به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین سوم

نام و نام خانوادگی	فربد سیاهکلی – سعید شکوفا
شماره دانشجویی	810198418 - 810198510
تاریخ ارسال گزارش	1401.09.18

فهرست

شکل‌ها

جدول‌ها

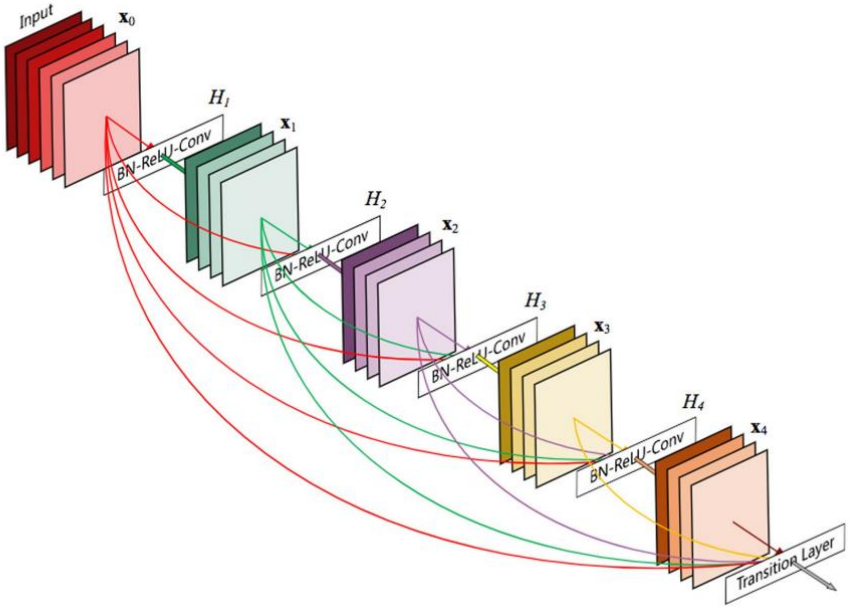
No table of figures entries found.

پاسخ 1. یادگیری انتقالی

۱-۱. آشنایی با یادگیری انتقالی transfer learning

$$8 + 0 = 0 \rightarrow 8\%4 = 0 \rightarrow densenet$$

1) این مقاله در زمان همه فراگیری ویروس کرونا منتشر شده است و هدف آن تشخیص سریع بیماری بدون استفاده از تست RT-PCT است. برای این کار از عکس های CT استفاده کزده است که این عکس ها از بیمارستانی در برزیل جمع آوری شده اند. عکس ها به صورت covidو non-covid لیبل زده شده اند و از data augmentation استفاده کرده(زیاد کردن دیتا بدون جمع آوری آن) و دیتا ها را بالانس کرده تا شبکه بایاس نداشته باشد. از شبکه densenet نیز استفاده کرده زیرا که در این شبکه از هر لایه تمام خروجی لایه های پیشین خود را به عنوان ورودی میگیرد و سبب می شود دقت بالایی داشته باشد. همچنین قبل از دادن عکس ها به شبکه آن ها را پردازش میکند به این صورت که آن ها رو rotate,zoom,flip (برای جلوگیری از overfitشدن)و همچنین سایز عکس ها را نیز با ورودی شبکه تنظیم میکند و سپس عکس ها را به شبکه داده و train میکند.
2) شبکه densenet از چندین لایه convoloutionبه همراه max pool استفاده میکند :



شکل 1 معماری شبکه

نکته قابل توجه در این شبکه این است که هر لایه کانولوشن از تمام لایه های قبل از خود ورودی میگرید و بر اساس آن ها خروجی را محاسبه میکند که یک مزیت است. در واقع تمام ورودی های ما قبل از هر لایه وارد آن لایه میشوند و شبکه قدرت بیشتری پیدا میکند البته این کار سبب می شود که با دادن ورودی به شبکه زمان بیشتری طول بکشد تا خروجی دهد و همچنین چون وزن ها بیشتر می شوند train کردن این شبکه نیز بیشتر طول خواهد کشید و دیتای بیشتری هم برای train شدن لازم دارد.همچنین در آخر شبکه densenet 1000 نورون وجود دارد که چون ما دو کلاس عکس داریم از یک linear 1000 به 2 استفاده میکنین و در اخر از loss function، soft max استفاده میکنیم تا شبکه عصبی را trainکنیم

پیش پردازش های لازم برای ورودی :

از آنجا که در دیتا های دانلود شده عکس های زیادی وجود دارد که هر کدام رزولوشن متفاوتی دارند و همچنین برخی از عکس ها در فرمت های متفاوتی غیر از RGB گرفته شده اند که تعداد کانال متفاوتی دارند(مثلا خاکستری هستند که 1 کانال دارند و یا موردی دیگر هستند که 4 کانال دارند) پس باید تمام عکس ها به RGB تبدیل شوند که سه کانال داشته باشند (طبق ورودی شبکه) همچنین رزولوشن عکس ها را نیز باید با ورودی شبکه تنظیم کرد که این کار ها اجباری است و باید انجام شود. پردازش های دیگری نیز به صورت رندوم روی عکس ها اعمال شده است مانند چرخش عکس ها ، فلیپ کردن و زوم کردن که برای جلوگیری از overfit شدن و بهتر train شدن شبکه انجام می شوند.

3) شبکه عصبی درست شده دو نورون خروجی دارد که یکی از نورون ها احتمال مثبت بودن covid را پیش بینی میکند و نورون دیگر منفی بودن covid (یا به عبارتی مثبت بودن non-covid) را پیش بینی میکند. پس در واقع دو دسته عکس را از هم تفکیک میکند که covid و non-covid می باشند. در صورتی که عکس داده شده به این شبکه در این دو کلاس نباشد باز هم شبکه دو خروجی احتمال را محاسبه کرده و هر کدام که احتمال بیشتری داشته باشد را به عنوان خروجی میدهد.

برای اینکه اگر عکس نامربوطه به شبکه داده شد شبکه خروجی covid یا non-covid ندهد می توان برای خروجی ها treshhold تعریف کرد یعنی اگر احتمال covid از مقدار معینی بیشتر بود آن گاه شبکه بگوید که این عکس مربوط به covid است و برای non-covid نیز مجددا همین کار را تکرار کرد. باید توجه کرد که مقدار معین treshhold باید به درستی انتخاب شود که شبکه قابلیت خود را در تشخیص از دست ندهد.

4) دیتاست را ازKaggle لود میکنیم :

```
[ ]  ! pip install -q kaggle
     from google.colab import files
     files.upload()
```

Choose Files kaggle.json

- **kaggle.json**(application/json) - 64 bytes, last modified: 11/30/2022 - 100% done

```
Saving kaggle.json to kaggle.json
{'kaggle.json': b'{"username":"sd3power","key":"3000c4b93d151ddd45b80bce853a84c5"}'}
```

```
! mkdir ~/.kaggle
! cp kaggle.json ~/.kaggle/
! chmod 600 ~/.kaggle/kaggle.json
! kaggle datasets list
```

```
!kaggle datasets download -d ssarkar445/covid-19-xray-and-ct-scan-image-dataset
!unzip covid-19-xray-and-ct-scan-image-dataset.zip
```

شکل 2 لود دیتاست از روی کگل بر روی کولب

دیتا را unzip کرده و تمام ادرس های covid وnon-covid را در دو لیست به همین نام ها قرار میدهیم زیرا که نمی شود تمام عکس ها را import کرد و اگر این کار را انجام دهیم خطای حافظه رخ می دهد پس ادرس همه عکس ها را نگه میداریم و موقع train شبکه 64 تا 64 تا عکس ها را باز میکنیم و به شبکه میدهیم تا trainشود.

```
train=torch.utils.data.DataLoader(train_loader,64,True)
test=torch.utils.data.DataLoader(test_loader,128,False)
```

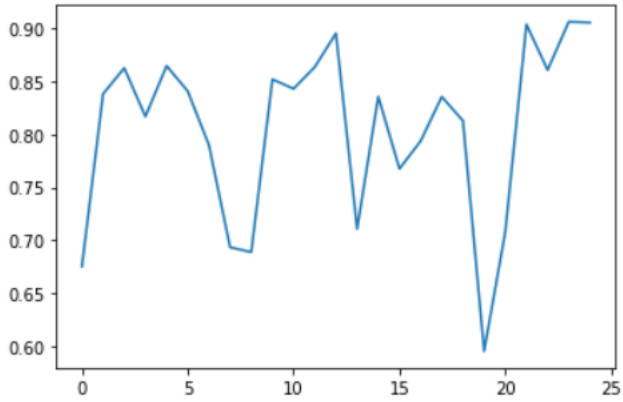
شکل 3 ساخت دیتالودر برای دیتای آموزش و تست

5) شبکه پیاده سازی شد وtrain شد :

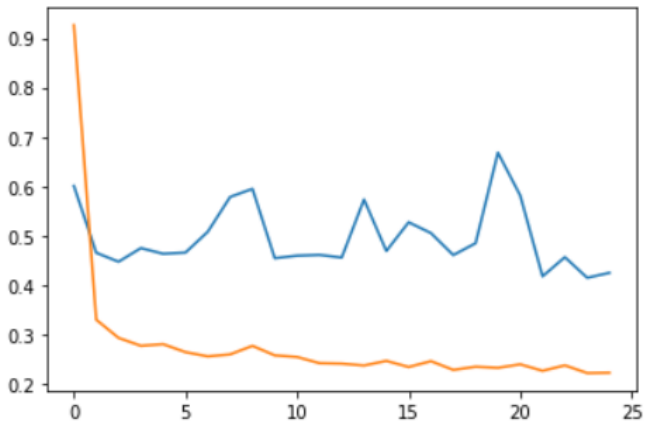
```
den=torchvision.models.densenet121(pretrained=True)
device=torch.device("cuda" if torch.cuda.is_available() else "cpu")
model=nn.Sequential(
                    den,
                    nn.ReLU(),
                    nn.Linear(1000,2))
model.to(device)
optim=torch.optim.Adam(model.parameters(),lr=0.01)
```

شکل 4 تعریف مدل و اپتیمایزر

منحنی های خواسته شده در داده های test :



شکل 5 نمودار **test accuracy**



شکل 6 نمودار **test loss**و **train loss**

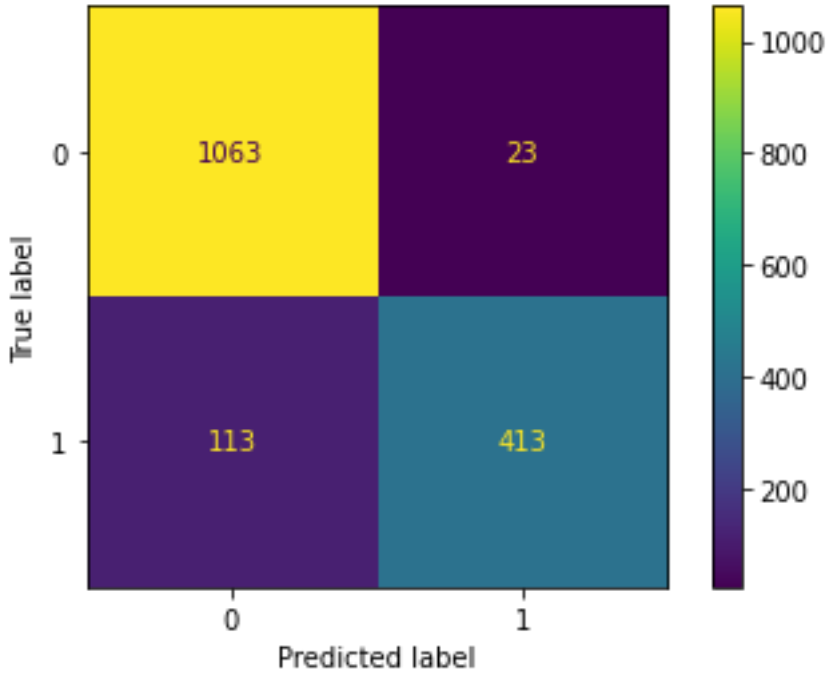
نمودار آبی مربوط به داده های تست و نمودار نارنجی مربوط به داده های **train** می باشد

نتایج بهترین مدل به دست آمده(برای داده های تست):

```
epoch 23 train loss : 0.22140486331859438
epoch 23 test loss : 0.41443236745320833
epoch 23 F1 : 0.9064713716506958
epoch 23 Acc : 0.9064713716506958
epoch 23 Pre : 0.9064713716506958
epoch 23 Model Saved
```

شکل 7 نتایج بهترین مدل

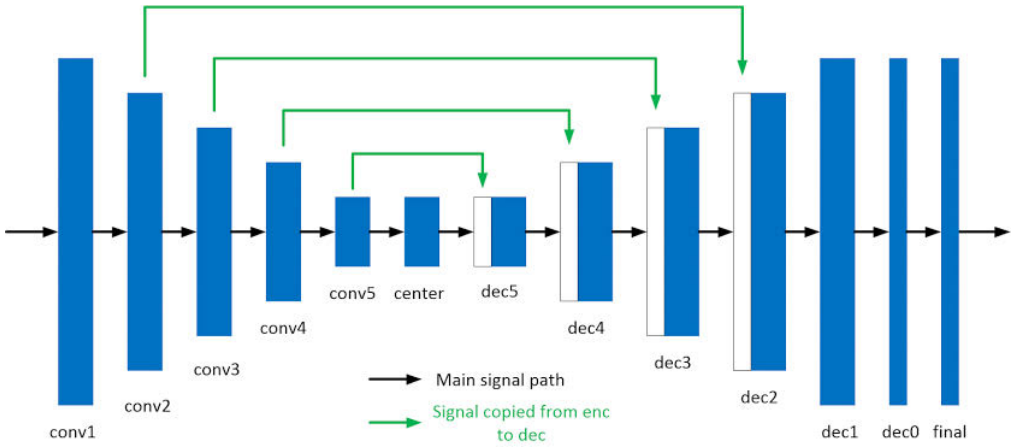
شکل ماتریس Confusion نیز به صورت زیر می شود که نشان می دهد شبکه از دقت و صحت قابل قبولی برخوردار است:



شکل 8 ماتریس **Confusion** شبکه

پاسخ ۲ – تشخیص چهره مسدود شده

1) این شبکه (PSP Net) در واقع یک encoder و یک decoder دارد ولی به جای اینکه شبکه mlp این کار را انجام دهد با convoloution این کار انجام شده است و encoderو decoder در واقع لایه های کانولوشنی هستند.



شکل 9 نمای شبکه **PSPNet**

همانطور که دیده می شود علاوه بر encode و decoder از لایه های encode بع لایه های decodeمی بریم که بتوان اشیا را به طور دقیق تشخیص داد.

2) بله تفاوت وجود دارد و به میزان شفافیت object بستگی دارد برای مثال عینک طبی با دست متفاوت خواهد بود و همچنین به میزان بزرگی و رنگ objectنیز بستگی دارد.
3) کلاس بندی داده ها به (NatOccوRandOcc) این است که میان عکس هایی که تولید شده اند و با عکس های دنیای واقعی مقایسه کنند و برای train کردن شبکه تفاوتی ایجاد نمیکند و همچنین randOcc لازم است چرا که برخی از unseen ها در دنیای واقعی را پوشش می دهد. ولی تقسیم کردن دیتاrandOcc به دو دسته ساده و wild بهتر است چرا که شبکه بهتر train میشود بر اساس اینکه داده شفاف است یا کدر.
4) بهتر است از شبکه های Feature-based methods استفاده کنیم چرا که لزومی ندارد وقتی تفاوت ها چشمگیر هستند از پارتیشن بندی استفاده کنیم.

(5

Table 4. **Overall Performance:** Results of PSPNet [39], DeepLabv3+ [9] and SegFormer [35] with different combination of datasets. The best results for each validation set are marked in bold. The metrics are mIoU (higher is better).

	Quantity	**RealOcc** (mIoU)			**COFW (Train)** (mIoU)			**RealOcc-Wild** (mIoU)		
		PSPNet	DeepLabv3+	SegFormer	PSPNet	DeepLabv3+	SegFormer	PSPNet	DeepLabv3+	SegFormer
C-Original	29,200	89.52	88.13	88.33	89.64	88.62	91.36	85.21	82.05	85.24
C-CM	29,200	96.15	96.13	97.42	91.82	92.77	**94.87**	91.33	91.01	95.16
C-WO	24,602	89.38	89.01	91.36	89.53	88.97	92.24	83.86	84.14	86.72
C-WO + C-WO-NatOcc	24,602 + 49,204	96.65	96.51	97.30	90.71	91.21	94.30	91.34	91.70	94.17
C-WO + C-WO-NatOcc-SOT	24,602 + 49,204	96.35	96.59	97.18	**92.32**	91.74	93.55	**93.26**	92.69	94.27
C-WO + C-WO-RandOcc	24,602 + 49,204	95.09	95.21	96.53	90.82	91.35	93.14	89.54	89.68	92.84
C-WO + C-WO-Mix	24,602 + 73,806	96.55	96.66	97.37	90.99	91.20	93.74	92.14	91.84	94.40
C-CM + C-WO-NatOcc	29,200 + 49,204	**97.28**	**97.33**	97.95	91.61	92.66	94.86	92.13	**93.81**	**95.43**
C-CM + C-WO-NatOcc-SOT	29,200 + 49,204	97.17	97.29	**98.02**	92.07	**92.91**	94.60	92.84	93.73	94.53

شکل 10 مقایسه کارایی **PSPNet** و **DeepLAB**

همانطور که دیده می شود این دو شبکه کارایی نزیک به هم دارند و برای دیتا ست های متفاوت نتیجه های متفاوتی می دهند و به نظر می رسد که با ترکیب دیتا ها شبکه DeepLab کارکرد بهتری دارد ولی استفاده از دیتا ست های مجزا شبکه PSPNet کارکرد بهتری داشته و بسته به دیتا باید نوع شبکه انتخاب شود.

.

پاسخ ۳ – تشخیص بلادرنگ اشیا

1)

برای شخصی سازی داده ها ابتدا نیاز به annotate کردن آنها در فرمت yolo داریم. بدین منظور می توان از پلتفرم هایی مانند Makesense.ai استفاده کرد که خروجی به فرمت یولو برای هر عکس ورودی خواهند داد. سپس لازم است تا دیتا را به سه دسته train/test/val تقسیم کنیم. همین تقسیم بندی منطقا باید برای فایل های annotation نیز انجام شود.

2)

حال نیاز است تا یک فایل با پسوند yaml ساخته شود تا در آن آدرس فولدرها و همچنین لیستی از اسامی لیبل ها قرار گیرد. فایل yaml برای این مسئله به صورت زیر تعریف شده است.

```yaml
train: /content/YOLOv6/images/train
val: /content/YOLOv6/images/val
test: /content/YOLOv6/images/test

nc: 12
names: ['black-bishop', 'black-king', 'black-knight', 'black-pawn', 'black-queen', 'black-rook', 'white-bishop',
        'white-king', 'white-knight', 'white-pawn', 'white-queen', 'white-rook']
```

شکل 11 تعریف فایل **yaml** مختص دیتاست شخصی

برای اجرای آموزش شبکه یولو بر روی دیتاست مخصوص، کافیست تا از Script پایتون train.py استفاده کرده و در آن مقادیر زیر به عنوان ورودی تعریف شوند:

- Batch size
- Input size
- Number of epochs
- Address to the .yaml file
- Type of model

```
# run this cell to begin training
!python tools/train.py --batch 32 --conf /content/YOLOv6/configs/yolov6m.py --epochs 100 --img-size 416 --data /content/YOLOv6/data.y
```

شکل 12 آموزش شبکه با اسکریپ **train.py** و تعیین هایپرپارامترهای مربوطه

پس از اجرا شبکه شروع به یادگیری کرده و با اسکریپ eval.py نیز میتوان ارزیابی شبکه را انجام داد.

پس از صد و پنجاه ایپاک ترین مدل را ذخیره کردیم .

```
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.522
 Average Precision  (AP) @[ IoU=0.50       | area=   all | maxDets=100 ] = 0.748
 Average Precision  (AP) @[ IoU=0.75       | area=   all | maxDets=100 ] = 0.661
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.578
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.524
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.449
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.730
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.731
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.711
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.728
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
Results saved to runs/train/exp1
Epoch: 148 | mAP@0.5: 0.7482461064814664 | mAP@0.50:0.95: 0.5223456287864977

     Epoch  iou_loss  dfl_loss  cls_loss
   149/149    0.4601    0.4262    0.8295: 100% 7/7 [00:04<00:00,  1.52it/s]
Inferencing model in train datasets.: 100% 1/1 [00:02<00:00,  2.08s/it]
```

شکل 13 نتیجه نهایی آموزش بر روی **YOLOv6m**

همانطور که دیده میشود مقدارloss شبکه در تصویر بالا آورده شده است. که با افزایش تعداد ایپاک شبکه همچنان نیز میتوانست به دقت بهتری برسد.

3) نمونه هایی از inference شبکه آموزش دیده شده بر روی دیتای تست: بدین منظور از اسکریپت infer.py استفاده شد.

شکل 14 خروجی شبکه

شکل 15 خروجی شبکه **YOLOv6**

فایل خروجی اینفرنس برای تمامی عکس های تست نیز **به پیوست** قرار گرفته است.