یادگیری عمیق

پروژه پایانی

طبقه بندی تصاویر پزشکی کووید ۱۹

نیکی نزاکتی - بابک بهکام کیا
98521099 - 98522094

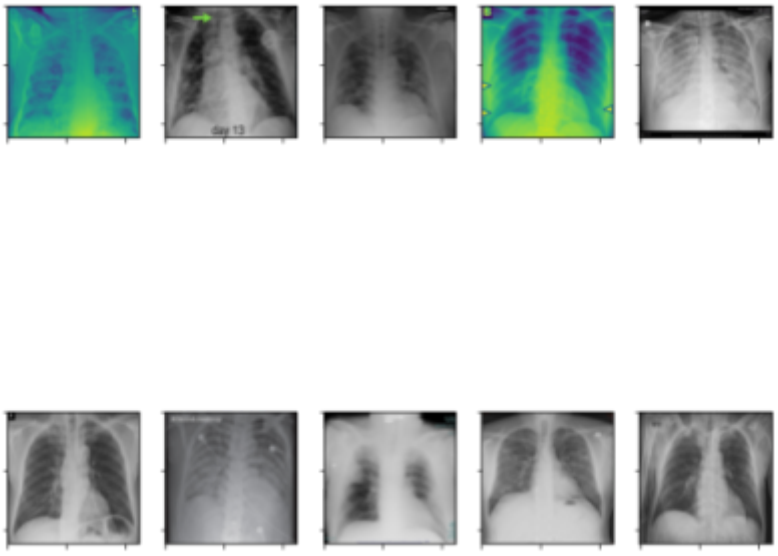
شرح پروژه

هدف از این پروژه انجام داده افزایی بر دیتاست مقاله “From Deep-COVID: Predicting COVID-19 Chest X-Ray Images Using Deep Transfer Learning” و پیاده سازی Transfer Learning بر معماری Resnet 18 برای انجام طبقه بندی تصاویر پزشکی کووید ۱۹ است.

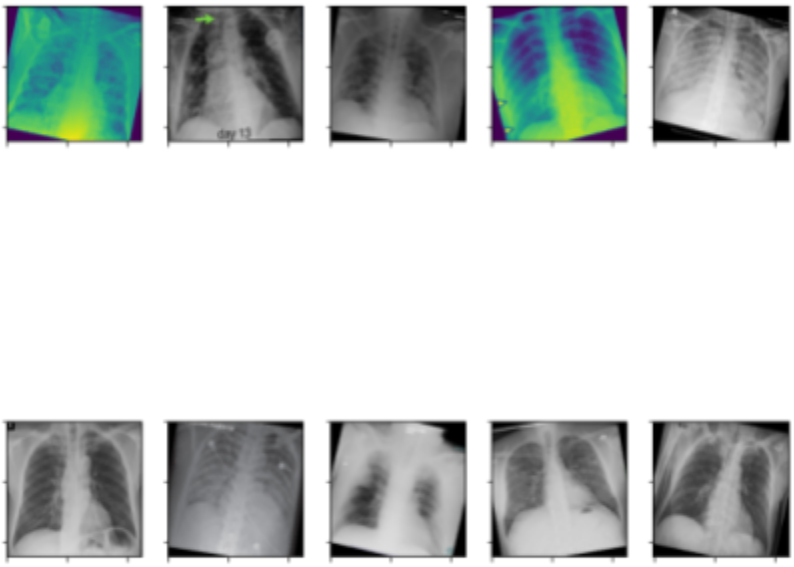
داده افزایی

در مقاله “Deep-COVID: Predicting COVID-19 From Chest X-Ray Images Using Deep Transfer Learning” از تصاویر اشعه ایکس قفسه سینه مجموعه داده ی COVID-Xray-5k که شامل 2048 تصویر آموزشی و 3100 تصویر آزمایشی است، استفاده شده است. این مجموعه داده خود شامل دو مجموعه داده می شود که یکی از آنها مجموعه داده ی Covid-Chestxray-Dataset است که حاوی مجموعه ای از تصاویر کووید-19 است. در ابتدا 84 تصویر برای بیماران دارای کووید-19 (covid) و 2000 تصویر برای بیمارانی که کووید-19 ندارند (non-covid)، را دارا هستیم. همانطور که مشخص است، تعداد داده های کووید-19 بسیار کم است. به همین دلیل از داده افزایی کمک میگیریم و از 4 نوع داده افزایی استفاده کرده و با خود داده ها، داده های آموزش کووید را 5 برابر می کنیم.

- برای انجام data augmentation ابتدا تصاویر را هم اندازه می کنیم و به اندازه ی ۲۲۴x۲۲۴ در می آوریم. برای این کار از transforms.Resize استفاده می کنیم.

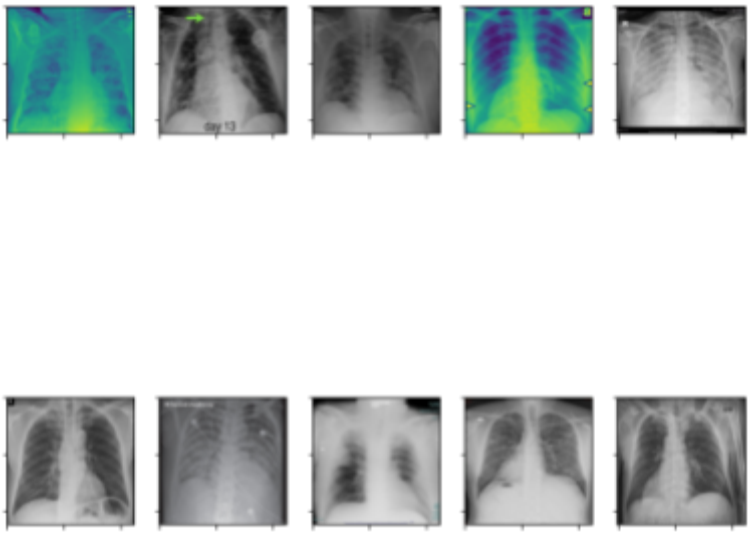


تصاویر تغییر سایز یافته



تصاویر چرخش یافته

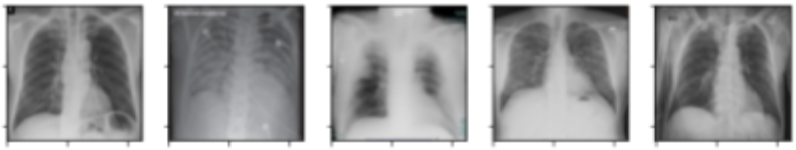
- سپس اولین داده افزایی به نام transforms_rotate را پیاده سازی می کنیم. در این داده افزایی از تابع RandomRotation استفاده می شود و تصویر را با یک زاویه ۱۵ درجه میچرخاند. این عملیات نیز توسط ماژول torchvision.transforms و با استفاده از transforms.RandomRotation انجام می شود.



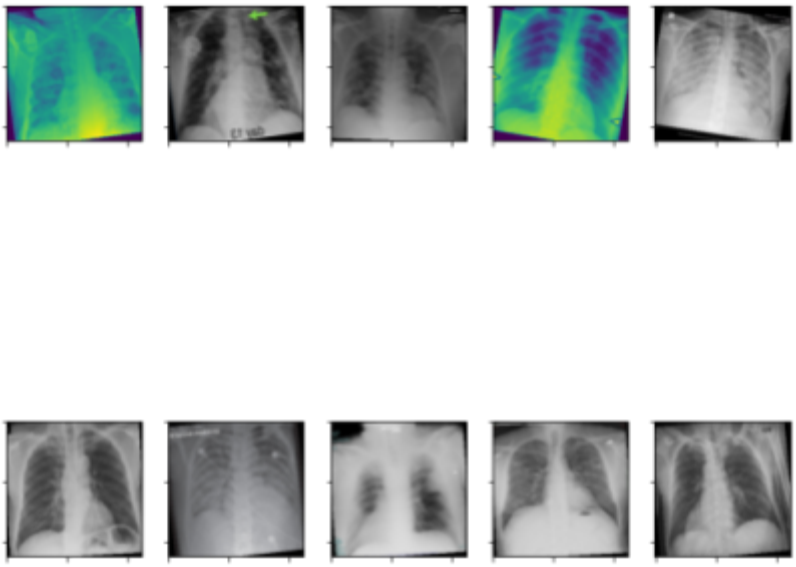
تصاویر flip یافته

- در ادامه از داده افزایی به نام transforms_flip استفاده کردیم و تصویر را با استفاده از torchvision.transforms.RandomHorizontalFlip تغییر می دهیم.

- در مرحله بعد از transforms.GaussianBlur با کرنل با اندازه ۳ و سیگما ۳ استفاده می کنیم که نویز گاوسی بر روی تصاویر اعمال میکند.



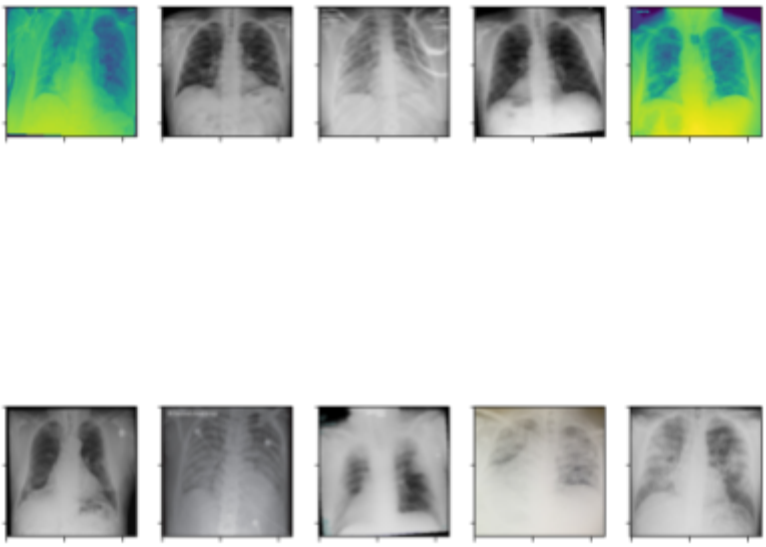
تصاویر نویزی شده



تصاویر حاصل از ۳ عملیات

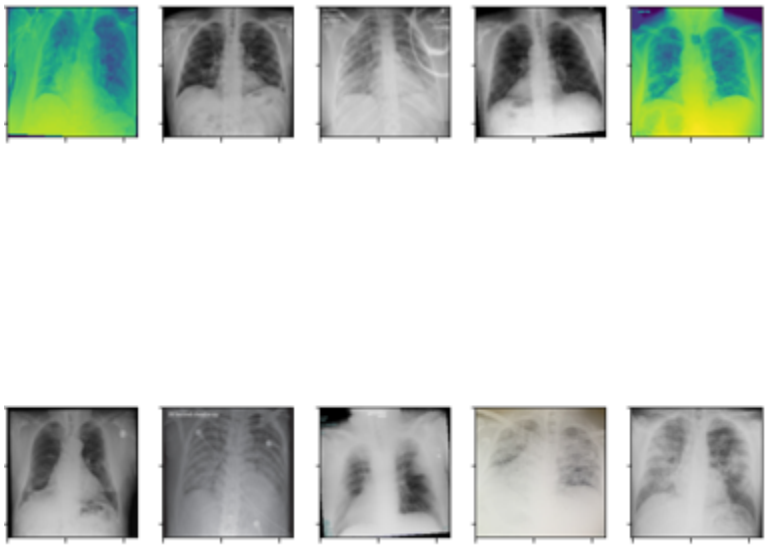
- در انتها از ترکیبی از سه داده افزایی استفاده شده در قبل، یعنی افزایش نویز، Horizontal Flip و چرخش استفاده می کنیم. این دادهافزایی را transforms_rotate نامیده و این بار از زاویه چرخش ۱۰ استفاده می کنیم.

سپس تصاویر حاصل از عملیات داده افزایی را به تصاویر تغییر سایز یافته اولیه اضافه کرده و shuffle می کنیم.



۱۰ نمونه از تصاویر نهایی حاصل از داده افزایی

در نهایت ۴۲۰ عکس جدید داریم که آن ها را به دیتاست آموزش کووید اضافه می کنیم. ۱۰ نمونه از آن ها به صورت زیر هستند:



۱۰ نمونه از تصاویر نهایی حاصل از داده افزایی

مدل:
به طور کلی در این پروژه ما از مدل Squeeznet که از قبل آموزش دیده است استفاده کردیم.

ابتدا همانند کدی که در مقاله 1 پیاده سازی شده است، transformer های مورد نیاز برای دیتا را تعریف می کنیم. این transformer ها شامل resize کردن و normal کردن و تبدیل ورودی به tensor است.

```
data_transforms = {
    'train': transforms.Compose([
        transforms.Resize(224),
        transforms.RandomResizedCrop(224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]) ,
    'test': transforms.Compose([
        transforms.Resize(224),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ])}
```

در مرحله بعد دیتاست از قبل آماده شده را در کولب load می کنیم و در ادامه DataLoader ها را تعریف می کنیم.
در ادامه کار، ابتدا امکان استفاده کردن از gpu را بررسی می کنیم و در صورت موفقیت، متغیر device را برابر cuda قرار می دهیم.

در مرحله بعد تابع train پیاده سازی شده در مقاله را در کولب اجرا می کنیم. در این تابع همان مراحل روتین pytorch برای آموزش مدل پیاده سازی شده اند.

در نهایت به بخش مدل می رسیم. ابتدا هایپر پارامتر ها را تعریف می کنیم.

hyper parameters

```
[16]  epochs = 30
      batch_size = 16
      dropout_rate = 0.5
```

در مرحله بعد squeezenet را در کولب load می کنیم و قابلیت آموزش پارامتر های بخش features این مدل را غیر فعال می کنیم.

```
model_conv = torchvision.models.squeezenet1_0(pretrained=True)
for param in model_conv.features.parameters():
    param.requires_grad = False
```

در ادامه لایه classifier این مدل را طبق گفته های آقای فتحی در گروه درس، تغییر می دهیم. در ادامه نیز دوباره کد مقاله برای این بخش را قرار می دهیم.

```
model_conv.classifier =nn.Sequential(
    nn.Dropout(p=dropout_rate),
    nn.Conv2d(512, 16, kernel_size=1),
    nn.ReLU(inplace=True),
    nn.AdaptiveAvgPool2d(output_size=(1, 1))
)



model_conv = model_conv.to(device)
criterion = nn.CrossEntropyLoss()

# Observe that only parameters of final layer are being optimized as
# opoosed to before.
optimizer_conv = optim.SGD(model_conv.classifier.parameters(), lr= 0.001, momentum= 0.9)

# Decay LR by a factor of 0.1 every 7 epochs
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_conv, step_size=7, gamma=0.1)


if __name__ == "__main__":
    model_conv, train_acc, valid_acc = train_model(model_conv, criterion, optimizer_conv,
                        exp_lr_scheduler, batch_size, num_epochs= epochs)
    model_conv.eval()
    torch.save(model_conv, './squeznet.pt')
```

نتایج:

در این بخش ابتدا تابعی برای محاسبه Sensitivity و Specificity، تعریف می کنیم.

$$\textbf{Sensitivity} = \frac{\#\text{Images correctly predicted as COVID-19}}{\#\text{Total COVID-19 Images}},$$
$$\textbf{Specificity} = \frac{\#\text{Images correctly predicted as Non-COVID}}{\#\text{Total Non-COVID Images}}. \quad (2)$$

همچنین تابع compute_prob را برای ذخیره نتایج مدل پیاده سازی کردیم تا بتوانیم با استفاده از تابع plot_probability_dist، توزیع تخمین های مدل را رسم کنیم.

تابع Confusion_Matrix برای رسم آن و تابع ROC_AUC برای رسم نمودار ROC پیاده سازی شده اند. قابل ذکر است که این دو تابع و تابع plot_probability_dist، از کد مقاله برداشته شده است.

مقادیر threshold را نیز با توجه به مقادیر ذکر شده در مقاله انتخاب کرده ایم.

Threshold	Sensitivity	Specificity
0.1	100%	89.9%
0.15	98%	92.9%
0.2	96.0%	94.6%
0.4	92%	97.6%
0.5	87%	98.3%

نتیجه ما:

Threshold	Sensitivity	Specificity
0.1	88%	99.5%
0.15	87%	99.6%
0.2	86%	99.7%
0.4	80%	99.8%
0.5	78%	99.9%