به نام خدا

دانشگاه تهران - دانشگاه صنعتی خواجه نصیرالدین طوسی تهران

دانشکده مهندسی برق و کامپیوتر

طبقه‌بندی تصاویر اشعه ایکس قفسهٔ سینه

نام و نام خانوادگی	محمدجواد احمدی
شمارهٔ دانشجویی	۴۰۱۰۰۰۸۶

فهرست مطالب

فهرست تصاویر

فهرست جداول

پرسش ۲. طبقه‌بندی تصاویر اشعهٔ ایکس قفسهٔ سینه

۱ پاسخ پرسش دوم

توضیح پوشهٔ کدهای طبقه‌بندی تصاویر اشعهٔ ایکس قفسهٔ سینه

کدهای مربوط به این قسمت، علاوه بر پوشهٔ محلی کدها در این لینک گوگل‌کولب آورده شده است. مدل‌های ذخیره‌شده و برخی خروجی‌ها هم از طریق این لینک گوگل‌درایو در دسترس هستند.

۱.۱ آماده‌سازی و پیش‌پردازش داده‌ها

۱.۱.۱ پاسخ قسمت الف

ابعادی که شبکهٔ Efficientnet در ورودی خود می‌پذیرد، بسته به نسخهٔ خاص شبکه متفاوت است. به‌صورت کلی گفته شده که مدل‌های کارآمد آن می‌توانند ورودی‌هایی با اندازه‌های مختلفی از ۲۲۴ تا ۶۷۲ را بپذیرند. کوچک‌ترین نسخهٔ این شبکه (Efficientnet-B0) اندازهٔ ورودی ۲۲۴ را می‌پذیرد و امکان دریافت ورودی با ابعاد ۱۲۸ را هم دارد. به هر حال بنابه آن‌چه در مقاله ذکر شده، ساختار مورد استفاده در این سوال ابعاد ورودی 128×128 می‌پذیرد و به همین دلیل، ورودی تصاویر به این ابعاد پیش‌پردازش می‌شوند.

هم‌چنین مقاله به روش‌های مختلفی برای افزونه‌کردن داده‌ها اشاره کرده است که در جدول ۱ آورده شده است. در ادامه سعی می‌کنیم توضیحاتی در خصوص هر یک از این روش‌ها ارائه دهیم.

جدول ۱: روش‌های تقویت داده مورد استفاده در مقاله

Techniques of Data Augmentation	Values
Re-scale	1.0/255
Shear Range	0.2
Width Shift Range	0.2
Height Shift Range	0.2
Rotation Range	30
Horizontal Flip	True
Zoom Range	0.2

- Re-scale: مقیاس‌بندی مجدد روشی است که در آن مقادیر پیکسلی یک تصویر به محدوده‌ای جدید مقیاس می‌شود. مقدار 1.0/255 نشان می‌دهد که این مقادیر پیکسلی در تصویر بر عدد 255 تقسیم می‌شوند و این باعث می‌شود که مقادیر پیکسل به محدوده‌ای جدبد و بین 0 و 1 مقیاس شوند. این روش کمک می‌کند تا اطمینان داشته باشیم که مقادیر

پیکسلی تصویر در یک مقیاس مشابه هستند و می‌توانند به راحتی توسط شبکهٔ عصبی پردازش شوند. هم‌چنین این روش به مدل کمک می‌کند تا تأثیر شرایط نوری و تغییرات رنگ را بهتر مدیریت کند. با تغییر مقیاس مقادیر پیکسل به محدوده‌ای کوچکتر، نیازهای محاسباتی مدل کاهش می‌یابد و آموزش مدل سریع‌تر، مقاوم‌تر، دقیق‌تر و کارآمدتر می‌شود. این موضوع هم‌چنین به جلوگیری از مسائلی مانند محودشدن گرادیان‌ها و انفجار شیب‌ها که ممکن است زمانی که مقادیر پیکسل خیلی بزرگ هستند، رخ دهد، کمک می‌کند.

- Shear: تغییر موقعیت پیسکل‌های یک تصویر در جهاتی خاص که منتج به یک تصویر برسی یا معوج می‌شود یک روش دیگر برای افرونه‌سازی داده است. مقدار 0.2 نشان‌دهندهٔ حداکثر زاویه بر حسب رادیان است که می‌توان یک تصویر را برش داد. برش به این معنی است که تصویر در امتداد یک محور، عمدتا برای ایجاد یا اصلاح زوایای ادراک، منحرف می‌شود. این روش کمک می‌کند تا رایانه‌ها بتوانند ببینند انسان‌ها چگونه چیزها را از زوایای مختلف می‌بینند. با اعمال روش برش روی داده‌های آموزشی، مدل می‌تواند نسبت به تغییرات شئ، که می‌تواند به دلیل اعوجاج پرسپکتیو یا زوایای دوربین رخ دهد، مقاوم‌تر شود. علاوه بر این، این روش می‌تواند به افزایش تنوع داده‌های آموزشی کمک کند، که این می‌تواند عملکرد تعمیم‌پذیرانهٔ مدل را بهبود بخشد.

- Width Shift: این روش، یک روش افزایش داده است که شامل تغییر موقعیت پیکسل‌ها در یک تصویر به صورت افقی در امتداد عرض تصویر است. در جدول ۱، مقدار 0.2، حداکثر کسری از عرض کل را نشان می‌دهد که یک تصویر را می‌توان به صورت افقی جابجا کرد. در طول عملیات جابجایی عرض، هر پیکسل در یک ردیف بر اساس کسری تصادفی از عرض کل تصویر، مقدار مشخصی جابجا می‌شود. میزان جابجایی در ردیف‌ها متفاوت است، به طوری که این جابجایی در لبه‌ها حداکثر و در مرکز حداقل است. تصویر به دست آمده با حرکت اشیاء در تصویر در جهت افقی، ظاهری تغییر یافته دارد. با اعمال تغییر عرض به داده‌های آموزشی، مدل می‌تواند نسبت به تغییرات در موقعیت اشیاء در تصویر، که می‌تواند به دلیل حرکات دوربین یا تغییر در پرسپکتیو رخ دهد، قوی‌تر شود. علاوه بر این، می‌تواند به افزایش تنوع داده‌های آموزشی کمک کند، که این می‌تواند تعمیم‌پذیری مدل را بهبود بخشد. با اِعمال درجات مختلف تغییر افقی به داده‌های آموزشی، مدل می‌تواند یاد بگیرد که اشیا را از زوایای مختلف و موقعیت‌های مختلف تشخیص دهد.

- Height Shift: این روش، یک روش افزایش داده است که شامل تغییر موقعیت پیکسل‌ها در یک تصویر به‌صورت عمودی در امتداد ارتفاع تصویر است. در جدول ۱، مقدار 0.2، حداکثر کسری از ارتفاع کل را نشان می‌دهد که یک تصویر را می‌توان به صورت عمودی جابجا کرد. در طول عملیات تغییر ارتفاع، هر پیکسل در یک ستون بر اساس کسری تصادفی از ارتفاع کل تصویر، مقدار مشخصی جابجا می‌شود. میزان جابجایی در ستون‌ها متفاوت است، به طوری که در لبه‌ها حداکثر و در مرکز حداقل تغییرات رخ می‌دهد. تصویر به دست آمده با حرکت اشیاء در تصویر در جهت عمودی، ظاهری جابجا شده دارد. این روش برای تقویت داده‌ها برای مدل‌های یادگیری عمیق مفید است. با اعمال جابجایی عمودی در داده‌های آموزشی، مدل می‌تواند نسبت به تغییرات در موقعیت اشیاء در تصویر، که می‌تواند به دلیل حرکات دوربین یا تغییر در پرسپکتیو رخ دهد، مقاوم‌تر شود. علاوه بر این، این روش می‌تواند به افزایش تنوع داده‌های آموزشی کمک کند، که این می‌تواند تعمیم‌پذیری مدل را بهبود بخشد. با اِعمال درجات مختلف تغییر عمودی به داده‌های آموزشی، مدل می‌تواند یاد بگیرد که اشیاء را از زوایای مختلف و موقعیت‌های مختلف تشخیص دهد.

- Rotation: چرخش یک روش افزایش داده است که شامل چرخش تصویر با درجات خاصی است که به طور تصادفی از یک محدودهٔ مشخص انتخاب می شود. در جدول ۱، مقدار 30 حداکثر درجهٔ چرخش مجاز برای تصویر را در جهت عقربه‌های ساعت یا خلاف جهت عقربه‌های ساعت نشان می‌دهد. در حین عملیات چرخش، پیکسل‌های تصویر به دور مرکز آن می‌چرخند. میزان چرخش به طور تصادفی بین -30 تا 30 درجه تغییر می‌کند و در نتیجه یک تصویر چرخشی

با جهت‌گیری جدید ایجاد می‌شود. اشیاء در تصویر به دلیل چرخش در موقعیت‌های مختلف ظاهر می‌شوند، اما شکل و اندازهٔ آن‌ها ثابت می‌ماند. چرخش یک روش مفید برای تقویت داده‌ها برای مدل‌های یادگیری عمیق است. با اعمال چرخش به داده‌های آموزشی، مدل می‌تواند نسبت به تغییرات جهت‌گیری اشیاء در تصویر، که می‌تواند به‌دلیل زوایای دوربین یا اعوجاج پرسپکتیو رخ دهد، مقاوم‌تر شود. علاوه بر این، این روش می‌تواند به افزایش تنوع داده‌های آموزشی کمک کند، که این می‌تواند تعمیم‌پذیری مدل را بهبود بخشد. با اِعمال درجات مختلف چرخش به داده‌های آموزشی، مدل می‌تواند یاد بگیرد که اشیاء را از زوایا و جهت‌گیری‌های مختلف تشخیص دهد.

- Horizontal Flip: این روش شامل چرخاندن تصویر به صورت افقی در امتداد محور عمودی است. در جدول ۱، مقدار True نشان می‌دهد که تصویر در طی فرآیند افزایش داده‌ها به صورت افقی برگردانده می‌شود. در طی عملیات چرخش افقی، پیکسل‌های تصویر در امتداد محور عمودی معکوس می‌شوند. سمت چپ تصویر به سمت راست و سمت راست به سمت چپ تبدیل می‌شود. این منجر به یک تصویر برگردانده می‌شود که یک تصویر آینه‌ای از تصویر اصلی است. اشیاء در تصویر به دلیل چرخش در موقعیت مخالف ظاهر می‌شوند، اما شکل و اندازهٔ آن‌ها ثابت می‌ماند. با اعمال چرخش افقی روی داده‌های آموزشی، مدل می‌تواند نسبت به تغییرات در جهت گیری اشیاء در تصویر، که می‌تواند به دلیل زوایای دوربین یا اعوجاج پرسپکتیو رخ دهد، مقاوم‌تر شود. علاوه بر این، این روش می‌تواند به افزایش تنوع داده‌های آموزشی کمک کند، که این می‌تواند تعمیم‌پذیری مدل را بهبود بخشد.
- Zoom: تغییر بزرگنمایی یک روش افزایش داده است که شامل بزرگنمایی یا کوچکنمایی یک تصویر با یک میزان خاص است. در جدول ۱، مقدار 0.2 نشان‌دهندهٔ حداکثر میزانی است که با آن می‌توان تصویر را بزرگ یا کوچک کرد. در طول عملیات زوم، پیکسل‌های تصویر بسته به فاکتور انتخابی بزرگ یا کوچک می‌شوند. میزان بزرگنمایی به طور تصادفی بین محدودهٔ مشخص شده متفاوت است و در نتیجه یک تصویر بزرگ‌نمایی یا کوچک‌نمایی‌شده با مقیاس جدید ایجاد می‌شود. اشیاء در تصویر بسته به فاکتور بزرگنمایی بزرگتر یا کوچکتر به نظر می‌رسند. با اعمال زوم روی داده‌های آموزشی، مدل می‌تواند نسبت به تغییرات در مقیاس اشیاء در تصویر، که می‌تواند به دلیل فاصلهٔ دوربین یا تغییر وضوح رخ دهد، مقاوم‌تر شود. علاوه بر این، این روش می‌تواند به افزایش تنوع داده‌های آموزشی کمک کند، که این می‌تواند عملکرد تعمیم مدل را بهبود بخشد. با معرفی درجات مختلف زوم به داده‌های آموزشی، مدل می‌تواند تشخیص اشیاء را در مقیاس‌ها و فواصل مختلف بیاموزد.

۲.۱.۱ پاسخ قسمت ب

در گام اول، مجموعه‌داده را از طریق لینک داده‌شده دانلود کرده و روی گوگل‌درایو بارگذاری می‌کنیم. سپس دستوراتی برای بارگیری دیتاست و خروج آن از حالت فشرده اجرا می‌کنیم. فراخوانی این فایل بدون نیاز به Mountکردن و استفاده از gdown در محیط گوگل‌کولب ممکن است با خطا مواجه شود که با استفاده از دستورات زیر مشکل حل می‌شود (منبع). در انتهای دستورات کدی می‌نویسیم تا فایل فشرده، پس از خروج از حالت فشرده و ذخیره در پوشه‌ای روی فضای گوگل‌کولب حذف شود:

```
!pip install --upgrade --no-cache-dir gdown
!gdown 1JwIyR97fXRfaciFjm4NLualTzq2XaDTg

import zipfile
zip_file_path = '/content/archive.zip'
folder_path = '/content/Dataset'
```

```
# Extract the zip file to the specified folder
with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    zip_ref.extractall(folder_path)

import os
file_path = "/content/archive.zip"
if os.path.exists(file_path):
    os.remove(file_path)
    print(f"{file_path} has been deleted successfully.")
else:
    print(f"{file_path} does not exist.")
```

پس از انجام این کار دستورات برنامهٔ ۱ را می‌نویسیم تا متوجه شویم در هر پوشه چه تعداد داده وجود دارد. نتایج در ذیل کدها آورده شده است.

Program 1: Find Numbers of Data Code

```
import os

# Define the path to the parent folder
parent_folder_path = "/content/Dataset"

# Recursively iterate through all subfolders and files in the parent folder
for dirpath, dirnames, filenames in os.walk(parent_folder_path):
    # Get the number of JPEG files in the current directory
    jpeg_count = sum(1 for filename in filenames if filename.lower().endswith('.jpeg'))

    # Print out the results for the current directory
    if jpeg_count > 0:
        # Get the relative path to the current directory
        relative_path = os.path.relpath(dirpath, parent_folder_path)

        # Print out the results
        print(f"{jpeg_count} (.jpeg) files are in {os.path.join(parent_folder_path, relative_path
    )}")

--------------------------------------------------------------------

8 (.jpeg) files are in /content/Dataset/chest_xray/__MACOSX/chest_xray/val/PNEUMONIA
8 (.jpeg) files are in /content/Dataset/chest_xray/__MACOSX/chest_xray/val/NORMAL
3875 (.jpeg) files are in /content/Dataset/chest_xray/__MACOSX/chest_xray/train/PNEUMONIA
1341 (.jpeg) files are in /content/Dataset/chest_xray/__MACOSX/chest_xray/train/NORMAL
390 (.jpeg) files are in /content/Dataset/chest_xray/__MACOSX/chest_xray/test/PNEUMONIA
234 (.jpeg) files are in /content/Dataset/chest_xray/__MACOSX/chest_xray/test/NORMAL
8 (.jpeg) files are in /content/Dataset/chest_xray/val/PNEUMONIA
8 (.jpeg) files are in /content/Dataset/chest_xray/val/NORMAL
3875 (.jpeg) files are in /content/Dataset/chest_xray/train/PNEUMONIA
```

```
1341 (.jpeg) files are in /content/Dataset/chest_xray/train/NORMAL
8 (.jpeg) files are in /content/Dataset/chest_xray/chest_xray/val/PNEUMONIA
8 (.jpeg) files are in /content/Dataset/chest_xray/chest_xray/val/NORMAL
3875 (.jpeg) files are in /content/Dataset/chest_xray/chest_xray/train/PNEUMONIA
1341 (.jpeg) files are in /content/Dataset/chest_xray/chest_xray/train/NORMAL
390 (.jpeg) files are in /content/Dataset/chest_xray/chest_xray/test/PNEUMONIA
234 (.jpeg) files are in /content/Dataset/chest_xray/chest_xray/test/NORMAL
390 (.jpeg) files are in /content/Dataset/chest_xray/test/PNEUMONIA
234 (.jpeg) files are in /content/Dataset/chest_xray/test/NORMAL
```

در ادامه دستوراتی برای ایجاد پوشه‌هایی با عناوین مشخص نوشته‌ایم که از تکرار آن در گزارش صرف‌نظر می‌کنیم. حال دستورات زیر را می‌نویسیم تا تمامی داده‌های موجود در پوشه‌ها و زیرپوشه‌های دیتاست در دو پوشهٔ اصلی که مربوط به کلاس‌های دیتاست هستند ذخیره شوند. مثلاً تمام داده‌های NORMAL در زیرپوشه‌های train، val و test، در یک پوشه با نام NORMAL تجمیع می‌شوند. بدین‌ترتیب یک مجموعه کامل از داده‌ها بدون استفاده از تقسیم‌بندی پیش‌فرض خواهیم داشت. این کار را به این دلیل انجام دادیم که در صورت سوال از ما خواسته شده بود تا داده‌ها را مطابق درصدبندی مشخصی در سه دستهٔ train، val و test تقسیم کنیم.

```python
import os
import shutil
from torch.utils.data import Dataset, DataLoader
from torchvision.datasets.folder import default_loader

# Set the path to your input folder here
input_folder_path = "/content/Dataset/chest_xray/chest_xray"

# Define the classes in your dataset
classes = ["NORMAL", "PNEUMONIA"]

# Define the path to the output folder where you want to save the combined data
output_folder_path = os.path.join(input_folder_path, "AllData")

# 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 ["train", "test", "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)

# Define a custom PyTorch dataset to load the combined data
class CustomDataset(Dataset):
    def __init__(self, root, classes, transform=None, loader=default_loader):
        self.root = root
        self.classes = classes
        self.transform = transform
        self.loader = loader
        self.samples = []

        # Loop through the NORMAL and PNEUMONIA classes and their respective image folders in the output folder
        for class_name in classes:
            class_folder_path = os.path.join(root, class_name)
            for file_name in os.listdir(class_folder_path):
                if file_name.endswith(".jpeg"):
                    self.samples.append((os.path.join(class_folder_path, file_name), classes.index(class_name)))

    def __getitem__(self, index):
        path, target = self.samples[index]
        sample = self.loader(path)
        if self.transform is not None:
            sample = self.transform(sample)
        return sample, target

    def __len__(self):
        return len(self.samples)

# Load the combined data using the custom PyTorch dataset
dataset = CustomDataset(output_folder_path, classes)

# Use the PyTorch DataLoader to create batches of data for training/testing
dataloader = DataLoader(dataset, batch_size=32, shuffle=True)
```

حال برنامه‌ای مشابه برنامهٔ ۱ اجرا می‌کنیم و نتیجه به شرح زیر است:

```
4273 (.jpeg) files are in /content/Dataset/chest_xray/chest_xray/AllData/PNEUMONIA
1583 (.jpeg) files are in /content/Dataset/chest_xray/chest_xray/AllData/NORMAL
```

در ادامه دستورات زیر را می‌نویسیم تا تمامی داده‌ها را به ابعاد گفته‌شده در مقاله (128×128) پیش‌پردازش کنیم:

```
import os
import torch
import torchvision
import torchvision.transforms as transforms
from torchvision.datasets import ImageFolder

# Define the path to the input and output folders
input_folder_path = "/content/Dataset/chest_xray/chest_xray/AllData"
output_folder_path = "/content/AllDataPreprocess"

# Define the transformation pipeline to apply to the images
transform = transforms.Compose([
    transforms.Resize((128, 128)),
    transforms.ToTensor()
])

# Load the data from the input folder using the ImageFolder dataset
data = ImageFolder(input_folder_path, transform=transform)

# 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 NORMAL and PNEUMONIA classes and their respective image folders in the input folder
for class_name in data.classes:
    class_folder_path = os.path.join(output_folder_path, class_name)
    if not os.path.exists(class_folder_path):
        os.makedirs(class_folder_path)

    # Loop through the images in the class folder, apply the transformation pipeline, and save the new images to the output folder
    for i in range(len(data)):
        if data.targets[i] == data.class_to_idx[class_name]:
            image, _ = data[i]
            file_name = data.samples[i][0].split("/")[-1]
            output_path = os.path.join(class_folder_path, file_name)
            torchvision.utils.save_image(image, output_path)
```

برای این که تکنیک نرمال‌سازی داده‌ها را هم در مجموعه‌داده‌های خود پوشش دهیم، میانگین و انحراف معیار داده‌ها را با استفاده از دستورات زیر محاسبه می‌کنیم. نتایج ذیل کد درج شده است.

```python
import os
import numpy as np
import torch
import torchvision.transforms as transforms
from PIL import Image

# Define the path to the folder containing the images
folder_path = "/content/AllDataPreprocess"

# Define the transforms to be applied to each image
transform = transforms.Compose([
    transforms.Resize((128, 128)),
    transforms.ToTensor()
])

# Define the accumulator variables for the mean and standard deviation
normal_images = []
pneumonia_images = []

# Loop through all files in the folder
for folder_name in ["NORMAL", "PNEUMONIA"]:
    folder = os.path.join(folder_path, folder_name)
    for file_name in os.listdir(folder):
        # Check if the file is an image (JPEG or PNG)
        if file_name.endswith(".jpg") or file_name.endswith(".jpeg") or file_name.endswith(".png"
    ):
            # Open the image file and apply the transforms
            image_path = os.path.join(folder, file_name)
            image = Image.open(image_path)
            image = transform(image)

            # Add the image to the corresponding accumulator variable
            if folder_name == "NORMAL":
                normal_images.append(image)
            else:
                pneumonia_images.append(image)

# Concatenate all images and calculate the mean and standard deviation
normal_images = torch.stack(normal_images)
pneumonia_images = torch.stack(pneumonia_images)
all_images = torch.cat([normal_images, pneumonia_images], dim=0)
mean = torch.mean(all_images, dim=(0, 2, 3))
std = torch.std(all_images, dim=(0, 2, 3))

# Output the mean and standard deviation
```

```
print(f"mean={mean.tolist()}, std={std.tolist()}")

------------------------------------------------

mean=[0.4815806746482849, 0.4815806746482849, 0.4815806746482849],
std=[0.23505905270576477, 0.23505905270576477, 0.23505905270576477]
```

در ادامه از دستورات زیر استفاده می‌کنیم که داده‌ها علاوه بر تغییر ابعاد نرمال هم شوند. توجه داشته باشید که هر یک از این دستورات مجموعه‌داده‌هایی جدید ایجاد می‌کند که هریک برای مراحل بعدی می‌توانند به‌صورت جداگانه مورد استفاده قرار گیرند. مثلاً ما در این‌جا هردوی مجموعه‌داده‌های پیش‌پردازش‌شده و پیش‌پردازش و نرمال‌شده را به‌صورت جداگانه در اختیار خواهیم داشت.

```
import os
import torch
import torchvision
import torchvision.transforms as transforms
from torchvision.datasets import ImageFolder

# Define the path to the input and output folders
input_folder_path = "/content/AllDataPreprocess"
output_folder_path = "/content/AllDataPreprocessNormal"

# Define the transformation pipeline to apply to the images
transform = transforms.Compose([
    transforms.Resize((128, 128)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.4815806746482849, 0.4815806746482849, 0.4815806746482849], std
    =[0.23505905270576477, 0.23505905270576477, 0.23505905270576477])
])

# Load the data from the input folder using the ImageFolder dataset
data = ImageFolder(input_folder_path, transform=transform)

# 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 NORMAL and PNEUMONIA classes and their respective image folders in the input
    folder
for class_name in data.classes:
    class_folder_path = os.path.join(output_folder_path, class_name)
    if not os.path.exists(class_folder_path):
        os.makedirs(class_folder_path)

    # Loop through the images in the class folder, apply the transformation pipeline, and save
    the new images to the output folder
```

```
    for i in range(len(data)):
        if data.targets[i] == data.class_to_idx[class_name]:
            image, _ = data[i]
            file_name = data.samples[i][0].split("/")[-1]
            output_path = os.path.join(class_folder_path, file_name)
            torchvision.utils.save_image(image, output_path)
```

ملاحظاتی در باب تقویت مجموعه‌داده

برای تقویت داده روی‌کردهای مختلفی را در پیش می‌گیریم. یک کار ایجاد مجموعه‌دادهٔ افزونه‌شده برای استفاده در مراحل بعدی است. این کار با اهداف مطرح‌شده در این قسمت مطابقت دارد. کار دیگری که با توجه به این که جمع تعداد داده‌های آموزشی، اعتبارسنجی و آزمون استفاده‌شده در پیاده‌سازی مقاله (بخش IV) با جمع داده‌های مجموعه‌دادهٔ دانلودشده مطابقت می‌کند انجام دادیم، استفاده از روش‌های افزونه‌سازی داده حین انجام فرآیند آموزش مدل است که همگی در ادامه جزئیات و مقدمات آماده‌سازی آن بحث خواهد شد (بخش ۴.۱).

برای تقویت مجموعه‌داده با استفاده از روش‌های آورده‌شده در جدول ۱ از دستورات زیر استفاده می‌کنیم. مشابه قسمت‌های قبل در نهایت دو مجموعه‌داده ذخیره می‌کنیم. یکی با حالت نرمال‌شده به میانگین و انحراف معیار مجموعه‌داده و دیگری با حالت غیرنرمال‌شده به این صورت. برای حالت دوم خط مربوط به نرمال‌سازی به این روش کامنت شده است.

```
import os
import torch
import torchvision
import torchvision.transforms as transforms
from torchvision.datasets import ImageFolder

# Define the path to the input and output folders
input_folder_path = "/content/Dataset/chest_xray/chest_xray/AllData"
output_folder_path = "/content/AllDataPreprocessNormalAug"

# Define the transformation pipeline to apply to the images
transform = transforms.Compose([
    transforms.Resize((128, 128)),
    transforms.RandomAffine(degrees=30, shear=0.2, scale=(0.8, 1.2), translate=(0.2, 0.2)),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    # transforms.Normalize(mean=[0.0, 0.0, 0.0], std=[1.0/255, 1.0/255, 1.0/255])
    transforms.Normalize(mean=[0.4815806746482849, 0.4815806746482849, 0.4815806746482849], std
    =[0.23505905270576477, 0.23505905270576477, 0.23505905270576477])
])

# Load the data from the input folder using the ImageFolder dataset
data = ImageFolder(input_folder_path, transform=transform)

# 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 NORMAL and PNEUMONIA classes and their respective image folders in the input folder
for class_name in data.classes:
    class_folder_path = os.path.join(output_folder_path, class_name)
    if not os.path.exists(class_folder_path):
        os.makedirs(class_folder_path)

    # Loop through the images in the class folder, apply the transformation pipeline, and save the new images to the output folder
    for i in range(len(data)):
        if data.targets[i] == data.class_to_idx[class_name]:
            image, _ = data[i]
            file_name = data.samples[i][0].split("/")[-1]
            output_path = os.path.join(class_folder_path, file_name)
            torchvision.utils.save_image(image, output_path)
```

در نهایت دستورات زیر را نوشته‌ایم تا با تجمیع دیتاست‌های عادی و پردازش‌شده و دیتاست تقویت‌شده، یک مجموعه‌دادهٔ افزونه‌شده بسازیمو مشابه مراحل قبلی این‌جا هم هر دو حالت نرمال‌سازی را در نظر گرفته‌ایم:

```
import os
import shutil

# Path to the new folder
new_folder_path = "/content/FinalDataset"

# Path to the first and second folders
first_folder_path = '/content/AllDataPreprocess'
second_folder_path = '/content/AllDataPreprocessAug'

# Create the new subfolders
os.makedirs(os.path.join(new_folder_path, "NORMAL"))
os.makedirs(os.path.join(new_folder_path, "PNEUMONIA"))

# Copy the files from the first folder
for filename in os.listdir(os.path.join(first_folder_path, "NORMAL")):
    new_filename = filename[:-5] + "_1" + filename[-5:]
    shutil.copy(os.path.join(first_folder_path, "NORMAL", filename),
                os.path.join(new_folder_path, "NORMAL", new_filename))
for filename in os.listdir(os.path.join(first_folder_path, "PNEUMONIA")):
    new_filename = filename[:-5] + "_1" + filename[-5:]
    shutil.copy(os.path.join(first_folder_path, "PNEUMONIA", filename),
                os.path.join(new_folder_path, "PNEUMONIA", new_filename))

```

```
# Copy the files from the second folder
for filename in os.listdir(os.path.join(second_folder_path, "NORMAL")):
    new_filename = filename[:-5] + "_2" + filename[-5:]
    shutil.copy(os.path.join(second_folder_path, "NORMAL", filename),
                os.path.join(new_folder_path, "NORMAL", new_filename))
for filename in os.listdir(os.path.join(second_folder_path, "PNEUMONIA")):
    new_filename = filename[:-5] + "_2" + filename[-5:]
    shutil.copy(os.path.join(second_folder_path, "PNEUMONIA", filename),
                os.path.join(new_folder_path, "PNEUMONIA", new_filename))
```

در نهایت، دستورات زیر را نوشتیم تا مجموعه‌داده‌های تشکیل‌شده با نسبت گفته‌شده در صورت سوال به دسته‌های آموزشی، اعتبارسنجی و آزمون تقسیم شوند. تعداد داده‌های هر دسته ذیل کد آورده شده است:

```
import os
import shutil
import random

# Set the path to the "FinalDataset" folder
data_dir = "/content/FinalDataset"

# Set the path to the output directory
output_dir = "/content/FinalDatasetTVT"

# 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):
    if i < num_train_normal:
        shutil.copy(os.path.join(data_dir, "NORMAL", file), os.path.join(output_dir, "train", "Normal"))
    elif i < num_train_normal + num_val_normal:
        shutil.copy(os.path.join(data_dir, "NORMAL", file), os.path.join(output_dir, "val", "Normal"))
    else:
        shutil.copy(os.path.join(data_dir, "NORMAL", file), os.path.join(output_dir, "test", "Normal"))
for i, file in enumerate(pneumonia_files):
    if i < num_train_pneumonia:
        shutil.copy(os.path.join(data_dir, "PNEUMONIA", file), os.path.join(output_dir, "train", "PNEUMONIA"))
    elif i < num_train_pneumonia + num_val_pneumonia:
        shutil.copy(os.path.join(data_dir, "PNEUMONIA", file), os.path.join(output_dir, "val", "PNEUMONIA"))
    else:
        shutil.copy(os.path.join(data_dir, "PNEUMONIA", file), os.path.join(output_dir, "test", "PNEUMONIA"))

------------------------------------------------------------

1709 (.jpeg) files are in /content/FinalDatasetNormalTVT/val/PNEUMONIA
633 (.jpeg) files are in /content/FinalDatasetNormalTVT/val/NORMAL
5127 (.jpeg) files are in /content/FinalDatasetNormalTVT/train/PNEUMONIA
1899 (.jpeg) files are in /content/FinalDatasetNormalTVT/train/NORMAL
1710 (.jpeg) files are in /content/FinalDatasetNormalTVT/test/PNEUMONIA
634 (.jpeg) files are in /content/FinalDatasetNormalTVT/test/NORMAL
```

۲.۱ توضیح لایه‌های مختلف معماری شبکه

در مقاله تصریح شده که بخش اصلی EfficientNet، بلوک‌های Mobile Inverted Bottleneck Conv (MBconv) و squeeze-and-excitation است. بلوک MBconv یک بلوک کانولوشن با ساختار معکوس و گلوگاه است که برای حفظ دقت شبکه و

کاهش پیچیدگی محاسباتی استفاده می‌شود. بلوک فشار و تحریک (SE) نیز یک بلوک گلوگاهی است که برای توجه به اهمیت کانال‌ها در یادگیری استفاده می‌شود. این بلوک ابتدا اطلاعات ورودی را با یک لایهٔ کانولوشنی یک‌دریک فشرده می‌کند و سپس وزن اهمیت هر کانال را با استفاده از یک لایهٔ کاملا متصل محاسبه می‌کند. سپس با استفاده از یک لایه تحریک معکوس، اهمیت هر کانال در تولید خروجی افزایش می‌یابد و شبکه روی کانال‌های مهم‌تر تمرکز می‌کند. ایدهٔ اصلی EfficientNet بر این است که از یک مدل بنچمارک با کیفیت و فشرده استفاده کرده تا با استفاده از تعداد مشخصی از ضرایب مقیاس‌دهی، به طور پیوسته تمامی پارامترهای آن را ارزیابی کند. این مقاله از این مدل پیش‌آموزش‌دیده با مجموعه‌دادهٔ Imagenet استفاده کرده است. دلیل اصلی استفاده از EfficientNet دستیابی به عملکرد مطلوب در وظایف بینایی رایانه و در عین حال به حداقل رساندن نیازهای محاسباتی و حافظه است. این کار با استفاده از یک روش مقیاس‌بندی انجام می‌پذیرد که عمق، عرض و وضوح معماری شبکه عصبی را به طور یکنواخت مقیاس می‌کند. این رویکرد مقیاس‌بندی امکان داد و ستد بهتر بین دقت و منابع محاسباتی را در مقایسه با مدل‌های دیگر فراهم می‌کند. از مزایای EfficientNet می‌توان به این اشاره کرد که این مدل در وظایف مختلف بینایی رایانه به عمل‌کرد مطلوبی دست پیدا کرده درحالی‌که از منابع محاسباتی کم‌تری در مقایسه با سایر مدل‌ها استفاده کرده. هم‌چنین این مدل را می‌توان به راحتی بسته به کاربردهای خاص کوچک، بزرگ و سازگار با کاربردهای متفاوت کرد. طراحی ماژولار این مدل امکان شخصی‌سازی و انطباق آسانش با مجموعه‌داده‌ها و برنامه‌های مختلف را فراهم می‌کند. علاوه بر این‌ها، این مدل را می‌توان به عنوان یک مدل از پیش‌آموزش‌دیده برای روی‌کرد یادگیری انتقالی استفاده کرد که این می‌تواند میزان داده‌های مورد نیاز و زمان مورد نیاز برای آموزش را تا حد زیادی کاهش دهد. به‌صورت کلی، شبکهٔ EfficientNet با استفاده از روش اندازه‌گیری و مقیاس‌بندی ترکیبی، توانسته است پیشرفت‌های چشمگیری را در کاهش پارامترها و عملیات در ثانیه (FLOPS) به همراه افزایش عملکرد دقت شگفت‌انگیز برای وظایف مختلف بینایی رایانه به دست آورد. این شبکه با کاهش هزینهٔ محاسباتی و استفاده از باتری، این قابلیت را دارد که مصرف انرژی در دستگاه‌های پردازشی را کاهش دهد. به دلیل هزینه‌های زیاد برای حاشیه‌نویسی داده‌های آموزشی، جمع‌آوری داده‌های آموزشی وسیع یک کار دشوار است. مقاله با استفاده از روش یادگیری انتقالی که به راحتی برای این شبکه قابل عملیاتی‌شدن است و هم‌چنین روش‌های مختلف افزایش داده‌های آموزشی، به همراه جلوگیری از بیش‌برازش، این مشکل را برطرف کرده است.

مقاله لایه‌های اولیهٔ EfficientNet را فریز و ثابت نگه داشته تا ویژگی‌ها یا الگوهای رایج سطح پایین با استفاده از همان‌ها از تصاویر اشعهٔ ایکس قفسهٔ سینه استخراج شوند. اما لایه‌های انتهایی اصلاح و تدقیق شده‌اند. در شبکه های عصبی کانولوشنی، لایه‌های اولیه ویژگی‌های بسیار بنیادی و اساسی مانند خطوط، منحنی‌ها و... را یاد می‌گیرند که تقریباً به همهٔ انواع تصاویر تعمیم می‌یابند. در لایه‌های بعدی، ویژگی‌ها به تدریج منحصر به مجموعه داده‌ای که مدل بر اساس آن آموزش داده می‌شود می‌گردد. هدف از تنظیم دقیق این است که این ویژگی‌های منحصربه‌فرد را برای بهبود عملکرد مدل روی مجموعه‌داده‌های تغذیه‌شدهٔ کنونی اشعه ایکس قفسه سینه تنظیم کنیم. بنابراین، در بالای مدل پایه، یک طبقه‌بند جدید طراحی کرده و برای تنظیم وزن‌ها بر اساس توزیع‌ها و الگوهای جدید آموزش را انجام می‌دهیم. این لایه‌های جدید شامل این موارد هستند:

- Global Average Pooling 2D: این لایه معمولاً در شبکه‌های عصبی کانولوشنی برای کارهای طبقه‌بندی تصویر استفاده می‌شود و هدفش کاهش ابعاد فضایی نگاشت‌های ویژگی تولیدشده توسط لایه‌های کانولوشنی در عین حفظ اطلاعات مهم است. نگاشت ویژگی‌ای با ابعاد (H x W x C (height x width x number of feature maps (or channels در نظر بگیرید. Global Average Pooling 2D یک عملیات ادغام میانگین فضایی را در ابعاد عرض و ارتفاع اِعمال می‌کند که منجر به نگاشت ویژگی‌ای با ابعاد 1 x 1 x C می‌گردد. این با محاسبهٔ میانگین هر کانال در تمام مکان‌های فضایی در نگاشت ویژگی به دست می‌آید. مزیت اصلی استفاده از این لایه این است که تعداد پارامترهای شبکه را کاهش می‌دهد که این می‌تواند به جلوگیری از بیش‌برازش و به بهبود تعمیم‌پذیری شبکه کمک کند. علاوه بر این، این لایه می‌تواند تفسیرپذیری شبکه را بهبود بخشد، زیرا یک نگاشت ویژگی با ابعاد 1 x 1 x C را می‌توان برای فهمیدن اینکه کدام کانال‌ها

برای یک کار خاص مهم هستند، راحت‌تر تجسم کرد.

- Dense layer (with activation 'RELU'): لایهٔ متراکم نوعی لایهٔ شبکهٔ عصبی است که هر نورون ورودی را به هر نورون خروجی متصل می‌کند. از این لایه به عنوان یک لایهٔ تماماً متصل نیز یاد می‌شود. لایهٔ متراکم یک تبدیل خطی به داده‌های ورودی اعمال می‌کند و به دنبال آن یک تابع فعال‌سازی اعمال می‌شود. تابع فعال سازی مشخص شده در مقاله ReLU است. این تابع فعال‌ساز تمام مقادیر منفی در خروجی تبدیل خطی را صفر می‌کند و همهٔ مقادیر مثبت را بدون تغییر نگه می‌دارد. به عبارت دیگر خروجی لایه متراکم با فعال سازی ReLU بیشینهٔ ۰ و حاصل تبدیل خطی خواهد بود.
- Dropout: این ابتکار یک تکنیک مناسب برای جلوگیری از بیش‌برازش در شبکه‌های عصبی است. این لایه در فرآیند آموزش شبکه عصبی، با احتمال مشخص، گره‌های مختلفی را خاموش می‌کند. خاموشی تصادفی این گره‌ها باعث می‌شود که شبکهٔ عصبی به صورت اجباری با اطلاعات کمتری آموزش ببیند. در شبکه‌های عصبی با تعداد پارامترهای بسیار زیاد، بسیاری از این پارامترها ممکن است غیرضروری باشند. به طور مشابه، در برخی موارد، تعداد داده‌های آموزش نیز ممکن است کافی نباشد. این موارد می‌تواند منجر به ایجاد بیش‌برازش شود که به معنی آموزش خیلی خوب روی داده‌های آموزش ولی عملکرد ناکافی بر روی داده‌های آزمون است.

۳.۱ پیاده‌سازی شبکه

از آن‌جا که در مقاله تصریح شده که برای پیاده‌سازی از Keras استفاده شده و نحوهٔ نگارش مقاله هم همین را نشان می‌دهد، جهت سادگی و نزدیکی بیش‌تر به مقاله پیاده‌سازی را با کتابخانهٔ Keras انجام می‌دهیم. برای پیاده‌سازی شبکه دستورات آورده‌شده در برنامهٔ ۲ را نوشته‌ایم. در گام اول و با استفاده از کتابخانهٔ ،lrKeras کلاس `ImageDataGenerator` را ایجاد می‌کنیم. این کلاس برای تبدیل تصاویر به شکلی قابل استفاده برای شبکهٔ عصبی کانولوشنی استفاده می‌شود. به دلیل اینکه شبکه‌های عصبی کانولوشنی حساس به تغییرات مختلف در داده‌های ورودی هستند، از روش‌هایی مانند تغییر اندازه، چرخش، جابجایی و غیره می‌توان برای افزونه‌سازی دادها استفاده کرد. روش‌هایی که برای این موضوع انتخاب شده‌اند در بخش ۱.۱ به تفصیل شرح داده شده‌اند و از تکرار آن‌ها در این‌جا خودداری می‌کنیم. در ادامه با استفاده از تابع `flow_from_directory` ، داده‌ها از دایرکتوری‌های آموزش، اعتبارسنجی و آزمون خوانده می‌شوند. در هر دسته‌بندی، تصاویر با ابعاد مشخص شده در پارامتر `target_size` (همان ابعاد 128×128 ذکرشده در مقاله) تبدیل و بارگیری می‌شوند. در مجموعه داده‌های آموزشی، پارامتر `batch_size` با مقدار ۱۲۸ تعیین شده است که به این معنی است که تصاویر به صورت دسته‌ای از ۱۲۸ داده به شبکهٔ مورد نظر مقاله وارد می‌شوند. در هر دسته‌بندی، نوع کلاس تصاویر با مقدار باینری به عنوان پارامتر `class_mode` تعیین شده است، زیرا طبقه‌بندی دوکلاسه است. در ادامه، با استفاده از تابع `next`، داده‌ها به صورت دسته‌بندی‌شده، در متغیرهای مربوط به تقسیم‌بندی‌های آموزش، اعتبارسنجی و آموزش ذخیره می‌شوند. این دستورات برای جداسازی داده‌ها و برچسب‌های مربوط به دسته‌های مختلف در مجموعه‌داده مورد استفاده قرار می‌گیرد. در اینجا، سه مجموعه داده شامل مجموعه آموزش، اعتبارسنجی و آزمون وجود دارد. در نتیجه، با استفاده از این دستورات، می توانیم داده ها و برچسب های مربوط به هر یک از مجموعه ها را جدا کنیم و آن ها را برای استفاده در مدل آماده کنیم.

در ادامه به تعریف مدل و شبکهٔ مورد استفاده در مقاله می‌پردازیم. ابتدا از کتابخانهٔ Keras، مدل پیش‌آموزش‌دیدهٔ EfficientNetB2 را با وزن‌های پیش‌آموزش‌دیده روی مجموعه‌دادهٔ ImageNet فراخوانی می‌کنیم. این مدل به عنوان شبکه‌ای قدرتمند در طبقه‌بندی تصاویر مورد استفاده قرار می‌گیرد. سپس، به این مدل، لایه‌های جدیدی اضافه می‌کنیم تا شبکه را برای طبقه‌بندی تصاویر مجموعه‌دادهٔ مقاله آموزش دهیم. در این بخش، خروجی شبکه پیش‌آموزش‌دیده را به عنوان ورودی لایه GlobalAveragePooling2D

می‌دهیم تا ویژگی‌های تصاویر با استفاده از میانگین‌گیری سراسری از نظر فضایی تبدیل و مناسب‌سازی شوند. سپس، یک لایهٔ تماماًمتصل با ۱۲۸ نورون و تابع فعال‌سازی ReLU و یک لایهٔ Dropout با احتمال ۰.۳ به شبکه اضافه می‌کنیم. سپس دوباره یک لایهٔ تماماًمتصل با ۶۴ نورون و تابع فعال‌سازی ReLU و یک لایهٔ پمrDropout دیگر با احتمال ۰.۲ به شبکه اضافه می‌کنیم. در انتها هم، یک لایهٔ تماماًمتصل با یک نورون و تابع فعال‌سازی سیگموئید به شبکه اضافه می‌کنیم تا خروجی‌های شبکه بین ۰ و ۱ باشند و به عنوان احتمال طبقه‌بندی تصاویر مورد استفاده قرار گیرند. در ادامه، لایه‌های پیش‌آموزش‌دیده را فریز می‌کنیم. یعنی اجازه نمی‌دهیم که وزن‌های این لایه‌ها در فرایند آموزش شبکه تغییر کنند و کاری می‌کنیم تا فقط لایه‌های جدیدی که در مرحله قبل اضافه کردیم آموزش داده شوند. البته همان‌طور که در کد آمده دستورات را نوشته‌ایم تا حالت‌های مختلف دیگری مانند فریزنکردن کل لایه‌ها و یا فریزنکردن بخشی از لایه‌ها را هم بررسی کنیم.

در ادامه و برای محاسبه وزن کلاس‌ها بر اساس داده‌های آموزش و تعداد نمونه‌های هر کلاس و ایجاد توازن کلاسی، دستوراتی را می‌نویسیم. دستور اول برای یافتن برچسب‌های یکتای کلاس‌های موجود در مجموعهٔ آموزشی استفاده شده است. سپس با استفاده از تابع `compute_class_weight` از کتابخانهٔ `sklearn.utils.class_weight`، وزن‌های کلاس‌ها محاسبه شده و با تنظیم balanced در متغیر `class_weights` ذخیره می‌شود. در دستور بعدی، با استفاده از تابع `dict.zip`، وزن‌های کلاس محاسبه شده در مرحله قبل، با لیبل‌های مربوطه به هر کلاس در قالب یک دیکشنری ذخیره شده است. این دیکشنری حاوی یک لیست از تمام کلاس‌ها است که برای هر کلاس، وزن محاسبه شده در قالب مقداری از دیکشنری ذخیره شده است. با انجام این مراحل، وزن هر کلاس در فرایند آموزش مدل، بر اساس توزیع کلاس‌ها در داده‌های آموزش، محاسبه شده و برای اعمال وزن به نمونه‌ها در طی فرآیند آموزش مدل، به کار گرفته می‌شود.

در ادامه و برای بهبود عملکرد مدل شبکهٔ عصبی در هنگام آموزش callback تعریف می‌کنیم.

با استفاده از `ModelCheckpoint`، مدل شبکه عصبی در هر دوره (epoch) پس از آموزش، ذخیره می‌شود. این فرآیند ذخیره‌سازی با توجه به این نکته رخ می‌دهد که در صورتی که مقدار تابع اتلاف برای دادهٔ اعتبارسنجی نسبت به دورهٔ قبل کاهش پیدا کرده باشد. بنابراین، یک فایل با نام `best_model.h5` تعریف می‌شود که در آن وزن مدل (weights) در هر دوره روی دیسک ذخیره می‌شود. پارامتر `save_best_only` برای این است که فقط بهترین نمونه از وزن‌های مدل ذخیره شود (بر اساس اتلاف اعتبارسنجی). پارامتر `monitor` برای این است که به کدام معیار برای بررسی بهبود نمونه‌ها توجه شود. در کد ما معیار `val_loss` یعنی اتلاف اعتبارسنجی انتخاب شده است. پارامتر `mode` برای مشخص کردن اینکه به دنبال بهبود معیار یا کاهش آن یا کاری دیگر هستیم تعریف شده است، در این حالت `min` تعریف کرده‌ایم؛ چراکه به دنبال کم شدن `val_loss` هستیم. پارامتر `verbose` نیز برای نمایش پیام‌های پیشرفت در آموزش است. در ادامهٔ همین فرآیندها، دستوری برای کاهش نرخ یادگیری در صورت عملکرد ضعیف مدل در طول آموزش تعریف می‌کنیم و با استفاده از `ReduceLROnPlateau`، نرخ یادگیری در صورت دیدن عملکرد ضعیف مدل در هنگام آموزش، به طور خودکار کاهش پیدا می‌کند. در کد تعریف‌شده، پارامتر `monitor` برای این است که به کدام معیار برای بررسی عملکرد مدل توجه شود. ما معیار `val_loss` را انتخاب کرده‌ایم. پارامتر `factor` برای تعیین میزان کاهش نرخ یادگیری استفاده می‌شود. در این حالت مقدار این پارامترِ مثلا ۰.۳ برای کاهش نرخ یادگیری به مقدار ۰.۳ برابر نرخ یادگیری قبلی استفاده می‌شود. پارامتر `patience` برای تعیین تعداد دوره‌هایی است که در صورت عدم بهبود عملکرد مدل، منتظر می‌مانیم تا نرخ یادگیری کاهش پیدا کند. در این حالت، پس از ۱۰ دوره بدون بهبود در اتلاف اعتبارسنجی، نرخ یادگیری کاهش می‌یابد. پارامتر `min_lr` نیز برای تعیین نرخ یادگیری حداقل استفاده شده است، به این معنا که نرخ یادگیری در هیچ صورتی نمی‌تواند کمتر از این مقدار شود. پارامتر `verbose` نیز برای نمایش پیام‌های پیشرفت در آموزش است. در انتها `EarlyStopping` را تعریف کرده‌ایم تا اگر اتلاف اعتبارسنجی برای بازه‌ی مشخصی (متوالی) بهبود نداشت، از ادامهٔ آموزش جلوگیری شود. از این دستور به طور معمول در مواردی استفاده می‌شود که مدل بیش از حد بزرگ شده باشد و احتمال دارد که به جای بهبود، بیش‌برازش رخ دهد. در نهایت دستورات `model.fit` را برای آموزش مدل با داده‌های آموزش و اعتبارسنجی

تعریف می‌کنیم. این تابع با استفاده از داده‌های آموزش و اعتبارسنجی و همچنین سایر پارامترهای تعیین شده، مدل را آموزش می‌دهد. در انتها هم دستوراتی برای ارزیابی عمل‌کرد مدل روی داده‌های آزمون نوشته‌ایم و ضمن پیش‌بینی و طبقه‌بندی داده‌ها، شاخص‌های مقاله را محاسبه کرده و ماتریس درهم‌ریختگی و نمودارهای دقت، اتلاف و غیره را نمایش داده‌ایم.

Program 2: Implementation Code

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout
from tensorflow.keras.models import Model
from efficientnet.tfkeras import EfficientNetB2
import numpy as np
from sklearn.metrics import roc_auc_score, recall_score, f1_score, precision_score

train_datagen = ImageDataGenerator(
    rescale=1.0/255,
    shear_range=0.2,
    width_shift_range=0.2,
    height_shift_range=0.2,
    rotation_range=30,
    horizontal_flip=True,
    zoom_range=0.2
)

val_datagen = ImageDataGenerator(rescale=1.0/255)
test_datagen = ImageDataGenerator(rescale=1.0/255)

train_generator = train_datagen.flow_from_directory(
    '/content/FinalDatasetTVT2/train',
    target_size=(128, 128),
    batch_size=128,
    class_mode='binary')

val_generator = val_datagen.flow_from_directory(
    '/content/FinalDatasetTVT2/val',
    target_size=(128, 128),
    batch_size=128,
    class_mode='binary')

test_generator = test_datagen.flow_from_directory(
    '/content/FinalDatasetTVT2/test',
    target_size=(128, 128),
    batch_size=1174,
    class_mode='binary')

# separate data and labels for training set
```

```
x_train, y_train = train_generator.next()

# separate data and labels for validation set
x_val, y_val = val_generator.next()

# separate data and labels for test set
x_test, y_test = test_generator.next()

# load pre-trained model
base_model = EfficientNetB2(weights='imagenet', include_top=False, input_shape=(128, 128, 3))

# add new layers
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)

# freeze pre-trained layers
for layer in base_model.layers:
    layer.trainable = False

# from math import ceil
# base_model = EfficientNetB2(include_top=False, weights='imagenet')
# # Determine the index of the layer to unfreeze based on the percentage
# unfreeze_index = ceil(0.7 * len(base_model.layers))
# # Freeze the weights of the base model up to the unfreeze index
# for layer in base_model.layers[:unfreeze_index]:
#     layer.trainable = False
# # Unfreeze the weights of the remaining layers
# for layer in base_model.layers[unfreeze_index:]:
#     layer.trainable = True

import numpy as np
from sklearn.utils.class_weight import compute_class_weight

# class_labels = np.unique(y_train)
class_weights = compute_class_weight(
                                        class_weight = "balanced",
                                        classes = np.unique(y_train),
                                        y = y_train
```

```
                                    )
class_weights = dict(zip(np.unique(y_train), class_weights))

# define callbacks
from keras.callbacks import ModelCheckpoint, ReduceLROnPlateau, EarlyStopping

# save the model weights after each epoch if the validation loss decreased
checkpoint = ModelCheckpoint('best_model.h5',
                             save_best_only=True,
                             save_weights_only=True,
                             monitor='val_loss',
                             mode='min', verbose=1)

# reduce learning rate when the validation loss plateaus
reduce_lr = ReduceLROnPlateau(monitor='val_loss',
                              factor=0.2,
                              patience=10,
                              min_lr=0.0001, verbose=1)

# stop training if the validation loss doesn't improve for 30 consecutive epochs
early_stop = EarlyStopping(monitor='val_loss', patience=10)

# compile the model
model.compile(optimizer=Adam(learning_rate=0.001), loss='binary_crossentropy', metrics=['accuracy
    '])

# train the model
history = model.fit(x_train, y_train,
                    validation_data=(x_val, y_val),
                    epochs=30,
                    batch_size=128,
                    callbacks=[checkpoint, reduce_lr, early_stop],
                    class_weight=class_weights
                    )

# evaluate the model on the test set
loss, accuracy = model.evaluate(x_test, y_test, batch_size=128)

print('Test loss:', loss)
print('Test accuracy:', accuracy)

# evaluate the model on the test set
y_pred = model.predict(x_test)
y_pred_classes = np.round(y_pred)
auc_score = roc_auc_score(y_test, y_pred)
```

```
recall = recall_score(y_test, y_pred_classes)
f1 = f1_score(y_test, y_pred_classes)
precision = precision_score(y_test, y_pred_classes)

print('Test AUC:', auc_score)
print('Test Recall:', recall)
print('Test F1-score:', f1)
print('Test Precision:', precision)

# plot confusion matrix
cm = confusion_matrix(y_test, y_pred_classes)
sns.heatmap(cm, annot=True, fmt='g')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.savefig('A5.pdf')
plt.show()

# plot PR curve
precision, recall, _ = precision_recall_curve(y_test, y_pred)
plt.plot(recall, precision)
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.savefig('A4.pdf')
plt.show()

# plot ROC curve
fpr, tpr, _ = roc_curve(y_test, y_pred)
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.savefig('A3.pdf')
plt.show()

# plot accuracy and loss curves for training and validation sets
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.savefig('A2.pdf')
plt.show()

```

```
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.savefig('A1.pdf')
plt.show()

model.save('my_model.h5')
```

۴.۱ نتایج پیاده‌سازی

آماده‌سازی نهایی مجموعه‌داده

برای آماده‌سازی نهایی مجموعه‌داده با توجه به دستورات و برنامهٔ آورده‌شده در قسمت نتایج پیاده‌سازی، دستورات آورده‌شده در این پارتیشن از کدهای گوگل‌کولب را نوشته‌ایم. در این دستورات ابتدا تمامی داده‌ها در دو پوشه‌کلاس عمده و در پوشهٔ `AllData` ذخیره می‌شوند و سپس با توجه به نسبت‌های گفته‌شده برای داده‌های آموزش، اعتبارسنجی و آزمون و به دو صورت مختلف در مجموعه‌دادهٔ `FinalDatasetTVT2` و `FinalDatasetTVT4` ذخیره می‌شوند. هم‌چنین دستوراتی برای تبدیل داده‌های از حالت `grayscale` به `RGB` نوشته شده است. هم‌چنین دستوراتی برای فراخوانی برخی کتابخانه‌ها و ابزارهای موردنیاز برای قسمت‌های بعد در این قسمت تعبیه شده است.

۱.۴.۱ پاسخ قسمت‌های الف و ب

در این قسمت سه پیاده‌سازی را مد نظر خود قرار داده‌ایم، آزمایش اول عیناً منطبق بر مقاله است، در آزمایش دوم صرفاً پارامتر `patience` در `EarlyStopping` را تغییر داده‌ایم و در آزمایش سوم لایه‌های شبکهٔ پایه را فریز نکرده‌ایم و به نوعی مدل را از ابتدا آموزش داده‌ایم و تدقیق کرده‌ایم.

- آزمایش اول: نمودارهای حاصل از پیاده‌سازی در شکل ۱، شکل ۲ و شکل ۳ نماش داده شده‌اند و سایر نتایج خواسته شده در صورت سوال به شرح زیر است:

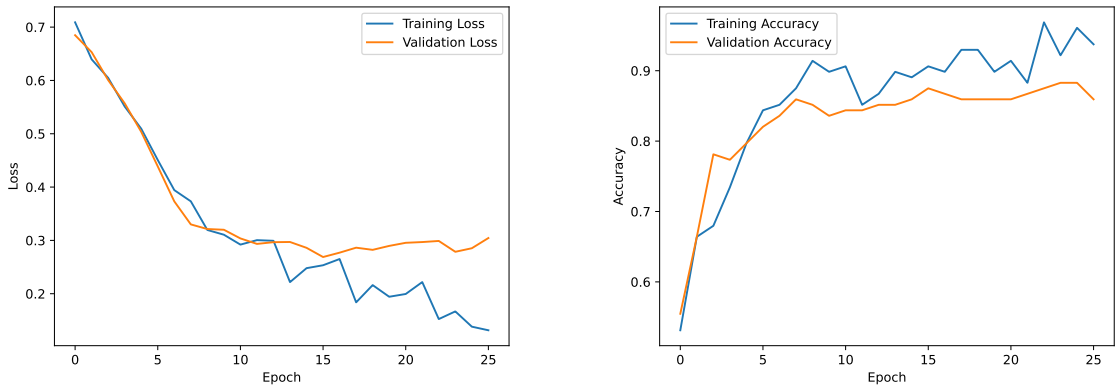
```
Test loss: 0.2954768240451813
Test accuracy: 0.8764906525611877
Test AUC: 0.9499426908834421
Test Recall: 0.8691588785046729
Test F1-score: 0.9112063686466625
Test Precision: 0.9575289575289575
```

- آزمایش دوم: نمودارهای حاصل از پیاده‌سازی در شکل ۴، شکل ۵ و شکل ۶ نماش داده شده‌اند و سایر نتایج خواسته شده در صورت سوال به شرح زیر است:

```
Test loss: 0.2669734060764313
Test accuracy: 0.8969335556030273
Test AUC: 0.957561864456592
Test Recall: 0.927570093457944
Test F1-score: 0.9291983616149796
Test Precision: 0.9308323563892146
```

- آزمایش سوم: نمودارهای حاصل از پیاده‌سازی در شکل ۷، شکل ۸ و شکل ۹ نماش داده شده‌اند و سایر نتایج خواسته شده در صورت سوال به شرح زیر است:

```
Test loss: 0.35489439964294434
Test accuracy: 0.8943781852722168
Test AUC: 0.9510190736495621
Test Recall: 0.9357476635514018
Test F1-score: 0.9281575898030128
Test Precision: 0.9206896551724137
```



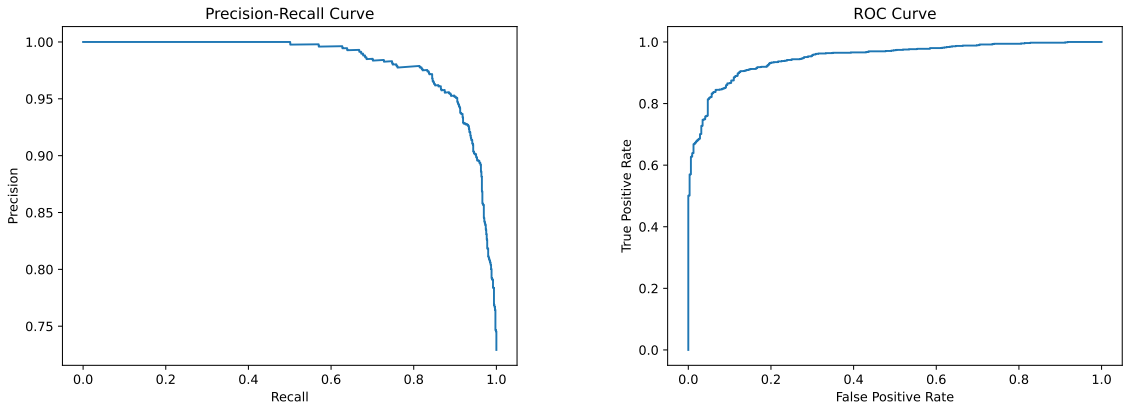
(آ) نمودار دقت (ب) نمودار تابع اتلاف

شکل ۱: نمودارهای دقت و اتلاف برای داده‌های آموزش و اعتبارسنجی (آزمایش اول).

۲.۴.۱ پاسخ قسمت ج

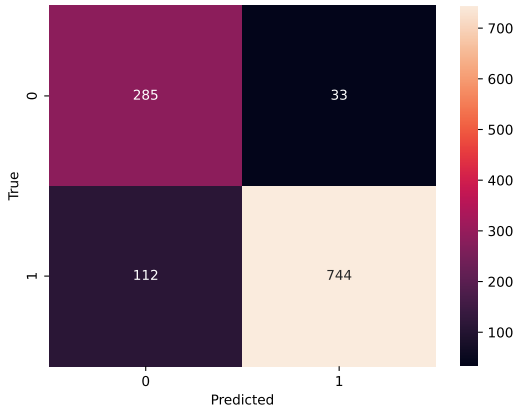
در این قسمت به‌صورت جداگانه به تفسیر و توضیح نتایج و نمودارها می‌پردازیم.

- نمودار **Precision-Recall**: از این نمودار برای بررسی توانایی مدل در تشخیص دو کلاس مثبت و منفی استفاده می‌شود. هر دو کلاس می‌توانند به دو صورت درست و نادرست تشخیص داده شوند. در این نمودار، محور افقی بازیابی (Recall) است و محور عمودی نمودار دقت (Precision) است. دقت نسبت تعداد پیش‌بینی‌های صحیح مثبت به تعداد کل پیش‌بینی‌های مثبت است و یادآوری یا بازخوانی نسبت تعداد پیش‌بینی‌های صحیح مثبت به تعداد کل موارد واقعی مثبت است. این نمودار برای مواردی که تعداد نمونه‌های دو کلاس با یکدیگر نامتعادل است (مانند مجموعه‌دادهٔ همین مقاله)،



(آ) نمودار ROC (ب) نمودار PR

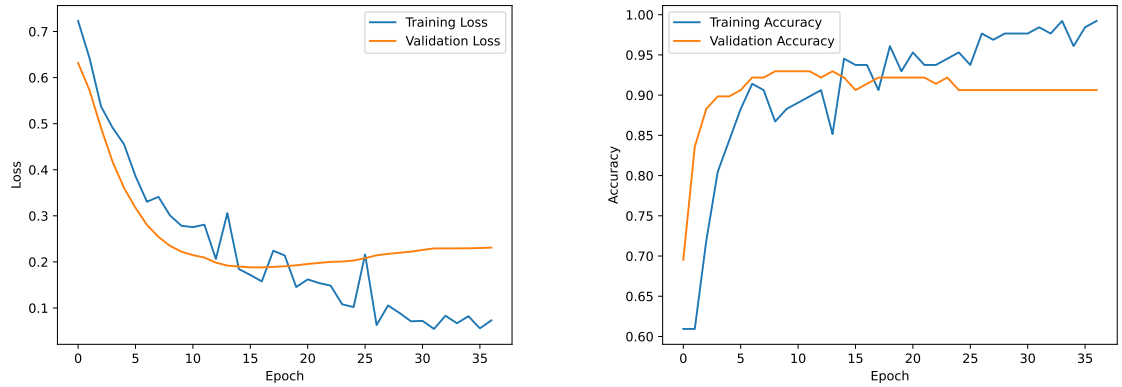
شکل ۲: نمودارهای ROC و PR (آزمایش اول).



شکل ۳: ماتریس درهم‌ریختگی پیاده‌سازی (آزمایش اول).

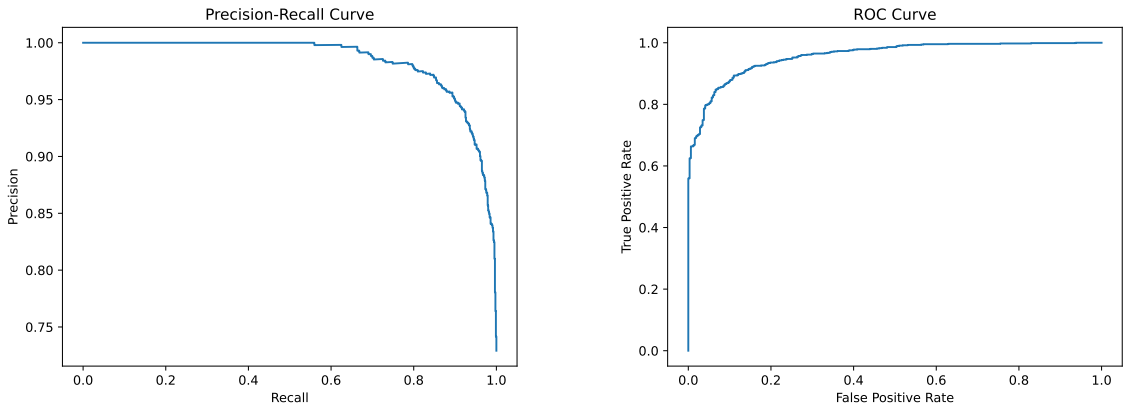
بسیار مفید است. در چنین مواردی، دقت اندازه‌گیری مناسبی برای ارزیابی عملکرد مدل نیست زیرا ممکن است این اندازه‌گیری به‌طور گمراه‌کننده‌ای نشان‌گر عملکرد خوب مدل شود. به عنوان مثال، یک مدلی که در یک مجموعه‌دادهٔ نامتعادل، تمام نمونه‌ها را به عنوان نمونه‌های اکثریت پیش‌بینی کند، ممکن است دقت بالایی داشته باشد، اما در ارزیابی کارایی پیش‌بینی برای کلاس اقلیت، عملکرد ضعیفی خواهد داشت. در حالتی که مدل عمل‌کرد خوبی دارد، هر دو مقدار دقت و بازیابی بسیار بالا باشند این نمودار به خطوط نزدیک به ۱ نزدیک است و به سمت راست و بالا تمایل دارد. بنابراین، در صورتی که مدل تمام داده‌های کلاس مثبت را به درستی تشخیص دهد و هیچ دادهٔ کلاس منفی را به عنوان کلاس مثبت شناسایی نکند، Precision و Recall هر دو ۱ خواهند بود و نقطه (۱،۱) روی نمودار قرار می‌گیرد. اگر نمودار به دو خط انتهایی یک بپسبد حالت ایده‌آل رخ داده است. با توجه به نتایج این نمودار در تصاویر ۲(ب)، ۵(ب) و ۸(ب) مشخص است که عمل‌کرد پیاده‌سازی ما مطلوب و نزدیک به خروجی نتایج مقاله است.

- نمودار **ROC**: حالت ایده‌آل برای این نمودار موقعی است که خط منحنی از گوشهٔ بالا و سمت چپ عبور کند. در این



(آ) نمودار دقت (ب) نمودار تابع اتلاف

شکل ۴: نمودارهای دقت و اتلاف برای داده‌های آموزش و اعتبارسنجی (آزمایش دوم).

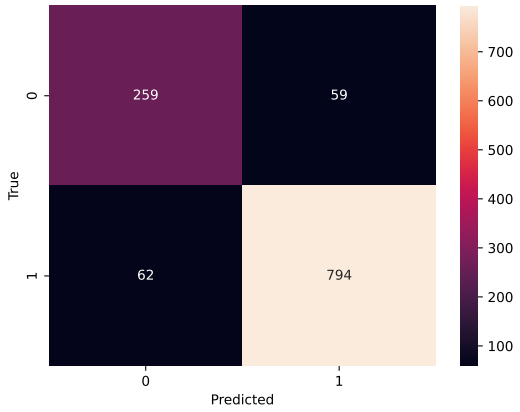


(آ) نمودار ROC (ب) نمودار PR

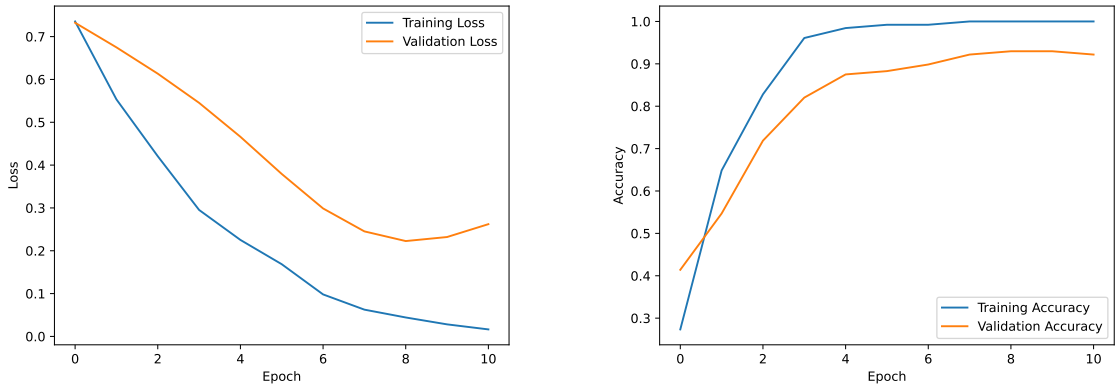
شکل ۵: نمودارهای ROC و PR (آزمایش دوم).

حالت عمل‌کرد مدل کاملاً درست و TP و FP برای تمام آستانه‌ها به‌ترتیب ۱ و ۰ است. ناحیهٔ زیر نمودار ROC می‌تواند به عنوان یک معیار جامع برای ارزیابی کلی عملکرد مدل در طبقه‌بندی دوکلاسه استفاده شود. این معیار با در نظر گرفتن همه مقادیر آستانه، عملکرد کلی مدل را بررسی می‌کند و مقدار بالاتر ناحیه زیر نمودار ROC بهترین عملکرد را نشان می‌دهد. با توجه به نتایج این نمودار در تصاویر ۲(آ)، ۵(آ) و ۸(آ) مشخص است که عمل‌کرد پیاده‌سازی ما مطلوب و نزدیک به خروجی نتایج مقاله است.

- نمودارهای دقت و اتلاف: هرچه نمودار دقت افزایش‌تر باشد و هرچه نمودار اتلاف کاهشی‌تر باشد عمل‌کرد مدل بهتر بوده .باید دقت کرد که گاهی ممکن است در عین کاهش اتلاف داده‌های آموزش، این شاخص در داده‌های اعتبارسنجی کاهش پیدا نمی‌کند. از آن‌جا که از روش‌های مختلفی برای جلوگیری از این اتفاق از جمله پایان زودهنگام و تغییر خودکار نرخ یادگیری در پیاده‌سازی خود استفاده کرده‌ایم مشاهده می‌شود که علی‌رغم سختی و پیچیدگی بالای مسأله نتایج پیاده‌سازی



شکل ۶: ماتریس درهم‌ریختگی پیاده‌سازی (آزمایش دوم).

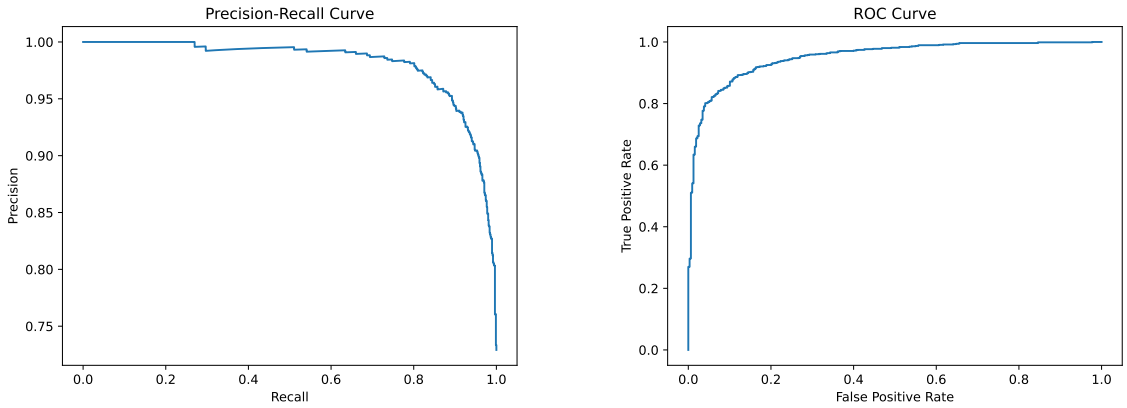


(ب) نمودار تابع اتلاف (آ) نمودار دقت

شکل ۷: نمودارهای دقت و اتلاف برای داده‌های آموزش و اعتبارسنجی (آزمایش سوم).

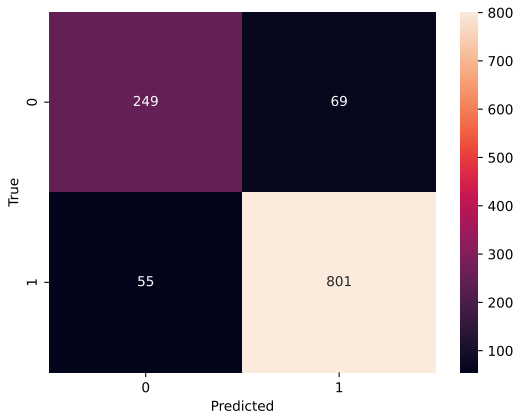
نشان‌داده‌شده برای هر سه آزمایش که در تصاویر شکل ۱، شکل ۴ و شکل ۷ نمایش داده شده، مطلوب و نزدیک به نتایج مقاله است.

- ماتریس درهم‌ریختگی: این ماتریس یک معیارجامع برای استخراج نتایج ارزیابی مدل روی داده‌های آزمون است. هرچه اعداد روی قطر اصلی این ماتریس بزرگ‌تر باشد عمل‌کرد مدل بهتر بوده. همان‌طور که از نتایج پیاده‌سازی مربوطه در تصاویر شکل ۳، شکل ۶ و شکل ۹ مشخص است، عمل‌کرد پیاده‌سازی مطلوب و نزدیک به مقاله بوده است.
- شاخص‌های ارزیابی: شاخص Precision نشان می‌دهد که چه تعداد از داده‌های تشخیص داده شده به عنوان مثبت درست تشخیص داده شده‌اند. به عبارت دیگر، این شاخص بیانگر تعداد مثبت درست تشخیص داده شده توسط مدل است. مقدار مطلوب این شاخص باید حداکثر و یک باشد. شاخص Recall هم نشان می‌دهد که چه تعداد از داده‌های مثبت در دیتاست به درستی تشخیص داده شده‌اند. lrRecall بیانگر تعداد مثبت واقعی در داده‌های مدل است. مقدار مطلوب این شاخص نیز باید حداکثر و یک باشد. شاخص F1 score ترکیبی از precision و recall است و به صورت



(آ) نمودار ROC (ب) نمودار PR

شکل ۸: نمودارهای ROC و PR (آزمایش سوم).



شکل ۹: ماتریس درهم‌ریختگی پیاده‌سازی (آزمایش سوم).

میانگین هندسی این دو شاخص محاسبه می‌شود. این شاخص در واقع نشان می‌دهد که چه تعادل و مصالحه‌ای بین precision و recall وجود دارد. مقدار مطلوب این شاخص باید بالای ۵.۰ باشد و بهتر است به سمت ۱ نزدیک شود. شاخص Accuracy هم نشان می‌دهد که چه تعداد از کل داده‌ها درست تشخیص داده شده‌اند. این شاخص بیانگر تعداد کل داده‌های درست تشخیص داده شده توسط مدل است. مقدار مطلوب این شاخص باید حداکثر و یک باشد. نتایج آورده‌شده در پاسخ قسمت‌های الف و ب که در ؟؟ هم تکرار شده‌اند، بیان‌گر عمل‌کرد مطلوب پیاده‌سازی ما و نزدیکی آن به نتایج مقاله است. در این قسمت دستورات زیر را نوشتم تا هرکدام از این شاخص‌ها را به تفکیک کلاس‌ها و برای حالات مختلف‌تری هم داشته باشیم.

```
from sklearn.metrics import classification_report

# get predicted classes
y_pred = model.predict(x_test)
```

جدول ۲: مقایسهٔ نتایج ارزیابی پیاده‌سازی و مقاله

Index	Paper	ImpTest1	ImpTest2	ImpTest3	ImpTest4
Accuracy	0.96	0.88	0.90	0.90	0.89
Precision	0.97	0.96	0.93	0.92	0.93
Recall	v 0.96	0.87	0.93	0.94	0.91
F1-score	0.96	0.92	0.93	0.93	0.92
AUC	0.991	0.95	0.96	0.95	0.94

```
y_pred_classes = np.round(y_pred)

# create classification report
target_names = ['NORMAL', 'PNEUMONIA']
print(classification_report(y_test, y_pred_classes, target_names=target_names))
```

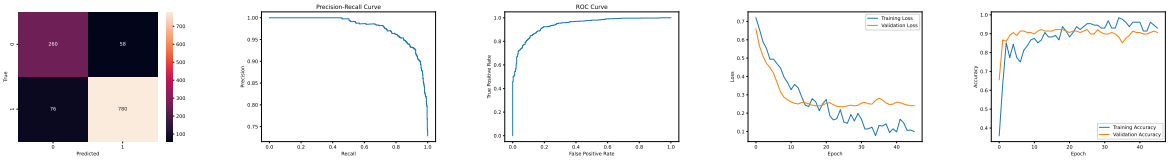
با انجام یک آزمایش دیگر به‌عنوان آزمایش چهارم (که مطابق با پارامترهای مقاله است) نتایج به شرح زیر است که نشان‌دهندهٔ عمل‌کرد مطلوب پیاده‌سازی مدل است:

```
37/37 [==============================] - 2s 43ms/step
              precision    recall  f1-score   support

      NORMAL       0.77      0.82      0.80       318
   PNEUMONIA       0.93      0.91      0.92       856

    accuracy                           0.89      1174
   macro avg       0.85      0.86      0.86      1174
weighted avg       0.89      0.89      0.89      1174
```

نتایج نموداری این آزمایش را هم در شکل ۱۰ نشان داده‌ایم.

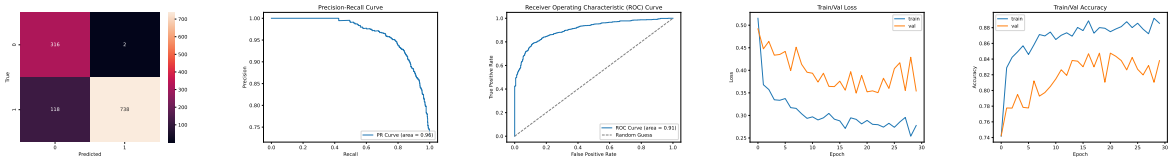


(آ) نمودار دقت (ب) نمودار اتلاف (ج) نمودار ROC (د) نمودار PR (ه) ماتریس درهم‌ریختگی

شکل ۱۰: نتایج پیاده‌سازی آزمایش چهارم.

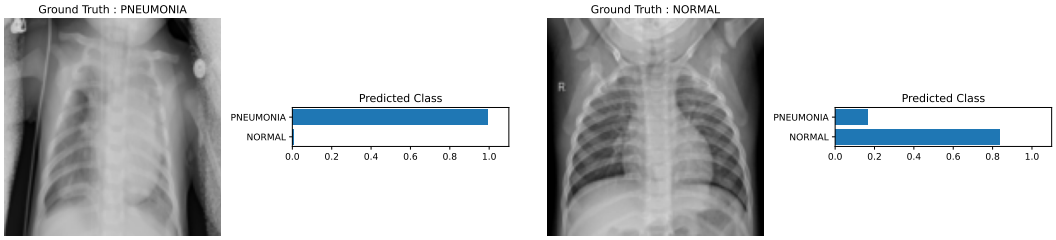
راه دوم – پیاده‌سازی سوال دوم با پایتورچ

برای سوال دوم، کد موجود در این مخزن گیت‌هاب را با توجه به خواسته‌های سوال تغییر داده و با اهداف سوال و مقاله مناسب‌سازی کردم. این پیاده‌سازی به‌عنوان راه‌حل دوم در این گوگل‌کولب در دسترس است. لازم به ذکر است که به‌دلیل علاقه‌مندی این مجموعه‌کد کاملاً فهمیده و بر اساس نیاز سوالات تغییراتی اساسی داده شده است (مثلاً برخی کلاس‌ها برای نمایش نمودارهای موردنیاز کاملاً تغییر داده شده‌اند)؛ اما چون استخوان‌بندی کلی آن از گیتهاب برداشته شده است از توضیحات بیش‌تر خودداری کرده‌ام. به هرحال برخی نتایج آن را در شکل ۱۱ و شکل ۱۲ آورده‌ام.



(آ) نمودار دقت (ب) نمودار اتلاف (ج) نمودار ROC (د) نمودار PR (ه) ماتریس درهم‌ریختگی

شکل ۱۱: نتایج پیاده‌سازی راه‌حل دوم.



(آ) نمونهٔ یک (ب) نمونهٔ دو

شکل ۱۲: نتایج پیاده‌سازی راه‌حل دوم.