را برابر با ۱./۲۵۵ قرار داده است. این لایه با ضرب هر پیکسل به مقدار scale عدد هر پیکسل را بین صفر و یک نرمال می کند به این صورت که اگر مقدار پیکسل ۰ باشد نرمال شده آن نیز صفر و اگر ۲۵۵ باشد نرمال شده آن ۱ خواهد بود.

ب) Shearing : این تکنیک تصاویر در جهت افقی یا عمودی می کشد و باعث تغییر شکل در تصویر اولیه می شود. از این تکنیک به مقدار ۰.۲ استفاده شده است.

پ) Width Shift : در این تکنیک عرض تصاویر به اندازه ۰.۲ به چپ یا راست جا به جا میشود.

ت) Height Shift : در این تکنیک تصاویر به اندازه ۰.۲ به بالا یا پایین جا به جا میشود.

ث) Rotation : در این تکنیک تصاویر به اندازه ۳۰+ یا ۳۰- درجه می چرخند.

ج) Horizontal Flip : با فعال کردن این تکنیک اجازه وارونه کردن افقی تصاویر داده می شود.

چ)Zoom : در این تکنیک تصاویر به اندازه ۰.۲ بزرگ یا کوچک می شوند.

استفاده از این تکنیک ها هر کدام دلایل خاص خود را دارد برای مثال در مورد ب ، باعث می شود مدل نسبت به شکل ظاهری تصویر مقاوم تر شود و عمومیت مدل بیشتر شود یا برای پ و ت ، باعث می شود مدل نسبت به تغییرات موقعیت اشیا حساسیت کم تری داشته باشد. در مورد ث، مدل نسبت به چرخش های جزئی مقاوم می شود و در مورد ج مدل نسبت به تقارن های موجود در تصویر مقاوم میشود و در نهایت در چ مدل نسبت به تغییرات اندازه اشیا مقاوم می شود.

سوال ۴ :

در ابتدا مجموعه داده را بر روی گوگل کولب دانلود می کنیم. با توجه به سرعت بیشتر پردازش پردازنده گرافیکی و عدم دسترسی به پردازنده گرافیکی در سیستم شخصی ، تصمیم بر اجرای پروژّه بر روی کولب گرفته شد. بعد از دریافت فایل فشرده مجموعه داده با دستورات لینوکسی آن را از حالت فشرده خارج می کنیم و یک سری از پوشه های اضافی این مجموعه داده را که عکس های تکراری در آن ها ریخته شده است را از مجموعه پاک می کنیم :

```
! unzip -qq /content/chest-xray-pneumonia.zip
!rm -rf /content/chest_xray/__MACOSX
!rm -rf /content/chest_xray/test
!rm -rf /content/chest_xray/train
!rm -rf /content/chest_xray/val
```

سپس با توجه به اینکه تقسیم بندی برای آموزش ، ارزیابی و اعتبار سنجی اولیه مجموعه داده مورد تایید ما نیست با استفاده از کتابخانه های os و shutil ، دایرکتوری AllData را می سازیم و تمامی عکس های موجود در هر سه دایرکتوری train,test,val را داخل این دایرکتوری ذخیره میکنیم. با توجه به اینکه در هر یک از سه دایرکتوری موجود، دو دایرکتوری برای Pneumonia , Normal وجود دارد عکس های هر دایرکتوری را در دایرکتوری هم نام با خودش در AllData ذخیره می کنیم :

```
import os
import shutil
# Define the input and output folder paths
input_folder_path = "/content/chest_xray/chest_xray"
output_folder_path = os.path.join(input_folder_path, "AllData")
classes = ["NORMAL", "PNEUMONIA"]

# Create the output folder if it doesn't exist
if not os.path.exists(output_folder_path):
    os.makedirs(output_folder_path)
```

```
# Loop through the train, test, and val folders
for split in ["test", "train", "val"]:
    split_folder_path = os.path.join(input_folder_path, split)
    # Loop through the NORMAL and PNEUMONIA folders in each split folder
    for class_name in classes:
        class_folder_path = os.path.join(split_folder_path, class_name) # Loop
through the image files in each class folder and copy them to the output folder
        for file_name in os.listdir(class_folder_path):
            if file_name.endswith(".jpeg"):
                src_path = os.path.join(class_folder_path, file_name)
                dst_path = os.path.join(output_folder_path, class_name,
file_name)
         # Create the class folder in the output folder if it doesn't exist
                if not os.path.exists(os.path.join(output_folder_path,
class_name)):
                    os.makedirs(os.path.join(output_folder_path, class_name))
                # Copy the image file to the output folder
                shutil.copyfile(src_path, dst_path)
```

حال با بررسی آیتم های موجود در هر دایرکتوری در می یابیم که ۴۲۷۳ عکس Pneumonia و ۱۵۸۳ عکس Normal داریم.

در ادامه این مجموعه عکس ها با استفاده از کتابخانه های os,shutil,random به نسبت ۶۰ درصد برای آموزش، ۲۰ درصد برای ارزیابی و ۲۰ درصد برای اعتبار سنجی تقسیم می کنیم و در دایرکتوری جدید FinalDataset که در هر آن سه دایرکتوری train,test,val را می سازیم ذخیره می کنیم. توجه داریم که عکس هارا بر میزنیم(shuffle می کنیم) سپس تقسیم بندی را انجام می دهیم.

```
import os
import shutil
import random

# Set the path to the "FinalDataset" folder
data_dir = "/content/chest_xray/chest_xray/AllData"
# Set the path to the output directory
output_dir = "/content/FinalDataset"

# Set the train/validation/test split ratios
train_ratio = 0.6
val_ratio = 0.2
test_ratio = 0.2

# Create the output directories
os.makedirs(os.path.join(output_dir, "train", "NORMAL"), exist_ok=True)
os.makedirs(os.path.join(output_dir, "train", "PNEUMONIA"), exist_ok=True)
os.makedirs(os.path.join(output_dir, "val", "NORMAL"), exist_ok=True)
os.makedirs(os.path.join(output_dir, "val", "PNEUMONIA"), exist_ok=True)
```

```
os.makedirs(os.path.join(output_dir, "test", "NORMAL"), exist_ok=True)
os.makedirs(os.path.join(output_dir, "test", "PNEUMONIA"), exist_ok=True)

# Get the list of image files in each class folder
normal_files = os.listdir(os.path.join(data_dir, "NORMAL"))
pneumonia_files = os.listdir(os.path.join(data_dir, "PNEUMONIA"))

# Shuffle the lists to randomize the order
random.shuffle(normal_files)
random.shuffle(pneumonia_files)

# Calculate the number of images for each split
num_normal = len(normal_files)
num_pneumonia = len(pneumonia_files)

num_train_normal = int(num_normal * train_ratio)
num_train_pneumonia = int(num_pneumonia * train_ratio)

num_val_normal = int(num_normal * val_ratio)
num_val_pneumonia = int(num_pneumonia * val_ratio)

num_test_normal = int(num_normal * test_ratio)
num_test_pneumonia = int(num_pneumonia * test_ratio)

# Copy the image files to the output directories for each split
for i, file in enumerate(normal_files):
    src_path = os.path.join(data_dir, "NORMAL", file)
    if i < num_train_normal:
        dst_path = os.path.join(output_dir, "train", "NORMAL", file)
    elif i < num_train_normal + num_val_normal:
        dst_path = os.path.join(output_dir, "val", "NORMAL", file)
    else:
        dst_path = os.path.join(output_dir, "test", "NORMAL", file)
    shutil.copy(src_path, dst_path)

for i, file in enumerate(pneumonia_files):
    src_path = os.path.join(data_dir, "PNEUMONIA", file)
    if i < num_train_pneumonia:
        dst_path = os.path.join(output_dir, "train", "PNEUMONIA", file)
    elif i < num_train_pneumonia + num_val_pneumonia:
        dst_path = os.path.join(output_dir, "val", "PNEUMONIA", file)
    else:
        dst_path = os.path.join(output_dir, "test", "PNEUMONIA", file)
    shutil.copy(src_path, dst_path)

print("Data has been split into train, val, and test sets.")
```

```
train_dir='/content/FinalDataset/train'
test_dir = '/content/FinalDataset/test'
val_dir = '/content/FinalDataset/val'
train_normal_dir = os.path.join(train_dir, 'NORMAL')

train_pen_dir = os.path.join(train_dir, 'PNEUMONIA')

validation_normal_dir = os.path.join(val_dir, 'NORMAL')

validation_pen_dir = os.path.join(val_dir, 'PNEUMONIA')

test_normal_dir = os.path.join(test_dir, 'NORMAL')

test_pen_dir = os.path.join(test_dir, 'PNEUMONIA')
print('total training NORMAL images:', len(os.listdir(train_normal_dir)))
print('total training PNEUMONIA images:', len(os.listdir(train_pen_dir)))
print('total validation NORMAL images:',
len(os.listdir(validation_normal_dir)))
print('total validation PNEUMONIA images:',
len(os.listdir(validation_pen_dir)))
print('total test NORMAL images:', len(os.listdir(test_normal_dir)))
print('total test PNEUMONIA images:', len(os.listdir(test_pen_dir)))
```

حال ما مجموعه داده ای داریم که به نسبت های خواسته شده تقسیم بندی شده است.

```
total training NORMAL images: 949
total training PNEUMONIA images: 2563
total validation NORMAL images: 316
total validation PNEUMONIA images: 854
total test NORMAL images: 318
total test PNEUMONIA images: 856
```

در این تقسیم بندی ما داده هایمان را به سه بخش آموزش، ارزیابی و اعتبارسنجی تقسیم کردیم. حال مدل را بر روی داده های آموزش، آموزش می دهیم و با داده های اعتبارسنجی بررسی می کنیم که آیا مدل به خوبی آموزش دیده است یا خیر. در همین حین داده های ارزیابی یا تست را به هیچ وجه به مدل نشان نمی دهیم تا دچار نشتی داده نشویم. تحلیل های لازمه را بر روی نتایج حاصل از داده های آموزش و اعتبارسنجی انجام می دهیم و در پایان برای بررسی عملکرد مدل، داده های ارزیابی را به مدل می دهیم تا عملکرد را بررسی کند. دقت می کنیم که نتیجه حاصل از عملکرد مدل روی داده های ارزیابی در گزارشات نهایی اعلام می شود.

در ادامه با استفاده از کلاس ImageDataGenerator که در کتابخانه کراس وجود دارد عملیات داده افزایی (مطابق با جدول ارائه شده در مقاله و fill_mode='nearest' که بر اساس نزدیک ترین نقطه، نقاط مبهمش را پر می کند را تعریف کرده و نحوه نرمال سازی عکس ها را نیز نرمال تعریف می کنیم تا بین [۰,۱] قرار بگیرند.

```
train_datagen = ImageDataGenerator(rescale=1./255,
                                   rotation_range=30,
                                   width_shift_range=0.2,
                                   height_shift_range=0.2,
                                   shear_range=0.2,
                                   horizontal_flip=True,
```

```
                                    zoom_range=0.2,
                                    fill_mode='nearest')

val_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)
```

توجه می کنیم که برای داده های ارزیابی و اعتبار سنجی نیازی به داده افزایی وجود ندارد.

سپس از متد flow_from_directory برای تولید دسته ای داده ها از دایرکتوری های مورد نظر استفاده می کنیم. برای هر قسمت آدرس دایرکتوری مربوطه داده شده است و ابعاد تصویر به ۱۲۸*۱۲۸ تغییر خواهد کرد همچنین در اجرای اول دسته های ۶۴ تایی در نظر گرفته شده و در اجرای دوم دسته های ۱۲۸ تایی در نظر گرفته شده است و چون ما ۲ کلاس داریم از class_mode=’binary’ استفاده کرده ایم. توجه شود چون می خواستیم در عملیات ارزیابی (تست) دسته های تکی داشته باشیم، مقدار دسته ها را برابر با تعداد عکس یعنی ۱۱۷۴ گذاشته ایم.

```
train_generator=train_datagen.flow_from_directory('/content/FinalDataset/train,
                                                  target_size=(128, 128),
                                                  batch_size=128 #OR 64,

                                                  class_mode='binary')
val_generator = val_datagen.flow_from_directory('/content/FinalDataset/val',
                                                target_size=(128, 128),
                                                batch_size=128  #OR 64,

                                                class_mode='binary')
test_generator = test_datagen.flow_from_directory('/content/FinalDataset/test',
                                                  target_size=(128, 128),
                                                  batch_size=1174,

                                                  class_mode='binary')
```

حال عملیات پیش پردازش ما به اتمام رسیده و داده های حاصل از این مرحله را می توان در مراحل بعدی استفاده کرد.

سوال ۵ :

یادگیری انتقالی یعنی از مدلی که قبلا روی کلاس های دیگر آموزش دیده را برای Task جدید و نسبتا مرتبط استفاده کنیم. گاهی ما داده های کافی برای آموزش یک مدل نداریم در نتیجه به سراغ مدل های از پیش آموزش دیده می رویم و آن ها را در مدل خودمان استفاده می کنیم. اما چطور ؟ فرض کنید می خواهیم مدلی بسازیم که وظیفه آن طبقه بندی سگ و گربه است. همچنین می دانیم مدل های از پیش آموزش داده شده ای مثل Resnet AlexNet و ... وجود دارند که بر روی مجموعه داده ی Imagenet که شامل میلیون ها تصویر و ۱۰۰۰ کلاس می باشند آموزش دیده اند. می دانیم که در CNN ویژگی ها سلسله مراتبی به دست می آیند در نتیجه لایه های اول ویژگی های ساده تر مثل خط و لبه و لایه های آخر ویژگی های پیچیده تر مثل کله ی گربه یا کله ی سگ و ... را به دست می آورند. فرض کنیم از معماری AlexNet میخواهیم استفاده کنیم. می دانیم که این معماری ۸ لایه دارد که شامل ۵ لایه کانولوشون و ۳ لایه FC دارد که این ۳ لایه وظیفه ی طبقه بندی را در ۱۰۰۰ کلاس بر عهده دارند. در یادگیری انتقالی ما این ۳ لایه را دور می ریزیم و FC های خودمان را جای آن می گذاریم. یعنی از معماری ها و مدل های از پیش آموزش داده شده به عنوان یک feature extractor استفاده می کنیم سپس مجموعه داده خودمان را

یک بار از این مدل عبور می دیم تا ویژگی های مفید برای مساله ما را به دست بیاورد. سپس در پایان فیچر ها را از لایه های classifier خودمان عبور می دهیم تا نتایج را بررسی کنیم. لزوما اجباری بر دور ریختن لایه های FC نیست در مواردی می توان لایه های CNN را نیز دور ریخت و از آنجا به ادامه آموزش پرداخت. در نهایت اگر مجموعه داده کوچکی داریم بهتر است از انتقال یادگیری استفاده کنیم تا کارایی بهتری داشته باشیم. برای مثال در مثال طبقه بندی گربه و سگ ، استفاده از انتقال یادگیری دقت طبقه بندی را مقدار قابل توجه ای بهتر می کند.

سوال ۶ :

در مقاله اشاره شده که از EfficientNet استفاده شده است، که از بلوک‌های Mobile Inverted Bottleneck Conv (MBconv)و squeeze-and-excitation (SE) تشکیل شده است. بلوک MBconvیک ساختار کانولوشن معکوس است که دقت شبکه را حفظ و پیچیدگی محاسباتی را کاهش می‌دهد. بلوک SE اهمیت کانال‌ها را در یادگیری مشخص می‌کند و با تمرکز بر کانال‌های مهم، خروجی را بهبود می‌بخشد. ایده اصلی EfficientNet بر این است که از یک مدل بنچمارک با کیفیت و فشرده استفاده کرده تا با استفاده از تعداد مشخصی از ضرایب مقیاس دهی، به طور پیوسته تمامی پارامترهای آن را ارزیابی کند. این مقاله از این مدل پیش آموزش دیده با مجموعه داده Imagenet استفاده کرده است. دلیل اصلی استفاده از EfficientNet دستیابی به عملکرد مطلوب در وظایف بینایی کامپیوتر و در عین حال به حداقل رساندن نیازهای محاسباتی و حافظه است. این مدل به‌عنوان یک مدل پیش‌آموزش‌دیده برای یادگیری انتقالی قابل استفاده است و نیاز به داده‌های آموزشی و زمان آموزش را کاهش می‌دهد . EfficientNet با روش‌های افزایش داده‌های آموزشی و جلوگیری از بیش‌برازش، مشکل جمع‌آوری داده‌های وسیع را برطرف می‌کند. لایه‌های اولیه این مدل برای استخراج ویژگی‌های سطح پایین ثابت نگه داشته شده و لایه‌های انتهایی برای تنظیم دقیق اصلاح شده‌اند تا عملکرد مدل بر روی تصاویر اشعه ایکس قفسه سینه بهبود یابد. در این مقاله، لایه های اولیه EfficientNet ثابت (اصطلاحا فریز) نگه داشته و لایه های انتهایی را اصلاح کرده است. در شبکه های کانولوشنی لایه های ابتدایی ویژگی های سطح پایین مثل خط و لبه ها را استخراج می کنند و لایه های انتهایی ویژگی های سطح بالا را تشخیص می دهند. در این مقاله ابتدا لایه های مربوط به طبقه بندی ای که روی مجموعه imagenet ساخته شده را دور میریزیم و طبقه بند خاص خود را تعریف می کنیم که در ادامه به آن خواهیم پرداخت. سپس لایه های ابتدایی مدل EfficientNet را ثابت نگه می داریم چرا که با توجه به مساله مورد نظر ما لایه های اولیه ویژگی های مفیدی را برای ما پیدا میکنند اما سایر لایه ها که ویژگی ها سطح بالا و مربوط به مجموعه داده imagenet را استخراج می کنند را Fine-Tune میکنیم تا ویژگی های مفید برای مجموعه داده فعلی ما استخراج شود. در پایان لایه های جدیدی را که در زیر به آن ها می پردازیم اضافه می کنیم تا طبقه بندی را انجام دهند.

Global Average Pooling ۲D : هدف استفاده از این لایه در شبکه‌های عصبی کانولوشنی این است که یک لایه‌ی فشرده‌تر و با ابعاد کمتر در انتهای شبکه ایجاد کند که به کاهش تعداد پارامترهای مدل و جلوگیری از بیش‌برازش کمک می‌کند. این لایه یک میانگین‌گیری ساده بر روی تمام ویژگی‌های فضایی هر کانال انجام می‌دهد، به طوری که اندازه خروجی به تعداد کانال‌های ورودی کاهش می‌یابد.

Dense Layer : در این لایه ها هر نورون ورودی به همه ی نورون های خروجی متصل می شود که به عنوان تماما متصل (FC) از آن ها نام می بریم. یک ترکیب خطی بر روی ورودی ها اعمال می شود و سپس به تابع فعال ساز داده می شود که در این مقاله ReLU استفاده شده است. این تابع فعال ساز برای خروجی های منفی، صفر و مقادیر مثبت را بدون تغییر نگه می دارد. لازم به ذکر است در لایه آخر با توجه به اینکه مساله ما یک مساله دودویی است و بین ۲ کلاس تصمیم گیری رخ می دهد از یک نورون و تابع فعال ساز sigmoid استفاده شده است.

Dropout: این تکنیک برای جلوگیری از بیش‌برازش در شبکه‌های عصبی استفاده می‌شود. در این روش، طی فرآیند آموزش، برخی گره‌ها به صورت تصادفی خاموش می‌شوند. این خاموشی تصادفی باعث می‌شود شبکه با اطلاعات کمتری آموزش ببیند. در شبکه‌های عصبی با پارامترهای زیاد، بسیاری از پارامترها ممکن است غیرضروری باشند و در برخی موارد، داده‌های آموزشی نیز ممکن است کافی نباشد. این موارد می‌توانند منجر به بیش‌برازش شوند که به معنی عملکرد خوب روی داده‌های آموزش و عملکرد ضعیف روی داده‌های آزمون است.

پیاده سازی شبکه

در این مرحله به ساخت مدل می پردازیم. در ابتدا با توجه به پیاده سازی مقاله از مدل EfficientNetB۲ استفاده می کنیم. ابتدا مدل را لود می کنیم.

```
from keras.applications.efficientnet import EfficientNetB2
```

سپس یک مدل پایه با وزن های imagenet و سایز ورودی ۳*۱۲۸*۱۲۸ می سازیم و با پارامتر include_top='false' لایه های مربوط به طبقه بندی این مدل را که بر روی ۱۰۰۰ کلاس imagenet تعریف شده است را دور می ریزیم.

```
base_model = EfficientNetB2(weights='imagenet', include_top=False,
input_shape=(128, 128, 3))
```

سپس هم می توانیم با رویکرد Sequential مدل را تعریف کنیم و هم از طریق Functional API که با توجه به عملیات Fine-Tune مد نظر از رویکرد دوم استفاده می کنیم. ابتدا تمام لایه های مدل EfficientNetB۲ را می گذاریم و بعد از یک لایه GAP قرار می دهیم. در ادامه یک لایه تمام متصل با ۱۲۸ نورون و تابع فعال ساز ReLU تعریف می کنیم. در لایه بعدی یک لایه Dropout با احتمال ۰.۳ قرار می دهیم و سپس یک لایه تمام متصل دیگر با ۶۴ نورون و تابع فعال ساز ReLU قرار می دهیم. سپس یک لایه Dropout دیگر با احتمال ۰.۲ قرار می دهیم. در لایه پایانی یک لایه تمام متصل تک نورونی با تابع فعال ساز Sigmoid قرار می دهیم. در این لایه عمل پیش بینی رخ خواهد داد که اگر مقدار خروجی این نورون بالای ۰.۵ باشد Pneumonia و اگر زیر این مقدار باشد Normal خواهد بود. و در نهایت مدل را با ورودی های مدل پایه و خروجی را برابر با لایه پیش بینی (لایه آخر) تعریف می کنیم.

```
'''
# Sequential Model
model = models.Sequential()
model.add(base_model)
model.add(GlobalAveragePooling2D())
model.add(layers.Dense(128, activation='relu'))
model.add(Dropout(0.3))
model.add(layers.Dense(64, activation='relu'))
model.add(Dropout(0.2))
model.add(layers.Dense(1, activation='sigmoid'))
'''
# Or Functional API
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(128, activation='relu')(x)
x = Dropout(0.3)(x)
x = Dense(64, activation='relu')(x)
x = Dropout(0.2)(x)
predictions = Dense(1, activation='sigmoid')(x)
model = Model(inputs=base_model.input, outputs=predictions)
```

در ادامه قابل آموزش بودن مدل پایه را برابر با False قرار می دهیم تا این لایه ها آموزش نبینند و وزن های آن تغییر نکند. سپس یک خلاصه از مدل می گیریم تا وضعیت مدل را بررسی کنیم.
مدل فعلی شرایط زیر را دارا می باشد:

```
Total params: 7957242 (30.35 MB)
Trainable params: 188673 (737.00 KB)
Non-trainable params: 7768569 (29.63 MB)
```

مقدار پارامتر های قابل آموزش در این مدل با مدل مورد استفاده در مقاله بسیار متفاوت است. پس ابتدا تمام لایه های مدل پایه را قابل آموزش می کنیم و سپس تغییرات زیر را در مدل اعمال می کنیم.
الف) یک بار با صحیح و خطا سعی می کنیم تعداد لایه های فریز شده مدل پایه استفاده شده در مقاله را حدس زده سپس مدل پایه را بر اساس آن فریز کنیم. البته احتمالا راه های مناسب تری نیز وجود داشته باشد که بنده از آن آگاهی ندارم. نتیجه این عملیات این شد که ۱۰ لایه اول مدل پایه را فریز کرده و مابقی لایه ها را قابل آموزش کردیم . در ادامه خلاصه ای از مدل را خواهید دید.

```
base_model.trainable = True
print("Number of layers in the base model: ", len(base_model.layers)) #Output = 340
fine_tune_at = 10
for layer in base_model.layers[:fine_tune_at]:
  layer.trainable = False
```

```
Total params: 7957242 (30.35 MB)
 Trainable params: 7888387 (30.09 MB)
Non-trainable params: 68855 (268.97 KB)
```

تعداد پارامتر های قابل آموزش این مدل بسیار شبیه به مدل استفاده شده در مقاله است. نتایج مربوط به این مدل تحت عنوان مدل ۱ بررسی خواهد شد.

ب) در این مرحله ۷۰ درصد از مدل پایه که دارای ۳۴۰ لایه است(برابر با ۲۳۸) را فریز کرده و ۳۰ درصد ما بقی را قابل آموزش می کنیم.

```
fine_tune_at = int(len(base_model.layers)*0.7)
for layer in base_model.layers[:fine_tune_at]:
  layer.trainable = False
```

```
Total params: 7957242 (30.35 MB)
Trainable params: 6383963 (24.35 MB)
Non-trainable params: 1573279 (6.00 MB)
```

نتایج مربوط به این مدل تحت عنوان مدل ۲ بررسی خواهد شد.

در ادامه با توجه به نامتوازن بودن کلاس های مجموعه داده، از تابع compute_class_weight که در کتابخانه scikit-learn وجود دارد استفاده می کنیم تا وزن های کلاسی را محاسبه و تنظیم کنیم تا مدل به درستی آموزش ببیند. در این تابع از پارامتر های مختلفی استفاده شده که در ادامه به آن ها می پردازیم.
class_weight='balanced' : این پارامتر به compute_class_weight می‌گوید که وزن‌های کلاسی باید به صورت متوازن محاسبه شوند. این به معنای این است که وزن‌ها به گونه‌ای تنظیم می‌شوند که تاثیر تعداد نمونه‌های هر کلاس در مجموعه داده را متعادل کنند.
classes=np.unique(train_generator.classes): این پارامتر مجموعه‌ای از کلاس‌های یکتا در داده‌های آموزشی را مشخص می‌کند. np.unique(train_generator.classes) تمام کلاس‌های موجود در داده‌های آموزشی را استخراج می‌کند.
y=train_generator.classes: این پارامتر شامل لیستی از کلاس‌های مربوط به هر نمونه در داده‌های آموزشی است.

سپس وزن های محاسبه شده را به صورت جفت های (index,weight) بر می گرداند که در یک دیکشنری ذخیره می کنیم.

```
import numpy as np
from sklearn.utils.class_weight import compute_class_weight
class_weights = compute_class_weight(
    class_weight='balanced',
    classes=np.unique(train_generator.classes),
    y=train_generator.classes
)
class_weights = dict(enumerate(class_weights))
```

سپس با استفاده از Callback های موجود در کراس ، ۳ کالبک زیر را تعریف می کنیم.

ModelCheckpoint : این Callback در هر ایپاک مدل را ذخیره می کند اگر که مقدار val_loss از ایپاک قبلی بهتر شده باشد. مدلی که قرار است ذخیره شود به نام best_model.h۵ خواهد بود و با پارامتر save_best_only=True فقط بهترین مدل را ذخیره خواهیم کرد.پارامتر save_weights_only تعیین می کند که فقط وزن ها را ذخیره کنیم. با پارامتر Monitor تعیین می کنیم چه معیاری بررسی شود که در اینجا از val_loss استفاده شده و با پارامتر mode تعیین می کنیم که بالا بودن ان معیار خوب است یا پایین بودن آن.

```
checkpoint = ModelCheckpoint('best_model.h5', save_best_only=True,
save_weights_only=True, monitor='val_loss', mode='min', verbose=1)
```

ReduceLROnPlateau: با استفاده از این Callback نرخ یادگیری را در فرآیند آموزش بررسی می کنیم تا اگر این عملکرد خوب نباشد نرخ یادگیری را به صورت خودکار کاهش دهیم.معیار مورد استفاده در اینجا val_loss است و با پارامتر factor تعیین می کنیم که هربار چقدر نرخ یادگیری را کاهش دهد که ما ۰.۲ استفاده کرده ایم. پارامتر patience تعیین می کند که چند ایپاک منتظر بماند تا عملکرد مدل بهبود یابد و در صورت عدم بهبود فرایند آموزش مقدار نرخ یادگیری را کاهش دهد. در اینجا ۱۰ ایپاک منتظر می ماند تا عملکرد مدل را بهبود یابد و در صورت عدم بهبودی ، نرخ یادگیری را فعلی را در ۰.۲ ضرب کرده و نرخ یادگیری را به حاصل این ضرب کاهش می دهد. پارامتر min_lr هم تعیین می کند کم ترین مقدار ممکن برای نرخ یادگیری چقدر خواهد بود که ما ۰.۰۰۰۰۱ قرار داده ایم.

```
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2,
patience=10, min_lr=0.00001, verbose=1)
```

EarlyStopping: از این Callback استفاده می کنیم تا در صورتی که عملکرد مدل رو به ضعیفی باشد از ادامه آموزش جلوگیری کنیم تا دچار بیش برازش نشویم. با استفاده از پارامتر monitor مشخص کرده ایم که معیار val_loss را بررسی کند و با پارامتر patience تعیین کردیم که اگر ۱۰ ایپاک معیار val_loss بهتر از بهترین عملکرد نشد آموزش را متوقف کند. mode نیز تعیین می کند که برای معیار انتخابی بیشتر بودن بهتر است یا کم تر بودن.

```
early_stop = EarlyStopping(monitor='val_loss', patience=10,mode='min')
```

حال مدل را با بهینه ساز adam و نرخ یادگیری ۰.۰۰۰۱ به همراه تابع ضرر binary_crossentropy به دلیل دو کلاسه بودن مجموعه داده و معیار accuracy کامپایل می کنیم.

```
model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.0001),
loss='binary_crossentropy', metrics=['accuracy'])
```

سپس عملیات آموزش را بر روی مجموعه داده آموزش و اعتبارسنجی با ۳۰ ایپاک و دسته های ۱۲۸ تایی به همراه callback هایی که تعریف کرده ایم انجام می دهیم.

```
history = model.fit(train_generator, validation_data=val_generator,
epochs=30,
batch_size=128,
callbacks=[checkpoint, reduce_lr, early_stop],class_weight=class_weights
)
```

جدول۱ مدل ها و پارامتر های آن

	اندازه دسته های عکس ها	ابعاد ورودی	لایه های قابل آموزش مدل پایه	بهینه ساز و نرخ یادگیری	ReduceLrOnPlateau	EarlyStopping	Loss	Epoch
مدل ۱	۶۴	۱۲۸*۱۲۸	۳۳۰	Adam,۰.۰۰۰۱	۰.۲ with patience ۱۰	Patience ۱۰	Binary-crossentropy	۳۰
مدل ۲	۱۲۸	۱۲۸*۱۲۸	۱۰۲	Adam,۰.۰۰۰۱	۰.۲ with patience ۱۰	Patience ۱۰	Binary-crossentropy	۳۰

نتایج پیاده سازی

سوال ۱ :

حال به خواسته های سوال ۱ از این قسمت می پردازیم. یادآوری می کنیم که مدل ۱ ما دسته های ۶۴ تایی از عکس را تولید می کند و مدل ۲ دسته های ۱۲۸ تایی و همچنین مدل ۱ فقط ۱۰ لایه اول مدل پایه را فریز کرده و مدل ۲ ۷۰ درصد ابتدایی شبکه فریز شده است .

مدل ۱ :

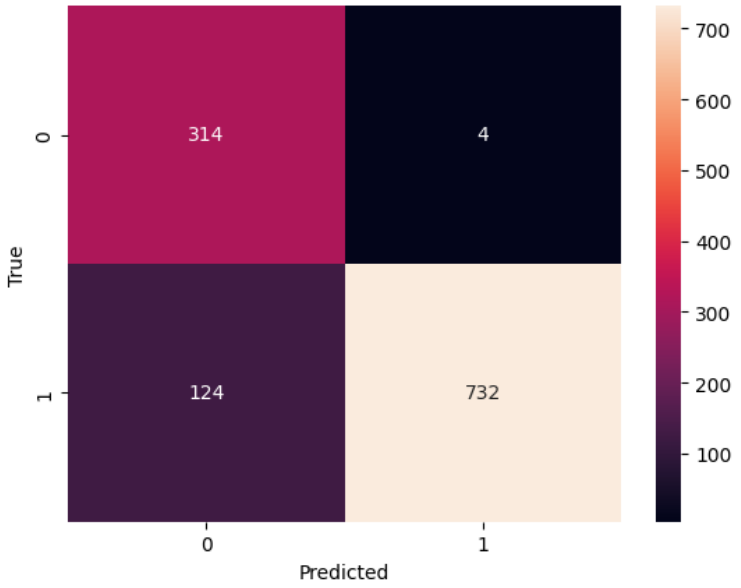
```
Test loss: 0.3850992023944855
Test accuracy: 0.8909710645675659
Test AUC : 0.9903529653794158
Test recall: 0.8551401869158879
Test f1 : 0.9195979899497487
Test precision : 0.9945652173913043
```

مدل ۲ :

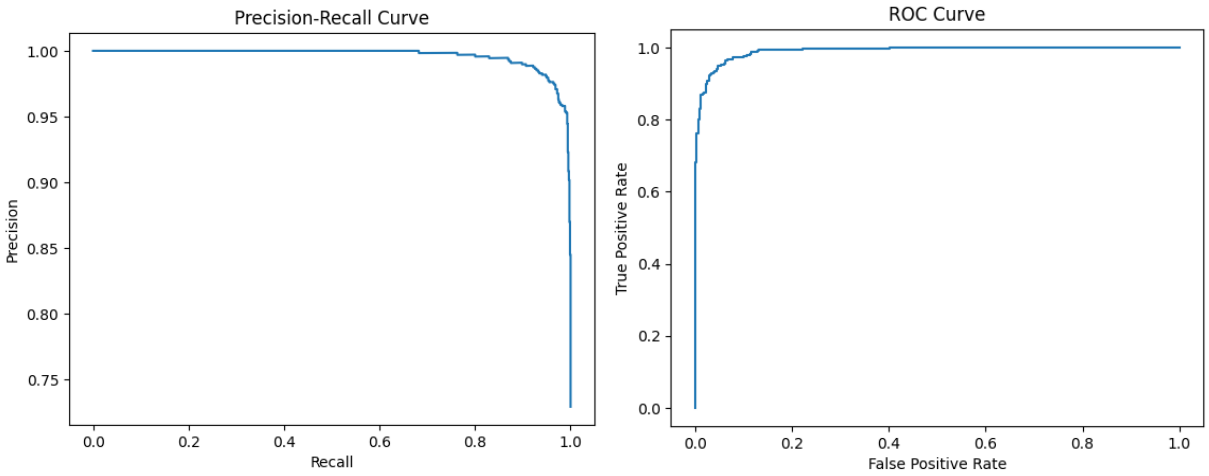
```
Test loss: 0.3084425628185272
Test accuracy: 0.8798977732658386
Test AUC : 0.9338924645859048
Test recall: 0.9264018691588785
Test f1 : 0.918355529820498
Test precision : 0.9104477611940298
```

سوال ۲ :

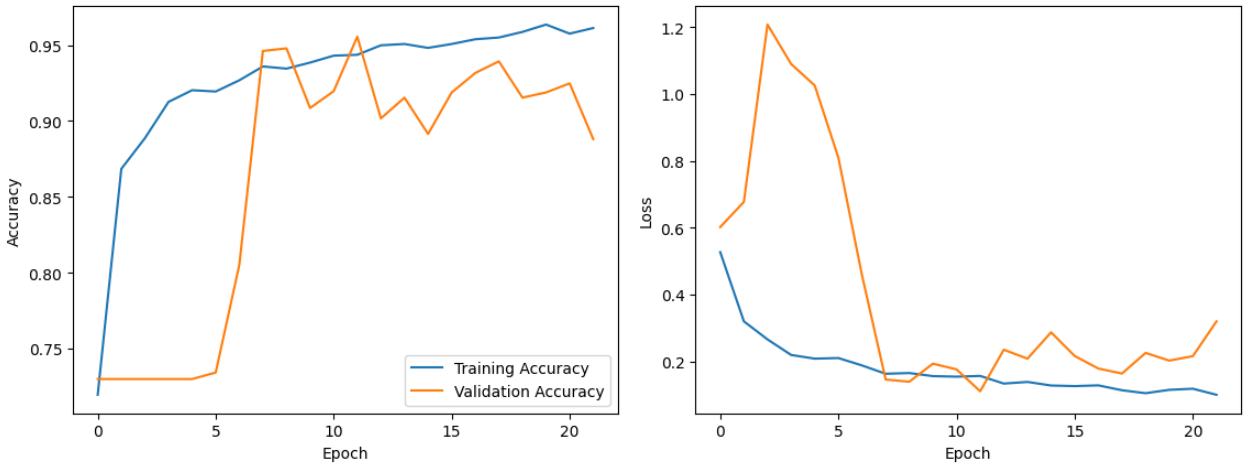
نمودار های مربوط به مدل ۱ :



شکل۱ نمودار **confusion matrix** مدل ۱

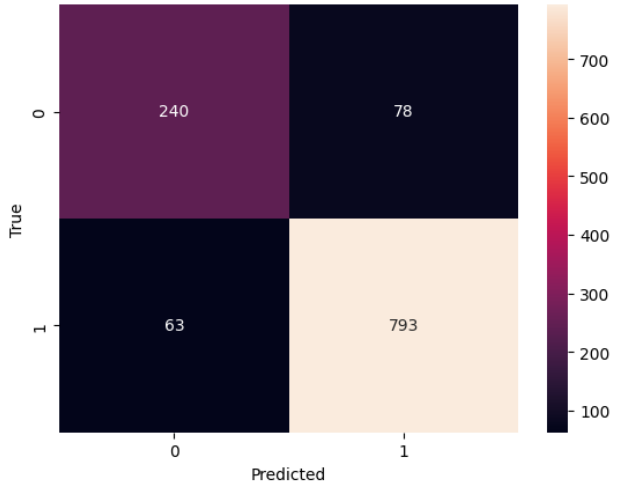


شکل۲ نمودار **ROC** و **Precision-recall** مدل ۱

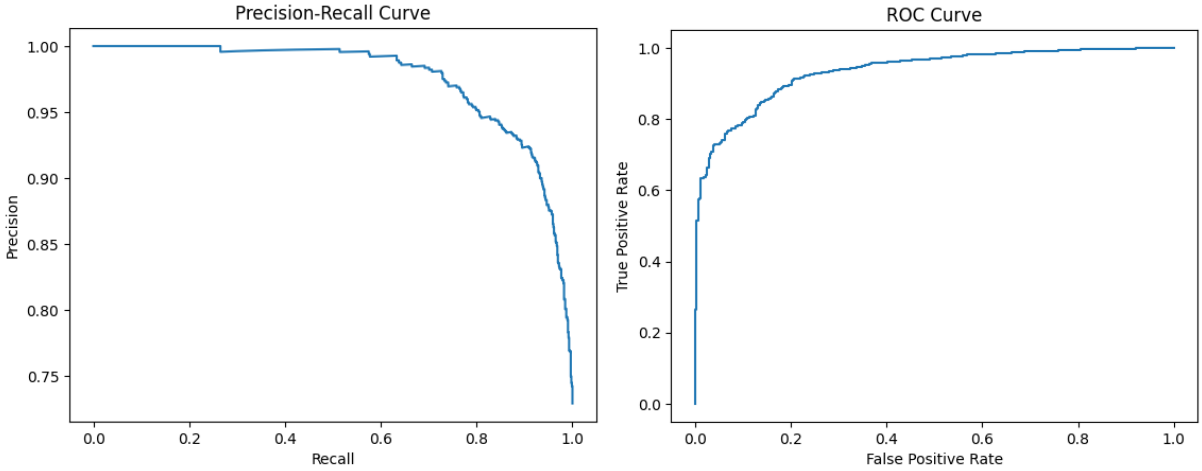


شکل۳ نمودار دقت و اتلاف مدل ۱

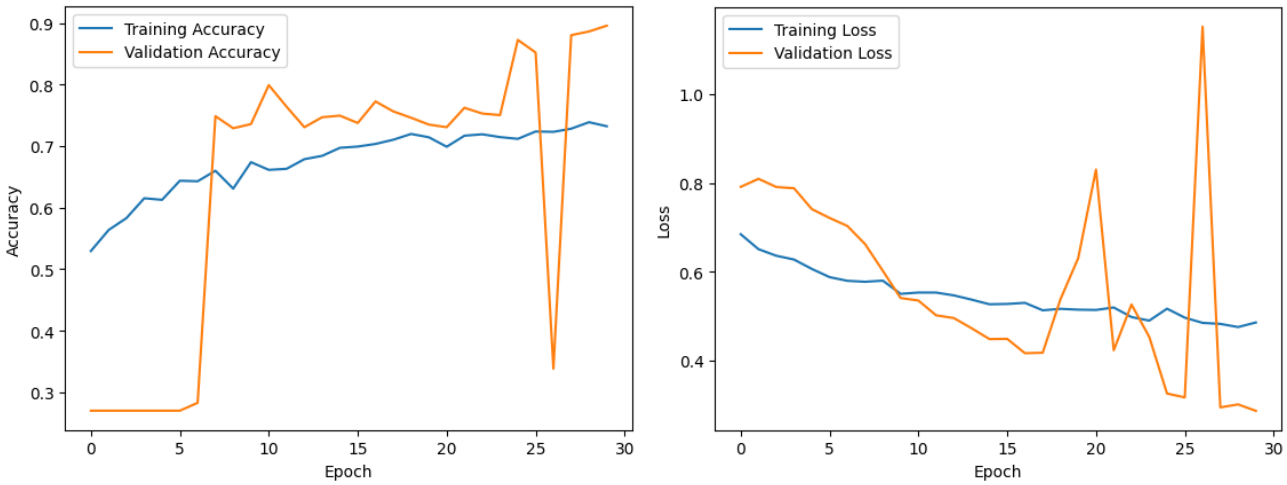
نمودار های مربوط به مدل ۲ :



شکل۴ نمودار **Confusion matrix** مدل ۲



شکل۵ نمودار **ROC** و **Precision-Recall** مدل ۲



شکل۶ نمودار دقت و اتلاف مدل ۲

سوال ۳ :

در بخش اول به بررسی Confusion Matrix می پردازیم که به جدول زیر می رسیم.

جدول ۲ مقایسه Confusion matrix ها

عنوان	مدل ۱	مدل ۲
TN	۳۱۴	۲۴۰
FP	۴	۷۸
FN	۱۲۴	۶۳
TP	۷۳۲	۷۹۳

در مدل اول تعداد بیشتری نمونه سالم را به درستی تشخیص داده است(۳۱۴ نمونه) و تعداد کمی از نمونه های سالم را ناسالم(۴ نمونه) تشخیص داده است. همچنین تعداد ۱۲۴ نمونه بیمار را به اشتباه سالم و ۷۳۲ نمونه بیمار را به درستی تشخیص داده است. در مدل دوم ۲۴۰ نمونه سالم را به درستی تشخیص داده و ۷۸ نمونه سالم را بیمار تشخیص داده است. همچنین ۶۳ بیمار را سالم را تشخیص و ۷۹۳ نمونه بیمار را به درستی تشخیص داده است. مدل اول FP کمتری دارد (نسبت ۴ به ۷۸) یعنی تعداد افراد سالمی که بیمار تشخیص داده بهتر است اما FN بیشتری دارد (نسبت ۱۲۴ به ۶۳) یعنی تعداد بیمارانی که سالم تشخیص داده است بیشتر است. با توجه به مساله و بیماری برای ما مهم است که بیمار به اشتباه سالم تشخیص داده نشود پس می توان گفت در این زمینه مدل ۲ بهتر عمل کرده است.

در بخش بعدی به تحلیل نمودار های ROC میپردازیم. این نمودار در محور عمودی TPR یا Sensitivity را مشخص کرده که بیانگر این است که مدل چه تعداد از موارد مثبت را درست تشخیص داده است. در محور افقی FPR قرار گرفته که نشان دهنده این است که چه تعداد از موارد منفی را اشتباه تشخیص داده است.
در مدل اول، نمودار به شدت به سمت محور عمودی منحرف شده که بیانگر دقت بالای مدل است. مدل اول تعداد خوبی از موارد مثبت را درست تشخیص داده است. مدل دوم هم عملکرد مشابهی دارد اما به خوبی مدل اول نمی باشد.
نمودار Precision Recall : در این نمودار محور افقی recall و محور عمودی precision قرار گرفته است و برای مجموعه داده های نامتوازن استفاده می شود. بهترین حالت این است که از بالا و سمت چپ عبور کند. در این نمودار در مدل اول عملکرد مناسب تری نسبت به مدل دوم مشاهده می شود .

نمودار دقت و اتلاف : هرچه نمودارِدقت افزایشی تر باشد وهرچه نموداراتلاف کاهشی ترباشد عملکرد مدل بهتر است. گاهی ممکن است در عین کاهش اتلاف داده های آموزش، این معیار در داده های اعتبارسنجی کاهش پیدا نکند که در این حالت مدل دچار بیش برازش شده است. در پیاده سازی از روش های مختلفی با کمک Callback ها سعی کردیم این مشکل را بر طرف کنیم. با توجه به این نمودار ها برای هر دو مدل می توان گفت مدل اول آموزش بهتری داشته است ولی در مدل دوم دقت آموزش کم تر از مدل اول است که می تواند دلایل مختلفی داشته باشد. در کل با توجه به پیچیدگی مساله و مدل به نظر می رسد مدل اول نسبتا خوب آموزش دیده است.

معیار های ارزیابی :

معیار accuracy : این معیار نشان می دهد چه تعداد از کل داده ها درست پیش بینی شده است.

معیار precision : این معیار بیان می کند که چه تعداد از پیش بینی های به عنوان مثبت بوده است.

معیار recall : این معیار نشان می دهد که چه تعداد از داده های مثبت موجود به درستی پیش بینی شده است.

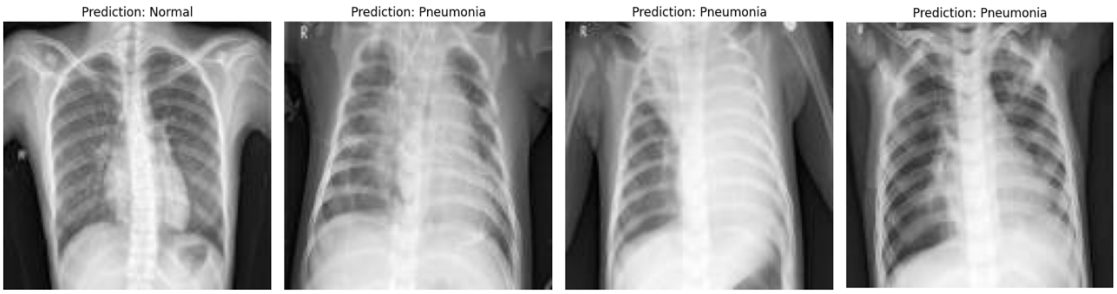
معیار f۱-score : میانگین هندسی precision و recall را نشان می دهد که بهتر از به ۱ نزدیک باشد.

در ادامه جدول مقایسه این معیار ها را خواهیم دید.

جدول ۳ عملکرد مدل ها بر اساس معیار های مختلف

معیار	مقاله	مدل ۱	مدل ۲
Accuracy	0.96	0.89	0.87
Precision	0.97	0.99	0.91
Recall	0.96	0.85	0.92
F1-Score	0.96	0.91	0.91
AUC	0.991	0.990	0.933

با بررسی معیار های مختلف می توان به این نتیجه رسید که مدل اول کمی بهتر از مدل دوم کار کرده است.
در نهایت ۴ عکس تصادفی (۱ مورد نرمال ۳ مورد بیمار) را به هر دو مدل دادیم که مدل اول هر ۴ عکس را صحیح پیش بینی کرد ولی مدل دوم عکس نرمال را بیمار تشخیص داد. در ادامه فقط نتیجه حاصل از مدل ۱ را مشاهده می کنید.



شکل۷ خروجی پیش بینی شده توسط مدل شماره ۱ بر روی ۴ عکس تصادفی